Université de Paris

École doctorale 386 Sciences Mathématiques de Paris Centre

Institut de Recherche en Informatique Fondamentale

Classical and Quantum Cryptanalysis for Euclidean Lattices and Subset Sums

Par: Yixin Shen

Thèse de doctorat d'informatique

Dirigée par Frédéric Magniez

Présentée et soutenue publiquement le 11 Mai 2021

Devant un jury composé de :

Frédéric Magniez	CNRS, Université de Paris	Directeur
Elham Kashefi	CNRS, Sorbonne Université	Rapporteuse
Damien Stehlé	Ecole Normale Supérieure de Lyon	Rapporteur
Léo Ducas	Centrum Wiskunde & Informatica	Examinateur
Michele Mosca	University of Waterloo	Examinateur
María Naya-Plasencia	Inria de Paris	Examinatrice
Simon Perdrix	CNRS, Université de Lorraine	Examinateur
Jean-Pierre Tillich	Inria de Paris	Président

To My Mum

Remerciements

Je tiens d'abord à remercier Frédéric Magniez, mon directeur de thèse, qui m'a accompagnée pendant ces trois dernières années avec beaucoup de patience et de pédagogie. Merci pour tout le temps que tu m'as consacré malgré ton rôle super occupé de directeur du labo et merci pour tous les bons conseils que tu m'as donnés. Ces trois dernières années ont été une expérience très enrichissante scientifiquement et humainement pour moi. J'espère que tu les as appréciées tout autant que moi.

I would like to thank Elham Kashefi and Damien Stehlé for accepting the heavy task of reviewing my thesis. A special thank to Damien, who read all the details thoroughly and found several small issues. I would also like to thank Léo Ducas, Michele Mosca, María Naya-Plasencia, Simon Perdrix, Jean-Pierre Tillich for accepting to be part of the jury and for asking a lot of hard but really interesting questions during my defense.

I want to thank my coauthors, especially those with whom I collaborated closely: Divesh Aggarwal, Xavier Bonnetain, Rémi Bricout, Yanlin Chen, Kamil Khadiev, Rajendra Kumar, Phong Q. Nguyen and André Schrottenloher. You have been an inspiration to me and I hope I served as an example to you as much as you have been an example for me. Un remerciement particulier à André et à Xavier avec qui j'ai passé de bons moments à Shenzhen, en Australie et à Dagstuhl pendant lesquels on a pu beaucoup discuter de la recherche et d'autres sujets et tester beaucoup de super bons restaurants.

During my thesis, I had some wonderful occasions to travel around the world and meet new people. I would like to thank in particular Zhengfeng Ji and Divesh Aggarwal for inviting me to visit them in Sydney and in Singapore, Claude Crépeau for inviting me to his annual crypto workshop in Barbados and María Naya-Plasencia for inviting me to the Dagstuhl quantum cryptanalysis workshop.

J'ai également eu l'occasion, pendant mon stage de master 1 et ma thèse, de passer au total un an au laboratoire JFLI au Japon. Je remercie Phong Q. Nguyen d'avoir rendu ces visites possibles qui m'ont permis de découvrir ce pays magnifique et d'apprendre sa langue et sa riche culture. Je remercie en particulier mes collègues de labo Thomas Silverston et Vincent Nozick qui m'ont fait découvrir beaucoup d'endroits extraordinaires à Tokyo et qui m'ont également donné beaucoup de conseils utiles pour ma future carrière de recherche.

Ces années auraient été différentes sans l'ambiance extraordinaire de mon laboratoire IRIF. Merci à Simona Etinski, Zeinab Galal, Sander Gribling, Alex Grilo, Yassine Hamoudi, Jonas Landman, Baptiste Louf, Alessandro Luongo, Etienne Mallet, Simon Mauras, Alexandre Nolin, Ami Paz, Mikaël Rabie, Olivier Stietel, Anupa Sunny, Daniel Szilagyi, Zhouningxin Wang pour tous les moments sympathiques passés ensemble au coin café du quatrième étage avec ou sans des jeux de cartes dans les mains.

This thesis would also be less fun if I didn't encounter all the friends that I met during the

conferences and the summer schools. Nick Spooner, Arjan Cornelissen, Miruna Rosca, Radu Titiu, Stephan Gocht, Joseph Swernofsky, Christian Majenz, Alexandru Cojocaru, Hamoon Mousavi, Vitor Pereira, Gilyén András, for each of you I have a story to tell. I hope I can see all of you again soon in a post Covid world.

Je souhaite également remercier ceux grâce à qui j'ai pris le chemin de la thèse, en particulier mes professeurs de mathématiques et d'informatique de classe préparatoire Anne-Laure Biolley, Bernard Randé et Roger Mansuy. Vous m'avez enseigné la rigueur et m'avez fait acquérir une base scientifique solide indispensable à la réalisation de cette thèse. Grâce à vos passions et vos empathies, mes premières années seule en France ont également été moins difficiles.

En dehors de la recherche, le badminton a été une partie de ma vie indispensable pour maintenir mon équilibre mental pendant la thèse. J'ai pu rencontrer beaucoup d'amis au sein du CPS10 et en dehors. J'aimerais en particulier remercier Ying et Thierry pour les bons restaurants découverts ensemble, Henri-Frédéric pour de nombreux débats sur la géopolitique au tour d'un verre sur le quai du canal, même si on n'est pas toujours d'accord. J'aimerais aussi remercier Isabelle, Alizée, Adèle et Anne-Sophie pour les tournois de badminton remportés ensemble.

En dehors de tous ces cadres, je remercie Pierre-Cyril Aubin, Cyril Bernard, Xavier Bonnetain, Dexiong Chen, Ana-Maria Cretu, Christine Finas, Antoine Doche, Shuai Fang, Arnaud Gallant, Yunzhi Gao, Manuel Gaulhiac, Antony Goh, Jérémy Goh, Amal Hamdani, Daniel Haziza, Chia-Man Hung, Jade Jiang, Ezékiel Kahn, Houzhi Li, Hugo Marival, Adina Nedelcu, André Schrottenloher, Maud Szusterman, Xavière de Truchis, Joseph-André Turk, Bogdan Ursu, Aurélien Velleret, Yang Yu, Yiyang Yu, Xiangpeng Zhang, Alexandre Zhou pour les amitiés que vous m'avez portées toutes ces années.

Amaury, tu es entré dans ma vie pendant cette thèse. Tu es une inspiration constante qui me montre quotidiennement comment être un bon chercheur. Merci d'être un oracle si fort qui me stimule intellectuellement ainsi que de ton aide dans beaucoup de mes travaux. Merci pour ton soutien dans la vie quotidienne et d'avoir partagé beaucoup de mon stress pendant cette période pas évidente. Et surtout merci pour ton amour infaillible qui m'a rendu une meilleure personne. Nous avons encore un long chemin à parcourir ensemble.

Last but not least, I would like to thank Martin R. Albrecht for hiring me as a postdoc. I hope we will establish a lot of fruitful collaborations in the years to come.

Cette thèse conclue une longue période de ma vie passée en France (11 ans). Je remercie tous ceux rencontrés sur le chemin qui m'ont permis de m'améliorer et de mieux me connaître moi-même. Merci pour toutes les mémoires partagées ensemble, je les porterai avec moi dans les aventures à venir.

Résumé

La cryptographie post-quantique est une branche de la cryptographie qui vise à concevoir des systèmes cryptographiques non quantiques (c'est-à-dire classiques), qui sont protégés contre un adversaire possédant un ordinateur quantique. Dans cette thèse, nous nous concentrons sur l'étude de deux problèmes fondamentaux pour la cryptographie post-quantique : le problème du plus court vecteur (SVP) et le problème de la somme de sous-ensembles aléatoires.

Le SVP demande de trouver le plus court vecteur non nul d'un réseaux euclidien donné. Il sert de jauge pour quantifier la sécurité de la cryptographie reposant sur les réseaux euclidiens, qui est considérée comme prometteuse pour l'ère post-quantique. Les principales approches pour résoudre le SVP sont les algorithmes de tamisage, qui utilisent un temps et un espace exponentiels, et les algorithmes d'énumération, qui utilisent un temps superexponentiel et un espace polynomial. Même si les algorithmes de tamisage sont asymptotiquement les algorithmes connus les plus rapides pour le SVP, la complexité en mémoire, en grande dimension, a historiquement été un facteur limitant pour exécuter ces algorithmes. Certains travaux récents ont montré comment utiliser de nouvelles astuces pour rendre possible l'utilisation du tamisage sur des réseaux euclidiens à haute dimension dans la pratique et bénéficier de leur temps de fonctionnement efficace. Néanmoins, la question de l'obtention d'un algorithme qui réalise le « meilleur des deux mondes », c'est-à-dire un algorithme qui fonctionne en temps exponentiel et ne nécessite qu'une quantité polynomiale de mémoire, reste ouverte depuis longtemps. En l'absence d'un tel algorithme, il est souhaitable d'avoir un compromis entre le temps et la quantité de mémoire requise qui interpole entre les meilleurs algorithmes de tamisage actuels et les meilleurs algorithmes d'énumération actuels.

Dans cette thèse, nous donnons tout d'abord un compromis temps-mémoire prouvable qui interpole approximativement entre les garanties de ressources des algorithmes de tamisage et celles des algorithmes d'énumération. Nous montrons ensuite l'algorithme quantique prouvable le plus rapide connu qui résout le SVP en temps $2^{0.9535n+o(n)}$ et en mémoire classique $2^{0.5n+o(n)}$ avec seulement un nombre $\mathrm{poly}(n)$ de qubits. Nous montrons également le meilleur algorithme prouvable classique connu dont la complexité en mémoire est de $2^{0.5n+o(n)}$.

Quant aux algorithmes d'énumération, on pensait auparavant qu'ils pouvaient bénéficier d'une accélération quadratique quantique grâce à l'algorithme de Grover. Nous montrons que cette accélération quadratique peut effectivement être obtenue pour l'algorithme d'énumération de base et ses variantes d'élagage cylindrique et discret, mais avec des algorithmes quantiques plus sophistiqués tels que la marche quantique sur les arbres. Notre accélération quantique s'applique également à l'élagage extrême où l'on répète le processus d'énumération sur plusieurs bases réduites : une approche naïve ne ferait que réduire le coût classique de mt (où m est le nombre de bases et t est le coût d'un seul processus d'énumération) à $m\sqrt{t}$ opérations quantiques, mais nous le ramenons à $\sqrt{mt}$.

Le problème de la somme de sous-ensembles aléatoires sous-tend également certains schémas cryptographiques visant à la sécurité post-quantique, bien qu'il soit principalement de nature académique. Dans un contexte de cryptanalyse, il sert souvent d'outil car de nombreux problèmes sur lesquels se fonde la cryptographie moderne peuvent être exprimés comme des variantes (vectorielles) du problème de la somme de sous-ensembles. Plus récemment, il a également été démontré qu'il peut être utilisé comme élément de base dans certains algorithmes de décalage caché quantique, qui ont des applications dans la cryptanalyse quantique de schémas symétriques et de schémas reposant sur les isogénies.

Dans cette thèse, nous présentons de nouveaux algorithmes classiques et quantiques pour la résolution du problème de la somme de sous-ensembles aléatoires. Nous obtenons d'abord l'algorithme classique le plus rapide connu en temps $\widetilde{O}\left(2^{0.283n}\right)$. Ensuite, nous améliorons l'état de l'art des algorithmes quantiques pour ce problème dans plusieurs directions. Nous concevons un algorithme avec un temps asymptotique $\widetilde{O}\left(2^{0.236n}\right)$, avec l'avantage d'utiliser une mémoire *classique* avec accès aléatoire quantique. Les algorithmes connus précédemment utilisaient des marches quantique, et nécessitaient une mémoire *quantique* avec accès aléatoire quantique. Nous proposons également des marches quantiques plus rapides pour ce problème en temps $\widetilde{O}\left(2^{0.216n}\right)$. Ce temps dépend d'une heuristique concernant le temps de mise à jour dans les marches quantiques. Les algorithmes précédents pour ce problème dépendaient eux aussi de cette heuristique. Nous montrons comment surmonter partiellement cette heuristique, et nous obtenons un algorithme quantique de temps $\widetilde{O}\left(2^{0.218n}\right)$ ne nécessitant que les heuristiques standard du problème de la somme de sous-ensembles aléatoires.

Mots clefs: Complexité de calcul, Algorithmes quantiques, Cryptographie post-quantique, Cryptographie basée sur les réseaux euclidiéns, Problème du plus court vector, Problème de la somme de sous-ensembles

Abstract

Post-quantum cryptography refers to a branch of cryptography aimed at designing non-quantum (ie. classical) cryptographic systems which are secure against an adversary having a quantum computer. In this thesis we focus on the study of two fondamental problems for post-quantum cryptography: the Shortest Vector problem (SVP) and the random Subset-Sum problem.

The SVP asks to find the shortest non-zero vector of a given lattice. It serves as a gauge to quantify the security of lattice-based cryptography which is widely considered to be promising for the post-quantum era. The main approaches to solve the SVP are the sieving algorithms, which use exponential time and space, and the enumeration algorithms, which use superexponential time and polynomial space. Even though sieving algorithms are asymptotically the fastest known algorithms for SVP, the memory requirement, in high dimension, has historically been a limiting factor to run these algorithms. Some recent works have shown how to use new tricks to make it possible to use sieving on high-dimensional lattices in practice and benefit from their efficient running time. Nevertheless, it has been a long standing open question to obtain an algorithm that achieves the "best of both worlds", i.e. an algorithm that runs in exponential time and requires only polynomial amount of memory. In the absence of such an algorithm, it is desirable to have a smooth tradeoff between time and memory requirement that interpolates between the current best sieving algorithms and the current best enumeration algorithms.

In this thesis, we first give a provable time memory trade-off which roughly interpolates between the resource guarantees of sieving algorithms and those of enumeration algorithms. We then show the fastest state-of-the-art provable quantum algorithm that solves SVP using $2^{0.9535n+o(n)}$ time and requires $2^{0.5n+o(n)}$ classical memory but only poly(n) qubits. We also provide the fastest classical algorithm in the provable setting whose memory is $2^{0.5n+o(n)}$.

As for the enumeration algorithms, it was previously commonly believed that they can benefit from a quantum quadratic speed-up using Grover's algorithm. We show that the quadratic speed-up can indeed be achieved for lattice enumeration and its cylinder and discrete pruning variants, but with more sophisticated quantum algorithms such as quantum walk on trees. Our quantum speed-up applies further to extreme pruning where one repeats enumeration over many reduced bases: a naive approach would only decrease the classical cost mt (where m is the number of bases and t is the number of operations of a single enumeration) to $m\sqrt{t}$ quantum operations, but we bring it down to $\sqrt{mt}$.

The random subset-sum problem also underlies some cryptographic schemes aiming at post-quantum security, although it is mostly of academic nature. In a cryptanalysis context, it often serves as a cryptanalytic tool as many problems on which modern cryptography is build can be expressed as some (vectorial) versions of the subset sum problem. More

recently it is also shown that it can be used as a building block in some quantum hidden shift algorithms, which have applications in quantum cryptanalysis of isogeny-based and symmetric cryptographic schemes.

In this thesis, we present new classical and quantum algorithms for solving random subset-sum instances. We first show the fastest state-of-the-art classical algorithm using $\widetilde{O}\left(2^{0.283n}\right)$ time. Next, we improve the state of the art of quantum algorithms for this problem in several directions. We devise an algorithm with asymptotic running time $\widetilde{O}\left(2^{0.236n}\right)$, with the advantage of using *classical* memory with quantum random access. The previously known algorithms all used the quantum walk framework, and required *quantum* memory with quantum random access. We also propose faster quantum walks for subset-sum with time complexity $\widetilde{O}\left(2^{0.216n}\right)$. This time is dependent on a heuristic on quantum walk updates that is also required by the previous algorithms. We show how to partially overcome this heuristic, and we obtain a quantum algorithm with time $\widetilde{O}\left(2^{0.218n}\right)$ requiring only the standard subset-sum heuristics.

Key-words: Computational Complexity, Quantum algorithms, Post-quantum cryptography, Lattice-based crytography, Shortest Vector Problem, Random Subset-Sum Problem

Contents

List of Publications

[ACKS21] Divesh Aggarwal, Yanlin Chen, Rajendra Kumar, and Yixin Shen. Improved (provable) algorithms for the shortest vector problem via bounded distance decoding. In Markus Bläser and Benjamin Monmege, editors, *38th International Symposium on Theoretical Aspects of Computer Science, STACS 2021, March 16-19, 2021, Saarbrücken, Germany (Virtual Conference)*, volume 187 of *LIPIcs*, pages 4:1–4:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.

[BBSS20] Xavier Bonnetain, Rémi Bricout, André Schrottenloher, and Yixin Shen. Improved classical and quantum algorithms for subset-sum. In Shiho Moriai and Huaxiong Wang, editors, *Advances in Cryptology - ASIACRYPT 2020 - 26th International Conference on the Theory and Application of Cryptology and Information Security, Daejeon, South Korea, December 7-11, 2020, Proceedings, Part II*, volume 12492 of *Lecture Notes in Computer Science*, pages 633–666. Springer, 2020.

[ABI+20] Andris Ambainis, Kaspars Balodis, Janis Iraids, Kamil Khadiev, Vladislavs Klevickis, Krisjanis Prusis, Yixin Shen, Juris Smotrovs, and Jevgenijs Vihrovs. Quantum lower and upper bounds for 2d-grid and dyck language. In Javier Esparza and Daniel Král', editors, *45th International Symposium on Mathematical Foundations of Computer Science, MFCS 2020, August 24-28, 2020, Prague, Czech Republic*, volume 170 of *LIPIcs*, pages 8:1–8:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.

[ANS18] Yoshinori Aono, Phong Q. Nguyen, and Yixin Shen. Quantum lattice enumeration and tweaking discrete pruning. In Thomas Peyrin and Steven D. Galbraith, editors, *Advances in Cryptology - ASIACRYPT 2018 - 24th International Conference on the Theory and Application of Cryptology and Information Security, Brisbane, QLD, Australia, December 2-6, 2018, Proceedings, Part I*, volume 11272 of *Lecture Notes in Computer Science*, pages 405–434. Springer, 2018.

Chapter 1 Introduction

Cryptography is the study of techniques for secure communication in the presence of malicious third parties. One of the main goals of cryptography is to guarantee the confidentiality of messages, by encrypting them before transmission. Historically, the first encryption procedures belong to the category of symmetric key encryption schemes, where the sender and the receiver share a common key to encrypt and decrypt the message. To do so, they either need to meet in person, or use a trusted third party, before being able to communicate safely. These requirements are clearly too restrictive for today's use of cryptography. Fortunately, Diffie and Hellman invented the idea of public key cryptographic in 1976 [DH76]. The principle is that each user now has its own secret key, which it keeps for itself, but it also has a public key, which it makes publicly available. The public key enables others to encrypt a message that only the user with the corresponding secret key will be able to decrypt. Public key encryption schemes are usually far less efficient than symmetric key ones. In today's cryptographic communication protocols, a public key scheme is typically used at the beginning of the conversation to exchange a shared secret symmetric key between two users. A symmetric key scheme is used with this shared secret symmetric key for the rest of the conversation. The security of public key encryption schemes is typically based on some mathematical assumptions, i.e. under the assumption that some mathematical problems are hard to solve, one can prove that no attacker (for a certain model of attacker) can recover the underlying message if it only knows the encrypted message and the public key. For the Diffie-Hellman protocol mentioned above, the hard problem that it relies on is the discrete logarithm problem. The presumed hardness of this problem (in practice on elliptic curves) has since been used for many other cryptographic constructions. Another very important problem for cryptography is integer factorization. The famous RSA encryption scheme that we are still using today is based on a problem related to it.

In the 1970s and 80s, the notion of quantum computer was suggested by several physicists such as Richard Feynman, Paul Benioff, David Deutsch or Charles H. Bennett. Feynman's idea was that using a quantum computer to simulate quantum phenomenon could be much more suitable than current computers, for which this problem is insurmountable. This idea remained ignored for a long time, as the realization of such a computer seemed improbable, and the applications were restricted. Everything changed in 1994, when Peter Shor's seminal work [Sho97] showed that a quantum computer could effectively solve the usual problems of public key cryptography (namely factorization and discrete logarithm), and therefore break most cryptography deployed on the internet. This launched the development of quantum computing, and post-quantum cryptography.

Post-quantum cryptography refers to a branch of cryptography aimed at designing non-quantum, ie. classical cryptographic systems which are secure against an adversary having access to a quantum computer. For public key cryptography, alternatives to RSA and Diffie-Hellman are needed because of Shor's quantum attacks that run in time polynomial in the size of the key. Long considered a rather marginal subject, post-quantum cryptography has now been on the international scene for several years, with first the announcement by the NSA in 2015 of its transition to post-quantum cryptography, then the launch in 2016 by NIST of an international competition to standardize post-quantum cryptosystems [NIS]. Many companies are interested in the viability of post-quantum cryptography: for example, Google experimented for several months a post-quantum version (based on Euclidean lattices) of the famous TLS protocol massively deployed on the Internet [GOO].

To date, there are several candidates for post-quantum cryptography, mainly lattice-based, code-based, hash-based, and multivariate cryptography (based on polynomial systems with several variables), and more recently, supersingular elliptic curve isogeny cryptography. Although these candidates are for the most part relatively old (dating back to the invention of the RSA), they raise some questions. All of these candidates suffer from significantly longer key sizes than RSA and elliptical curve cryptography, and are thus less efficient. Furthermore, their long-term security remains open, for quantum but also classic adversaries: many of these systems have been broken in practice, especially in code-based and multivariate cryptography. As for hash-based cryptography, we only know how to construct signature schemes from it, but not encryption schemes. Furthermore, only a limited number of signatures can be signed using each private key. Lattice-based cryptography thus seems to be the most promising post-quantum candidate. Among the round 3 finalists of the NIST Post-Quantum Cryptography Standardization procedure, 3 out of 4 Public-key Encryption and Key-establishment schemes and 2 out of 3 Digital Signature schemes are lattice-based.

Euclidean lattices can not only be used to construct various cryptographic primitives such as hash functions [LMPR08], signatures [GPV08, Lyu12], encryption schemes [Reg06, GPV08], pseudo random functions [BPR12], but also to build more sophisticated cryptographic primi-

[0]Image from Physics World's "Alice and Bob communicate without transferring a single photon".

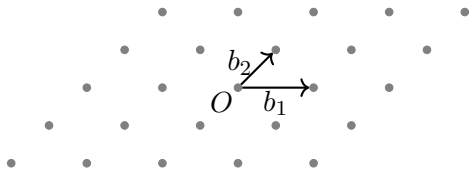


Figure 1.1: A 2D lattice with a basis $(\boldsymbol{b}_1, \boldsymbol{b}_2)$ at the origin O.

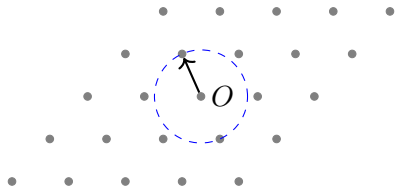


Figure 1.2: A lattice and a shortest nonzero vector: the blue circle does not contain any nonzero in its interior.

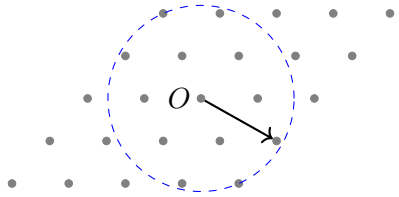


Figure 1.3: A lattice and a vector whose length is smaller than $\gamma = 2$ times the length of the shortest nonzero vector.

tives including fully homomorphic encryption schemes (FHE) [Gen09], identity based encryption (IBE) [GPV08, CHKP10], attribute based encryption (ABE) [AFV11], group signatures [GKV10] and so on. Most of those primitives are based on two mathematical problems known as the Short Integer Solution problem (SIS) [Ajt96] and the Learning with Errors problem (LWE) [Reg05]. These problems can be seen as randomized variants of two fundamental problems on lattices: the Shortest Vector Problem (SVP) and the Closest Vector Problem (CVP). Moreover, most strategies for cryptanalysing the previous cryptographic primitives rely on lattice reduction techniques known as the BKZ algorithm. BKZ internally uses an algorithm to solve (near) exact SVP in lower-dimensional lattices. Therefore, understanding the complexity SVP is critical to choosing security parameters of cryptographic primitives. In this thesis, we will study several approaches to solve SVP, and CVP to a lesser extent.

Lattices. A (Euclidean) *lattice* $\mathcal{L}$ is a discrete subgroup of $\mathbb{R}^m$, or equivalently the set $\mathcal{L}(\mathbf{b}_1, \ldots, \mathbf{b}_n) = \{\sum_{i=1}^n x_i \mathbf{b}_i \ : \ x_i \in \mathbb{Z}\}$ of all integer combinations of n linearly independent vectors $\mathbf{b}_1, \ldots, \mathbf{b}_n \in \mathbb{R}^m$. Such $\mathbf{b}_i$ form a *basis* of $\mathcal{L}$. All the bases have the same number n of elements, called the *dimension* or *rank* of $\mathcal{L}$. The lattice $\mathcal{L}$ is said to be *full-rank* if $n = m$ (see figure 1.1).

One of the fundamental algorithmic problems related to lattices is to find a shortest non-zero element of an arbitrary lattice (with respect to its Euclidean norm), given an arbitrary basis of this lattice. This problem is referred to as the shortest vector problem (SVP) (see Figure 1.2). A natural approximate version of this problem exists, where the problem consists in finding a non-zero vector of the lattice whose Euclidean norm is no more than γ times the norm of a shortest non-zero vector. This problem is referred to as the γ-approximate shortest vector problem (γ-approx-SVP) (see Figure 1.3). Starting from the '80s, the use of approximate and exact solvers for SVP (and other lattice problems) gained prominence for their applications in algorithmic number theory [LLL82], convex optimization [Jr.83, Kan87, FT87], coding theory [dB89], and cryptanalysis tool [Sha84, Bri84, LO85].

The SVP is a well studied computational problem in both its exact and approximate versions.

It is known to be NP-hard to approximate within any constant factor by a randomized reduction, and hard to approximate within a factor $n^{c/\log\log n}$ for some $c > 0$ under reasonable complexity-theoretic assumptions (NP $\nsubseteq$ RSUBEXP) [Ajt98, Mic98, Kho05, HR07]. For an approximation factor $2^{\mathcal{O}(n)}$, one can solve SVP in time polynomial in n using the celebrated LLL lattice basis reduction algorithm [LLL82]. In cryptography, we are interested in approximating SVP within factors polynomial in n. The fastest known algorithms for this regime rely on (a variant of) the BKZ lattice basis reduction algorithm [Sch87, SE94a, AKS01, GN08, HPS11, ALNS20, LN20], which can be seen as a generalization of the LLL algorithm and gives a $O(\beta^{n/\beta})$ approximation in $2^{\mathcal{O}(\beta)}$ poly(n) time for any $2 \leqslant \beta \leqslant n$. The smallest approximation factor for which the BKZ algorithm runs in polynomial time is $2^{\mathcal{O}(\frac{n\log\log n}{\log n})}$, which is better than the LLL algorithm.

As one would expect from the hardness results above, all known algorithms for solving exact SVP require at least exponential time. In fact, the fastest known algorithms also require exponential space. There has been some recent evidence [AS18a] showing that one cannot hope to get a $2^{o(n)}$ time algorithm for SVP if one believes reasonable complexity theoretic conjectures such as the (Gap) Exponential Time Hypothesis. Most of the known algorithms for SVP can be broadly classified into two classes: (i) the algorithms that require memory polynomial in n but run in time $n^{\mathcal{O}(n)}$ and (ii) the algorithms that require memory $2^{\mathcal{O}(n)}$ and run in time $2^{\mathcal{O}(n)}$.

The first class, initiated by Kannan [Kan87, Hel85, HS07, MW15], combines lattice basis reduction with exhaustive enumeration inside Euclidean balls. While enumerating vectors requires $2^{\mathcal{O}(n\log n)}$ time, it is much more space-efficient than other kinds of algorithms for exact SVP. These algorithms can be further made much faster in practice using some heuristic techniques, in particular the pruning technique [SE94a, GNR10, Che13].

Another class of algorithms, and currently the fastest, is based on sieving. First developed by Ajtai, Kumar, and Sivakumar [AKS01], they generate many lattices vectors and then divide-and-sieve to create shorter and shorter vectors iteratively. A sequence of improvements [Reg04, PS09, ADRS15, AS18b], has led to a $2^{n+o(n)}$ time and space algorithm by sieving the lattice vectors and carefully controlling the distribution of output, thereby after several sieving steps outputting a set of lattice vectors that contains the shortest vector with overwhelming probability. There are also variants [NV08, MV10] of the sieving algorithms mentioned above that, under some heuristic assumptions, give a $(4/3)^{n+o(n)}$ time complexity and a $(4/3)^{n/2+o(n)}$ memory complexity. They assume, for example, that the direction of the vectors is uniform on the unit sphere, and the vectors in the sieved lists are independent. A long line of work, including [BGJ13, Laa15a, Laa15b, BDGL16] decreases this time complexity down to $(3/2)^{n/2+o(n)}$ while still keeping the same memory complexity (also $(4/3)^{n/2+o(n)}$). Other variants (tuple-sieving) are designed to lower the memory complexity [BLS16, HK17], still under heuristic assumptions.

Another important problem related to lattices is the closest vector problem (CVP). Given an arbitrary basis of a lattice $\mathcal{L}$ and a target point t in the space, one needs to find a point of $\mathcal{L}$ closest to t (see figure 1.4). In the approximate variant of this problem, called γ-approx-CVP, the goal is to find a point of $\mathcal{L}$ at distance at most $\gamma \cdot \text{dist}(\mathcal{L}, t)$ from t, where $\gamma \geqslant 1$ and $\text{dist}(\mathcal{L}, t)$ is the minimal distance between t and a point of $\mathcal{L}$ (see Figure 1.5). The best algorithms to solve γ-approx-CVP are asymptotically as efficient as for γ-approx-SVP. In other words, the best algorithm to solve exact CVP has a complexity exponential in the dimension n of the lattice and the smallest approximation factor for which we have a polynomial time

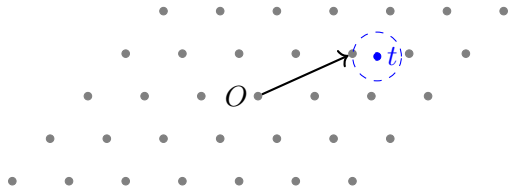


Figure 1.4: A lattice, a target vector t and a closest lattice vector to t.

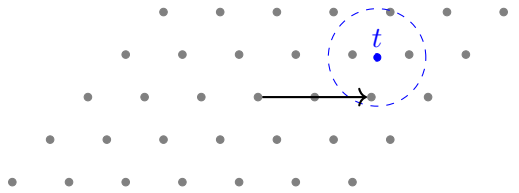


Figure 1.5: A lattice, a target vector t and a lattice vector at dist at most $\gamma \cdot \mathrm{dist}(\mathcal{L}, t)$ from t, where $\gamma < 2$.

algorithm is $2^{\mathcal{O}(\frac{n \log\log n}{\log n})}$.

Subset Sum. In this thesis, we also study the *subset-sum problem*. The subset-sum problem, also known as *knapsack problem* is the following: given n integers $\mathbf{a} = (a_1, \dots a_n)$ and a target integer S, find an n-bit vector $\mathbf{e} = (e_1, \dots e_n) \in \{0,1\}^n$ such that $\mathbf{e} \cdot \mathbf{a} = \sum_i e_i a_i = S$. The *density* of the knapsack instance is defined as $d = n/(\log_2 \max_i a_i)$ and, for a random instance $\mathbf{a}$, it is related to the number of solutions that one can expect.

The decision version of the subset-sum problem is NP-complete [GJ79]. In the low density case, we can transform it to the problem of finding a particular short vector in an integer lattice, and apply the LLL lattice reduction algorithm to solve it in polynomial time [LO83]. In the high density case, subexponential time algorithms also exist using Wagner's algorithm [Lyu05] which is in some sense a more general version of the BKW algorithm [BKW03] also used to solve the LWE problem [ACF+15]. When the density is close to 1, the best algorithms known are exponential-time, which is why we name these instances "hard" knapsacks.

In cryptographic applications, one usually encounters random instances rather than worst case instances. Such instances of the subset-sum problem are defined by choosing $\mathbf{a} = (a_1, \dots a_n)$ uniformly at random from $(\mathbb{Z}/M\mathbb{Z})^n$ for some integer M and setting $S = \mathbf{e} \cdot \mathbf{a} \mod M$ for a binary $\mathbf{e} \in \{0,1\}^n$ where half of the coordinates are equal to one. Note that we are now focusing on the modular version of the subset-sum problem, where we want to find a solution for $S = \mathbf{e} \cdot \mathbf{a} \mod M$. Up to polynomial factors, solving modular knapsacks and knapsacks over the integers is equivalent: any algorithm that realizes one task can be used to solve the other. The hardest case where the density equals 1 corresponds to $M = 2^n$. In a breakthrough paper, Howgrave-Graham and Joux [HJ10] showed how to solve these specific random instances in time $2^{0.337n}$, breaking the Horowitz-Sahni bound of $2^{0.5n}$ for worst case instances [HS74]. This bound was later improved by Becker, Coron and Joux [BCJ11] down to $2^{0.291n}$ time and memory, which was the best known runtime before the work in this thesis. In the quantum setting, [BJLM13] and [HM18] are the corresponding quantum algorithms of [HS74] and [BCJ11]. They run in respective time $2^{0.241n}$ and $2^{0.226n}$ and use quantum walks on Johnson graphs.

The random subset-sum problem underlies some cryptographic schemes aiming at post-quantum security (see *e.g.* [LPS10]), although it is mostly of academic nature. In a cryptanalysis context, it often serves as a cryptanalytic tool as many problems on which modern cryptography is built can be expressed as some (vectorial) versions of the subset sum problem. This is the case for the syndrome decoding problem in code-based cryptography, where the fastest known algorithms [BJMM12, BM18] heavily rely on the fastest algorithms known to solve

the subset-sum problem. It is also the case for the short integer solution problem and for the learning parity with noise problem (LPN) [Ale03]. More recently it was also shown that the subset-sum problem can be used as a building block in some quantum hidden shift algorithms [Bon19], which have applications in quantum cryptanalysis of isogeny-based [BS20] and symmetric cryptographic schemes [BN18].

1.1 Contributions

During my PhD, I have been interested in the difficulty of problems related to lattices. Namely, I have focused on both provable and heuristic algorithms for solving SVP in classical and quantum settings. Some of my results also apply to CVP. I also worked on the random subset-sum problem and developed better classical and quantum algorithms for it. The quantum algorithms I have developed fall into three categories, depending the type of quantum memory they use: plain quantum circuits (where we only use a set of universal quantum gates on qubits), *classical memory with quantum random access* (QRACM) and *quantum memory with quantum random access* (QRAQM). While low depth construction of QRACM has been proposed (albeit with a blow up in the number of required qubits), no physical architecture with low depth has been proposed for QRAQM. Typically, Grover search requires QRACM but MNRS-type quantum walks need QRAQM [MNRS11]. It is of interest to design algorithms with the minimum possible requirements on the quantum memory models. Whenever possible, I have tried to only use the plain quantum circuit model or the more realistic QRACM model. See Section 2.2.2 for a more detailed discussion on different types of quantum memory.

1.1.1 Provable Time-Space Trade-off for SVP

We first present in Chapter 3 a new randomized algorithm for SVP that provides a smooth trade-off between the time complexity and memory requirement without any heuristic assumptions. This algorithm is obtained by giving a new randomized algorithm for sampling lattice vectors from the Discrete Gaussian distribution up to the smoothing parameter that runs in time $q^{11n+o(n)}$ and requires $\mathrm{poly}(q) \cdot q^{16n/q^2}$ space, where q can be any integer in $[4, \sqrt{n}]$. The smoothing parameter quantifies how smooth the Discrete Gaussian distribution is: The smaller the parameter is, the more discrete the distribution is and the harder it is to sampler from it. Using the standard reduction from Bounded Distance Decoding (BDD) with preprocessing (where an algorithm solving the problem is allowed unlimited preprocessing time on the lattice before the algorithm receives the target vector) to Discrete Gaussian Sampling (DGS) from [DRS14] and a reduction from SVP to BDD given in [CCL18], we obtain a randomized algorithm that solves SVP in time $q^{11n+o(n)}$ and in space $\mathrm{poly}(n) \cdot q^{\frac{16n}{q^2}}$.

If we take $k = q^2$, then the time complexity of our SVP algorithm becomes $k^{5.5n+o(n)}$ and the space complexity $\mathrm{poly}(n) \cdot k^{(8n/k)}$. Our trade-off without any heuristic assumptions is thus the same (up to a constant in the exponents) as what was claimed[1] by Kirchner and Fouque [KF16] and proven in [HK17] *under heuristic assumptions*.

Our trade-off for the DGS is obtained by extending the classical coset-based sieving algorithm of [ADRS15]. Instead of efficiently pairing vectors in the same cosets of $\mathcal{L}/2\mathcal{L}$ to

[1] [KF16] never passed peer review. We will discuss their results in Section 3.5

obtain smaller ones (which takes a lot of space), we try to combine many vectors at random until we find one in $q\mathcal{L}$. Combining vectors at random is very space efficient but potentially very time inefficient. The main observation is that if we combine sufficiently many vectors, then the sum behaves likes a uniformly distributed variable over the cosets of $\mathcal{L}/q\mathcal{L}$ and hence, on average, we will not have to wait too long to successfully combine vectors. Indeed, above the smoothing parameter, the distribution over the cosets of a random sample from the discrete Gaussian distribution is almost uniform. Furthermore, again above the smoothing parameter, the sum of independent discrete Gaussian variables is very close to a discrete Gaussian distribution.

1.1.2 Faster Provable Classical and Quantum Algorithms for SVP

In Chapter 4, we present the fastest quantum algorithm without any heuristic assumptions for SVP. To date, the fastest classical provable algorithm for SVP [ADRS15] runs in time and space $2^{n+o(n)}$. Previously, no faster provable quantum algorithm was known. We show a quantum algorithm that solves SVP in time $2^{0.9532n+o(n)}$ and classical space $2^{0.5n+o(n)}$ with an additional number of qubits polynomial in n. Our algorithm is a follow-up work of [CCL18] in which the authors provided a quantum algorithm in time $2^{1.2553n+o(n)}$ and space $2^{0.5n+o(n)}$. [CCL18] provided a reduction from SVP to BDD. We improve over [CCL18] by constructing a better BDD oracle that costs less on each query. The BDD oracle is obtained by first producing many samples from a discrete Gaussian distribution above the smoothing parameter and then applying a classical reduction from BDD to DGS. The quality of the reduction fundamentally depends on the width s of the Gaussian, which needs to be equal to the smoothing parameter η_ε of the lattice for some ε. Unfortunately, the most efficient DGS sampler above the smoothing parameter stops at $\sqrt{2}\eta_{1/2}$. The solution proposed by [CCL18] is to find ε such that $\eta_\varepsilon > \sqrt{2}\eta_{1/2}$, unfortunately this requires to choose a much smaller ε than necessary. We avoid this step by construction a super lattice $\mathcal{L}'$ of $\mathcal{L}$ such that $\eta_\varepsilon(\mathcal{L}') \leqslant \eta_\varepsilon(\mathcal{L})/\sqrt{2}$. This way, by using the best DGS sampler, we can reach $\sqrt{2}\eta_\varepsilon(\mathcal{L}') = \eta_\varepsilon(\mathcal{L})$ without changing the parameter ε. We need an extra rejection sampling step to only keep vectors in $\mathcal{L}$ but not in $\mathcal{L}'$ but the impact on the complexity is small.

We also present a classical randomized algorithm for SVP that improves over the algorithm from [CCL18] and results in the fastest classical algorithm that has a space complexity $2^{0.5n+o(n)}$. We improve over [CCL18] by showing how to reduce the number of queries to the BDD oracles by making smarter queries. Our algorithm solves SVP in time $2^{1.730n+o(n)}$ and space $2^{0.5n+o(n)}$. The classical technique to reduce SVP to BDD involves constructing a α-BDD oracle and then using this oracle to enumerate all lattice points in a sufficiently large ball. Intuitively speaking, if $\alpha = 1/p$ for some integer p then we will need to make p^n queries to enumerate all points. Previous work, including our quantum algorithm, construct a 1/3-BDD oracle and hence make 3^n queries. Ideally, we would like to take $p = 2$ but we cannot build a 1/2-BDD oracle. Our main observation is that instead of enumerating all points inside a large ball, it is more efficient to enumerate all points inside many smaller balls that, together, cover the large ball. By doing so, each small ball only requires 2^n queries and by carefully choosing the radius α, we can use less than $(3/2)^n$ balls, resulting in an overall net gain.

The time complexity of these two results are obtained using a known upper bound of a quantity $\beta(\mathcal{L})$ related to the kissing number of a lattice (see Section 2.4) which is $2^{0.402n}$. In practice most lattices have a much smaller $\beta(\mathcal{L})$ which is often $2^{o(n)}$. In that case, our classical

Time Complexity	Space Complexity	Reference
$n^{\frac{n}{2e}+o(n)}$	poly(n)	[Kan87, HS07]
$2^{n+o(n)}$	$2^{n+o(n)}$	[ADRS15]
$2^{2.05n+o(n)}$	$2^{0.5n+o(n)}$	[CCL18]
$2^{1.7397n+o(n)}$	$2^{0.5n+o(n)}$	This thesis, [ACKS21]

Table 1.1: Classical randomized provable algorithms for the shortest vector problem.

Time Complexity	Space Complexity	Reference
$2^{1.799n+o(n)}$	$2^{1.286n+o(n)}$ classical memory with quantum random access (QRACM) + poly(n) qubits	[LMvdP15a]
$2^{1.2553n+o(n)}$	$2^{0.5n+o(n)}$ classical memory + poly(n) qubits	[CCL18]
$2^{0.9535n+o(n)}$	$2^{0.5n+o(n)}$ classical memory + poly(n) qubits	This thesis, [ACKS21]

Table 1.2: Quantum provable algorithms for the shortest vector problem. [LMvdP15a] uses the QRACM model. [CCL18] and our quantum algorithm need only polynomial qubits and $2^{0.5n+o(n)}$ classical space.

algorithm runs in time $2^{1.292n}$ and our quantum algorithm runs in time $2^{0.750n}$. More generally, we study the dependency of the complexity of the algorithms on the quantity $\beta(\mathcal{L})$.

We summarize known provable classical randomized algorithms and quantum algorithms respectively in Table 1.1 and Table 1.2. Note that classical algorithms can also be seen as quantum algorithms without using any quantum power.

1.1.3 Quantum Quadratic Speed-up for Enumeration Algorithms for SVP and CVP

In Chapter 5, we show that lattice enumeration and its cylinder and discrete pruning variants can all be quadratically sped up on a quantum computer, unlike sieving. This is done by a careful interpretation and analysis of enumeration as tree algorithms. Interestingly, we show that this speedup also applies to extreme pruning [GNR10] where one repeats enumeration over many reduced bases: a naive approach would only decrease the classical cost mt (where m is the number of bases and t is the number of operations of a single enumeration) to $m\sqrt{t}$ quantum operations, but we bring it down to $\sqrt{mt}$.

First, we clarify the application of Montanaro's algorithm [Mon15] to enumeration with cylinder pruning: the analysis of [Mon15] assumes that the degree of the tree is bounded by a constant, which is tailored for constraint satisfaction problems, but is not the setting of

lattice enumeration. To tackle enumeration, we add basic tools such as binary tree conversion and dichotomy: we obtain that if a lattice enumeration (with or without cylinder pruning) searches over a tree with T nodes, the best solution can be found by a quantum algorithm using roughly $\sqrt{T}$ poly-time operations, where there is a polynomial overhead, which can be decreased if one is only interested in finding one solution. This formalizes earlier brief remarks of [ADPS16, dPLP16, ABB+17] and applies to both SVP and CVP.

Our main result is that the quantum quadratic speed-up also applies to the recent discrete pruning enumeration introduced by Aono and Nguyen [AN17] as a generalization of Schnorr's sampling algorithm [Sch03]. To do so, we tweak discrete pruning and use an additional quantum algorithm, namely that of Ambainis and Kokainis [AK17] from STOC '17 to estimate the size of trees. Roughly speaking, given a parameter T, discrete pruning selects T branches (optimizing a certain metric) in a larger tree, and derives T candidate short lattice vectors from them. Our quantum variant directly finds the best candidate in roughly $\sqrt{T}$ operations.

As mentioned previously, we show that the quadratic speed-up of both enumerations also applies to the extreme pruning setting (required to exploit the full power of pruning): if one runs cylinder pruning over m trees, a quantum enumeration can run in $\sqrt{T}$ poly-time operations where T is the sum of the m numbers of nodes, rather than $\sqrt{mT}$ naively; and there is a similar phenomenon for discrete pruning.

Enumeration algorithms are an instance of the general algorithmic technique called branch and bound. After our paper, [Mon20] proposed a quantum speedup of branch and bound algorithms which basically uses the same ingredients as in Section 5.5.

As a side result, we develop two tweaks to discrete pruning [AN17], to make it more powerful and more efficient. The first tweak enables to solve CVP in such a way that most of the technical tools introduced in [AN17] can be reused. This works for the approximation form of CVP, but also its exact version formalized by the *Bounded Distance Decoding* problem (BDD), which appears in many cryptographic applications such as LWE. In BDD, the input is a lattice basis and a lattice point shifted by some small noise whose distribution is crucial. We show how to handle the most important noise distributions, such as LWE's Gaussian distribution and finite distributions used in GGH [GGH97] and lattice attacks on DSA [NS02]. Enumeration, which was historically only described for SVP, can trivially be adapted to CVP, and so does [GNR10]'s cylinder pruning [LN13]. However, discrete pruning [AN17] appears to be less simple.

The second tweak deals with the selection of optimal discrete pruning parameters, and is crucial for our quantum variant. Intuitively, given an integer $T > 0$, the problem is to find the T "best" integral vectors $\mathbf{t} \in \mathbb{N}^n$ which minimize some objective function $f(\mathbf{t})$. Aono and Nguyen [AN17] introduced a fast practical algorithm to do so for a very special useful choice of f, but the algorithm was heuristic: no good bound on the running time was known. We show that their algorithm can actually behave badly in the worst case, *i.e.* it may take exponential time. But we also show that by a careful modification, the algorithm becomes provably efficient and even optimal for that f, and heuristically for more general choices of f: the running time becomes essentially T operations.

Our theoretical analysis has been validated by experiments, which shows that in practical BDD situations, discrete pruning is as efficient as cylinder pruning. Since discrete pruning has interesting features (such as an easier parallelization and an easier generation of parameters), it might become the method of choice for large-scale blockwise lattice reduction.

1.1.4 Better Quantum and Classical Algorithms for the Random Subset Sum Problem

In Chapter 6, we improve classical and quantum subset sum algorithms based on representations. Such algorithms work by representing the knapsack solution in many different ways, as a sum of vectors in $\{0,1\}^n$ or $\{-1,0,1\}^n$. See Section 6.1 for more details. We write these algorithms as sequences of "merge-and-filter" operations, where lists of subknapsacks are first merged with respect to an arbitrary constraint, then *filtered* to remove the subknapsacks that cannot be part of a solution.

First, we propose a more time-efficient classical subset-sum algorithm based on representations. We relax the constraints of BCJ and introduce "2"s in the representations, obtaining a better time complexity exponent of 0.283.

Most of our contributions concern quantum algorithms. As a generic tool, we introduce *quantum filtering*, which speeds up the filtering of representations with a quantum search. We use this improvement in all our new quantum algorithms.

We give an improved quantum walk based on quantum filtering and our extended $\{-1,0,1,2\}$ representations. Our best runtime exponent is 0.216, under the quantum walk update heuristic of [HM18], which uses the average update time rather than the worst update time in the MNRS quantum walk framework. Next, we show how to overcome this heuristic, by designing a new data structure for the vertices in the quantum walk, and a new update procedure with guaranteed time. We remove this heuristic from the previous algorithms [BJLM13, HM18] with no additional cost. However, we find that removing it from our quantum walk increases its cost to 0.218.

All previously known quantum algorithms for the subset sum problems use quantum walks on Johnson graphs, thus require quantum memory with quantum random access (QRAQM). No quantum algorithms using a weaker quantum memory model, such as classical memory with quantum random access (QRACM) was known. We devise a new quantum subset-sum algorithm based on HGJ, with time $\widetilde{O}\left(2^{0.236n}\right)$. It is the first quantum time speedup on classical algorithms that is *not* based on a quantum walk. The algorithm performs instead a depth-first traversal of the HGJ tree, using quantum search as its only building block. Hence, by construction, it does not require the additional heuristic of [HM18] *and* it only uses QRACM, giving also the first quantum time speedup for subset-sum in this memory model.

A summary of our contributions is given in Table 1.3[2].

All these complexity exponents are obtained by numerical optimization. Our code is available at `https://github.com/xbonnetain/optimization-subset-sum` .

1.1.5 Other Work

During my PhD studies, I was also involved in another project that has resulted into a publication. In a joint work with Ambainis et al [ABI+20], we studied the quantum query complexity of the two following problems:

Quantum complexity of regular languages. Consider the problem of recognizing whether an n-bit string belongs to a given regular language. This models a variety of

[2] After this work, Alexander May has informed us that the thesis [Bö11] contains unpublished results using more symbols, with the best exponent of 0.2871 obtained with the symbol set $\{-2,-1,0,1,2\}$.

Table 1.3: Time exponents of previous and new algorithms for subset-sum, classical and quantum, rounded upwards. We note that the removal of Heuristic 6.17 in [BJLM13, HM18] comes from our new analysis in Section 6.6.5. QW: Quantum Walk. QS: Quantum Search. CF: Constraint filtering (not studied in this thesis). QF: Quantum filtering.

Time exponent	Representations	Memory model	Techniques	Requires Heuristic 6.17	Reference
			Classical		
0.3370	$\{0,1\}$	RAM			[HJ10]
0.2909	$\{-1,0,1\}$	RAM			[BCJ11]
0.287	$\{-1,0,1\}$	RAM	CF		[Oze16]
0.2830	$\{-1,0,1,2\}$	RAM			Sec. 6.2.3
			Quantum		
0.241	$\{0,1\}$	QRAQM	QW	**No**	[BJLM13] + Sec. 6.6.5
0.226	$\{-1,0,1\}$	QRAQM	QW	**No**	[HM18] + Sec. 6.6.5
0.2356	$\{0,1\}$	**QRACM**	QS + QF	No	Sec 6.4.3
0.2156	$\{-1,0,1,2\}$	QRAQM	QW + QF	Yes	Sec. 6.5.3
0.2182	$\{-1,0,1,2\}$	QRAQM	QW + QF	No	Sec. 6.6.5

computational tasks that can be described by regular languages. In the quantum case, the most commonly used model for studying the complexity of various problems is the query model. For this setting, Aaronson, Grier and Schaeffer [AGS18] recently showed that any regular language L has one of three possible quantum query complexities on inputs of length n: $\Theta(1)$ if the language can be decided by looking at $O(1)$ first or last symbols of the word; $\tilde{\Theta}(\sqrt{n})$ if the best way to decide L is Grover's search (for example, for the language consisting of all words containing at least one letter a); $\Theta(n)$ for languages in which we can embed counting modulo some number p which has quantum query complexity $\Theta(n)$.

As shown in [AGS18], a language being of complexity $\tilde{O}(\sqrt{n})$ (which includes the first two cases above) is equivalent to it being star-free. Star-free languages are defined as the languages which have regular expressions not containing the Kleene star (if it is allowed to use the complement operation). Star-free languages are one of the most commonly studied subclasses of regular languages and there are many equivalent characterizations of them. One of the star-free languages mentioned in [AGS18] is the Dyck language (with one type of parenthesis) with a constant bounded height. The Dyck language is the set of balanced strings of parentheses `(` and `)`. If at no point the number of opening parentheses exceeds the number of closing parentheses by more than k, we denote the problem of determining if an input of length n belongs to this language by $\textsc{Dyck}_{k,n}$. The language is a fundamental example of a context-free language that is not regular. When more types of parenthesis are allowed, the famous Chomsky–Schützenberger representation theorem shows that any context-free language is the homomorphic image of the intersection of a Dyck language and a regular language.

Our results. We show that an exponential dependence of the complexity on k is unavoidable. Namely, for the balanced parentheses language:

- there exists $c > 1$ such that, for all $k \leqslant \log n$, the quantum query complexity is $\Omega(c^k\sqrt{n})$;

- if $k = c \log n$ for an appropriate constant c, the quantum query complexity is $\Omega(n^{1-\varepsilon})$.

Thus, the exponential dependence on k is unavoidable and distinguishing sequences of balanced parentheses of length n and depth $\log n$ is almost as hard as distinguishing sequences of length n and arbitrary depth.

Similar lower bounds have recently been independently proven by Buhrman et al. [BPS19].

Additionally, we give an explicit algorithm for the decision problem $\textsc{Dyck}_{k,n}$ with $O\big(\sqrt{n}(\log n)^{0.5k}\big)$ quantum queries. The algorithm also works when k is not a constant and is better than the trivial upper bound of n when $k = o\Big(\frac{\log(n)}{\log\log n}\Big)$.

Finding paths on a grid. The second problem that we consider is graph connectivity on subgraphs of the 2D grid. Consider a 2D grid with vertices (i, j), $i \in \{0, 1, \ldots, n\}, j \in \{0, 1, \ldots, k\}$ and edges from (i, j) to $(i + 1, j)$ and $(i, j + 1)$. The grid can be either directed (with edges in the directions of increasing coordinates) or undirected. We are given an unknown subgraph G of the 2D grid and we can perform queries to variables x_u (where u is an edge of the grid) defined by $x_u = 1$ if u belongs to G and 0 otherwise. The task is to determine whether G contains a path from $(0, 0)$ to (n, k).

Our interest in this problem is driven by the edit distance problem. In the edit distance problem, we are given two strings x and y and have to determine the smallest number of operations (replacing one symbol by another, removing a symbol or inserting a new symbol) with which one can transform x to y. If $|x| \leqslant n, |y| \leqslant k$, the edit distance is solvable in time $O(nk)$ by dynamic programming [WF74]. If $n = k$ then, under the strong exponential time hypothesis (SETH), there is no classical algorithm computing edit distance in time $O(n^{2-\varepsilon})$ for $\varepsilon > 0$ [BI15] and the dynamic programming algorithm is essentially optimal.

However, SETH does not apply to quantum algorithms. Namely, SETH asserts that there is no algorithm for general instances of SAT that is substantially better than naive search. Quantumly, a simple use of Grover's search gives a quadratic advantage over naive search. This leads to the question: can this quadratic advantage be extended to edit distance (and other problems that have lower bounds based on SETH)?

Since edit distance is quite important in classical algorithms, the question about its quantum complexity has attracted a substantial interest from various researchers. Boroujeni et al. [BEG+18] invented a better-than-classical quantum algorithm for approximating the edit distance which was later superseded by a better classical algorithm of [CDG+18]. However, there has been no quantum algorithms computing the edit distance exactly (which is the most important case).

The main idea of the classical algorithm for edit distance is as follows:

- We construct a weighted version of the directed 2D grid (with edge weights 0 and 1) that encodes the edit distance problem for strings x and y, with the edit distance being equal to the length of the shortest directed path from $(0, 0)$ to (n, k).

- We solve the shortest path problem on this graph and obtain the edit distance.

As a first step, we can study the question of whether the shortest path is of length 0 or more than 0. Then, we can view edges of length 0 as present and edges of length 1 as absent. The question "Is there a path of length of 0?" then becomes "Is there a path from $(0, 0)$ to (n, k) in which all edges are present?". A lower bound for this problem would imply a similar lower

bound for the shortest path problem and a quantum algorithm for it may contain ideas that would be useful for a shortest path quantum algorithm.

Our results. We use our lower bound on the balanced parentheses language to show an $\Omega(n^{1.5-\varepsilon})$ lower bound for the connectivity problem on the directed 2D grid. This shows a limit on quantum algorithms for finding edit distance through the reduction to shortest paths. More generally, for an $n \times k$ grid ($n > k$), our proof gives a lower bound of $\Omega((\sqrt{n}k)^{1-\varepsilon})$.

The trivial upper bound is $O(nk)$ queries, since there are $O(nk)$ variables. There is no nontrivial quantum algorithm, except for the case when k is very small. Then, the connectivity problem can be solved with $O(\sqrt{n}\log^k n)$ quantum queries [Kļe17] [3] but this bound becomes trivial already for $k = \Omega(\frac{\log n}{\log\log n})$.

For the undirected 2D grid, we show a lower bound of $\Omega((nk)^{1-\varepsilon})$, whenever $k \geqslant \log n$. Thus, the naive algorithm is almost optimal in this case. We also extend both of these results to higher dimensions, obtaining a lower bound of $\Omega((n_1 n_2 \dots n_d)^{1-\varepsilon})$ for an undirected $n_1 \times n_2 \times \ldots \times n_d$ grid in d dimensions and a lower bound of $\Omega(n^{(d+1)/2-\varepsilon})$ for a directed $n \times n \times \ldots \times n$ grid in d dimensions.

In a recent work, an $\Omega(n^{1.5})$ lower bound for edit distance was shown by Buhrman et al. [BPS19], assuming a quantum version of the Strong Exponential Time hypothesis (QSETH). As part of this result they give an $\Omega(n^{1.5})$ query lower bound for a different path problem on a 2D grid. Then QSETH is invoked to prove that no quantum algorithm can be faster than the best algorithm for this shortest path problem. Neither of the two results follow directly one from another, as different shortest path problems are used.

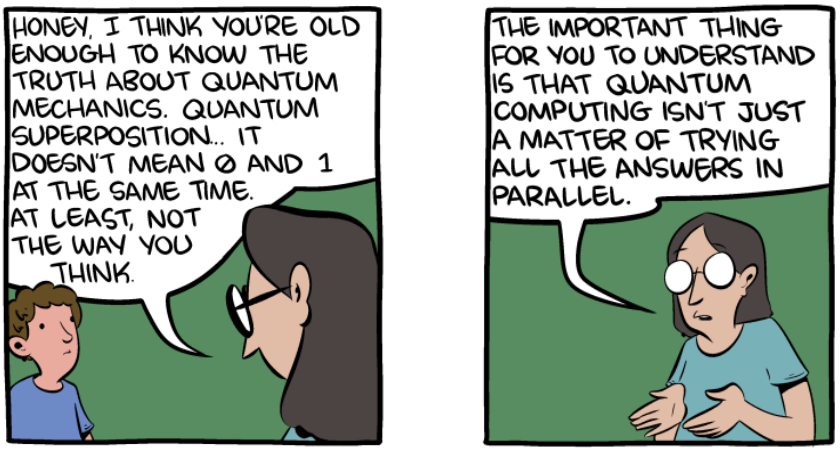
[3] Aaronson et al. [AGS18] also give a bound of $O(\sqrt{n}\log^{m-1} n)$ but in this case m is the rank of the syntactic monoid which can be exponentially larger than k.

Chapter 2 Preliminaries

2.1 Basic Notations

$\mathbb{N}$ is the set of non negative integers. $\mathbb{Z}$ is the set of integers. $\mathbb{R}$ is the set of real numbers. $\mathbb{C}$ is the set of complex numbers. $\mathbb{Z}_n$ is the ring of residue classes modulo n. The vectors are written in bold. For any finite set U, its number of elements is $\#U$. For any measurable subset $S \subseteq \mathbb{R}^n$, its volume is $\text{vol}(S)$. The Euclidean norm of a vector $\mathbf{v} \in \mathbb{R}^n$ is $\|\mathbf{v}\|$. We denote by $B_n(\mathbf{c}, R)$ the n-dimensional Euclidean ball of radius R and center $\mathbf{c}$, whose volume is $\text{vol}(B_n(R)) = R^n \frac{\pi^{n/2}}{\Gamma(n/2+1)}$. If $\mathbf{c}$ is omitted, we mean $\mathbf{c} = \mathbf{0}$. The distance between a point $\mathbf{x}$ and a set S is defined by $\text{dist}(\mathbf{x}, S) := \inf\{\|\mathbf{x} - \mathbf{y}\| : \mathbf{y} \in S\}$. We denote by $\mathbb{E}()$ the expectation and $\mathbb{V}()$ the variance of a random variable. We denote by ln the natural logarithm and log the logarithm in base 10. We make frequent use of the usual o and O notation for asymptotics. We also use the soft-O notation that removes polylogarithmic factors: $f(n) = \widetilde{O}(g(n))$ if and only if $\exists k \in \mathbb{N}$ such that $f(n) = O(g(n)\log^k(g(n)))$. The *Hamming weight* of a vector $\boldsymbol{e} \in \mathbb{R}^n$ is the number of coefficients that are different from the zero.

2.2 Quantum Computing



[0]Image from SMBC's "The Talk" and Scott Aaronson.

2.2.1 Introduction

In this thesis, when it comes to quantum algorithms, we always assume that we are working in some Hilbert space $\mathcal{H}$ of some finite-dimension n, and we use Dirac's bra-ket notation. A vector $\mathbf{x} = (x_1, \cdots, x_n) \in \mathcal{H}$ is denoted as $|\mathbf{x}\rangle = \Sigma_i x_i |i\rangle$, where $(|i\rangle)_{i\in\{0,\cdots,n-1\}}$ is the canonical basis of $\mathcal{H}$. This notation comes from the inner product $\langle x|y\rangle$, which can be seen as a linear form $\langle x|$ applied to a vector $|y\rangle$.

A qubit is a unit vector in the Hilbert space $\mathbb{C}^2$ with two basis vectors $\{|0\rangle, |1\rangle\}$. It can be denoted as $|\phi\rangle = \alpha_0|0\rangle + \alpha_1|1\rangle$, where α_0, α_1 are complex numbers such that $|\alpha_0|^2 + |\alpha_1|^2 = 1$. We can *measure* $|\phi\rangle$ in the basis $\{|0\rangle, |1\rangle\}$, this will output $|0\rangle$ with probability $|\alpha_0|^2$ and $|1\rangle$ with probability $|\alpha_1|^2$. The α_i's are called the *amplitudes* of the $|i\rangle$'s in $|\phi\rangle$.

We can combine different Hilbert spaces using tensor product: if $|0\rangle, |1\rangle, \cdots, |n-1\rangle$ are an orthonormal basis of space $\mathcal{H}_A$ and $|0\rangle, |1\rangle, \cdots, |m-1\rangle$ are an orthonormal basis of space $\mathcal{H}_B$, then the tensor product space $\mathcal{H} = \mathcal{H}_A \otimes \mathcal{H}_B$ is an nm-dimensional space spanned by the set of states $\{|i\rangle \otimes |j\rangle \mid i \in \{0, \cdots, n-1\}, j \in \{0, \cdots, m-1\}\}$. We often write $|i\rangle|j\rangle$ instead of $|i\rangle \otimes |j\rangle$ for convenience. This allows us to consider systems with n qubits in the Hilbert space $\mathbb{C}^{2^n} = (\mathbb{C}^2)^{\otimes n}$. Intuitively, a quantum state in this space can be viewed as a superposition of the 2^n possible values of n classical bits. More formally, a quantum state $|\phi\rangle$ is of the form $\sum_{w=w_1\cdots w_n\in\{0,1\}^n} \alpha_w |w\rangle$ where $|w\rangle = |w_1\rangle \otimes \cdots \otimes |w_n\rangle$. We can measure $|\phi\rangle$ in the basis $(|w\rangle)_{w=w_1\cdots w_n\in\{0,1\}^n}$ which is called the computational basis: it will output $|w\rangle$ with probability $|\alpha_w|^2$.

Instead of measuring $|\phi\rangle$, we can also apply some operation to it and change it to some other state $|\psi\rangle = \sum_i \beta_i |i\rangle$. Quantum mechanics only allows *unitary* operators. Any unitary transformation is a reversible operation. For a unitary operator U, we denote by $U^\dagger$ its conjugate transpose. Since U is unitary, $U^\dagger = U^{-1}$. This quantum reversible computation may seem at odds with how we define classical circuits, using irreversible gates such as OR and AND. But in fact, any classical computation can be made reversible by replacing any irreversible gate $x \mapsto f(x)$ by the reversible gate $(x, y) \mapsto (x, y + f(x))$, and running it on the input $(x, 0)$ and producing $(x, f(x))$. In other words, we can make classical computation reversible by storing all intermediate steps of the computation. This modification is not free of cost, as Corollary 2.2 shows.

Quantum gates. In the following, we present some elementary quantum unitary operators that acts on a small number of qubits. We call them *quantum gates* and they can be represented by matrices in the canonical basis $|0\rangle, |1\rangle$. The Hadamard gate is

$$H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

and it satisfies: $H|0\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$ and $H|1\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$. The T gate is:

$$T = \begin{bmatrix} 1 & 0 \\ 0 & e^{i\pi/4} \end{bmatrix}.$$

An example of a 2-qubit gate is the controlled-not gate called CNOT. Depending on whether the first bit of the input is 1, it negates the second bit or does nothing, ie $\text{CNOT}|0\rangle|b\rangle = |0\rangle|b\rangle$

and $\text{CNOT}|1\rangle|b\rangle = |1\rangle|1-b\rangle$. The CNOT gate can be represented by the following matrix form in the basis $|0\rangle\otimes|0\rangle, |0\rangle\otimes|1\rangle, |1\rangle\otimes|0\rangle, |1\rangle\otimes|1\rangle$:

$$\text{CNOT} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}.$$

The Toffoli gate, a three-qubit gate, is defined for any $a, b, c \in \{0,1\}$ by

$$\text{Toffoli}|a\rangle|b\rangle|c\rangle = \begin{cases} |a\rangle|b\rangle|1\oplus c\rangle & \text{if } a = b = 1; \\ |a\rangle|b\rangle|c\rangle & \text{otherwise.} \end{cases}$$

Universality. It is known that the set of all 1-qubit operations together with the 2-qubit CNOT gate is *universal*, meaning that any unitary operator can be built from these gates. However, in practice, we cannot build all 1-qubit operators to infinite precision since there are continuously many of them. We restrict ourselves to a universal set of gates in the sense of approximation, meaning that any unitary can be approximated arbitrarily well using only these gates. This is the case for the set of gates H, CNOT and T.

The Toffoli gate together with ancilla preparation is universal for classical computation. This means that any classical circuit can be implemented by a quantum one. It comes from the fact that the NAND gate is universal for classical computing and can be implemented by a Toffoli gate: $\text{Toffoli}|a\rangle|b\rangle|1\rangle = |a\rangle|b\rangle|\text{NAND}(a,b)\rangle$. However, the total number of quantum elementary gates used can be higher than the classical one because of the reversibility of the quantum computation (see Corollary 2.2).

Quantum circuit model. In the quantum circuit model, the time complexity is the circuit size, which is the total number of elementary quantum gates. The space complexity is the number of qubits used. We will assume that the elementary quantum gates come from a fixed universal set. Up to constant factors, the complexity does not depend on the universal set that we have chosen.

Given a function $f : \{0,1\}^n \to \{0,1\}^m$, we say that a quantum circuit, implementing a unitary U that acts on $n+\ell+m$ qubits, *computes f with probability α* if for every x, a measurement on the last m qubits of $U|x\rangle|0^\ell\rangle|0^m\rangle$ outputs $f(x)$ with probability at least α. The exact location of the qubits that we measure for the output actually does not matter, since we can also apply SWAP gates (implementable by elementary gates) to swap them to the m last positions. The extra ℓ qubits that are not part of the input/output are called *ancilla qubits* (or work space).

Quantum query model. We sometimes use the standard form of the *quantum query model*: given a unitary $\mathcal{O}$, we say that a circuit computes f with *oracle access* to $\mathcal{O}$ if by augmenting the model with the unitary $\mathcal{O}$, we can construct a circuit computing f. The number of *queries* on $\mathcal{O}$ is the number of unitary $\mathcal{O}$ in the circuit. If we find an efficient algorithm for a problem in query complexity and we are given an explicit circuit realizing the black-box transformation of the oracle $\mathcal{O}$, we will have an efficient algorithm for an explicit computational problem.

Quantum algorithm. So far, we have focused on circuits computing a function with a fixed inputs/outputs size. In general, the input/output size of a problem is not fixed and we need a family of circuits. Furthermore, it is common to alternate between phases of classical computations and quantum computations. In principle, it is always possible to turn a classical computation into a quantum one and combine all quantum algorithms into one quantum circuit with a single measurement at the end. Nevertheless, this transformation has a non-negligible cost (Corollary 2.2) that we want to avoid.

We say that a *quantum algorithm computes a function* $F : \{0,1\}^* \to \{0,1\}^*$ *with probability* α if there is a classical algorithm $\mathcal{A}$ *with quantum evaluation* that outputs $F(w)$ with probability α on input w. By quantum evaluation we mean that the algorithm can, any number of times during the computation, build a quantum circuit and evaluate it, that is measure the state $U|0\rangle$ where U is the unitary implemented by the circuit. The *time complexity* $T(n)$ is the classical time complexity of $\mathcal{A}$ plus the time complexity of the circuits (that is the number of gates). The *classical space complexity* $S(n)$ is the space complexity of $\mathcal{A}$ (ignoring quantum evaluations). The *quantum space complexity* $Q(n)$ is the maximal space complexity of all circuits (that is the maximum number of qubits used). In the natural way, we say that a quantum algorithm has oracle access to $\mathcal{O}$ if it produces circuits with oracle access to $\mathcal{O}$. The query complexity of the algorithm is the sum of the query complexity of the circuits.

It is sometimes necessary to consider relations instead of functions, for instance in search problems where there are several possible answers. Given a relation $R \subseteq \{0,1\}^* \times \{0,1\}^*$, we say a quantum algorithm *solves the relation problem* R *with probability* α if there is a quantum algorithm computing a function $F : \{0,1\}^* \to \{0,1\}^*$ with probability α such that $(x, F(x)) \in R$ for all inputs x.

Now that the quantum model of computation is properly defined, we can express the fact that every classical computation can be implemented by a quantum computer, although at a non-negligible cost.

Theorem 2.1 ([Ben89, LS90])**.** *Given any* $\varepsilon > 0$ *and any classical computation with running time* T *and space complexity* S*, there exists an equivalent reversible classical computation with running time* $O(T^{1+\varepsilon}/S^{\varepsilon})$ *and space complexity* $O(S(1 + \ln(T/S)))$*.*

Corollary 2.2. *Given any* $\varepsilon > 0$ *and any classical computation with running time* T *and space complexity* S*, there exists an equivalent quantum circuit of size* $O(T^{1+\varepsilon}/S^{\varepsilon})$ *using* $O(S(1 + \ln(T/S)))$ *qubits.*

Quantum Errors. Quantum algorithms are probabilistic in nature and therefore prone to return wrong results with small probability. But quantum circuits also suffer from two more intrinsic sources of errors in quantum circuits: universal gates approximation and noise. Recall that a set of gate is universal if any unitary operator can be approximated arbitrarily well. Hence, by definition, a quantum circuit built from universal gates will only be an approximation of the true transformation that we analyze. Furthermore, quantum circuits do not exist in isolation and are subject to decoherence when interacting with their environments. This process, as well as other quantum noises, can be modeled by several types of faults or noise and is the subject of active research. See e.g. [HLGJ20] for example for a study on the effects of decoherence on QRACM (defined in the next section) in full generality. One

important question in this area is whether one can build efficient quantum error correction to achieve fault-tolerant quantum computations (see e.g. [Got09] for an introduction on this topic). In this thesis, we will ignore all errors in quantum circuits. This assumption is justified by the theoretical nature of the problems studied and the fact that the cost of error correction is negligible compared to the cost of our algorithms (which is always exponential).

2.2.2 Access to Memory

All the quantum algorithms considered in this thesis run in the quantum circuit model. "Baseline" quantum circuits are simply built using a universal quantum gate set. A requirement for many quantum algorithms to process data efficiently is to be able to access classical data in quantum superposition. Such algorithms use quantum random-access memory, often denoted as qRAM, and require the circuit model to be augmented with the so-called "qRAM gate". These qRAM gates are assumed to have a time complexity polylogarithmic in the amount of classical data stored, so that each call is not time consuming. This model is inspired by the classical RAM model where we usually assume memory access in time $O(1)$. Algorithms for subset-sum, lattice sieving and generic decoding problem using qRAM gates achieve time speedups with respect to their classical counterparts.

Given an input register $1 \leqslant i \leqslant r$, which represents the index of a memory cell, and many quantum registers $|x_1, \dots x_r\rangle$, which represent stored data, the qRAM gate fetches the data from register x_i, possibly in superposition:

$$|i\rangle|x_1, \dots x_r\rangle|y\rangle \mapsto |i\rangle|x_1, \dots x_r\rangle|y \oplus x_i\rangle \ .$$

This then enables various quantum data structures, such as the ones depicted in [BJLM13] and [Amb07], with fast access, even if the queries and the data are superpositions. Following the terminology of [Kup13], we consider three types of qRAMs:

- If the input i is classical, then this is the plain quantum circuit model. We can implement it using a universal quantum gate set.
- If the x_j are classical, we have *classical memory with quantum random access* (QRACM). The qRAM gate becomes
$$|i\rangle|y\rangle \mapsto |i\rangle|y \oplus x_i\rangle \ .$$
- In general, we have *quantum memory with quantum random access* (QRAQM). This is the most powerful quantum memory model where the data are also in superposition.

It is possible to implement a QRACM using a universal quantum gate set, albeit at a considerable cost. Given a classical data set $\{x_1, \cdots, x_r\}$, one can construct, in time $\tilde{O}(r)$, a circuit using $\tilde{O}(r)$ qubits that implements a QRACM for this data set. The obtained circuit then allows query in the form $|i\rangle|y\rangle \mapsto |i\rangle|y \oplus x_i\rangle$ and has circuit depth $O(\text{polylog}(r))$ [GLM08, KP20, MGM20, HLGJ20]. To the best of our knowledge, no physical architecture of circuit depth $O(\text{polylog}(r))$ has been proposed for the QRAQM model. Note that even low depth implementation of QRACM has at least $\Omega(r)$ gates, hence has time complexity at least $\Omega(r)$ by our definition. Therefore, the assumption that the qRAM gates have time complexity $\text{polylog}(r)$ is very strong and corresponds to parallel evaluation of the circuit.

All known quantum algorithms for subset-sum with a quantum time speedup over the best classical one require QRAQM. For comparison, speedups on heuristic lattice sieving algorithms exist in the QRACM model [LMvdP15b, KMPM19], including the best one to date [Laa15a].

2.2.3 Quantum Search

One of the most well-known quantum algorithms is Grover's unstructured search algorithm [Gro96a]. Suppose we have a set of objects named $\{0, 1, \ldots, N-1\}$, of which some are *targets*. We say that an oracle $\mathcal{O}$ *identifies the targets* if, in the classical (resp. quantum) setting, $\mathcal{O}(i) = 1$ (resp. $\mathcal{O}|i\rangle = -|i\rangle$) when i is a target and $\mathcal{O}(i) = 0$ (resp. $\mathcal{O}|i\rangle = |i\rangle$) otherwise. Given such an oracle $\mathcal{O}$, the goal is to find a target $j \in \{0, 1, \ldots, N-1\}$ by making queries to the oracle $\mathcal{O}$. Note that this is a relation problem.

It is very common in quantum search problems to assume that the oracle identifies targets by swapping the phase: $\mathcal{O}|i\rangle = (-1)^{b_i}|i\rangle$ where $b_i = 1$ for targets and $b_i = 0$ for other objects. At the same time, many quantum algorithms[1] (and QRACM in particular) provide a transformation $\mathcal{O}'$ such that $\mathcal{O}'|i\rangle|0\rangle = |i\rangle|b_i\rangle$. The latter can always be transformed into the former by applying $\mathcal{O}'$ to the state $|-\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$ and discarding the second qubit.

In the search problem, one tries to minimize the number of queries to the oracle. In the classical case, one needs $O(N)$ queries to solve such a problem. Grover, on the other hand, provides a quantum algorithm, that solves the search problem with only $O(\sqrt{N})$ queries [Gro96b] when there is one target, and $O(\sqrt{N/t})$ when there are exactly t targets. When the number of targets is unknown, Brassard *et al.* [BBHT05] provided a modified Grover algorithm that solves the search problem with $O(\sqrt{N/t})$ expected queries and $O(\sqrt{N})$ queries in the worst case with some arbitrarily small probability of failure. Furthermore, virtually all quantum search algorithms not only return a solution to the search problem, but one solution among the t possibilities, uniformly at random. This property can be used to find all solutions (see Remark 2.5). We present here a generalization of Grover's algorithm called amplitude amplification [BHMT02].

Theorem 2.3 (Amplitude amplification, from [BHMT02])**.** *Suppose we have a set of N objects of which some are targets. Let $\mathcal{O}$ be a quantum oracle that identifies the targets. Let $\mathcal{A}$ be a quantum circuit using no intermediate measurements, ie $\mathcal{A}$ is reversible. Let a be the initial success probability of $\mathcal{A}$, that is the probability that a measurement of $\mathcal{A}|0\rangle$ outputs a target. There exists a quantum algorithm that calls $O\left(\sqrt{1/a}\right)$ times $\mathcal{A}$, $\mathcal{A}^\dagger$ and $\mathcal{O}$, uses as many qubits as $\mathcal{A}$ and $\mathcal{O}$, and outputs a target with probability greater than $1 - a$.*

Grover's algorithm is a particular case of this theorem where $\mathcal{A}$ produces a uniform superposition of all objects, in which case $a = \frac{1}{N}$. The theorem then states that we can find a target with probability $1 - \frac{1}{N}$ using $O(\sqrt{N})$ calls to the oracle $\mathcal{O}_f$.

As we will use quantum search as a subprocedure, we make some remarks to justify that, up to additional polynomial factors in time, we can consider it runs with no errors and allows to return all the solutions efficiently.

Remark 2.4 (Error in a sequence of quantum searches). Throughout this thesis, we will assume that a quantum search in a search space of size S with T solutions runs in exact time

[1] This is in fact what we obtain when we transform a classical circuit into a quantum circuit.

$\sqrt{S/T}$. In practice, there is a constant overhead, but since S and T are always exponential for the problems considered in this thesis, the difference is negligible. Furthermore, this is a probabilistic procedure and it will return a wrong result with a probability of the order $\sqrt{T/S}$. As we can test if an error occurs, by checking if the returned object is marked, we can make it negligible by redoing the quantum search polynomially many times.

Remark 2.5 (Finding all solutions). Assume that a quantum search algorithm returns a solution among the T possibilities, selected uniformly at random. Finding all solutions is then an instance of the coupon collector problem with T coupons [NS60]; all coupons are collected after $O(T\log(T))$ trials on average. However, *in the QRACM model*, this logarithmic factor disappears: we can run the search of Lemma 2.3 with a new test function that returns 0 if the output of $\mathcal{A}$ is incorrect, *or* if it is correct but has already been found. The change to the runtime is negligible, and thus, we collect all solutions with only $O(T)$ searches.

It is also possible to find the minimum of a set of elements stored in the QRACM model using less queries than in the classical case.

Theorem 2.6 ([DH96], Theorem 1)**.** *Let $\mathcal{O}$ be a quantum oracle such that $\mathcal{O}|i\rangle|0\rangle = |i\rangle|x_i\rangle$ for some N numbers $x_1, \ldots, x_N$. Then there exists a quantum algorithm that finds an index j such that x_j is the minimum among all x_i with probability at least $1/2$ and with $O(\sqrt{N})$ queries to $\mathcal{O}$.*

In particular, the assumption of the previous theorem can immediately be satisfied in the QRACM model, by using a qRAM to store $x_1, \ldots, x_N$.

2.2.4 Quantum Walk Algorithms

Quantum walks can be used to generalize quantum search. They allow to obtain polynomial speedups on many partially structured problems, with sometimes optimal results (*e.g.* Ambainis' algorithm for element distinctness [Amb07]). In this section, we will consider walks in the MNRS framework [MNRS11].

Let $G = (V, E)$ be an undirected, connected, regular[2] graph, such that some vertices of G are "marked". Let ε be the fraction of marked vertices, that is a vertex chosen uniformly at random has a probability ε of being marked. The eigenvalues of a graph are the eigenvalues of its adjacency matrix. Let δ be the spectral gap of G, which is defined as the difference between its two largest eigenvalues.

In a *classical* random walk on G, we can start from any vertex and reach the stationary distribution in approximately $\frac{1}{\delta}$ random walk steps. Then, such a random vertex is marked with probability ε. Assume that we have a procedure `Setup` that samples a uniform vertex to start with in time S, `Check` that verifies if a vertex is marked or not in time C and `Update` that performs a walk step in time U, then we will have found a marked vertex in expected time $\mathsf{S} + \frac{1}{\varepsilon}\left(\frac{1}{\delta}\mathsf{U} + \mathsf{C}\right)$.

Quantum walks reproduce the same process, except that their internal state is not a vertex of G, but a superposition of vertices. The walk starts in the uniform superposition $\sum_{v\in V}|v\rangle$, which must be generated by the quantum version of the `Setup` procedure. It repeats

[2] This is sufficient for our purposes and simplifies the analysis. The MNRS framework does not need this assumption.

$\sqrt{1/\varepsilon}$ iterations using U and C that, similarly to amplitude amplification, move the amplitude towards the marked vertices. An update produces, from a vertex, the superposition of its neighbors. Each iteration does not need to repeat $\frac{1}{\delta}$ vertex updates and, instead, takes a time equivalent to $\sqrt{1/\delta}$ updates to achieve an almost uniform distribution.

Theorem 2.7 (Quantum walk [MNRS11])**.** *Let $G = (V, E)$ be a regular graph with spectral gap $\delta > 0$. Let $\varepsilon > 0$ be a lower bound on the probability that a vertex chosen uniformly at random of G is marked. For a random walk on G, let $\mathsf{S}, \mathsf{U}, \mathsf{C}$ be the setup, update and checking cost. Then there exists a quantum algorithm that with constant probability finds a marked vertex in time*

$$\mathsf{S} + \frac{1}{\sqrt{\varepsilon}}\left(\frac{1}{\sqrt{\delta}}\mathsf{U} + \mathsf{C}\right).$$

2.2.5 Quantum Walk on Trees

In the MNRS framework, the graph is assumed to be known in advance, and moreover the initial state of the quantum walk is the stationary distribution of the corresponding random walk. Since the graph is also assumed to be regular for our purposes, the stationary distribution is simply the uniform distribution among all vertices.

Here we would like to use quantum walks in a context where the input graph is a tree defined implicitly by a backtracking algorithm and hence is not known in advance, and where the walk starts at the root of the tree. Furthermore, the tree may no longer be regular.

Quantum Tree Algorithms. We consider a tree $\mathcal{T}$ of an unknown structure with a given root r, to which we are given local classical or quantum query access in the following way:

1. we are given an oracle which, given a node v, returns the number of children $d(v)$ for this node (if $d(v) = 0$, v is called a leaf);

2. we are given an oracle which, given a node v and $i \in [d(v)]$, returns the i-th child of v.

We denote by $V(\mathcal{T})$ its set of nodes, $L(\mathcal{T})$ its set of leaves, $d(\mathcal{T}) = \max_{v \in V(\mathcal{T})} d(v)$ its degree and $n(\mathcal{T})$ an upper-bound of its depth. When there is no ambiguity, we use d and n directly without the argument $\mathcal{T}$. We also denote by $\#\mathcal{T}$ the number of nodes of the tree $\mathcal{T}$.

Trees of unknown structure can come up in backtracking algorithms. Backtracking is a classical algorithm for solving problems such as constraint satisfaction problems (CSP), by performing a tree search in depth-first order. For CSP, each node represents a partial assignment and its children say how to extend this partial assignement. There is an oracle $\mathcal{P} : V(\mathcal{T}) \to \{true, false, indeterminate\}$ verifying whether the constraint is satisfied, such that $\mathcal{P}(v) \in \{true, false\}$ iff v is a leaf. A node $v \in V(\mathcal{T})$ is called marked if $\mathcal{P}(v) = true$. Here, $\mathcal{T}$ is the tree corresponding to a backtracking algorithm listing *all solutions* of the CSP. Note however, that a classical backtracking algorithm might find *a solution* in substantially less than $\#\mathcal{T}$ steps. Montanaro [Mon15] studied the quantum case:

Theorem 2.8 ([Mon15])**.** *There is a quantum algorithm* **ExistSolution**$(\mathcal{T}, T, \mathcal{P}, n, \varepsilon)$ *which given $\varepsilon > 0$, a tree $\mathcal{T}$ such that $d(\mathcal{T}) = O(1)$, a black box function $\mathcal{P}$, and upper bounds T and n on the number of nodes and the depth of $\mathcal{T}$, determines if $\mathcal{T}$ contains a marked node by*

making $O(\sqrt{Tn}\log(1/\varepsilon))$ *queries to* $\mathcal{T}$ *and to the black box function* $\mathcal{P}$*, with a probability of correct answer* $\geqslant 1-\varepsilon$*. It uses* $O(1)$ *auxiliary operations per query and uses* $\texttt{poly}(n)$ *qubits.*

Theorem 2.9 ([Mon15])**.** *There is a quantum algorithm* **FindSolution**$(\mathcal{T},\mathcal{P},n,\varepsilon)$ *which, given* $\varepsilon>0$*, a tree* $\mathcal{T}$ *such that* $d(\mathcal{T})=O(1)$*, a black box function* $\mathcal{P}$*, and an upper bound* n *on the depth of* $\mathcal{T}$*, outputs* x *such that* $\mathcal{P}(x)$ *is true, or "not found" if no such* x *exists by making* $O(\sqrt{\#\mathcal{T}}n^{3/2}\log(n)\log(1/\varepsilon))$ *queries to* $\mathcal{T}$ *and to the black box function* $\mathcal{P}$*, with correctness probability at least* $1-\varepsilon$*. It uses* $O(1)$ *auxiliary operations per query and uses* $\texttt{poly}(n)$ *qubits.*

Notice that Th. 2.9 does not require an upper-bound on $\#\mathcal{T}$ as input.

Remark 2.10. Montanaro's algorithm has the following drawback. Since classical backtracking algorithms are usually optimized to search the most promising branches first, a classical search algorithm may find a marked vertex after examining $T' \ll \#\mathcal{T}$ nodes of the tree. Since the running time of Montanaro's algorithm depends on $\#\mathcal{T}$, the quantum speedup that it achieves can be much less than quadratic (or there might be no speedup at all). This problem is dealt with in [AK17] where the authors give a quantum algorithm making $\tilde{O}(\sqrt{T'}n^{3/2})$ queries to find one marked node, where T' is the number of nodes actually visited by the classical algorithm. We will not need this improved algorithm in this thesis.

Remark 2.11. A unified framework for quantum walk has recently been proposed in [AGJ19] where both the MNRS quantum walk framework and the quantum backtracking algorithms on trees are included.

Ambainis and Kokainis [AK17] gave a quantum algorithm to estimate the size of a tree, under the same local black box query access model, with input a tree $\mathcal{T}$ and a candidate upper bound T_0 on $\#\mathcal{T}$. The algorithm must output an estimate for $\#\mathcal{T}$, *i.e.* either a number of $\hat{T}\leqslant T_0$ or a claim "$\mathcal{T}$ contains more than T_0 vertices". The estimate is δ-correct if:

1. the estimate is $\hat{T}\leqslant T_0$ which satisfies $|T-\hat{T}|\leqslant \delta T$ where T is the actual number of vertices;

2. the estimate is "$\mathcal{T}$ contains more than T_0 vertices" and the actual number of vertices T satisfies $(1+\delta)T > T_0$.

An algorithm solves the *tree size estimation problem up to precision* $1\pm\delta$ with correctness probability at least $1-\varepsilon$ if for any $\mathcal{T}$ and any T_0, the probability that it outputs a δ-correct estimate is at least $1-\varepsilon$.

Theorem 2.12 ([AK17])**.** *There is a quantum algorithm* **TreeSizeEstimation**$(\mathcal{T},T_0,\delta,\varepsilon)$ *which, given* $\varepsilon>0$*, a tree* $\mathcal{T}$*, and upper bounds* d *and* n *on the degree and the depth of* $\mathcal{T}$*, solves tree size estimation up to precision* $1\pm\delta$*, with correctness probability at least* $1-\varepsilon$*. It makes* $O\Big(\frac{\sqrt{nT_0}}{\delta^{1.5}}d\log^2(\frac{1}{\varepsilon})\Big)$ *queries to* $\mathcal{T}$ *and* $O(\log(T_0))$ *non-query gates per query. The algorithm uses* $\texttt{poly}(n,\log(d),\log(T_0),\log(\delta),\log(\log(1/\varepsilon)))$ *qubits.*

2.3 Probability

Probability distributions. Given two random variables X and Y on a set E, we denote by $\mathrm{d_{SD}}$ the *statistical distance* between X and Y, which is defined by

$$\begin{aligned}\mathrm{d_{SD}}(X,Y) &= \tfrac{1}{2}\sum_{z\in E}\left|\Pr_X[X=z]-\Pr_Y[Y=z]\right| \\ &= \sum_{z\in E\,:\,\Pr_X[X=z]>\Pr_Y[Y=z]}\left(\Pr_X[X=z]-\Pr_Y[Y=z]\right).\end{aligned}$$

We write X is ε-close to Y to denote that the statistical distance between X and Y is at most ε. Given a finite set E, we denote by U_E a uniform random variable on E, i.e. for all $x \in E$, $\Pr_{U_E}[U_E = x] = \frac{1}{|E|}$.

Data processing inequality. When analyzing the output distribution of an algorithm, it is often convenient to assume that the input distribution is ideal (e.g. uniform). On the other hand, we will want to run the algorithm on non-ideal input distribution (e.g. with a slight deviation from uniform). In this case, the output distribution will deviate from the ideal output distribution and it is important to quantify this divergence. The statistical distance satisfies the following useful inequality, known as the *data processing inequality*:

$$\mathrm{d_{SD}}(f(X), f(Y)) \leqslant \mathrm{d_{SD}}(X,Y)$$

for any two distributions X and Y and any (possibly randomized) algorithm f. In other words, the error does not increase under the application of f.

Gaussian distribution. The cumulative distribution function (CDF) of the Gaussian distribution of expectation 0 and variance σ^2 is $\frac{1}{2}(1+\mathrm{erf}(\frac{x}{\sigma\sqrt{2}}))$ where the error function is $\mathrm{erf}(z) := \frac{2}{\sqrt{\pi}}\int_0^z e^{-t^2}dt$. The multivariate Gaussian distribution over $\mathbb{R}^m$ of parameter σ selects each coordinate with a Gaussian distribution.

We need the following lemma on the distribution of the vector inner product which directly follows from the Leftover Hash Lemma [ILL89].

Lemma 2.13. *Let $\mathbb{G}$ be a finite abelian group, f be a positive integer and $\mathcal{Y} \subseteq \{0,1\}^f$. Define the inner product $\langle\cdot,\cdot\rangle : \mathbb{G}^f \times \mathcal{Y} \to \mathbb{G}$ by $\langle x, y\rangle = \sum_i x_i y_i$ for all $x \in \mathbb{G}^f, y \in \mathcal{Y}$. Let X, Y be independent and uniform random variables on $\mathbb{G}^f$, $\mathcal{Y}$ respectively. Then*

$$\mathrm{d}_{SD}((\langle X,Y\rangle, X), (U_{\mathbb{G}}, X)) \leqslant \frac{1}{2}\cdot\sqrt{\frac{|\mathbb{G}|}{|\mathcal{Y}|}},$$

where $U_{\mathbb{G}}$ is uniform in $\mathbb{G}$ and independent of X.

We will also need the Chernoff-Hoeffding bound [Hoe63].

Lemma 2.14. *Let $X_1, \ldots, X_M$ be the independent and identically distributed random boolean variables of expectation p. Then for $\varepsilon > 0$,*

$$\Pr\left[\frac{1}{M}\sum_{i=1}^{M} X_i \leqslant p(1-\delta)\right] \leqslant \left(\frac{e^{-\delta}}{(1-\delta)^{1-\delta}}\right)^{pM}.$$

2.4 Lattices

Lattices. A *lattice* $\mathcal{L}$ is a discrete subgroup of $\mathbb{R}^m$, or equivalently a set $\mathcal{L}(\mathbf{b}_1, \ldots, \mathbf{b}_n) = \{\sum_{i=1}^n x_i \mathbf{b}_i \ : \ x_i \in \mathbb{Z}\}$ of all integer combinations of n linearly independent vectors $\mathbf{B} = \{\mathbf{b}_1, \ldots, \mathbf{b}_n\} \subset \mathbb{R}^m$. Such $\mathbf{b}_i$'s form a *basis* of $\mathcal{L}$. Bases are not unique, one lattice basis may be transformed into another one by applying an arbitrary unimodular transformation. All the bases have the same number n of elements, called the *dimension* or *rank* of $\mathcal{L}$, and the same n-dimensional volume of the parallelepiped $\{\sum_{i=1}^n a_i \mathbf{b}_i \ : \ a_i \in [0, 1)\}$ they generate. We call this volume the *co-volume* of $\mathcal{L}$, denoted by $\mathrm{covol}(\mathcal{L})$. It is also sometimes call the *determinant* of the lattice, denoted by $\det \mathcal{L}$. This quantity is equal to the square root of the determinant of the Gram matrix $\mathbf{B}^T \mathbf{B}$: $\mathrm{covol}(\mathcal{L}) = \det \mathcal{L} = \sqrt{\det \mathbf{B}^T \mathbf{B}}$. The lattice $\mathcal{L}$ is said to be *full-rank* if $n = m$. When the lattice is full rank, $\mathbf{B}$ is a square matrix and we have $\mathrm{covol}(\mathcal{L}) = \det \mathcal{L} = |\det(\mathbf{B})|$.

We denote by $\lambda_1(\mathcal{L})$ the first minimum of $\mathcal{L}$, defined as the length of a shortest non-zero vector of $\mathcal{L}$.

For a rank n lattice $\mathcal{L} \subset \mathbb{R}^m$, the *dual lattice*, denoted $\mathcal{L}^*$, is defined as the set of all points in $\mathsf{span}(\mathcal{L})$ that have integer inner products with all lattice points,

$$\mathcal{L}^* = \{\mathbf{w} \in \mathsf{span}(\mathcal{L}) : \forall \mathbf{y} \in \mathcal{L}, \langle \mathbf{w}, \mathbf{y} \rangle \in \mathbb{Z}\} \ .$$

Similarly, for a lattice basis $\mathbf{B} = (\mathbf{b}_1, \ldots, \mathbf{b}_n)$, we define the dual basis $\mathbf{B}^* = (\mathbf{b}_1^*, \ldots, \mathbf{b}_n^*)$ to be the unique set of vectors in $\mathsf{span}(\mathcal{L})$ satisfying $\langle \mathbf{b}_i^*, \mathbf{b}_j \rangle = 1$ if $i = j$, and 0, otherwise. It can be shown that $\mathcal{L}^*$ is itself a rank n lattice and $\mathbf{B}^*$ is a basis of $\mathcal{L}^*$.

In what follows, unless explicitely stated, we consider full rank lattices.

Orthogonalization. For a basis $B = (\mathbf{b}_1, \ldots, \mathbf{b}_n)$ of a lattice $\mathcal{L}$ and $i \in \{1, \ldots, n\}$, we denote by π_i the orthogonal projection on $\mathrm{span}(\mathbf{b}_1, \ldots, \mathbf{b}_{i-1})^\perp$. The *Gram-Schmidt orthogonalization* of the basis B is defined as the sequence of orthogonal vectors $B^\star = (\mathbf{b}_1^\star, \ldots, \mathbf{b}_n^\star)$, where $\mathbf{b}_i^\star := \pi_i(\mathbf{b}_i)$. We can write each $\mathbf{b}_i$ as $\mathbf{b}_i^\star + \sum_{j=1}^{i-1} \mu_{i,j} \mathbf{b}_j^\star$ for some unique $\mu_{i,1}, \ldots, \mu_{i,i-1} \in \mathbb{R}$. Thus, we may represent the $\mu_{i,j}$'s by a lower-triangular matrix μ with unit diagonal. $\pi_i(\mathcal{L})$ is a lattice of rank $n + 1 - i$ generated by $\pi_i(\mathbf{b}_i), \ldots, \pi_i(\mathbf{b}_n)$, with $\mathrm{covol}(\pi_i(\mathcal{L})) = \prod_{j=i}^n \left\|\mathbf{b}_j^\star\right\|$.

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

Heuristic 2.15. *Given a full-rank lattice $\mathcal{L} \subseteq \mathbb{R}^n$ and a measurable set $S \subseteq \mathbb{R}^n$, the number of points in $S \cap \mathcal{L}$ is approximately* $\mathrm{vol}(S)/\mathrm{covol}(\mathcal{L})$.

Both rigorous results and counter-examples are known (see [AN17]). One should therefore experimentally verify its use, especially for pruned enumeration which relies on strong versions of the heuristic, where the set S is not fixed, depending on a basis of L.

Discrete Gaussian Distribution. For any $s > 0$, define $\rho_s(\boldsymbol{x}) = \exp(-\pi \|\boldsymbol{x}\|^2 / s^2)$ for all $\boldsymbol{x} \in \mathbb{R}^n$. We write ρ for ρ_1. For a discrete set S, we extend ρ to sets by $\rho_s(S) = \sum_{\boldsymbol{x} \in S} \rho_s(\boldsymbol{x})$. Given a lattice $\mathcal{L}$, the *discrete Gaussian* $D_{\mathcal{L},s}$ is the distribution over $\mathcal{L}$ such that the probability of a vector $\boldsymbol{y} \in \mathcal{L}$ is proportional to $\rho_s(\boldsymbol{y})$:

$$\Pr_{X \sim D_{\mathcal{L},s}}[X = \boldsymbol{y}] = \frac{\rho_s(\boldsymbol{y})}{\rho_s(\mathcal{L})}.$$

Kissing Number and related quantities. For any lattice $\mathcal{L} \subset \mathbb{R}^n$ and $d > 0$, let $N(\mathcal{L}, r)$ denote the number of nonzero lattice vectors of length at most r. A natural question is to bound this quantity in terms of r. When $r < \lambda_1(\mathcal{L})$, only the origin lies inside the ball so $N(\mathcal{L}, r) = 0$. When $r = \lambda_1(\mathcal{L})$, this quantity is known as the kissing number $\tau(\mathcal{L})$ of the lattice:

$$\tau(\mathcal{L}) = \{\boldsymbol{x} \in \mathbb{R}^n : \|\boldsymbol{x}\| = \lambda_1(\mathcal{L})\}.$$

Finally when $r \to \infty$, $N(\mathcal{L}, r) = \frac{r^n \mathrm{vol}(B_n(1))}{\det \mathcal{L}} + o(r^n)$ by the geometric interpretation of the determinant of a lattice. The precise behavior for intermediate values of r, however, is unclear and for that reason we introduce the quantity

$$\gamma(\mathcal{L}) = \inf\{\gamma : \forall r \geqslant 1,\ N(\mathcal{L}, r\lambda_1(\mathcal{L})) \leqslant \gamma \cdot r^n\}. \tag{2.1}$$

It is clear by the definition that $\gamma(\mathcal{L}) \geqslant \tau(\mathcal{L})$. The best known upper bound on this quantity comes from the breakthrough work of Kabatyanskii and Levenshtein [KL78] which implies [PS09] that

$$\gamma(\mathcal{L}) \leqslant 2^{0.401n + o(n)}.$$

The same work also implies that $\tau(\mathcal{L}) \leqslant 2^{0.401n+o(n)}$ although we will not need it in this thesis. Recently, in a breakthrough result, Serge Vlăduţ [Vlă19] gave a construction of an infinite set of lattices such that their kissing number is greater than $2^{0.0338n+o(n)}$. This is the first (and only known) construction of a family of lattices with kissing number $2^{\Omega(n)}$. Given this result (and the fact that it was hard to find such a family of lattices), one might conjecture that $\tau(\mathcal{L}) = c^{n+o(n)}$ for some constant c much smaller than $2^{0.401}$. Moreover, in practice most lattices have a much smaller kissing number which is often $2^{o(n)}$. Given the close connection between $\tau(\mathcal{L})$ and $\gamma(\mathcal{L})$, it is not unreasonable to conjecture that $\gamma(\mathcal{L})$ is also a $2^{o(n)}$ for most lattices. In view of the fact that $\gamma(\mathcal{L})$ can be anywhere between 1 and $2^{0.401n+o(1)}$, we will study the dependence of the time complexity of our algorithms on $\gamma(\mathcal{L})$ by introducing

$$\beta(\mathcal{L}) = \gamma(\mathcal{L})^{1/n}. \tag{2.2}$$

The upper bound above can then be reformulated as $\beta(\mathcal{L}) \leqslant 2^{0.401+o(1)}$ for any lattice $\mathcal{L}$.

Lattice problems. The following problem plays a central role in this thesis.

Definition 2.16. For $\delta = \delta(n) \geqslant 0$, σ a function that maps lattices to non-negative real numbers, and $m = m(n) \in \mathbb{N}$, δ-DGS_σ^m (the Discrete Gaussian Sampling problem) is defined as follows: The input is a basis $\mathbf{B}$ for a lattice $\mathcal{L} \subset \mathbb{R}^n$ and a parameter $s > \sigma(\mathcal{L})$. The goal is to output a sequence of m vectors whose joint distribution is δ-close to m independent samples from $D_{\mathcal{L},s}$.

We omit the parameter δ if $\delta = 0$, and the parameter m if $m = 1$. We stress that δ bounds the statistical distance between the *joint* distribution of the output vectors and m independent samples from $D_{\mathcal{L},s}$.

Definition 2.17. The search problem SVP (Shortest Vector Problem) is defined as follows: The input is a basis $\mathbf{B}$ for a lattice $\mathcal{L} \subset \mathbb{R}^n$. The goal is to output a nonzero vector $\boldsymbol{y} \in \mathcal{L}$ with minimum l_2-norm, that is $\|\mathbf{y}\| = \lambda_1(\mathcal{L})$.

Definition 2.18. The search problem γ-approx-SVP (γ-Approximate Shortest Vector Problem) is defined as follows: The input is a basis $\mathbf{B}$ for a lattice $\mathcal{L} \subset \mathbb{R}^n$ and $\gamma > 1$. The goal is to output a nonzero vector $\boldsymbol{y} \in \mathcal{L}$ such that $\|\mathbf{y}\| \leqslant \gamma\lambda_1(\mathcal{L})$.

Definition 2.19. The search problem CVP (Closest Vector Problem) is defined as follows: The input is a basis $\mathbf{B}$ for a lattice $\mathcal{L} \subset \mathbb{R}^n$ and a target vector $\mathbf{t} \in \mathbb{R}^n$. The goal is to output a vector $\mathbf{y} \in \mathcal{L}$ such that $\|\mathbf{y} - \mathbf{t}\| = \mathrm{dist}(\mathbf{t}, \mathcal{L})$.

Definition 2.20. The search problem γ-approx-CVP (γ-Approximate Closest Vector Problem) is defined as follows: The input is a basis $\mathbf{B}$ for a lattice $\mathcal{L} \subset \mathbb{R}^n$, a target vector $\mathbf{t} \in \mathbb{R}^n$ and $\gamma > 1$. The goal is to output a vector $\mathbf{y} \in \mathcal{L}$ such that $\|\mathbf{y} - \mathbf{t}\| \leqslant \gamma\mathrm{dist}(\mathbf{t}, \mathcal{L})$.

Definition 2.21. For $\alpha = \alpha(n) < 1/2$, the search problem α-BDD (Bounded Distance Decoding) is defined as follows: The input is a basis $\mathbf{B}$ for a lattice $\mathcal{L} \subset \mathbb{R}^n$ and a target vector $\mathbf{t} \in \mathbb{R}^n$ with $\mathrm{dist}(\boldsymbol{t}, \mathcal{L}) \leqslant \alpha \cdot \lambda_1(\mathcal{L})$. The goal is to output a vector $\mathbf{y} \in \mathcal{L}$ with $\|\mathbf{y}-\mathbf{t}\| = \mathrm{dist}(\mathbf{t}, \mathcal{L})$.

Note that while the other problems become more difficult as the approximation factor γ becomes smaller, α-BDD becomes more difficult as α gets larger.

For convenience, when we discuss the running time of algorithms solving the above problems, we will often ignore polynomial factors in the bit-length of the individual input basis vectors (i.e. we consider only the dependence on the ambient dimension n).

Smoothing parameter. For a lattice $\mathcal{L}$ and $\varepsilon > 0$, the *smoothing parameter* $\eta_\varepsilon(\mathcal{L})$ is the smallest s such that $\rho_{1/s}(\mathcal{L}^*) \leqslant 1 + \varepsilon$. Intuitively, this parameter provides the width beyond which the discrete Gaussian measure on a lattice behaves like a continuous one. The smoothing parameter has the following well-known property, namely that above the smoothing parameter, the discrete Gaussian measure is essentially invariant under shifts.

Lemma 2.22 ([Reg05, Claim 3.8])**.** *For any lattice $\mathcal{L} \subset \mathbb{R}^n$, $\boldsymbol{c} \in \mathbb{R}^n$, $\varepsilon > 0$, and $s \geqslant \eta_\varepsilon(\mathcal{L})$,*

$$\frac{1-\varepsilon}{1+\varepsilon} \leqslant \frac{\rho_s(\mathcal{L}+\boldsymbol{c})}{\rho_s(\mathcal{L})} \leqslant 1 \ .$$

Corollary 2.23. *Let $\mathcal{L} \subset \mathbb{R}^n$ be a lattice, q be a positive integer, and let $s \geqslant \eta_\varepsilon(q\mathcal{L})$. Let C be a random coset in $\mathcal{L}/q\mathcal{L}$ sampled such that for all $\boldsymbol{c} \in \mathcal{L}/q\mathcal{L}$, $\Pr[C = q\mathcal{L} + \boldsymbol{c}] = \frac{\rho_s(q\mathcal{L}+\boldsymbol{c})}{\rho_s(\mathcal{L})}$. Also, let U be a coset in $\mathcal{L}/q\mathcal{L}$ sampled uniformly at random. Then*

$$\mathrm{d}_{SD}(C, U) \leqslant 2\varepsilon \ .$$

Proof. By Lemma 2.22, we have that

$$\rho_s(q\mathcal{L}) \geqslant \rho_s(q\mathcal{L}+\mathbf{c}) \geqslant \frac{1-\varepsilon}{1+\varepsilon}\rho_s(q\mathcal{L}) \ ,$$

for any $\boldsymbol{c} \in \mathcal{L}/q\mathcal{L}$ and hence,

$$q^n\rho_s(q\mathcal{L}) \geqslant \sum_{\boldsymbol{c}\in\mathcal{L}/q\mathcal{L}} \rho_s(q\mathcal{L}+\boldsymbol{c}) = \rho_s(\mathcal{L})$$

Therefore,

$$\frac{\rho_s(q\mathcal{L}+\boldsymbol{c})}{\rho_s(\mathcal{L})} \geqslant \frac{1-\varepsilon}{1+\varepsilon}\cdot\frac{\rho_s(q\mathcal{L})}{\rho_s(\mathcal{L})} \geqslant \frac{1-\varepsilon}{1+\varepsilon}\cdot\frac{1}{q^n}\,.$$

We conclude that

$$\begin{aligned}
\mathrm{d_{SD}}(C,U) &= \sum_{\boldsymbol{c}\in\mathcal{L}/q\mathcal{L}\,:\,\Pr[C=\boldsymbol{c}]<\Pr[U=\boldsymbol{c}]} (\Pr[U=\boldsymbol{c}]-\Pr[C=\boldsymbol{c}]) \\
&\leqslant \sum_{\boldsymbol{c}\in\mathcal{L}/q\mathcal{L}\,:\,\Pr[C=\boldsymbol{c}]<\Pr[U=\boldsymbol{c}]} \Pr[U=\boldsymbol{c}]\left(1-\frac{1-\varepsilon}{1+\varepsilon}\right) \\
&\leqslant \sum_{\boldsymbol{c}\in\mathcal{L}/q\mathcal{L}} \Pr[U=\boldsymbol{c}]\frac{2\varepsilon}{1+\varepsilon} \\
&\leqslant \frac{2\varepsilon}{1+\varepsilon}\,,
\end{aligned}$$

as needed. □

Lemma 2.24 ([CDLP13, Lemma 2.4])**.** *For any lattice $\mathcal{L}\subset\mathbb{R}^n, \varepsilon\in(0,1)$ and $k>1$, we have $k\eta_\varepsilon(\mathcal{L}) > \eta_{\varepsilon^{k^2}}(\mathcal{L})$*

The following lemma shows that there is a relation between the first minimum of a lattice and the smoothing parameter of its dual lattice.

Lemma 2.25 (Variant of [ADRS15, Lemma 6.1])**.** *For any lattice $\mathcal{L}\subset\mathbb{R}^n$ and $\varepsilon\in(0,1)$,*

$$\sqrt{\frac{\ln(1/\varepsilon)}{\pi}} < \lambda_1(\mathcal{L})\eta_\varepsilon(\mathcal{L}^*) < \sqrt{\frac{\beta(\mathcal{L})^2 n}{2\pi e}}\cdot\varepsilon^{-1/n}\cdot(1+o(1)), \tag{2.3}$$

and if $\varepsilon \leqslant (e/\beta(\mathcal{L})^2+o(1))^{-\frac{n}{2}}$, we also have

$$\sqrt{\frac{\ln(1/\varepsilon)}{\pi}} < \lambda_1(\mathcal{L})\eta_\varepsilon(\mathcal{L}^*) < \sqrt{\frac{\ln(1/\varepsilon)+n\ln\beta(\mathcal{L})+o(n)}{\pi}}. \tag{2.4}$$

as n tends to infinity.

Remark 2.26. As noted in [ADRS15] below Lemma 6.1, the inequality (2.3) actually holds for all $\varepsilon\in(0,1)$ so we dropped the condition in the first case.

Remark 2.27. In Lemma 6.1 of [ADRS15], β comes from Lemma 4.2 of the same paper and only needs to satisfy the equation $|\mathcal{L}\cap T_r| \leqslant \beta^{n+o(n)}r^n$ for all r where $T_r = \{x\in\mathbb{R}^n : r\leqslant\|x\|\leqslant(1+\frac{1}{n})r\}$, assuming the lattice is normalized so that $\lambda_1(\mathcal{L})=1$. A trivial upper bound on $|\mathcal{L}\cap T_r|$ is $N(\mathcal{L},r')$, the number of point in $\mathcal{L}$ of radius at most $r'=(1+\frac{1}{n})r$. By our definitions (2.1) and (2.2), this is bounded by $\gamma(\mathcal{L})\big((1+\frac{1}{n})r\big)^n \leqslant \beta(\mathcal{L})^n e r^n$ and therefore we can replace β by our $\beta(\mathcal{L})$.

Micciancio and Peikert [MP13] showed the following result about the distribution resulting from the sum of many Gaussian samples. Essentially, above the smoothing parameter, the sum of independent Gaussian samples is very close to a Gaussian distribution.

Theorem 2.28 ([MP13, Theorem 3.3]). *Let $\mathcal{L}$ be an n-dimensional lattice, $\boldsymbol{z} \in \mathbb{Z}^m$ a nonzero integer vector, $s_i \geqslant \sqrt{2}\|\boldsymbol{z}\|_\infty \cdot \eta_\varepsilon(\mathcal{L})$, and $\mathcal{L} + \boldsymbol{c_i}$ arbitrary cosets of $\mathcal{L}$ for $i = 1 \cdots, m$. Let $\boldsymbol{y_i}$ be independent vectors with distributions $D_{\mathcal{L}+\boldsymbol{c_i},s_i}$, respectively. Then the distribution of $\boldsymbol{y} = \sum_{i=1}^m z_i \boldsymbol{y_i}$ is $m\varepsilon$ close to $D_{Y,s}$, where $Y = \gcd(\boldsymbol{z})\mathcal{L} + \sum_{i=1}^m z_i \boldsymbol{c_i}$, and $s = \sqrt{\sum (z_i s_i)^2}$.*

We will need the following theorems to sample vectors from a discrete Gaussian distribution with a large width.

Theorem 2.29 ([ADRS15, Proposition 2.17]). *For any $\varepsilon \leqslant 0.99$, there is a randomized algorithm that takes as input a lattice $\mathcal{L} \in \mathbb{R}^n$, $M \in \mathbb{Z}_{>0}$ (the desired number of output vectors), and $s > 2^{n \log\log n/\log n} \cdot \eta_\varepsilon(\mathcal{L})$, and outputs M independent samples from $D_{\mathcal{L},s}$ in time $M \cdot poly(n)$.*

We need to recall the definition of the *honest* Discrete Gaussian Sampling problem, introduced in [ADRS15].

Definition 2.30 ([ADRS15, Definition 5.1]). For $\varepsilon \geqslant 0$, σ a function that maps lattices to non-negative real numbers, and $m \in \mathbb{N}$, the *honest* Discrete Gaussian Sampling problem ε-hDGS$_\sigma^m$ is defined as follows: the input is a basis B for a lattice $\mathcal{L} \subset \mathbb{R}^n$ and a parameter $s > 0$. The goal is for the output distribution to be $\varepsilon-$close to $D_{\mathcal{L},s}^{m'}$ for some independent random variable $m' \geqslant 0$. If $s > \sigma(\mathcal{L})$ then m' must be equal to m.

Theorem 2.31 ([ADRS15, Theorem 5.11]). *Let σ be the function that maps a lattice $\mathcal{L}$ to $\sqrt{2}\eta_{1/2}(\mathcal{L})$. Then, there is an algorithm that solves $\exp(-\Omega(\kappa))$-hDGS$_\sigma^{2^{n/2}}$ in time $2^{n/2+\text{polylog}(\kappa)+o(n)}$ for an $\kappa \geqslant \Omega(n)$.*

Lemma 2.32 ([ADRS15, Lemma 5.12]). *There is a randomized polynomial-time algorithm that takes as input a lattice $\mathcal{L} \subset \mathbb{R}^n$ of rank n and an integer a with $n/2 \leqslant a < n$ and returns a super lattice $\mathcal{L}' \supset \mathcal{L}$ of index 2^a with $\mathcal{L}' \subseteq \mathcal{L}/2$ such that for any $\varepsilon \in (0,1)$, we have $\eta_{\varepsilon'}(\mathcal{L}') \leqslant \eta_\varepsilon(\mathcal{L})/\sqrt{2}$ with probability at least $1/2$ where $\varepsilon' := 2\varepsilon^2 + 2^{(n/2)+1-a}(1+\varepsilon)$.*

More context about the above results is available in Section 3.1. We will also make heavy use of the following reduction from α-BDD to DGS that was shown in [DRS14].

Theorem 2.33 ([DRS14, Theorem 3.1]). *For any $\varepsilon \in (0, 1/200)$, let $\phi(\mathcal{L}) \equiv \frac{\sqrt{\ln(1/\varepsilon)/\pi - o(1)}}{2\eta_\varepsilon(\mathcal{L}^*)}$. Then there exists an algorithm that solves CVP^ϕ using $m \cdot poly(n)$ arithmetic operations where $m = O(\frac{n\log(1/\varepsilon)}{\sqrt{\varepsilon}})$. Moreover, the preprocessing consists of m vectors sampled from $D_{\mathcal{L}^*,\eta_\varepsilon(\mathcal{L}^*)}$.*

Remark 2.34. We are going to use this reduction in the superpolynomial regime: typically m will be exponential in n. This leaves unclear the space complexity of the reduction. The reduction works by evaluating a polynomial number of times functions of the form $\sum_{i=1}^m f_i(\boldsymbol{x})$ where each f_i is a polynomial time computable function that depends on the i^{th} DGS sample. Furthermore, all the complexities above are in terms of arithmetic operations, not bit complexity. If we assume that all the DGS samples have poly(n) bit-size then the reduction has time complexity $m \cdot \text{poly}(n)$ and space complexity $O(\text{poly}(n) + \log m)$ *excluding the storage space of the m vectors provided by the DGS*. Finally, as noted in the proof of the theorem in [DRS14], only the preprocessing is probabilistic and with probability at least $1 - 2^{-\Omega(n)}$ over the choice of the samples, the algorithm will deterministically solve all CVP^ϕ instances.

One drawback of the theorem above is that it requires a sampler for the discrete Gaussian distribution *exactly at the smoothing parameter* $\eta_\varepsilon(\mathcal{L})$ which is generally not known. For these reasons, we provide a workaround in the form of Theorem 2.40 in the next section.

Finally, the following theorem proved in [CCL18] shows how to solve SVP by an exponential number of calls to a α-BDD oracle.

Theorem 2.35 ([CCL18, Theorem 8])**.** *Given a basis matrix $\mathbf{B} \subset \mathbb{R}^{n\times n}$ for lattice $\mathcal{L}(\mathbf{B}) \subset \mathbb{R}^n$, a target vector $\boldsymbol{t} \in \mathbb{R}^n$, an α-BDD oracle BDD_α with $\alpha < 0.5$, and an integer scalar $p > 0$. Let $f_p^\alpha : \mathbb{Z}_p^n \to \mathbb{R}^n$ be*

$$f_p^\alpha(\boldsymbol{s}) = -p \cdot \mathsf{BDD}_\alpha(\mathcal{L}, (\mathbf{B}\,\boldsymbol{s} - \boldsymbol{t})/p) + \mathbf{B}\,\boldsymbol{s}.$$

If $\operatorname{dist}(\boldsymbol{t}, \mathcal{L}) \leqslant \alpha\lambda_1(\mathcal{L})$, then the list $m = \{f_p^\alpha(\boldsymbol{s}) \mid \boldsymbol{s} \in \mathbb{Z}_p^n\}$ contains all lattice points at distance less than or equal to $p\alpha\lambda_1(\mathcal{L})$ to $\boldsymbol{t}$.

Remark 2.36. In the original statement in [CCL18], the list m contains all lattice points at distance *less than* $p\alpha\lambda_1(\mathcal{L})$ to $\boldsymbol{t}$. However by inspecting the proofs, the list m also contains all lattice points at distance $p\alpha\lambda_1(\mathcal{L})$ to $\boldsymbol{t}$.

2.5 Reduction from CVP to DGS

The goal of this section is to improve Theorem 2.33 to not requires a sampler for the discrete Gaussian distribution *exactly at the smoothing parameter* $\eta_\varepsilon(\mathcal{L})$. Indeed, we usually do not know $\eta_\varepsilon(\mathcal{L})$ and it is a nontrivial problem to even estimate it [CDLP13]. It was claimed in [ADRS15, Theorem 7.3] that the reduction still holds if we only provide a DGS oracle above the smoothing parameter. Unfortunately, this is not entirely satisfactory for two reasons: [ADRS15] contains no proof[3] and the modified statement does not make it clear that the parameters of the DGS instance do not depend on the CVP instance, like in Theorem 2.33. For these reasons, we provide a self-contained proof in the form of Theorem 2.40 in the next section. We first prove some technical lemmas on the discrete Gaussian distribution.

Lemma 2.37. *For any lattice $\mathcal{L} \subset \mathbb{R}^n$, $s > 0$ and $r \geqslant s\sqrt{n}/\lambda_1(\mathcal{L})$,*

$$\rho_s(\mathcal{L}\backslash B_n(r\lambda_1(\mathcal{L}))) \leqslant r^n \beta(\mathcal{L})^{n+o(n)} \rho_s(\mathcal{L}\backslash\{0\})^{r^2}.$$

Proof. Let $t = 1 + 1/n$, $R = r\lambda_1(\mathcal{L})$, $r_i = Rt^i$ and $T_i = B_n(r_{i+1})\backslash B_n(r_i)$ for all $i \in \mathbb{N}$. Then by definition of $\beta(\mathcal{L})$,

$$|\mathcal{L} \cap T_i| \leqslant |\mathcal{L} \cap B_n(r_{i+1})| \leqslant \beta(\mathcal{L})^{n+o(n)}(rt^{i+1})^n$$

It follows that

$$\rho_s(\mathcal{L}\backslash B_n(R)) = \sum_{i=0}^{\infty} \rho_s(\mathcal{L} \cap T_i) \leqslant \sum_{i=0}^{\infty} |\mathcal{L} \cap T_i| e^{-\pi \frac{r_i^2}{s^2}} \leqslant \beta(\mathcal{L})^{n+o(n)} \sum_{i=0}^{\infty} \underbrace{(rt^{i+1})^n e^{-\pi \frac{R^2}{s^2} t^{2i}}}_{=f(i)}$$

where $f(i) = (rt^{i+1})^n e^{-\pi \frac{R^2}{s^2} t^{2i}}$. But check that for all $i \in \mathbb{N}$,

$$\frac{f(i+1)}{f(i)} = t^n e^{-\pi \frac{R^2}{s^2} t^{2i}(t^2-1)} \leqslant e^{1-2\pi \frac{R^2}{ns^2}} \leqslant e^{1-3\pi} < 1/2$$

[3] All the authors of [DRS14] are all in [ADRS15] so the statement is still probably true.

where we have used that $R \geqslant s\sqrt{n}$, $t^2 - 1 \leqslant \frac{3}{n}$, $t^n \leqslant e$ and $t^{2i} \geqslant 1$. It follows that

$$\rho_s(\mathcal{L}\backslash B_n(R)) \leqslant \beta(\mathcal{L})^{n+o(n)} \cdot 2 \cdot f(0) \leqslant \beta(\mathcal{L})^{n+o(n)} \cdot 2(rt)^n e^{-\pi\frac{R^2}{s^2}} \leqslant r^n \beta(\mathcal{L})^{n+o(n)} \cdot e^{-\pi\frac{r^2\lambda_1(\mathcal{L})}{s^2}}.$$

On the other hand,

$$\rho_s(\mathcal{L}\backslash\{0\}) \geqslant e^{-\pi\frac{\lambda_1(\mathcal{L})^2}{s^2}}$$

so the result follows immediately. □

Lemma 2.38. *For any $c > 1$, for any lattice $\mathcal{L} \subset \mathbb{R}^n$, $\varepsilon \in (0, 1/e)$, we have $t\eta_\varepsilon(\mathcal{L}) \leqslant \eta_{\varepsilon^{t^2}e^{o(1)}}(\mathcal{L})$ where $t = 1 + \frac{1}{n^c}$.*

Proof. Let $s = \eta_\varepsilon(\mathcal{L})$, $M = n^c$ and $t = 1 + \frac{1}{M}$ with $c > 1$. Let $r = M$ and check that

$$\frac{\sqrt{n}}{s\lambda_1(\mathcal{L}^*)} \leqslant \sqrt{\frac{n\pi}{\ln(1/\varepsilon)}} \leqslant r \tag{2.5}$$

by Lemma 2.25, for large enough n since $\varepsilon \leqslant 1/e$. Let $\Gamma = (\mathcal{L}^*\backslash\{0\}) \cap B_n(r\lambda_1(\mathcal{L}^*))$, then

$$|\Gamma| \leqslant \beta(\mathcal{L}^*)^{n+o(n)} r^n.$$

Now observe that

$$\begin{aligned}
\rho_{1/ts}(\mathcal{L}^*\backslash\{0\}) &= \sum_{x\in\mathcal{L}^*\backslash\{0\}} \rho_{1/ts}(x) = \sum_{x\in\mathcal{L}^*\backslash\{0\}} \rho_{1/s}(x)^{t^2} \\
&\geqslant \sum_{x\in\Gamma} \rho_{1/s}(x)^{t^2} \geqslant |\Gamma|\left(\frac{1}{|\Gamma|}\sum_{x\in\Gamma}\rho_{1/s}(x)\right)^{t^2} && \text{by Jensen's inequality} \\
&= |\Gamma|^{1-t^2}\rho_{1/s}(\Gamma)^{t^2} \\
&= |\Gamma|^{1-t^2}\left(\rho_{1/s}(\mathcal{L}^*\backslash\{0\}) - \rho_{1/s}(\mathcal{L}^*\backslash B_n(r\lambda_1(\mathcal{L}^*)))\right)^{t^2} \\
&\geqslant \left(\beta(\mathcal{L}^*)^{n+o(n)} r^n\right)^{1-t^2}\left(\varepsilon - \beta(\mathcal{L}^*)^{n+o(n)} r^n \varepsilon^{r^2}\right)^{t^2}
\end{aligned}$$

by (2.5), Lemma 2.37 and since $\rho_{1/s}(\mathcal{L}^*\backslash\{0\}) = \varepsilon$. Now observe that $t^2 - 1 \leqslant 3/M$ and

$$\left(\beta(\mathcal{L}^*)^{n+o(n)} r^n\right)^{-3/M} = e^{-3n^{1-c}(\ln\beta(\mathcal{L}^*)+o(1)+c\ln n)} = e^{-3n^{1-c}O(\ln n)} = e^{-o(1)}$$

since $c > 1$. We also have that

$$\begin{aligned}
\beta(\mathcal{L}^*)^{n+o(n)} r^n \varepsilon^{r^2} &= e^{n\ln\beta(\mathcal{L}^*)+nc\ln(n)+o(n)-(r^2-1)\ln\frac{1}{\varepsilon}}\varepsilon \\
&\leqslant e^{O(n\ln n)-n^{2c}}\varepsilon && \text{since } \varepsilon \leqslant 1/e \\
&\leqslant e^{o(1)}\varepsilon
\end{aligned}$$

for large enough n. It follows that

$$\rho_{1/ts}(\mathcal{L}^*\backslash\{0\}) \geqslant e^{o(1)}(\varepsilon e^{o(1)})^{t^2} \geqslant \varepsilon^{t^2} e^{o(1)}$$

since $t \leqslant 2$. Therefore we must have $\eta_{\varepsilon^{t^2}e^{o(1)}}(\mathcal{L}) \geqslant ts$.

□

In order to state our theorem, we need to introduce a small variant on the notion of an honest Gaussian sampler. We note that any discrete Gaussian sampler can trivially be turned into a semi-honest sampler. The reason we need a semi-honest sampler is that we do not know the value of η_ε in general so we cannot guarantee that all calls will satisfy $s \geqslant \sigma$ when $\sigma = \eta_\varepsilon$.

Definition 2.39. For $\varepsilon \geqslant 0$, σ a function that maps lattices to non-negative real numbers, and $m \in \mathbb{N}$, the *semi-honest* Discrete Gaussian Sampling problem ε-shDGS$^m_\sigma$ is defined as follows: the input is basis B for a lattice $\mathcal{L} \subset \mathbb{R}^n$ and a parameter $s > 0$. The goal is for the output distribution to be $\varepsilon-$close to $D^m_{\mathcal{L},s}$ when $s \geqslant \sigma(\mathcal{L})$, and can be any distribution of m lattices vectors otherwise.

Theorem 2.40. *For any $\alpha > 0$, any $\varepsilon \leqslant \min(e^{-n^\alpha}, 1/200)$, let $\phi(\mathcal{L}) \equiv \frac{\sqrt{\ln(1/\varepsilon)/\pi - o(1)}}{2\eta_\varepsilon(\mathcal{L}^*)}$. There exists an algorithm that, on input $\mathcal{L}$ and with constant probability, constructs a deterministic CVP^ϕ oracle for $\mathcal{L}$ by doing $\mathrm{poly}(n)$ calls to a 0.5-shDGS$^m_{\eta_\varepsilon}$ sampler on the lattice $\mathcal{L}^*$ and requires storage space for $m \cdot \mathrm{poly}(n)$ lattice vectors, where $m = O(\frac{n \ln(1/\varepsilon)}{\sqrt{\varepsilon}})$. Each call to the oracle uses $m \cdot \mathrm{poly}(n)$ arithmetic operations and storage space for $O(\mathrm{poly}(n) + \ln m)$ lattice vectors excluding the storage space of the preprocessing.*

Remark 2.41. Similarly to Remark 2.34, if we assume that all the DGS samples have $\mathrm{poly}(n)$ bit-size then the reduction has time complexity $m \cdot \mathrm{poly}(n)$ and space complexity $O(\mathrm{poly}(n) + \ln m)$ *excluding the storage space of the m vectors provided by the DGS*. Furthermore, if basis vectors of $\mathcal{L}$ have bit-size $\mathrm{poly}(n)$ then we can ensure that all DGS samples have $\mathrm{poly}(n)$ bit-size by first generating more samples (say twice the amount) and throwing away all samples of norm larger than $\exp(\Omega(n^2))$. Since the vectors are sampled from a Gaussian with width at most $\exp(O(n))$ (since the basis vectors have size at most $2^{o(n)}$, the error induced by throwing away the tail of the distribution is smaller than $2^{-\Omega(n^2)}$.

Proof of Theorem 2.40. First we note that we can easily identify an interval $I = [a, b]$ such that $\eta_\varepsilon(\mathcal{L}^*) \in [a, b]$ and $\frac{b}{a} \leqslant 2^{n+o(n)}$. Indeed, by e.g. [Reg05, Lemma 2.11 and Claim 2.13] one has

$$\sqrt{\ln(1/\varepsilon)}\pi \leqslant \lambda_1(\mathcal{L})\eta_\varepsilon(\mathcal{L}^*) \leqslant \sqrt{n}$$

so $\eta_\varepsilon(\mathcal{L}^*) \in \frac{1}{\lambda_1(\mathcal{L})}[c, d]$ where $\frac{d}{c} = \sqrt{\frac{n}{\ln(1/\varepsilon)}} = O(\sqrt{n})$ since $\varepsilon \leqslant 1/200$. Furthermore, by running the LLL algorithm on $\mathcal{L}$ and taking the length of the shortest basis vector, we obtain a length ℓ such that $2^{-n}\ell \leqslant \lambda_1(\mathcal{L}) \leqslant \ell$. It follows that $\eta_\varepsilon(\mathcal{L}^*) \in [a, b] := [\frac{c}{\ell}, \frac{2^n d}{\ell}]$ and $\frac{b}{a} = 2^n \frac{d}{c} = 2^{n+o(n)}$.

Now let $c = 2+\alpha$, $\delta = 1+\frac{1}{n^c}$ and $N = \left\lceil \frac{\ln(b/a)}{\ln \delta} \right\rceil$. Check that $N = \mathrm{poly}(n)$ since $\frac{b}{a} = 2^{n+o(n)}$. Furthermore, if we let $s_i = a\delta^i$ for $i = 1, \ldots, N$ then there must exists some i_0 such that

$$\tfrac{1}{\delta} s_{i_0} \leqslant \eta_\varepsilon(\mathcal{L}^*) \leqslant s_{i_0}. \tag{2.6}$$

The preprocessing stage of the algorithm consists in calling the shDGS$^m_\sigma$ sampler with parameter s_i to obtain a lists L_i of m vectors, for each $i = 1, \ldots, N$, and storing all the lists. This requires $N = \mathrm{poly}(n)$ calls and we need to store $m \cdot \mathrm{poly}(n)$ vectors.

We now describe the oracle for CVP^ϕ. On input $\boldsymbol{t} \in \mathbb{R}^n$, for each $i = 1, \ldots, N$, the oracle calls the algorithm of Theorem 2.33 on input $\boldsymbol{t}$ and provides the list L_i to the algorithm in place of the DGS samples. Hence, for each i, we either obtain a lattice vector $\boldsymbol{y}_i$ or the algorithm from Theorem 2.33 fails and we let $\boldsymbol{y}_i = \boldsymbol{0}$. Finally, the oracle returns the point

closest to $\boldsymbol{t}$ in the list $\boldsymbol{y}_1, \ldots, \boldsymbol{y}_N$. The running time is clear so it remains to prove that the algorithm actually solves CVP^{ϕ} on $\mathcal{L}$.

We first note that when called on s_{i_0}, the $\mathrm{shDGS}_{\sigma}^{m}$ sampler will return m vectors that are $0.5-$close to m samples from $D_{\mathcal{L},s}^{m}$ since $s_{i_0} \geqslant \sigma(\mathcal{L}) = \eta_{\varepsilon}(\mathcal{L}^*)$ by (2.6). Furthermore, by Lemma 2.38 we have

$$t\eta_{\varepsilon}(\mathcal{L}^*) \leqslant \eta_{\varepsilon^{t^2} e^{o(1)}}(\mathcal{L}^*)$$

where $t = 1 + \frac{1}{n^c} = \delta$. It follows by (2.6) that

$$\eta_{\varepsilon}(\mathcal{L}) \leqslant s_{i_0} \leqslant \delta\eta_{\varepsilon}(\mathcal{L}^*) \leqslant \eta_{\varepsilon^{\delta^2} e^{o(1)}}(\mathcal{L}^*).$$

But the map $\varepsilon \mapsto \eta_{\varepsilon}(\mathcal{L})$ is continuous and decreasing, so it follows that

$$s_{i_0} = \eta_{\varepsilon'}(\mathcal{L}^*) \quad \text{for some } \varepsilon^{\delta^2} e^{o(1)} \leqslant \varepsilon' \leqslant \varepsilon.$$

Therefore by Theorem 2.33 and Remark 2.34, with constant probability over the choice of L_{i_0}, the (deterministic) algorithm of Theorem 2.33 solves CVP^{ψ} when given L_{i_0}, where

$$\psi(\mathcal{L}) = \frac{\sqrt{\ln(1/\varepsilon')/\pi - o(1)}}{2\eta_{\varepsilon'}(\mathcal{L}^*)}$$

assuming that $m = |L_{i_0}| \geqslant m' := O(\frac{n \ln(1/\varepsilon')}{\sqrt{\varepsilon'}})$ which holds because

$$\frac{n \ln(1/\varepsilon')}{\sqrt{\varepsilon'}} \leqslant \frac{n\delta^2 \ln(1/\varepsilon) + o(n)}{\sqrt{\varepsilon^{\delta^2} e^{o(1)}}} \leqslant \frac{n \ln(1/\varepsilon) + n^{1-c} \ln(1/\varepsilon) + o(n)}{\sqrt{\varepsilon}\varepsilon^{n^{-c}/2} e^{o(1)}} = O\left(\frac{n \ln(1/\varepsilon)}{\sqrt{\varepsilon}}\right)$$

since $n^{1-c} \ln \varepsilon = n^{1-c+\alpha} = o(1)$ and $\varepsilon^{n^{-c}/2} = e^{n^{\alpha-c}/2} = e^{o(1)}$. It follows that, with constant probability over the preprocessing, our oracle solves CVP^{ψ}. Check that $\varepsilon^{\delta^2} e^{o(1)} \geqslant \varepsilon^{\delta^2 + o(1)}$ since $\varepsilon < \frac{1}{e}$ and thus by Lemma 2.24,

$$\eta_{\varepsilon^{\delta^2} e^{o(1)}}(\mathcal{L}^*) \leqslant \eta_{\varepsilon^{\delta^2 + o(1)}}(\mathcal{L}^*) \leqslant (\delta^2 + o(1))\eta_{\varepsilon}(\mathcal{L}^*) \leqslant (1 + o(1))\eta_{\varepsilon}(\mathcal{L}^*)$$

since $\delta = 1 + o(1)$. Hence we have

$$\psi(\mathcal{L}) \geqslant \frac{\sqrt{\ln(1/\varepsilon)/\pi - o(1)}}{2(1 + o(1))\eta_{\varepsilon}(\mathcal{L}^*)} \equiv \phi(\mathcal{L}).$$

□

Chapter 3
Discrete Gaussian Sampling and the Shortest Vector Problem

The work in this chapter has been published at STACS 2021 [ACKS21] and is a joint work with Divesh Aggarwal, Yanlin Chen and Rajendra Kumar. It also appears in part in Rajendra Kumar's PhD thesis [Kum21]. This chapter contains more introductory material over the published version.

3.1 Introduction

A major class of algorithms to solve the Shortest Vector Problem is based on *sieving*. The idea is to start with (exponentially) many lattices vectors and then combine several vectors together to create shorter and shorter vectors. After sufficiently many iterations, the resulting list will hopefully contain a shortest vector. There are several very different ways in which vectors can be combined to create shorter vectors. The simplest one is to simply look for pairs of vectors that are approximately antipodal: the sum will then be closer to the origin than the original vectors [NV08]. This process can then be refined in many ways [MV10, BGJ13, Laa15a, Laa15b, BDGL16]. These approaches generally rely on some heuristics to analyse their performance. For example, they assume that the direction of the vectors is uniform on the unit sphere, and the vectors in the sieved lists are independent.

The process described above is purely geometric: we litterally generate shorter vectors by combining lattice vectors. Instead of maintaining a geometric invariant (length), one can aim to maintain a *probabilistic invariant* (i.e. a distribution) that is loosely related to the length of vectors. Such an invariant is the discrete Gaussian distribution: its parameter (the *width*) is statistically related to the length of vectors drawn according to this distribution. One can modify the sieving step so that if the list contains vectors drawn from a discrete Gaussian of width σ then the resulting vectors should still follow a discrete Gaussian but of smaller parameter, typically $\alpha\sigma$ for $\alpha < 1$. This approach has been very successful and has led [Reg04, PS09, ADRS15, AS18b] to the currently fastest *provable* algorithm for SVP, running in time $2^{n+o(n)}$, which is based on discrete Gaussian sampling.

Even though sieving algorithms are asymptotically the fastest known algorithms for SVP, the memory requirement, in high dimension, has historically been a limiting factor to run these algorithms. Some recent works [Duc18, ADH+19] have shown how to use new tricks to make it possible to use sieving on high-dimensional lattices in practice and benefit from their efficient running time [SVP].

Nevertheless, it would be ideal and has been a long standing open question to obtain an algorithm that achieves the "best of both worlds", i.e. an algorithm that runs in time $2^{\mathcal{O}(n)}$ as in sieving and requires memory polynomial in n as in enumeration. In the absence of such an

algorithm, it is desirable to have a smooth trade-off between time and memory requirement that interpolates between the current best sieving algorithms and the current best enumeration algorithms.

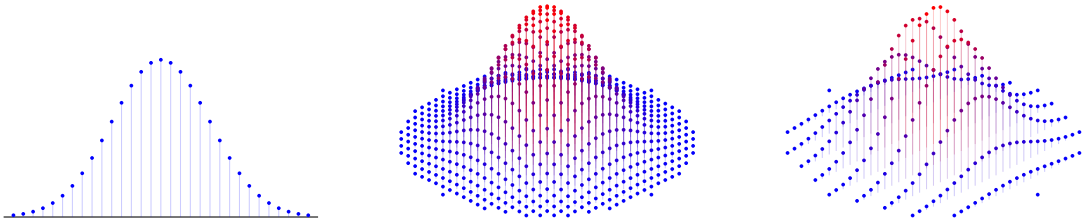
To this end, Bai, Laarhoven, and Stehlé [BLS16] proposed the tuple sieving algorithm, providing such a trade-off based on heuristic assumptions mentionned above. This algorithm was later proven to have time and space complexity $k^{n+o(n)}$ and $k^{n/k+o(n)}$, under the same heuristic assumptions [HK17]. One can vary the parameter k to obtain a smooth time/space trade-off. However, as noted in [HK17, p. 22], their analysis only applies to "fixed k". In particular, they cannot obtain the regime of subexponential space and superexponential time. Furthermore, it is desirable to obtain a provable variant of this algorithm that does not rely on any heuristics.

Kirchner and Fouque [KF16] attempted to do this. They claim an algorithm for solving SVP in time $q^{\Theta(n)}$ and in space $q^{\Theta(n/q)}$ for any positive integer $q > 1$. Unfortunately, their analysis falls short of supporting their claimed result, and the correctness of the algorithm is not clear. We refer the reader to Section 3.5 for more details.

In the rest of this chapter, we describe a provable time/space trade-off for discrete Gaussian sampling. Using this trade-off, we obtain a provable time/space trade-off for SVP. A detailed description of our contributions is available in Section 1.1.1.

3.2 Gaussian Sampling and the SVP

In order to understand discrete Gaussian sampling, and how it applies to the SVP, it useful to visualise what such a distribution looks like (see Figure 3.1). Its shape is strongly affected by the parameter s (the width). Similarly to the continuous Gaussian distribution, a smaller parameter will produce a "tigher" distribution around the origin. Unlike the continuous Gaussian, and under a certain threshold (called the *smoothing parameter* η_ε, see Section 2.4), the distribution will start to loose certain properties and become "less smooth". The parameter s represents a trade-off in different ways: in its shape, in the applications of the distribution and in how easy it is to sample (see Figure 3.2). The problem of sampling the discrete Gaussian distribution (known as DGS) therefore has many applications beyond the shortest vector problem.



$\mathcal{L} = \mathbb{Z}, s = 7$ $\mathcal{L} = \mathbb{Z}^2, s = 7$ $\mathcal{L} = \mathbb{Z} \times 4\mathbb{Z}, s = 7$

Figure 3.1: Examples of discrete Gaussian distribution in one and two dimensions with different parameters.

We now informally review how a DGS-based algorithm for SVP works. The first step of any such algorithm is to generate vectors according to a discrete Gaussian of very large width.

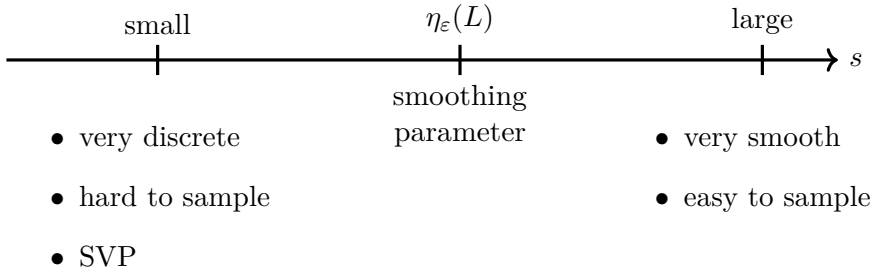


Figure 3.2: Trade-off of the discrete Gaussian depending on its parameter s.

When the width is sufficiently large (typically exponentially bigger than what is necessary to solve SVP), the discrete Gaussian is so smooth that one can essentially generate according to *continuous* Gaussian distribution and round to produce to a lattice vector. The width of the Gaussian obtained this way depends on the quality of the basis. One can obtain a suitably reduced basis in polynomial time by the LLL algorithm [Kle00, GPV08, BLP+13, ADRS15]. See Theorem 2.29 for more details. When solving the SVP, one will typically generate an exponential number of vectors according to this distribution. The second step is to repeatedly "sieve" this list to reduce the parameter s, usually down to the smoothing parameter. Indeed, for reasons explained below, it is much easier to sieve when s is above the smoothing parameter. The third step depends on the algorithm: one can either continue to sieve below the smoothing parameter, which requires great care [ADRS15], or use this sampler to construct an oracle for the BDD problem [DRS14, ADRS15]: Given an algorithm for DGS a constant factor c above the smoothing parameter, we can solve the problem of BDD where the target vector is within distance $\alpha\lambda_1(\mathcal{L})$ of the lattice, where the constant $\alpha < 0.5$ depends on the constant c. Additionally, using [CCL18], one can enumerate all lattice points within distance $q\delta$ to a target $\boldsymbol{t}$ by querying q^n times a BDD oracle with decoding distance δ (or $q^{n/2}$ times if we are given a quantum BDD oracle). Thus, by choosing $q = \lceil \lambda_1(\mathcal{L})/\delta \rceil$ and $\boldsymbol{t} = \mathbf{0}$, an algorithm for BDD immediately gives us an algorithm for SVP.

We now review the second step above in more details: sieving vectors to reduce the parameter s above the smoothing parameter. Without any loss of generality, we can always assume that $\mathcal{L}$ is a full-rank lattice. The key argument of any such algorithm is that combining discrete Gaussian distributions still produces a discrete Gaussian, when above the smoothing parameter. Here, the concept of *coset* usually plays an important role. Recall that if $\mathcal{L} \subset \mathbb{R}^n$ is a lattice, we can consider the quotient $\mathcal{L}/2\mathcal{L} = \{\mathbf{x} + 2\mathcal{L} : \mathbf{x} \in \mathcal{L}\}$. This quotient is finite of size exactly 2^n and each coset $\mathbf{x} + 2\mathcal{L}$ essentially encodes the parities of the coordinates of the vectors modulo $2\mathcal{L}$. In other words, if $\mathbf{b}_1, \ldots, \mathbf{b}_n$ is a basis of $\mathcal{L}$, then a lattice vector $\sum_{i=1}^n x_i \mathbf{b}_i$ belongs to a coset that only depends on the parities $x_1 \pmod 2, \ldots, x_n \pmod 2$. One idea to reduce the length of vectors is to take the average of two vectors. However the average is usually not a lattice vector but more can be said if we consider cosets. Indeed let $\mathbf{c} + \mathcal{L}$ be a coset of $\mathcal{L}/2\mathcal{L}$,

$$\text{if } \mathbf{x}, \mathbf{y} \in \mathbf{c} + 2\mathcal{L} \text{ then } \frac{\mathbf{x} + \mathbf{y}}{2} \in \mathcal{L}.$$

Furthermore, as the lemma below shows, this operation preserves the Gaussian distribution while reducing the parameter s to $s/\sqrt{2}$.

Lemma 3.1 ([ADRS15])**.** *Let $\mathcal{L}$ be a lattice and X_1, X_2 be two independent variables sampled from $D_{\mathcal{L},s}$ with $s > 0$, then for any $\mathbf{y} \in \mathcal{L}$,*

$$\Pr_{(X_1,X_2)\sim D^2_{\mathcal{L},s}}[(X_1 + X_2)/2 = \mathbf{y} | X_1 + X_2 \in 2\mathcal{L}] = \Pr_{Z\sim D_{\mathcal{L},s/\sqrt{2}}}[Z = \mathbf{y}].$$

This result suggests the following sieving algorithm, proposed by Aggarwal et al [ADRS15]. While superficially trivial, we point out that the complete analysis of this algorithm is quite sophisticated.

Algorithm 3.1 PairAndAverage(L)

Input: L is a list of vectors in $\mathcal{L}$

$L' \leftarrow$ empty list
for each unpaired vector $\mathbf{x}$ in L **do**
  if there exists another unpaired vector $\mathbf{y}$ with $\mathbf{x} + \mathbf{y} \in 2\mathcal{L}$ **then**
    add $(\mathbf{x} + \mathbf{y})/2$ to L'
  end if
end for
return L'

Analysis of the algorithm. Unfortunately, the algorithm above produces the wrong distribution despite the lemma. The problem comes from the fact that in the lemma above, X_1 and X_2 are two independent variables, hence the probability of X_1 and X_2 to be in the same coset $\mathbf{c} + 2\mathcal{L}$ (a necessary and sufficient condition for $X_1 + X_2$ to be in $2\mathcal{L}$) is proportional to $\rho_s(\mathbf{c} + 2\mathcal{L})^2$, where ρ_s is the Gaussian mass function (see Section 2.4). On the other hand, the algorithm above has a different behaviour: the number of pairs of vectors in the coset $\mathbf{c} + 2\mathcal{L}$ will be proportional to the number of times this coset appears in the list, that is $\rho_s(\mathbf{c} + 2\mathcal{L})$ instead of $\rho_s(\mathbf{c} + 2\mathcal{L})^2$. This mismatch between the "squared distribution" that we want and the "unsquared" distribution that we get is the main technical challenge in [ADRS15]. It is overcome by the use of a "square sampler" that only works above the smoothing parameter. In a later article, the authors introduced the notion of "mixture of Gaussians" and showed that this algorithm, while not producing a discrete Gaussian, still produces a distribution that can be used to solve the SVP in time 2^n [AS18b].

An interesting observation for us is that, *at the cost of introducing a small error on the distribution*, the algorithm above actually works whenever the parameter s is above the smoothing parameter. This is where the smoothness of the distribution comes in: a well-known consequence (Lemma 2.22) of s being above the smoothing parameter is that all cosets $\mathbf{c} + 2\mathcal{L}$ have the same weight (up to a small factor ε). It follows that the "squared distribution" problem disappears because all cosets have the same weight (up to ε) and so does the "square distribution".

Reducing the memory usage. Another drawback of Algorithm 3.1 (and its variants) is that the memory usage is clearly exponential in the dimension: we need to remember one vector per coset to efficiently pair them and there are 2^n cosets. Furthermore, to even have a chance at pairing two vectors, we need a list of size at least 2^n, otherwise all vectors could be

a different cosets. This problem can be mitigated somewhat by the use of towers of lattice [ADRS15] but it only brings the memory usage down to $2^{n/2}$ with no obvious way to improve it further. There is of course, an obvious but extremely inefficient way to reducing the memory usage: simply select two elements of the list at random and pair them if they are in the same coset, otherwise try again (see Algorithm 3.2). The complexity of this algorithm is terrible:

Algorithm 3.2 ExtremelyNaivePairAndAverage(L)

Input: L is a list of vectors in $\mathcal{L}$
1: $L' \leftarrow$ empty list
2: **while** L contains at least $2^n + 1$ elements **do**
3: $\quad \mathbf{x} \leftarrow$ a random element from L
4: $\quad \mathbf{y} \leftarrow$ a random element from L
5: $\quad$ **if** $\mathbf{x} + \mathbf{y} \in 2\mathcal{L}$ **then**
6: $\quad\quad$ add $(\mathbf{x} + \mathbf{y})/2$ to L'
7: $\quad\quad$ remove $\mathbf{x}$ and $\mathbf{y}$ from L
8: $\quad$ **end if**
9: **end while**
10: **return** L'

since all cosets have the same weight above the smoothing parameter, we expect that it will take 2^n trials to get two vectors $\mathbf{x}$ and $\mathbf{y}$ which sum is in $\mathcal{L}$. Hence the algorithm will take an expected time $2^n|L|$ compared to $|L|$ in the previous algorithm. On the other hand, it does not use any memory besides the one used to store the list, but the list still needs to be of size at least 2^n which is exponential. Therefore a natural question is whether we can have a smooth trade-off between Algorithm 3.1 and Algorithm 3.2.

Improving memory usage. One way to improve the space complexity, and that we pursue in this work, is to sum more than two vectors. Assume now that $|L| > 2^{n/2}$ (compared to 2^n previously) and pick a random element $\mathbf{x}$ in L. Now consider the $|L|^2$ possible sums of two vectors (excluding $\mathbf{x}$): they will fall into the 2^n cosets. But since $|L|^2 > 2^n$, and since the sum of two discrete Gaussians of parameter s above the smoothing parameter is very close to a discrete Gaussian of parameter $\sqrt{2}s$ (Theorem 2.28), we can expect one sum per coset. Therefore one of those sums $\mathbf{y} + \mathbf{z}$ falls in the same coset as that of $\mathbf{x}$ and the sum $\mathbf{x} + \mathbf{y} + \mathbf{z} \in 2\mathcal{L}$ can now be divided by two to obtain a vector of $\mathcal{L}$. Theorem 2.28 again guarantees that this sum is statistically close to being a Gaussian of parameter $\sqrt{3}s$ on $2\mathcal{L}$, hence $(\mathbf{x} + \mathbf{y} + \mathbf{z})/2 \in \mathcal{L}$ will be close to a discrete Gaussian of parameter $\sqrt{3}s/2$ on $\mathcal{L}$. Note that we have managed to reduce the space requirement on the list from 2^n down to $2^{n/2}$. Also note that while it works for three vectors, it does not work with four. Indeed, the sum of four Gaussians of parameter s will be close to a Gaussian of parameter $2s$, therefore dividing by two will again yield a Guassian of parameter s: we are not reducing the size of the vectors anymore.

A smooth time/memory trade-off. Going further requires one more idea: we can go beyond $2\mathcal{L}$ and look for vectors in $q\mathcal{L}$ for some parameter $q \geqslant 2$. If we sum $d + 1$ vectors then the sum will be close to a Gaussian of parameter $\sqrt{d+1}s/q$ after dividing by q, again by

Theorem 2.28. If we proceed as before, picking a first element at random and then looking for d vectors which sum is in the same coset, then there are roughly $\binom{|L|}{d} \approx |L|^d$ possible sums. Therefore, we only need $|L| \gtrsim q^{n/d}$ samples to find a collision on average (since there are q^n cosets in $\mathcal{L}/q\mathcal{L}$) and we can pick any $d \in [1, q^2 - 2]$ to ensure that we reduce the parameter of the Gaussian. If we always choose a list of size $|L| \approx q^{n/d}$ then the time complexity will be roughly $|L|^d \approx q^n$ while the space complexity is dominated by the size of the list, hence $q^{n/d}$. Setting $q = 3$ and $d = 1$, we obtain a 3^n time and space algorithm. Setting $d = q^2 - 2$ for any q, we obtain a q^n time algorithm in space q^{n/q^2}. In particular, for $q = \sqrt{n}$, we obtain a n^n algorithm in space $\mathrm{poly}(n)$.

In the following, we will explore this approach in details and see that there are significant technical challenges to overcome to make it work. In particular, the fact that we remove samples from the list after each match perturbs the uniformity of the list and of the cosets. Even though the error is small (exponentially so), this becomes a problem when we repeat this operation an exponential number of times. As a result, the trade-off that we obtain is not quite as good as the simplistic analysis above suggests.

3.3 Algorithm for Discrete Gaussian Sampling

We now present the main result of this chapter. For convenience, when we discuss the running time of the algorithms in this chapter, we ignore polynomial factors in the bit-length of the individual input basis vectors (i.e. we consider only the dependence on the ambient dimension n).

Theorem 3.2. *Let $n \in \mathbb{N}, q \geqslant 2, d \in [1, n]$ be positive integers, and let $\varepsilon > 0$. Let C be any positive integer. Let $\mathcal{L}$ be a lattice of rank n, and let $s \geqslant 2\sqrt{d}q\eta_\varepsilon(\mathcal{L})$. There is an algorithm that, given $N = 160d^2 \cdot C \cdot q^{n/d}$ independent samples from $D_{\mathcal{L},s}$, outputs a list of vectors that is $(4\varepsilon^{2d}N + 11Cq^{-5n/2})$-close to $Cq^{n/d}$ independent vectors from $D_{\mathcal{L},\frac{\sqrt{8d+1}}{q}s}$. The algorithm runs in time $C \cdot (10e \cdot d)^{8d} \cdot q^{8n+n/d+o(n)}$ and requires memory $\mathrm{poly}(d) \cdot q^{n/d}$.*

Proof. We prove the result for $C = 1$, and the general result follows by repeating the algorithm. Let $\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N\}$ be the N input vectors and let $\{\boldsymbol{c}_1, \ldots, \boldsymbol{c}_N\}$ be the corresponding cosets in $\mathcal{L}/q\mathcal{L}$. We will analyze Algorithm 3.3; note that it produces samples in a streaming fashion. The time complexity of the algorithm is

$$\frac{N}{2} \cdot \binom{N/2}{8d} \leqslant \frac{N}{2}\left(\frac{eN}{16d}\right)^{8d} \leqslant (10e \cdot d)^{8d} \cdot q^{8n+n/d+o(n)} ,$$

and memory requirement of the algorithm is immediate. We now show correctness: we will make repeated use of the data processing inequality (see Section 2.3) and accumulate the error terms until the end of the proof.

Let $\varepsilon' = \varepsilon^{2d}$ so that $s \geqslant \sqrt{2}\eta_{\varepsilon'}(q\mathcal{L})$ by Lemma 2.24. First, we can assume that the vectors $\boldsymbol{x}_i$ for $i \in [N]$ are obtained by first sampling $\boldsymbol{c}_i \in \mathcal{L}/q\mathcal{L}$ such that $\Pr[\boldsymbol{c}_i = \boldsymbol{c}] = \Pr[D_{\mathcal{L},s} \in q\mathcal{L} + \boldsymbol{c}]$ and then sampling the vector $\boldsymbol{x}_i$ according to $D_{q\mathcal{L}+\boldsymbol{c}_i,s}$. Indeed, let $X \sim D_{\mathcal{L},s}$ and C be a

Algorithm 3.3 TradeOffSieve(L)

Input: $L = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N\}$ is a list of N vectors in $\mathcal{L}$

1: $L_1 \leftarrow \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{\frac{N}{2}}\}$
2: $L_2 \leftarrow \{\boldsymbol{x}_{\frac{N}{2}+1}, \ldots, \boldsymbol{x}_N\}$
3: $Q \leftarrow 0$
4: **while** $Q < q^{n/d}$ and L_1 is not empty **do**
5: $\quad \boldsymbol{v} \leftarrow$ be the first vector in L_1.
6: $\quad$ Find $8d$ vectors (by trying all $8d$-tuples) $\boldsymbol{x}_{i_1}, \ldots, \boldsymbol{x}_{i_{8d}}$ from L_2 s.t. $\boldsymbol{c}_{i_1} + \cdots + \boldsymbol{c}_{i_{8d}} - \boldsymbol{v} \in q\mathcal{L}$.
7: $\quad$ **if** such vectors exist **then**
8: $\quad\quad$ Output the vector $\frac{\boldsymbol{x}_{i_1}+\cdots+\boldsymbol{x}_{i_{8d}}-\boldsymbol{v}}{q} \in \mathcal{L}$
9: $\quad\quad$ $Q \leftarrow Q + 1$.
10: $\quad\quad$ Remove vectors $\boldsymbol{x}_{i_1}, \cdots, \boldsymbol{x}_{i_{8d}}$ from L_2
11: $\quad$ **end if**
12: $\quad$ Remove vector $\mathbf{v}$ from L_1
13: **end while**

random coset in $\mathcal{L}/q\mathcal{L}$ sampled such that $\Pr[C = q\mathcal{L} + \boldsymbol{c}] = \frac{\rho_s(q\mathcal{L}+\boldsymbol{c})}{\rho_s(\mathcal{L})}$, then for any $\boldsymbol{x} \in \mathcal{L}$,

$$\Pr[X = \boldsymbol{x}] = \frac{\rho_s(\boldsymbol{x})}{\rho_s(\mathcal{L})} = \frac{\rho_s(q\mathcal{L}+\boldsymbol{c})}{\rho_s(\mathcal{L})}\frac{\rho_s(\boldsymbol{x})}{\rho_s(q\mathcal{L}+\boldsymbol{c})} = \Pr[C = \boldsymbol{c}] \Pr_{Y \sim D_{q\mathcal{L}+\boldsymbol{c},s}}[Y = \boldsymbol{x}]$$

where $\boldsymbol{c} = q\mathcal{L} + \boldsymbol{x}$. Moreover, by Corollary 2.23, this distribution is $2\varepsilon' N$-close to sampling $\boldsymbol{c}_i$ for $i \in [N]$, independently and uniformly from $\mathcal{L}/q\mathcal{L}$, and then sampling the vectors $\boldsymbol{x}_i$ according to $D_{q\mathcal{L}+\boldsymbol{c}_i,s}$. We now assume that the input is sampled from this distribution.

Second, we can assume that the algorithm initially gets only the corresponding cosets as input, and the vectors $\boldsymbol{x}_{i_j} \in q\mathcal{L} + \boldsymbol{c}_{i_j}$ for $j \in [8d]$, and $\boldsymbol{v} \in q\mathcal{L} + \boldsymbol{c}$ are sampled from $D_{q\mathcal{L}+\boldsymbol{c}_{i_j},s}$ and $D_{q\mathcal{L}+\boldsymbol{c},s}$ only before such a tuple is needed in Step 4 of the algorithm. Indeed, notice that the test at line 6 does not actually depend on the particular value of the $\boldsymbol{x}_{i_j}$ and $\boldsymbol{v}$ but only on their cosets. Since any input vector is used only once in Step 4, these samples are independent of all prior steps. This implies, by Theorem 2.28, that the vector obtained in Step 4 of the algorithm is $2\varepsilon'$-close to being distributed as $D_{\mathcal{L},s\frac{\sqrt{8d+1}}{q}}$.

It remains to show that our algorithm finds $q^{n/d}$ vectors (with high probability). Let $N' = \frac{N}{2}$ be an integer, X be a random variable uniform over $(\mathcal{L}/q\mathcal{L})^{N'}$, and let Y be a random variable independent of X and uniform over vectors in $\{0,1\}^{N'}$ with Hamming weight $8d$. The number of such vectors is

$$\binom{N'}{8d} \geqslant \left(\frac{N'}{8d}\right)^{8d} \geqslant q^{8n} . \tag{3.1}$$

Let U be a uniformly random coset of $\mathcal{L}/q\mathcal{L}$. By Lemma 2.13 and (3.1), we have

$$\mathrm{d_{SD}}((\langle X, Y\rangle, X), (U, X)) \leqslant \frac{1}{2} \cdot \sqrt{\frac{q^n}{q^{8n}}} = \frac{1}{2} q^{-7n/2} .$$

By Markov inequality we have

$$\Pr_{x \leftarrow X}\left[\mathrm{d_{SD}}(\langle x, Y\rangle, U) \geqslant \frac{q^{-n}}{10}\right] \leqslant \frac{10}{q^{-n}} \mathbb{E}_{x \leftarrow X}[\mathrm{d_{SD}}(\langle x, Y\rangle, U)]$$

$$= \frac{20}{q^{-n}} \, \mathrm{d_{SD}}((\langle X, Y\rangle, X), (U, X))$$
$$\leqslant \frac{20}{q^{-n}} \cdot \frac{1}{2} q^{-7n/2} = 10 q^{-5n/2}.$$

Hence, with probability at least $1 - 10q^{-5n/2}$ over the choice of $x \leftarrow X$, we have that $\mathrm{d_{SD}}(\langle x, Y\rangle, U) \leqslant \frac{q^{-n}}{10}$ and thus for any $\boldsymbol{v} \in \mathcal{L}/q\mathcal{L}$,

$$q^{-n} + \frac{q^{-n}}{10} > \Pr[\langle x, Y\rangle = \boldsymbol{v} \bmod q\mathcal{L}] > q^{-n} - \frac{q^{-n}}{10}. \tag{3.2}$$

It follows that, by introducing a statistical distance of at most $10q^{-5n/2}$ on the input, we can assume that the input vectors in list L_2 satisfy (3.2). Notice that after the algorithm found i vectors for any $i < q^{n/d}$, it has removed $8id$ vectors from L_2. We will show that for each vector from L_1 (which is uniformly sampled from $\mathcal{L}/q\mathcal{L}$) with constant probability we will find $8d$-vectors in Step 3.

After $i < q^{n/d}$ output vectors have been found, there are $M = N' - 8id$ vectors remaining in the list L_2. There are $\binom{M}{8d}$ different $8d$-combinations possible with vectors remaining in L_2.

$$\begin{aligned}
\binom{N'}{8d} \Big/ \binom{M}{8d} &= \frac{N' \cdots (N' - 8d + 1)}{M \cdots (M - 8d + 1)} \\
&< \left(\frac{N' - 8d}{N' - 8d(i+1)} \right)^{8d} \\
&< \left(1 + \frac{8dq^{n/d}}{N' - 8dq^{n/d} - 8d} \right)^{8d} \\
&= \left(1 + \frac{1}{10d - 1 - \frac{1}{q^{n/d}}} \right)^{8d} \qquad \text{since } N' = 80d^2 q^{n/d} \text{ for } C = 1 \\
&\leqslant \left(1 + \frac{1}{10d - 3/2} \right)^{8d} < \frac{5}{2}.
\end{aligned} \tag{3.3}$$

At the beginning of the algorithm, there are $\binom{N'}{8d}$ combinations, and hence by (3.2), each of the q^n cosets appears at least $0.9q^{-n}\binom{N'}{8d}$ times. After $i < q^{n/d}$ output vectors have been found, there are only $\binom{M}{8d}$ combinations left, and $\binom{N'}{8d} - \binom{M}{8d}$ possible combinations have been removed. We say that a coset $\boldsymbol{c}$ disappears if there is no set of $8d$ vectors in L_2 that add to $\boldsymbol{c}$. In order for a coset to disappear, all of the at least $0.9q^{-n}\binom{N'}{8d}$ combinations from the initial list must be removed. Hence, the number of cosets that disappear is at most $\frac{\binom{N'}{8d} - \binom{M}{8d}}{0.9q^{-n}\binom{N'}{8d}} < \frac{3/5}{0.9} q^n = \frac{2}{3} q^n$ distinct cosets by (3.3). Hence with probability at least $1/3$, we find $8d$ vectors $\boldsymbol{x}_{i_1}, \ldots, \boldsymbol{x}_{i_{8d}}$ from L_2 such that $\boldsymbol{x}_{i_1} + \cdots + \boldsymbol{x}_{i_{8d}} - \boldsymbol{v} \in q\mathcal{L}$. By Chernoff-Hoeffding bound with probability greater than $1 - e^{-d^2 q^{n/d}}$, the algorithm finds at least $q^{n/d}$ vectors. In total, the statistical distance from the desired distribution is

$$2\varepsilon' N + 2\varepsilon' q^{n/d} + 10 \cdot q^{-5n/2} + e^{-d^2 q^{n/d}} \leqslant 4\varepsilon' N + 11 \cdot q^{-5n/2}.$$

□

Corollary 3.3. *Let $n \in \mathbb{N}$, $q \in [4, \sqrt{n}]$ be an integer, and let $\varepsilon = q^{-32n/q^2}$. Let $\mathcal{L}$ be a lattice of rank n, and let $s \geqslant \eta_\varepsilon(\mathcal{L})$. There is an algorithm that outputs a list of vectors that is $q^{-\Omega(n)}$-close to q^{16n/q^2} independent vectors from $D_{\mathcal{L},s}$. The algorithm runs in time $q^{13n+o(n)}$ and requires memory $\mathrm{poly}(n) \cdot q^{16n/q^2}$.*

Proof. Choose d so that $16d - 16 < q^2 \leqslant 16d$, which is possible when $q \geqslant 4$, and let $\alpha = q/\sqrt{8d+1}$ — this is the ratio by which we decrease the Gaussian width in Theorem 3.2 — and note that $\alpha \geqslant 1.2$.

Let $p = \lceil 2\sqrt{d}q \rceil < q^2$ and k be the smallest integer such that $\alpha^k \cdot p \geqslant 2^{n \log\log n / \log n}$. Thus $k = O(n \log\log n / \log n)$. Let $g = \alpha^k p s \geqslant 2^{n \log\log n/\log n} \cdot \eta_\varepsilon(\mathcal{L})$. By Theorem 2.29, in time $N_0 \cdot \mathrm{poly}(n)$, we get $N_0 = (160d^2)^k q^{n/d}$ samples from $D_{\mathcal{L},g}$.

We now iterate k times the algorithm from Theorem 3.2. Initially we have N_0 vectors. At the beginning of the i-th iteration for $i \leqslant k-1$, we have $N_i := N_0 \cdot (160d^2)^{-i}$ vectors that are Δ_i-close to being independently distributed from $D_{\mathcal{L},\alpha^{-i}g}$, where $\alpha^{-i} g \geqslant \alpha p \cdot \eta_\varepsilon(\mathcal{L})$. Hence, we can apply Theorem 3.2 and get $N_{i+1} = N_i/160d^2$ vectors that are Δ_{i+1}-close to being independently distributed from $D_{\mathcal{L},\alpha^{-(i+1)}g}$, where $\Delta_{i+1} \leqslant \Delta_i + 4\varepsilon^{2d} N_i + 11(160d^2)^{k-i} q^{-5n/2}$. At each iteration we had $N_i \geqslant 160d^2 q^{n/d}$ vectors, a necessary condition to apply Theorem 3.2. Therefore after k iterations, we have at least $N_k = N_0/(160d^2)^k = q^{n/d}$ samples that are Δ_k-close to being independently distributed from $D_{\mathcal{L},\alpha^{-k}g}$, where

$$\begin{aligned}
\Delta_k &\leqslant 11 q^{-5n/2} \sum_{i=1}^{k} (160d^2)^{k-i} + \sum_{i=0}^{k-1} 4\varepsilon^{2d} N_i \\
&\leqslant 11(160d^2)^k q^{-5n/2} + 4q^{-4n} q^{n/d} \sum_{i=0}^{k-1} (160d^2)^{k-i} && \text{since } 16d \geqslant q^2 \\
&\leqslant \left(11 q^{-5n/2} + 4q^{-4n+n/d}\right)(160d^2)^{k+1} = q^{-5n/2+o(n)} && \text{since } (160d^2)^{k+1} = q^{o(n)}.
\end{aligned}$$

Any vector distributed as $D_{\mathcal{L},ps}$ is in $p\mathcal{L}$ with probability at least p^{-n}. We repeat the algorithm $2p^n = O(q^{2n})$ times to obtain $p^n \cdot 2 \cdot q^{n/d}$ vectors that are $2p^n q^{-5n/2+o(n)} = q^{-n/2+o(n)}$ close to $2p^n \cdot q^{n/d}$ independent samples from $D_{\mathcal{L},ps}$. Of these samples obtained, we only keep vectors that fall in $p\mathcal{L}$ and divide them by p. Let $M = p^n \cdot 2 \cdot q^{n/d}$. By Chernoff-Hoeffding (Lemma 2.14) with $P = p^{-n}$, and $\delta = \frac{1}{2}$, the probability to obtain less than $(1-\delta)PM = q^{n/d}$ samples is at most $\left(\frac{e^{-\delta}}{(1-\delta)^{1-\delta}}\right)^{PM} \leqslant e^{-\frac{1}{10}q^{n/d}}$. Furthermore, $d \leqslant \frac{q^2+16}{16}$ and $q \mapsto \frac{\ln q}{16+q^2}$ is decreasing for $q \geqslant 4$, hence for $q \leqslant \sqrt{n}$,

$$q^{n/d} \geqslant e^{16n \frac{\ln q}{16+q^2}} \geqslant e^{16n \frac{\ln \sqrt{n}}{16+n}} \geqslant e^{16 \ln \sqrt{n} - o(1)} = \Omega(n^8).$$

Hence with probability greater than $1 - e^{-\frac{1}{10}q^{n/d}} = 1 - q^{-\Omega(n^8)}$, we get $q^{n/d}$ vectors from the distribution $D_{\mathcal{L},s}$. The statistical distance from the desired distribution is $q^{-\Omega(n^8)} + q^{-n/2+o(n)} \leqslant q^{-n/2+o(n)}$. We repeat this for $\frac{q^{16n/q^2}}{q^{n/d}}$ times, to get q^{16n/q^2} vectors. The total statistical distance from the desired distribution is $\frac{q^{16n/q^2}}{q^{n/d}} \cdot q^{-n/2+o(n)} \leqslant q^{-\Omega(n)}$. The total running time is bounded by

$$q^{2n}\left(\frac{q^{16n/q^2}}{q^{n/d}}\right)\left(\mathrm{poly}(n) \cdot N_0 + \sum_{i=0}^{k-1} (10ed)^{8d} \cdot (160d^2)^{k-i} q^{8n+n/d+o(n)}\right) \leqslant q^{13n+o(n)}.$$

The memory usage is slightly more involved: we can think of the k iterations as a pipeline with k intermediate lists and we observe that as soon as a list (at any level) has more than $160d^2q^{16n/q^2}$ elements, we can apply Theorem 3.2 to produce q^{16n/q^2} vectors at the next level. Hence, we can ensure that at any time, each level contains at most $160d^2q^{16n/q^2}$ vectors, so in total we only need to store at most $k \cdot 160d^2q^{16n/q^2} = \mathrm{poly}(n)q^{16n/q^2}$ vectors, to which we add the memory usage of the algorithm of Theorem 3.2 which is bounded by $\mathrm{poly}(n) \cdot q^{n/d} \leqslant \mathrm{poly}(n) \cdot q^{16n/q^2}$. Finally, we run the filter $(p\mathcal{L})$ on the fly at the end of the k iterations to avoid storing useless samples. □

This tradeoff works for any $q \geqslant 4$, and the running time can be bounded by $q^{cn} + o(n)$ for some constants c that we have not tried to optimize.

3.4 Algorithms for BDD and SVP

In the previous section, we obtained a trade-off for Discrete Gaussian Sampling. By using existing reductions from BDD and SVP to DGS, we will show how to obtain a trade-off for those problems.

Theorem 3.4. *Let $n \in \mathbb{N}, q \in [4, \sqrt{n}]$ be a positive integer. Let $\mathcal{L}$ be a lattice of rank n, there exists an algorithm that creates a $0.1/q$-BDD oracle in time $q^{13n+o(n)}$ and space $\mathrm{poly}(n) \cdot q^{16n/q^2}$. Every call to this oracle takes time $\mathrm{poly}(n)q^{16n/q^2}$.*

Proof. Let $\varepsilon = q^{\frac{-32n}{q^2}}$ and $s \geqslant \eta_\varepsilon(\mathcal{L}^*)$. From Corollary 3.3, there exists an algorithm that outputs q^{16n/q^2} vectors whose distribution is statistically close to $D_{\mathcal{L}^*,s}$ in time $q^{13n+o(n)}$ and space $\mathrm{poly}(n) \cdot q^{16n/q^2}$. By repeating this algorithm $\mathrm{poly}(n)$ times, we can therefore build a $\mathsf{DGS}_{\eta_\varepsilon}^{\mathrm{poly}(n)q^{16n}/q^2}$ oracle such that each call takes time $q^{13n+o(n)}$ and space $\mathrm{poly}(n) \cdot q^{16n/q^2}$.

By Theorem 2.40 and Remark 2.41 we can construct a $\alpha-$BDD such that each call takes time $m \cdot \mathrm{poly}(n)$ and space $\mathrm{poly}(n)$, where $\alpha = \phi(\mathcal{L})/\lambda_1(\mathcal{L}) = \frac{\sqrt{\ln(1/\varepsilon)/\pi - o(1)}}{2\eta_\varepsilon(\mathcal{L}^*)\lambda_1(\mathcal{L})}$ and $m = O(\frac{n\log(1/\varepsilon)}{\sqrt{\varepsilon}}) = O(\frac{n^2}{q^2}q^{16n/q^2}\log q) \leqslant \mathrm{poly}(n)q^{16n/q^2}$. The preprocessing consists of $\mathrm{poly}(n)$ calls to the $\mathsf{DGS}_{\eta_\varepsilon}^m$ sampler described above and requires space $m \cdot \mathrm{poly}(n)$. Hence the total complexity is $\mathrm{poly}(n)q^{13n+o(n)} = q^{13n+o(n)}$ in time and $m \cdot \mathrm{poly}(n) = \mathrm{poly}(n)q^{16n/q^2}$ in space. By using inequality (2.3), in Lemma 2.25, we have that

$$\lambda_1(\mathcal{L})\eta_\varepsilon(\mathcal{L}^*) < \sqrt{\frac{\beta(\mathcal{L})^2 n}{2\pi e}} \cdot \varepsilon^{-1/n}(1 + o(1)).$$

Hence we can guarantee that

$$\alpha(\mathcal{L}) \geqslant \sqrt{\frac{\ln(1/\varepsilon)}{2n(\beta^2/e)\varepsilon^{-2/n}}} \cdot (1 - o(1)) \geqslant \frac{1}{q}\sqrt{\frac{32 \cdot e \cdot \ln q}{2\beta^2 q^{64/q^2}}} \cdot (1 - o(1)) \geqslant (10q)^{-1}.$$

□

Theorem 3.5. *Let $n \in \mathbb{N}, q \in [4, \sqrt{n}]$ be a positive integer. Let $\mathcal{L}$ be a lattice of rank n. There is a randomized algorithm that solves SVP in time $q^{13n+o(n)}$ and in space $\mathrm{poly}(n) \cdot q^{\frac{16n}{q^2}}$.*

Proof. By Theorem 3.4, we can construct a $\frac{0.1}{q}$-BDD oracle in time $q^{13n+o(n)}$ and in space $\mathrm{poly}(n) \cdot q^{\frac{16n}{q^2}}$. Each execution of the BBD oracle now takes $\mathrm{poly}(n)q^{16n/q^2}$ time. By Theorem 2.35, with $(10q)^n$ queries to $\frac{0.1}{q}$-BDD oracle, we can find the shortest vector. The total time complexity is $q^{13n+o(n)} + \mathrm{poly}(n)q^{16n/q^2} \cdot (10q)^n = q^{13n+o(n)}$. □

Remark 3.6. If we take $q = \sqrt{n}$, Theorem 3.5 gives a SVP algorithm that takes $n^{\mathcal{O}(n)}$ time and $\mathrm{poly}(n)$ space. When q is a large enough constant, for any constant $\varepsilon > 0$, there exists a constant $C = C(\varepsilon) > 2$, such that there is a 2^{Cn} time and $2^{\varepsilon n}$ space algorithm for DGS, and SVP. In particular, the time complexity of the algorithm in this regime is worse than the best known sieving algorithms.

3.5 Comparison with previous time/space trade-offs

We now compare our trade-off for SVP with other trade-offs in the litterature.

Kirchner and Fouque [KF16] begin their algorithm by sampling an exponential number of vectors from the discrete Gaussian distribution $D_{\mathcal{L},s}$ and then using a pigeon-hole principle, show that there are two distinct sums of d vectors (for an appropriate d) that are equal mod $q\mathcal{L}$, for some large enough integer q. This results in a $\{-1, 0, 1\}$ combination of input lattice vectors (of Hamming weight at most $2d$) in $q\mathcal{L}$; a similar idea was used in [BLS16] to construct their tuple sieving algorithm. In both algorithms, it is difficult to control (i) the distribution of the resulting vectors, (ii) the dependence between resulting vectors.

Bai et al [BLS16] get around the above issues by making a heuristic assumption that the resulting vectors behave like independent samples from a "nice enough" distribution. [HK17] proved that this heuristic indeed leads to the time-memory tradeoff conjectured in [BLS16], but don't prove correctness. As noted in [HK17, p. 22], their analysis only applies to "fixed k" which in our context means $q = O(1)$. In particular, they cannot obtain the regime of subexponential space and superexponential time.

Kirchner and Fouque, on the other hand, use the pigeon-hole principle to argue that there exist coefficients $\alpha_1, \ldots, \alpha_{2d} \in \{-1, 0, 1\}$ and $2d$ lattice vectors in the set of input vectors $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_{2d}$ such that $\frac{\sum_{i=1}^{2d} \alpha_i \boldsymbol{v}_i}{q} \in \mathcal{L}$. It is then stated that $\frac{\sum_{i=1}^{2d} \alpha_i \boldsymbol{v}_i}{q}$ has a nice enough Discrete Gaussian distribution. We observe that while the resulting distribution obtained will indeed be close to a discrete Gaussian distribution, we have no control over the parameter s of this distribution and it can be anywhere between $1/q$ and $\sqrt{2d}/q$ depending on the number of nonzero coordinates in $(\alpha_1, \ldots, \alpha_q)$. For instance, let $\boldsymbol{v}_1, \cdots, \boldsymbol{v}_5$ be input vectors which are all from $D_{\mathcal{L},s}$ for some large s and we want to find the collision in $q\mathcal{L}$ for some positive integer q. Suppose that we find a combination $\boldsymbol{w}_1 = \boldsymbol{v}_1 + \boldsymbol{v}_2 - (\boldsymbol{v}_1 + \boldsymbol{v}_3) \in q\mathcal{L}$ and another combination $\boldsymbol{w}_2 = \boldsymbol{v}_2 + \boldsymbol{v}_3 - (\boldsymbol{v}_4 + \boldsymbol{v}_5) \in q\mathcal{L}$, then by Theorem 2.28, one would expect that $\boldsymbol{w}_1/q \sim D_{\mathcal{L},\sqrt{2}s/q}$ and $\boldsymbol{w}_2/q \sim D_{\mathcal{L},\sqrt{4}s/q}$. This means that the output of the exhaustive search algorithm by Kirchner and Fouque will behave like samples taken from discrete Gaussian distributions with different parameters, making it difficult to keep track of the standard deviation after several steps of the algorithm, and to obtain samples from the Discrete Gaussian distribution at the desired parameter above the smoothing parameter. We overcome this issue by showing that there is a combination of the input vectors with a *fixed* Hamming weight that is in $q\mathcal{L}$.

There are other technical details that were overlooked in [KF16]. In particular, one needs be careful with respect to the errors, both in the probability of failure and the statistical distance of the input/output. Indeed the algorithm performs an exponential number of steps, it is not enough to show that the algorithm succeeds with "overwhelming probability" and that the output has a "negligible statistical distance" from the desired output. However, none of the claimed error bounds in [KF16] are proven and almost the entire proof of the Exhaustive Search (Theorem 3.4) is left to the reader.

Chapter 4
New Space Efficient Provable Quantum and Classical Algorithms for the SVP

For convenience, in this chapter, we will ignore polynomial factors in the bit-length of the individual input basis vectors of a lattice and only consider the dependence on the ambient dimension n. Since all our algorithm have running time at least $2^{\Omega(n)}$, this is the same as assuming that the bit-length of the basis vectors is $2^{o(n)}$. Since the basis vectors are usually taken to be of size polynomial in the dimension, this assumption induces no real loss of generality.

The work in this chapter has been published at STACS 2021 [ACKS21] and is a joint work with Divesh Aggarwal, Yanlin Chen and Rajendra Kumar. It also appears in part in Rajendra Kumar's PhD thesis [Kum21]. This chapter contains several improvements over the published version and has been significantly rewritten.

4.1 Introduction

In a recent work, Chen et al [CCL18] proposed an algorithm for SVP based on Discrete Gaussian sampling (DGS) in [ADRS15]. Recall that DGS consists in producing samples according to the discrete Gaussian distribution (see Definition 2.16 and Section 2.4). Their algorithm runs in time $2^{2.05n+o(n)}$ and the memory requirement is $2^{0.5n+o(n)}$. The quantum variant of their algorithm runs in time $2^{1.2553n+o(n)}$, has the same classical space complexity and uses only poly(n) numbers of qubits. Both their classical and quantum algorithms previously had the best space complexity among known provable algorithms that run in time $2^{O(n)}$. In this chapter, we improve upon their results and provide faster classical and quantum algorithms for SVP. Let us first recall the randomized reduction from SVP to BDD shown in [CCL18]. Recall that an α-BDD oracle takes as input a lattice $\mathcal{L}$, a target vector $\mathbf{t}$ with $\mathrm{dist}(\boldsymbol{t}, \mathcal{L}) \leqslant \alpha\lambda_1(\mathcal{L})$ and outputs a vector $\mathbf{y} \in \mathcal{L}$ such that $\|\mathbf{y} - \mathbf{t}\| = \mathrm{dist}(\mathbf{t}, \mathcal{L})$ (see Definition 2.21).

Theorem ([CCL18, Theorem 8])**.** *Given a basis matrix $\mathbf{B} \subset \mathbb{R}^{n\times n}$ for lattice $\mathcal{L}(\mathbf{B}) \subset \mathbb{R}^n$, a target vector $\boldsymbol{t} \in \mathbb{R}^n$, an α-BDD oracle BDD_α with $\alpha < 0.5$, and an integer scalar $p > 0$. Let $f_p^\alpha : \mathbb{Z}_p^n \to \mathbb{R}^n$ be*

$$f_p^\alpha(\boldsymbol{s}) = -p \cdot \mathsf{BDD}_\alpha(\mathcal{L}, (\mathbf{B}\,\boldsymbol{s} - \boldsymbol{t})/p) + \mathbf{B}\,\boldsymbol{s}.$$

If $\mathrm{dist}(\boldsymbol{t}, \mathcal{L}) \leqslant \alpha\lambda_1(\mathcal{L})$, then the list $m = \{f_p^\alpha(\boldsymbol{s}) \mid \boldsymbol{s} \in \mathbb{Z}_p^n\}$ contains all lattice points at distance less than or equal to $p\alpha\lambda_1(\mathcal{L})$ to $\boldsymbol{t}$.

The BDD oracle built in [DRS14] by discrete Gaussian samples has decoding distance $\alpha\lambda_1(\mathcal{L})$ for $\alpha < 1/2$ (there is a trade-off between α and the number of DGS samples needed),

and therefore, if we choose $\boldsymbol{t}$ to be $\mathbf{0}$, then p has to be at least 3 to ensure that the shortest vector is one of the vectors output by the list m.

In order to solve SVP with $p = 3$, it is sufficient to use a BDD oracle with decoding coefficient α equal to $1/3$. In [CCL18], the authors use the reduction from BDD to DGS by [DRS14] and use the Gaussian sampler of [ADRS15] to obtain many samples with standard deviation equal to $\sqrt{2}\eta_{1/2}$. This allows them to construct a 0.391-BDD but each query to the BDD oracle uses many DGS samples. This is wasteful since we really only need a 1/3-BDD. The reason why it is so expensive is that in the analysis they need to find ε such that $\eta_\varepsilon > \sqrt{2}\eta_{1/2}$ to apply the reduction, and it requires them to take ε much smaller than would be strictly necessary to construct a 1/3-BDD oracle; this smaller ε explains the bigger decoding radius.

We will improve the time complexity of the algorithm in two ways. First we show how to improve the BDD oracle so that each query becomes cheaper. We do so by building a DGS sampler directly at the smoothing parameter to avoid the $\sqrt{2}$ factor that makes the decoding radius unnecessarily large (and hence the BDD too expensive). Second, we show how to reduce the number of queries to the BDD oracle. Recall that in [CCL18], one needs to query $p^n = 3^n$ times the 0.391-BDD to construct the list m in the classical setting and $p^{n/2} = 3^{n/2}$ times in the quantum setting by the quantum minimum finding algorithm in Theorem 2.6. By making smarter queries, we show how to take $p = 2$ above (and therefore greatly reduce the number of queries). A detailed description of our contributions is available in Section 1.1.2.

For convenience, in this chapter, we will ignore polynomial factors in the bit-length of the individual input basis vectors (i.e. we consider only the dependence on the ambient dimension n) when discussing the complexity of algorithms for DGS, BDD or SVP.

4.2 Improved algorithms for BDD

We obtain a BDD oracle with decoding distance α by using the same reduction as above but making each call cheaper. This is achieved by building a sampler that directly samples at the smoothing parameter, hence avoiding the $\sqrt{2}$ factor, allowing us to take a bigger ε. In [ADRS15], it was shown how to construct a dense lattice $\mathcal{L}'$ whose smoothing parameter $\eta(\mathcal{L}')$ is $\sqrt{2}$ times smaller than the original lattice, and that contains all lattice points of the original lattice. Suppose that we first use such a dense lattice to construct a corresponding discrete Gaussian sampler with standard deviation equal to $s = \sqrt{2}\eta(\mathcal{L}')$. We then do the rejection sampling on condition that the output is in the original lattice $\mathcal{L}$. We thus have constructed a discrete Gaussian sampler of $\mathcal{L}$ whose standard deviation is $\sqrt{2}\eta(\mathcal{L}') = \eta(\mathcal{L})$. Nevertheless, $|\mathcal{L}'/\mathcal{L}|$ will be at least $2^{0.5n}$, which implies that this procedure needs at least $2^{0.5n}$ input vectors to produce an output vector.

The complexity of our BDD algorithms heavily depends on the quantity $\beta(\mathcal{L})$ which is related to the kissing number of the lattice (see Section 2.4). For this reason, we first provide complexity bounds that depend on $\beta(\mathcal{L})$ and then obtain complexity bounds in the worst case ($\beta(\mathcal{L}) \leqslant 2^{0.401+o(1)}$) as corollaries. We first show how to efficient sample a discrete Gaussian at the smoothing parameter.

Lemma 4.1. *There is a probabilistic algorithm that, given a lattice $\mathcal{L} \subset \mathbb{R}^n$, $m \in \mathbb{Z}_+$ and $s \geqslant \eta_{1/3}(\mathcal{L})$ as input, outputs m samples from a distribution $(m \cdot 2^{-\Omega(n^2)})$-close to $D_{\mathcal{L},s}$ in*

expected time $m \cdot 2^{n/2+o(n)}$ *and space* $(m + 2^{n/2}) \cdot 2^{o(n)}$*. Furthermore, all samples have* $\mathrm{poly}(n)$ *bit-size.*

Proof. Let $a = \frac{n}{2} + 4$. We repeat the following until we output m vectors. We use the algorithm in Lemma 2.32 to obtain a lattice $\mathcal{L}' \supset \mathcal{L}$ of index 2^a. We then run the algorithm from Theorem 2.31 with input $(\mathcal{L}', s)$ to obtain a list of vectors from $\mathcal{L}'$. We output the vectors in this list that belong to $\mathcal{L}$. The correctness of the algorithm, assuming it outputs anything, is clear as long as the samples obtained from Theorem 2.31 are (sufficiently) independent, which we will prove below.

By Theorem 2.31, we obtain, in time and space $2^{(n/2)+o(n)}$, $M \leqslant 2^{n/2}$ vectors that are $2^{-\Omega(n^2)}$-close to M vectors independently sampled from $D_{\mathcal{L}',s}$. The theorem guarantees that $M = 2^{n/2}$ if $s \geqslant \sqrt{2}\eta_{1/2}(\mathcal{L}')$. Also, by Lemma 2.32, with probability at least $1/2$, we have $s \geqslant \eta_{1/3}(\mathcal{L}) \geqslant \sqrt{2}\eta_{1/2}(\mathcal{L}')$. Note that when $s < \sqrt{2}\eta_{1/2}(\mathcal{L}')$, the samples obtained from Theorem 2.31 are still $2^{-\Omega(n^2)}$-close to M vectors independently sampled from $D_{\mathcal{L}',s}$ but M could be much lower than $2^{n/2}$ or even 0. On the other hand, if $s \geqslant \sqrt{2}\eta_{1/2}(\mathcal{L}')$ then $M = 2^{n/2}$.

Assume that $s \geqslant \sqrt{2}\eta_{1/2}(\mathcal{L}')$, which happens with probability at least $1/2$. From these $M = 2^{n/2}$ vectors, we will reject the vectors which are not in lattice $\mathcal{L}$. It is easy to see that the probability that a vector sampled from the distribution $D_{\mathcal{L}',s}$ is in $\mathcal{L}$ is at least $\rho_s(\mathcal{L})/\rho_s(\mathcal{L}') \geqslant \frac{1}{2^a}$ using Lemma 2.22. Thus, the probability that we obtain at least one vector from $\mathcal{L}$ (which is distributed as $D_{\mathcal{L},s}$) is at least

$$\frac{1}{2}\left(1 - (1 - 1/2^a)^{2^{n/2}}\right) = \frac{1}{2}\left(1 - (1 - 1/2^{n/2+4})^{2^{n/2}}\right) \geqslant \frac{1}{2} \cdot \left(1 - e^{-2^{n/2}/2^{n/2+4}}\right) = \frac{1}{2}(1 - e^{-1/16}).$$

It implies that after rejection of vectors, with constant probability we will get at least one vector from $D_{\mathcal{L},s}$. Thus, the expected number of times we need to repeat the algorithm is $O(m)$ until we obtain vectors $\boldsymbol{y_1}, \ldots, \boldsymbol{y_m}$ whose distribution is statistically close to being independently distributed from $D_{\mathcal{L},s}$. The time and space complexity is clear from the algorithm.

We can ensure that all samples have $\mathrm{poly}(n)$ bit-size by first generating more samples (say twice the amount) and throwing away all samples of norm larger than $\exp(\Omega(n^2))$. Since the vectors are sampled from a Gaussian with width at most[1] $\exp(O(n))$, the error induced by throwing away the tail of the distribution is smaller than $2^{-\Omega(n^2)}$. □

4.2.1 BDD when ε is small

In order to go further, we will make heavy use of Theorem 2.40 and Lemma 2.25 to relate the smoothing parameter to other parameters of the lattice. A small difficulty when applying Lemma 2.25 is the case distinction on ε. We will start by using inequality (2.4) which will require to take very small values of ε when sampling the discrete Gaussian.

Lemma 4.2. *For any sufficiently large* n*, any lattice* $\mathcal{L} \subset \mathbb{R}^n$ *and any* A *such that* $\frac{1}{2\ln 2} - b + o(1) \leqslant A \leqslant 1$*, there exists a randomized algorithm that creates a* α*-BDD oracle in time* $2^{(A+1)n/2+o(n)}$ *and space* $2^{0.5n+o(n)}$*, where* $\alpha = \frac{1}{2}\sqrt{\frac{A}{A+b}}$ *and* $b = \log_2 \beta(\mathcal{L})$*. Every call to this oracle takes time* $2^{An/2+o(n)}$ *and space* $\mathrm{poly}(n)$*, excluding the space of the preprocessed data.*

[1] Here we are using our assumption that the basis vectors have size at most $2^{o(n)}$.

Proof. Let $\varepsilon = 2^{-An}$, $A \leqslant 1$ to be fixed later. We know that $\eta_\varepsilon(\mathcal{L}^*) > \eta_{1/3}(\mathcal{L}^*)$ for any sufficiently large n ($n > \frac{1}{A}\log_2 3$) by the monotonicity of the smoothing parameter function. Hence the $\mathsf{DGS}^m_{\eta_{1/3}}$ sampler from Lemma 4.1 can be used as a $\mathsf{DGS}^m_{\eta_\varepsilon}$ sampler, for any $m \in \mathbb{N}$.

By Theorem 2.40 and Remark 2.41 we can construct a $\alpha-$BDD such that each call takes time $m \cdot \mathrm{poly}(n) = 2^{An/2+o(n)}$ and space $\mathrm{poly}(n)$, where $\alpha = \phi(\mathcal{L})/\lambda_1(\mathcal{L}) = \frac{\sqrt{\ln(1/\varepsilon)/\pi - o(1)}}{2\eta_\varepsilon(\mathcal{L}^*)\lambda_1(\mathcal{L})}$ and $m = O(\frac{n\log(1/\varepsilon)}{\sqrt{\varepsilon}}) = 2^{An/2+o(n)}$. The preprocessing consists of $\mathrm{poly}(n)$ calls to the $\mathsf{DGS}^m_{\eta_\varepsilon}$ sampler described above and requires space $m \cdot \mathrm{poly}(n)$. Hence the total complexity is $\mathrm{poly}(n) \cdot m \cdot 2^{n/2+o(n)} = 2^{(A+1)n/2+o(n)}$ in time and $2^{An/2+o(n)} \leqslant 2^{n/2+o(n)}$ in space. By using Lemma 2.25, inequality (2.4), only valid when $\varepsilon \leqslant (e/\beta(\mathcal{L})^2 + o(1))^{-\frac{n}{2}}$, we have that

$$\lambda_1(\mathcal{L})\eta_\varepsilon(\mathcal{L}^*) < \sqrt{\frac{\ln(1/\varepsilon) + n\ln\beta(\mathcal{L}) + o(n)}{\pi}}.$$

Hence we can guarantee that

$$\alpha = \frac{\sqrt{\ln(1/\varepsilon)/\pi - o(1)}}{2\eta_\varepsilon(\mathcal{L}^*)\lambda_1} > \frac{\sqrt{\ln(1/\varepsilon)/\pi - o(1)}}{2\sqrt{\frac{\ln(1/\varepsilon)+n\ln\beta(\mathcal{L})+o(n)}{\pi}}} = \frac{1}{2}\sqrt{\frac{\ln(1/\varepsilon)+o(1)}{\ln(1/\varepsilon)+n\ln\beta(\mathcal{L})}} = \frac{1}{2}\sqrt{\frac{A}{A+b}} + o(1)$$

where $b = \log_2\beta(\mathcal{L})$. Furthermore, as noted above, this inequality is only valid when $\varepsilon \leqslant (e/\beta^2 + o(1))^{-\frac{n}{2}}$, that is $A \geqslant \frac{1}{2\ln 2} - b + o(1)$. Finally, note that since $b \leqslant 0.401$, we must have $A \geqslant 0.32$ and the inequality holds as soon as $n \geqslant 5 \geqslant \frac{1}{A}\log_2 3$. Finally note that Theorem 2.40 requires $\varepsilon \leqslant 1/200$ which holds as soon as $n \geqslant 17 \geqslant \frac{1}{A}\ln 200$. □

We can reformulate the previous lemma by expressing the complexity in terms of α instead of some arbitrary constant A.

Corollary 4.3. *For any $n \geqslant 5$, any lattice $\mathcal{L} \subset \mathbb{R}^n$ and any α such that $\frac{1}{2}\sqrt{1-2b\ln 2} + o(1) \leqslant \alpha < \frac{1}{2}\sqrt{\frac{1}{1+b}}$, there exists a randomized algorithm that creates a α-BDD oracle in time $2^{(A+1)n/2+o(n)}$ and space $2^{0.5n+o(n)}$ such that every call to this oracle takes time $2^{An/2+o(n)}$, and space $\mathrm{poly}(n)$, excluding the space of the preprocessed data, where $A = \frac{4b\alpha^2}{1-4\alpha^2}$ and $b = \log_2\beta(\mathcal{L})$.*

Proof. Apply Lemma 4.2 for some A to be fixed later. Observe that $\alpha = \frac{1}{2}\sqrt{\frac{A}{A+b}}$ so $A = \frac{4b\alpha^2}{1-4\alpha^2}$. Now the constraints $\frac{1}{2\ln 2} - b + o(1) \leqslant A \leqslant 1$ become

$$\begin{aligned}\tfrac{1}{2\ln 2} - b + o(1) \leqslant \tfrac{4b\alpha^2}{1-4\alpha^2} &\Leftrightarrow (\tfrac{1}{2\ln 2} - b + o(1))(1-4\alpha^2) \leqslant 4b\alpha^2\\ &\Leftrightarrow \tfrac{1}{4\ln 2} - \tfrac{b}{2} + o(1) \leqslant \tfrac{\alpha^2}{\ln 2}\\ &\Leftrightarrow \tfrac{1}{2}\sqrt{1-2b\ln 2} + o(1) \leqslant \alpha\end{aligned}$$

and

$$\tfrac{4b\alpha^2}{1-4\alpha^2} \leqslant 1 \Leftrightarrow 4(1+b)\alpha^2 \leqslant 1 \Leftrightarrow \alpha \leqslant \tfrac{1}{2}\sqrt{\tfrac{1}{1+b}}.$$

□

4.2.2 BDD when ε is large

The inequality (2.4) in Lemma 2.25 tells us that if we take an extremely small ε to compute the BDD oracle, we can find a BDD oracle with $\alpha(\mathcal{L})$ almost $1/2$. However the time complexity for each call of the oracle will be very costly. On the other hand, if we use the inequality (2.3) in Lemma 2.25 with a larger ε, each call of the oracle will take much less time, but the constraint on the decoding coefficient α will be different. It is therefore important to study this second regime as well. Note that inequality (2.3) actually applies to all $\varepsilon \in (0,1)$ but is mostly useful when ε is large.

Lemma 4.4. *For any sufficiently large n, any lattice $\mathcal{L} \subset \mathbb{R}^n$ and any $\frac{1}{n}\log_2 3 \leqslant A \leqslant 1$, there exists a randomized algorithm that creates a α-BDD oracle in time $2^{(A+1)n/2+o(n)}$ and space $2^{0.5n+o(n)}$, where $\alpha = \frac{2^{-A}\sqrt{A}\sqrt{2e\ln 2}}{2\beta(\mathcal{L})} - o(1)$. Every call to this oracle takes time $2^{An/2+o(n)}$ and space $\mathrm{poly}(n)$, excluding the space of the preprocessed data.*

Proof. Let $\varepsilon = 2^{-An}$, $A \leqslant 1$ to be fixed later. We know that $\eta_\varepsilon(\mathcal{L}^*) > \eta_{1/3}(\mathcal{L}^*)$ for any sufficiently large n ($n > \frac{1}{A}\log_2 3$) by the monotonicity of the smoothing parameter function. Hence the $\mathsf{DGS}^m_{\eta_{1/3}}$ sampler from Lemma 4.1 can be used as a $\mathsf{DGS}^m_{\eta_\varepsilon}$ sampler, for any $m \in \mathbb{N}$.

By Theorem 2.40 and Remark 2.41 we can construct a $\alpha-$BDD such that each call takes time $m \cdot \mathrm{poly}(n) = 2^{An/2+o(n)}$ and space $\mathrm{poly}(n)$, where $\alpha = \phi(\mathcal{L})/\lambda_1(\mathcal{L}) = \frac{\sqrt{\ln(1/\varepsilon)/\pi - o(1)}}{2\eta_\varepsilon(\mathcal{L}^*)\lambda_1(\mathcal{L})}$ and $m = O(\frac{n\log(1/\varepsilon)}{\sqrt{\varepsilon}}) = 2^{An/2+o(n)}$. The preprocessing consists of $\mathrm{poly}(n)$ calls to the $\mathsf{DGS}^m_{\eta_\varepsilon}$ sampler described above and requires space $m \cdot \mathrm{poly}(n)$. Hence the total complexity is $\mathrm{poly}(n) \cdot m \cdot 2^{n/2+o(n)} = 2^{(A+1)n/2+o(n)}$ in time and $2^{An/2+o(n)} \leqslant 2^{n/2+o(n)}$ in space. By using inequality (2.3), in Lemma 2.25, we have that

$$\lambda_1(\mathcal{L})\eta_\varepsilon(\mathcal{L}^*) < \sqrt{\frac{\beta(\mathcal{L})^2 n}{2\pi e}} \cdot \varepsilon^{-1/n}(1+o(1)).$$

Hence we can guarantee that

$$\alpha = \frac{\sqrt{\ln(1/\varepsilon)/\pi - o(1)}}{2\eta_\varepsilon(\mathcal{L}^*)\lambda_1(\mathcal{L})} > \frac{1}{2}\sqrt{\frac{2e\ln\frac{1}{\varepsilon} - o(1)}{n}} \cdot \beta(\mathcal{L})^{-1}\varepsilon^{\frac{1}{n}} \cdot (1-o(1)) = \frac{2^{-A}\sqrt{A}\cdot\sqrt{2e\ln 2}}{2\beta(\mathcal{L})} - o(1).$$

□

Corollary 4.5. *For any $n \geqslant 2$, any integer $m > 0$, any lattice $\mathcal{L} \subset \mathbb{R}^n$ and any $\frac{\sqrt{e\ln 3}}{\sqrt{2n}\beta(\mathcal{L})} \leqslant \alpha \leqslant \frac{1}{2\beta(\mathcal{L})}$, where $b = \log_2 \beta(\mathcal{L})$, there exists a randomized algorithm that creates $(\alpha + o(1))$-BDD oracle in time $2^{(A+1)n/2+o(n)}$ and space $2^{n/2+o(n)}$, such that each call takes time $2^{An/2+o(n)}$ and space $\mathrm{poly}(n)$, excluding the space of the preprocessed data, where*

$$A = -\frac{1}{2\ln 2}W(-\frac{4\alpha^2\beta(\mathcal{L})^2}{e})$$

where W is the principal branch of the Lambert W function. Furthermore, the above expression of A is a continuous and increasing function of $\beta(\mathcal{L})$.

Proof. By Lemma 4.4, we can build an oracle for any $\frac{1}{n}\log_2 3 \leqslant A \leqslant 1$ such that the decoding radius is $\alpha = \frac{2^{-A}\sqrt{A}\sqrt{2e\ln 2}}{2\beta(\mathcal{L})} - o(1)$. Hence, we want to find A such that

$$\frac{2^{-A}\sqrt{A}\sqrt{2e\ln 2}}{2\beta(\mathcal{L})} = \alpha \qquad \text{and} \qquad \tfrac{1}{n}\log_2 3 \leqslant A \leqslant 1.$$

Let $f : A \mapsto 2^{-A}\sqrt{A}$ so that the first condition is equivalent to

$$f(A) = \tfrac{2\alpha\beta(\mathcal{L})}{\sqrt{2e\ln 2}}. \tag{4.1}$$

Now assume that (4.1) holds and let $y = -2A\ln(2)$, then it is equivalent to

$$e^y y = -2\ln(2)\tfrac{2\alpha^2\beta(\mathcal{L})^2}{e\ln 2},$$

that is

$$e^y y = -\tfrac{4\alpha^2\beta(\mathcal{L})^2}{e}. \tag{4.2}$$

This equation admits a solution if and only if

$$-\tfrac{4\alpha^2\beta(\mathcal{L})^2}{e} \geqslant -\tfrac{1}{e} \quad \Leftrightarrow \quad \alpha \leqslant \tfrac{1}{2\beta(\mathcal{L})}. \tag{4.3}$$

Assuming this is the case, (4.2) can admit up to two solutions. However, since the complexity increases with A, we want the solution that minimizes A, *i.e.* that maximizes y. The largest of the (up to) two solutions of (4.2) is always given by be the principal branch W of the Lambert W function:

$$y = W(-\tfrac{4\alpha^2\beta(\mathcal{L})^2}{e}) \quad \text{that is} \quad A = -\tfrac{1}{2\ln 2}W(-\tfrac{4\alpha^2\beta(\mathcal{L})^2}{e}) \tag{4.4}$$

and always satisfies $y \geqslant -1$. In particular, we always have $A \leqslant \frac{1}{2\ln 2}$. Now check that f is strictly increasing over $[0, \frac{1}{2\ln 2}]$. Hence, the condition $\frac{1}{n}\log_2 3 \leqslant A$ is equivalent to

$$\begin{aligned}
&f(\tfrac{1}{n}\log_2 3) \leqslant f(A) \\
&\Leftrightarrow f(\tfrac{1}{n}\log_2 3)^2 \leqslant \left(\tfrac{2\alpha\beta(\mathcal{L})}{\sqrt{2e\ln 2}}\right)^2 && \text{by (4.1)} \\
&\Leftrightarrow 2^{-\frac{2}{n}\log_2 3}\tfrac{1}{n}\log_2 3 \leqslant \tfrac{2\alpha^2\beta(\mathcal{L})^2}{e\ln 2} \\
&\Leftrightarrow \tfrac{e\ln 3}{2n\beta(\mathcal{L})^2}9^{-\frac{1}{n}} \leqslant \alpha^2 \\
&\Leftarrow \tfrac{e\ln 3}{2n\beta(\mathcal{L})^2} \leqslant \alpha^2. && (4.5)
\end{aligned}$$

In summary, we can always take A as in (4.4) assuming (4.3) and (4.5) hold. □

4.2.3 Putting everything together

We have analyzed the construction of α-BDDs in two regimes, based on Lemma 4.1. It is not a priori clear which construction is better and in fact we will see that it depends in a nontrivial way on the relation between α and $\beta(\mathcal{L})$.

Theorem 4.6. *For any sufficiently large n, any $m > 0$, any $\frac{\sqrt{e \ln 3}}{\sqrt{2n}\beta(\mathcal{L})} \leqslant \alpha < \frac{1}{2}\sqrt{\frac{1}{1+b}}$ and any lattice $\mathcal{L} \subset \mathbb{R}^n$, there exists a randomized algorithm that creates a $(\alpha + o(1))$-BDD oracle in time $2^{(A+1)n/2+o(n)}$ and space $2^{n/2}$, such that each call takes time $2^{An/2+o(n)}$ and space $\mathrm{poly}(n)$, excluding the space of the preprocessed data, where*

$$A = \begin{cases} -\frac{1}{2\ln 2} W(-\frac{4\alpha^2\beta(\mathcal{L})^2}{e}) & \text{when } b < \frac{1-4\alpha^2}{2\ln 2} \\ \frac{4\alpha^2}{1-4\alpha^2} b & \text{when } b \geqslant \frac{1-4\alpha^2}{2\ln 2} \end{cases}$$

where W is the principal branch of the Lambert W function and $b = \log_2 \beta(\mathcal{L})$. Furthermore, the above expression of A is a continuous and increasing function of b.

Proof. Let $\frac{\sqrt{e \ln 3}}{\sqrt{2n}\beta(\mathcal{L})} \leqslant \alpha < \frac{1}{2}\sqrt{\frac{1}{1+b}}$ and $b = \log_2 \beta(\mathcal{L})$. By Corollary 4.3, we can build an α-BDD if $\frac{1}{2}\sqrt{1 - 2b\ln 2} \leqslant \alpha$, in which case the complexity will depend on $A = A_1(\alpha, b) := \frac{4b\alpha^2}{1-4\alpha^2}$. By Corollary 4.5, we can build an α-BDD if $\alpha < \frac{1}{2}\sqrt{1 - 2b\ln 2}$, in which case the complexity will depend on $A = A_2(\alpha, b) := -\frac{1}{2\ln 2} W(-\frac{4\alpha^2\beta(\mathcal{L})^2}{e})$. In both cases, the BDD oracle can be created in time $2^{(A+1)n/2+o(n)}$, space $2^{0.5n+o(n)}$ and each call takes time $2^{An/2+o(n)}$. Now observe that

$$\alpha < \tfrac{1}{2}\sqrt{1 - 2b\ln 2} \;\Leftrightarrow\; 4\alpha^2 < 1 - 2b\ln 2 \;\Leftrightarrow\; b < \tfrac{1-4\alpha^2}{2\ln 2}.$$

Let $b^* := \frac{1-4\alpha^2}{2\ln 2}$, then there are two cases:

- If $b \geqslant b^*$ then $\frac{1}{2}\sqrt{1 - 2b\ln 2} \leqslant \alpha$ so only Corollary 4.3 applies and we can build a α-BDD. In this case the complexity exponent is $A_1(\alpha, b) = \frac{4\alpha^2}{1-4\alpha^2} b$.
- If $b < b^*$ then $\alpha < \frac{1}{2}\sqrt{1 - 2b\ln 2}$ so Corollary 4.5 applies but Corollary 4.3 does not for this particular value of α. However, we can apply Corollary 4.3 to build a α'-BDD oracle with $\alpha' \geqslant \alpha_1^{min}(b) := \frac{1}{2}\sqrt{1 - 2b\ln 2} > \alpha$. We will show that the α-BDD of Corollary 4.5 is always more efficient than the α'-BDD of Corollary 4.3 in this case and the complexity exponent will thus be $A_2(\alpha, b) = -\frac{1}{2\ln 2} W(-\frac{4\alpha^2\beta(\mathcal{L})^2}{e})$.

Assume that $b < b^*$, we claim that $A_1(\alpha, b) \geqslant A_2(\alpha', b)$ for any $\alpha' \geqslant \alpha_1^{min}(b)$. Indeed, on the one hand A_2 is an increasing function of b so

$$A_2(\alpha, b) < A_2(\alpha, \tfrac{5}{18\ln 2}) = -\tfrac{1}{2\ln 2} W(-4\alpha^2 e^{-4\alpha^2}) = \tfrac{4\alpha^2}{2\ln 2} = \tfrac{2\alpha^2}{\ln 2} \qquad \text{since } W(xe^x) = x.$$

On the other hand, A_1 is an increasing function of α so

$$A_1(\alpha, b) \geqslant A_1(\alpha_1^{min}(b), b) = \tfrac{1-2b\ln 2}{2\ln 2}$$

which is a decreasing function of b, therefore

$$A_1(\alpha, b) \geqslant A_1(\alpha_1^{min}(b^*), b^*) = \tfrac{1-2b^*\ln 2}{2\ln 2} = \tfrac{2\alpha^2}{\ln 2} > A_2(\alpha, b). \qquad \square$$

Corollary 4.7. *For any sufficiently large integer n, any integer $m > 0$, any lattice $\mathcal{L} \subset \mathbb{R}^n$ there exists a randomized algorithm that creates a 1/3-BDD oracle in time $2^{0.6604n+o(n)}$ and space $2^{0.5n+o(n)}$ such that very call to this oracle takes time $2^{0.1604n+o(n)}$ and space $\mathrm{poly}(n)$, excluding the space of the preprocessed data.*

Proof. By Theorem 4.6, the value of A increases with $b = \log_2 \beta(\mathcal{L})$. Since $b \leqslant 0.401 + o(1)$ and $0.401 \geqslant \frac{5}{18\ln 2}$, we always have $A \leqslant \frac{4}{5}0.401 + o(1) \leqslant 0.3208 + o(1)$ and we obtain the result. $\square$

4.3 Quantum algorithm for SVP

From [CCL18], we can enumerate all vectors of length $p \cdot \frac{1}{3}\lambda_1(\mathcal{L})$ by making p^n calls to a 1/3-BDD oracle. Although naively searching for the minimum in the set of vectors of length less than or equal to $p \cdot \frac{1}{3}\lambda_1(\mathcal{L})$, will find the origin with high probability, one can work around this issue by shifting the zero vector. Choosing an arbitrary nonzero lattice vector as the shift, we are guaranteed to obtain a vector of length at least λ_1 for $p \geqslant 3$. Hence by combining the 1/3-BDD oracle from Theorem 4.6 and the quantum minimum finding algorithm from Theorem 2.6, we can find the shortest vector.

Theorem 4.8. *For sufficiently large n, there is a quantum algorithm that solves SVP in time $2^{(A+\log_2 3+\varepsilon)n/2+o(n)}$ and classical-space $2^{0.5n+o(n)}$ with a polynomial number of qubits, where*

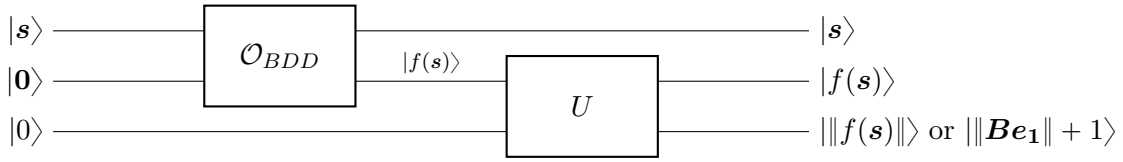
$$A = \begin{cases} -\frac{1}{2\ln 2} W(-\frac{4\beta(\mathcal{L})^2}{9e}) & \text{when } b < \frac{5}{18\ln 2} \\ \frac{4}{5} b & \text{when } b \geqslant \frac{5}{18\ln 2} \end{cases}$$

and $b = \log_2 \beta(\mathcal{L})$. In particular, there is a quantum algorithm that solves SVP in time $2^{0.9529n+o(n)}$ and classical-space $2^{0.5n+o(n)}$ with a polynomial number of qubits.

Proof. Let $\mathbf{B}$ be a basis of the lattice, $\mathsf{BDD}_{1/3}$ be a 1/3-BDD oracle and let $f : \mathbb{Z}_3^n \to \mathcal{L}$ be $f(\boldsymbol{s}) = -3 \cdot \mathsf{BDD}_{1/3}(\mathcal{L}, (\boldsymbol{Bs})/3) + \boldsymbol{Bs}$. The algorithm works on three quantum registers and our goal is to build a superposition of states of the form $|\boldsymbol{s}\rangle|f(\boldsymbol{s})\rangle|x\rangle$ where $x = \|f(\boldsymbol{s})\|$ most of the time (see the definition of U below). First apply Theorem 4.6 to construct 1/3-BDD oracle. This oracle runs in time $2^{An/2+o(n)}$ and space $\mathrm{poly}(n)$ excluding the preprocessed data. Hence, we can construct a classical circuit of size $2^{An/2+o(n)}$ and space $\mathrm{poly}(n)$ for this oracle, by hard-coding the preprocessed data in the circuit. Let $\varepsilon > 0$ and apply Corollary 2.2 to construct a quantum circuit $\mathcal{O}_{BDD}$ of size $2^{(A+\varepsilon)n/2+o(n)}$ on $\mathrm{poly}(n)$ qubits that satisfies $\mathcal{O}_{BDD}|\boldsymbol{s}\rangle|\mathbf{0}\rangle = |\boldsymbol{s}\rangle|f(\boldsymbol{s})\rangle$ for all $\boldsymbol{s} \in \mathbb{Z}_3^n$. We then construct another quantum circuit U satisfying

$$U(|\boldsymbol{\omega}\rangle|0\rangle) = \begin{cases} |\boldsymbol{\omega}\rangle|\|\boldsymbol{\omega}\|\rangle & \text{if } \boldsymbol{\omega} \neq \mathbf{0} \\ |\boldsymbol{\omega}\rangle|\|\boldsymbol{Be_1}\| + 1\rangle & \text{if } \boldsymbol{\omega} = \mathbf{0}, \end{cases}$$

Here $\boldsymbol{e_1} \in \mathbb{Z}^n$ is a vector whose first coordinate is one and the rest are zero. We then consider the following quantum circuit (we have not drawn ancilla qubits):



This circuit $\mathcal{O}$ has size $2^{(A+\varepsilon)n/2+o(n)}$, satisfies $\mathcal{O}|\boldsymbol{s}\rangle|\mathbf{0}\rangle|0\rangle = |\boldsymbol{s}\rangle|f(\boldsymbol{s})\rangle|\|f(\boldsymbol{s})\|\rangle$ if $f(\boldsymbol{s}) \neq 0$ and $\mathcal{O}|\boldsymbol{s}\rangle|\mathbf{0}\rangle|0\rangle = |\boldsymbol{s}\rangle|f(\boldsymbol{s})\rangle|\|\boldsymbol{Be_1}\| + 1\rangle$ and uses $\mathrm{poly}(n)$ qubits. We can now apply the quantum minimum finding algorithm from Theorem 2.6 on the first and third registers and the index $\boldsymbol{s}'$ of a shortest vector in this list. The output of the algorithm will be $f(\boldsymbol{s}')$. As a result of applying the quantum minimum finding algorithm from Theorem 2.6, the quantum algorithm takes time $3^{0.5n} \cdot 2^{(A+\varepsilon)n/2+o(n)} = 2^{(A+\log_2 3+\varepsilon)n/2+o(n)}$ and classical space $2^{0.5n+o(n)}$ with a polynomial number of qubits.

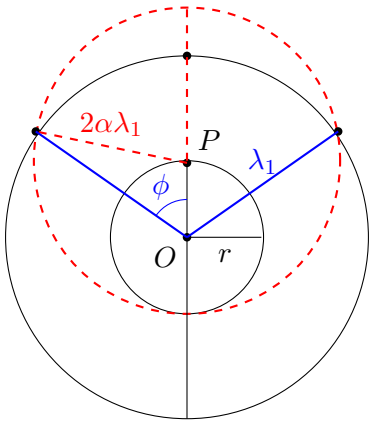


Figure 4.1: One can cover the sphere of radius λ_1 by balls of radius $2\alpha\lambda_1$, where $\frac{1}{3} \leqslant \alpha < \frac{1}{2}$, whose centers (here P) are at distance r from the origin O. Each such ball covers a spherical cap of half-angle ϕ.

Lastly, we show that the quantum algorithm will output a shortest non-zero vector with constant probability. Since $\|\boldsymbol{B}\boldsymbol{e_1}\| + 1 > \lambda_1(\mathcal{L})$, with at least $1/2$ probability one will find the index $\boldsymbol{i}$ such that $f(\boldsymbol{i})$ is a shortest nonzero vector by using the quantum minimum finding algorithm. Therefore it suffices to show that there is an index $\boldsymbol{i} \in \mathbb{Z}_3^n$ such that $\|f(\boldsymbol{i})\| = \lambda_1(\mathcal{L})$. By Theorem 2.35, the list $\{f(\boldsymbol{s}) | \boldsymbol{s} \in \mathbb{Z}_3^n\}$ contains all lattice points in a ball of radius $3 \cdot \frac{1}{3}\lambda_1(\mathcal{L}) = \lambda_1(\mathcal{L})$ from $\mathbf{0}$, including the lattice vector with length $\lambda_1(\mathcal{L})$. Hence with at least $1/2$ probability, the algorithm outputs a non-zero shortest lattice vector.

By the proof of Corollary 4.7, we can always take $A \leqslant 0.3208 + o(1)$ in the construction of the 1/3-BDD. Hence by taking a suitably small ε, the exponent of the running time becomes $(A + \log_2 3 + \varepsilon)n/2 + o(n) = 0.9529n + o(n)$. □

4.4 Solving SVP by spherical caps on the sphere

We now explain how to reduce the number of queries to the α-BDD oracle. Consider a uniformly random target vector $\boldsymbol{t}$ such that $\alpha(1 - \frac{1}{n})\lambda_1(\mathcal{L}) \leqslant \|\boldsymbol{t}\| < \alpha\lambda_1(\mathcal{L})$, it satisfies the condition of Theorem 2.35, *i.e.* $\mathrm{dist}(\mathcal{L}, \boldsymbol{t}) \leqslant \alpha\lambda_1(\mathcal{L})$. We enumerate all lattice vectors within distance $2\alpha\lambda_1(\mathcal{L})$ of $\boldsymbol{t}$ and keep only the shortest nonzero one. We show that for any $\alpha \geqslant \frac{1}{3}$, we will get the shortest nonzero vector of the lattice with probability at least $2^{-cn+o(n)}$ for some c that depends on α. By repeating this $2^{cn+o(n)}$ times, the algorithm will succeed with constant probability. The optimal choice of α is not obvious and is deferred to Section 4.5.

Theorem 4.9. *Assume we can create an α-BDD oracle, with $\alpha \geqslant \frac{1}{3}$, in time T_c, space S_c such that each call takes time T_o. Then there is a randomized algorithm that solves, with constant probability, SVP in space S_c and time*

$$T_c + \frac{2^{n+o(n)}T_o}{\sin^n \phi}$$

where $\cos\phi = \frac{1+s^2-4\alpha^2}{2s}$ *and* $s = \min(\alpha, \sqrt{1-4\alpha^2})$. *Furthermore, there is a quantum algorithm that solves* SVP *in classical space* S_c *using a polynomial number of qubit and time*

$$T_c + \frac{2^{n/2+o(n)}T_o}{\sin^{n/2}\phi}.$$

Proof. On input lattice $\mathcal{L}(\mathbf{B})$, use the LLL algorithm [LLL82] to get a number d (the norm of the first vector of the basis) that satisfies $\lambda_1(\mathcal{L}) \leqslant d \leqslant 2^{n/2}\lambda_1(\mathcal{L})$. For $i = 1, \ldots, n^2$, let $d_i = d/(1+\frac{1}{n})^i$. There exists a j such that $\lambda_1(\mathcal{L}) \leqslant d_j \leqslant (1+\frac{1}{n})\lambda_1(\mathcal{L})$. We repeat the following procedure for all $i = 1, \ldots, n^2$:

Fix $N \in \mathbb{N}$ to be fixed later. For $j = 1$ to N, pick a uniformly random vector $\boldsymbol{v_{ij}}$ on the surface of the ball of radius $r(1-\frac{1}{n})d_i$. By Theorem 2.35, we can enumerate 2^n lattice points using the function $f_{ij} : \mathbb{Z}_2^n \to \mathcal{L}$ defined by

$$f_{ij}(\boldsymbol{x}) = \mathbf{B}\,\boldsymbol{x} - 2 \cdot \mathsf{BDD}_\alpha(\mathcal{L}, (\mathbf{B}\,\boldsymbol{x} - \boldsymbol{v_{ij}})/2). \tag{4.6}$$

At each step we only store the shortest nonzero vector. At the end, we output the shortest among them. The running time of the algorithm is straightforward. we make 2^n queries to a α-BDD oracle that takes time T_o, we further repeat this n^2N times. Therefore the algorithm takes time $T_c + 2^n \cdot n^2N \cdot T_o$ and space S_c. This entire procedure is summarized in Algorithm 4.1.

Algorithm 4.1 Solving SVP by spherical caps on the sphere

Input: basis $\boldsymbol{B}$ of a lattice $\mathcal{L} \subset \mathbb{R}^n$
Input: an α-BDD oracle (for a well-chosen α)
Output: a shortest non-zero vector of $\mathcal{L}$

use LLL to get a number d that satisfies $\lambda_1(\mathcal{L}) \leqslant d \leqslant 2^{n/2}\lambda_1(\mathcal{L})$.
$\boldsymbol{z} \leftarrow$ any basis vector
for $i = 1, \ldots, n^2$ **do**
  $d_i \leftarrow d(1+\frac{1}{n})^{-i}$
  $r \leftarrow \min(\alpha, \sqrt{1-4\alpha^2})$
  $\cos\phi \leftarrow \frac{1+r^2-4\alpha^2}{2r}$
  $N \leftarrow \frac{A_n(\lambda_1)}{V_{n-1}(\lambda_1 \sin\phi)}$ ▷ see (4.7)
  for $j = 1, ..., N$ **do**
    $\boldsymbol{v}_{ij} \leftarrow$ random vector of norm $r(1-\frac{1}{n})d_i$
    for $\boldsymbol{x} \in \{0,1\}^n$ **do**
      $\boldsymbol{y} \leftarrow f_{ij}(\boldsymbol{x})$ ▷ see (4.6)
      if $\|\boldsymbol{y}\| < \|\boldsymbol{x}\|$ **then**
        $\boldsymbol{z} \leftarrow \boldsymbol{y}$ ▷ shorter vector
      end if
    end for
  end for
end for
return $\boldsymbol{z}$

To prove the correctness of the algorithm, it suffices to show that there exists an $i \in [n^2]$ for which the algorithm finds the shortest vector with high probability. Recall that there exists

an i such that $\lambda_1(\mathcal{L}) \leqslant d_i \leqslant (1+\frac{1}{n})\lambda_1(\mathcal{L})$ and let that index be k. We will show that for a uniformly random vector $\boldsymbol{v}$ of length $r(1-\frac{1}{n})d_k$, if we enumerate 2^n vectors by the function $f : \mathbb{Z}_2^n \to \mathcal{L}$,

$$f(\boldsymbol{x}) = \mathbf{B}\,\boldsymbol{x} - 2 \cdot \mathrm{BDD}_\alpha(\mathcal{L}, (\mathbf{B}\,\boldsymbol{x} - \boldsymbol{v})/2),$$

then with probability δ, whose expression is derived in the next paragraph, there exists $\boldsymbol{x} \in \mathbb{Z}_2^n$ such that $f(\boldsymbol{x})$ is the shortest nonzero lattice vector; we will then choose $N = 1/\delta$ so that repeating N times this process finds the shortest vector with probability bounded from below by a constant.

To that aim, we show that we can cover the sphere of radius λ_1 by N balls of radius $2\alpha\lambda_1$ whose centers are at distance $r(1-\frac{1}{n})d_k \leqslant r\lambda_1$ from the origin (see Figure 4.1 where we took $r = \alpha$). We have two concentric circles of radius $r(1-\frac{1}{n})d_k$ and λ_1, and let P be a uniformly random point on the surface of the ball of radius $r(1-\frac{1}{n})d_k$. A ball of radius $2\alpha\lambda_1$ at center P will cover the spherical cap with angle ϕ of the ball of radius λ_1. For convenience, write $r = s\lambda_1$ for some s. We can calculate the optimal choice of r by noting that if we take the center of the caps to be at distance r then the angle ϕ satisfies $\cos\phi = \frac{\lambda_1^2 + r^2 - 4\alpha^2\lambda_1^2}{2r\lambda_1} = f(s)$ by the law of cosines, where $f(x) = \frac{1+x^2-4\alpha^2}{2x}$. We want to maximize the angle ϕ, since the area we can cover increases with ϕ. For minimizing $\cos(\phi)$, we minimize f. Note however that we will apply Theorem 2.35 to enumerate all points inside a ball and this requires the center of the ball to be at distance at most $\alpha\lambda_1$ from the origin. Therefore we want to minimize f over $[0, \alpha]$. Check that $f(x)$ is decreasing until $\sqrt{1-4\alpha^2}$ and then increasing. We conclude that the optimal radius is when $s = \min(\alpha, \sqrt{1-4\alpha^2})$.

Now observe that the surface area of any such cap is lower bounded by the surface area of the base of the cap, which is a $(n-1)$-dimensional sphere of radius $\lambda_1 \sin\phi$. Hence the number of spherical caps required to cover the surface of sphere is in the order of $N := A_n(\lambda_1)/V_{n-1}(\lambda_1 \sin\phi)$ where A_n (resp. V_n) is the surface area (resp. volume) of a n-dimensional sphere:

$$A_n(\lambda_1) = \frac{2\pi^{n/2}\lambda_1^{n-1}}{\Gamma(n/2)}, \qquad V_{n-1}(\lambda_1 \sin\phi) = \frac{\pi^{(n-1)/2}\lambda_1^{n-1}\sin^{n-1}\phi}{\Gamma((n+1)/2)}. \tag{4.7}$$

Thus we have

$$N = \frac{A_n(\lambda_1)}{V_{n-1}(\lambda_1 \sin\phi)} = \frac{2^{o(n)}}{\sin^{n-1}\phi}.$$

If we randomly choose the center $\boldsymbol{v}$ of the sphere, the corresponding spherical caps will cover the shortest vector with probability at least $1/N$. By Theorem 2.35, the list $\{f(\boldsymbol{x}) \mid \boldsymbol{x} \in \mathbb{Z}_2^n\}$ will contain all lattice points within radius $2\alpha d_k$ from $\boldsymbol{v}$. Hence, the list will contain a shortest vector with probability $1/N$. By repeating this process N times, we can find the shortest vector with constant probability. Therefore, an upper bound of the total time complexity of our method can be expressed as

$$T_c + \frac{2^{n+o(n)}T_o}{\sin^n\phi}.$$

In the quantum case, we can apply the quantum minimum finding algorithm in Theorem 2.6 to speed up the spherical cap search algorithm: we can build a circuit to generate the random vectors $\boldsymbol{v_{ij}}$ above and therefore build a circuit that associates to every $(i, j, \boldsymbol{s})$ the lattice vector $f_{ij}(\boldsymbol{s})$. By a similar argument to that in proof of Theorem 4.8, we then apply the quantum

minimum finding algorithm on the set of $(i, j, \boldsymbol{s}) \in [n^2] \times [N] \times [2^n]$ and obtain the shortest vector of that list by making $\sqrt{n^2 N 2^n} = 2^{n/2+o(n)}\sqrt{N}$ queries to the BDD oracle. The same argument as above shows that the shortest nonzero vector of the lattice is in that list with constant probability. □

Corollary 4.10. *There is a randomized algorithm that solves SVP in time $2^{1.7397n+o(n)}$ and in space $2^{0.5n+o(n)}$ with constant probability.*

Proof. Apply Theorem 4.6 with[2] $\alpha = 0.4103$: since $0 \leqslant b = \log_2 \beta(\mathcal{L}) \leqslant 0.401$, we indeed have that $\alpha < \frac{1}{2}\sqrt{\frac{1}{1+b}}$ so we can create a $(\alpha + o(1))$-BDD in time $T_c = 2^{(A+1)n/2+o(n)}$, space $S_c = 2^{0.5n+o(n)}$ such that each call takes time $T_o = 2^{An/2+o(n)}$ where $A = A(b)$ is given by Theorem 4.6. The theorem also guarantees that $A(b)$ increases with b so $A \leqslant A(0.401)$. But $0.401 \geqslant \frac{1-4\alpha^2}{2\ln 2} \approx 0.2356$ so $A(0.401) = \frac{4\alpha^2}{1-4\alpha^2}0.401 \approx 0.8267$ by Theorem 4.6. Apply Theorem 4.9 to get a randomized algorithm that solves SVP in space S_c and in time

$$T := T_c + \frac{2^{n+o(n)}T_o}{\sin^n \phi}$$

where $\cos\phi = \frac{1+s^2-4\alpha^2}{2s}$ and $s = \min(\alpha, \sqrt{1-4\alpha^2})$. Therefore, $\frac{1}{\sin\phi} \approx 1.2537$ and

$$T = 2^{(A+1)n/2+o(n)} + 2^{n+An/2-n\log_2 \sin\phi+o(n)} = 2^{1.7397n+o(n)}.$$ □

4.5 Dependency of the SVP on a quantity related to the kissing number

In the previous sections, we obtained several algorithms for SVP and bounded their complexity using the only known bound on the quantity $\beta(\mathcal{L})$, which is related to the lattice kissing number (see Section 2.2): $\beta(\mathcal{L}) \leqslant 2^{0.402}$. The complexity of those algorithms is highly affected by this quantity and since $\beta(\mathcal{L})$ can be anywhere between 0 and $2^{0.402}$ (see Section 2.4), we will study the dependence of the time complexity in $\beta(\mathcal{L})$. Recall that $b = \log_2 \beta(\mathcal{L})$.

In order to avoid doing the analysis twice, we introduce a factor ν that is 1 for classical algorithms and $\frac{1}{2}$ for quantum algorithms. We now can reformulate the time complexity in Theorem 4.9 as

$$T_c + \frac{2^{\nu n+o(n)}T_o}{\sin^{\nu n} \phi} \tag{4.8}$$

where $\cos\phi = \frac{1+s^2-4\alpha^2}{2s}$ and $s = \min(\alpha, \sqrt{1-4\alpha^2})$. We instantiate the algorithm in Theorem 4.9 with α-BDD provided by Theorem 4.6 which satisfies

$$T_c = 2^{(A+1)/n+o(n)}, \qquad T_o = 2^{An/2+o(n)} \quad \text{and} \quad S_c = 2^{0.5n+o(n)}$$

where

$$A = \begin{cases} -\frac{1}{2\ln 2}W(-\frac{4\alpha^2\beta(\mathcal{L})^2}{e}) & \text{when } b < \frac{1-4\alpha^2}{2\ln 2} \\ \frac{4\alpha^2}{1-4\alpha^2}b & \text{when } b \geqslant \frac{1-4\alpha^2}{2\ln 2} \end{cases}.$$

[2] The optimal value of α was found numerically, see Section 4.5.

Since $\nu \in \{\frac{1}{2}, 1\}$, we can simplify the expression of the time complexity to

$$\frac{2^{(A/2+\nu)n+o(n)}}{\sin^{\nu n}\phi}. \tag{4.9}$$

However, the optimal choice of α is not obvious: by increasing the decoding radius, we reduce the number of queries but increase the cost of each queries. Since there seems to be no closed-form expression for the optimal value of α, we express as an optimization program. Formally, we have $T = 2^{c_1(b,\nu)n+o(n)}$ where

$$c(b,\nu) = \min_{\alpha\in[\frac{1}{3},\frac{1}{2})} \frac{A}{2} + \nu - \frac{\nu}{2}\log_2(1-\cos^2\phi)$$

where A and $\cos\phi$ are given by the expressions above that depend on α. We numerically computed the graph of this function and plotted the result on Figures 4.2 and 4.3 for the classical and quantum algorithms respectively.

For completeness, we can also consider a classical version of our quantum algorithm in Section 4.3 and we then obtain algorithm in time $2^{c_3(\nu,b)n+o(n)}$ where

$$c'(\nu,b) = \nu\log_2 3 + \frac{1}{2}\begin{cases} -\frac{1}{2\ln 2}W(-\frac{4\beta(\mathcal{L})^2}{9e}) & \text{when } b < \frac{5}{18\ln 2} \\ \frac{4}{5}b & \text{when } b \geqslant \frac{5}{18\ln 2} \end{cases}.$$

We also plotted the resulting graph on Figure 4.2 and Figure 4.3 to compare against the spherical caps algorithms. In particular, we obtain the following result when $\gamma(\mathcal{L}) = \beta(\mathcal{L})^n$ is subexponential in n:

Theorem 4.11. *For any family $(\mathcal{L}_n)_n$ of full-rank lattices such that $\mathcal{L}_n \subseteq \mathbb{R}^n$ and $\beta(\mathcal{L}_n)^n = 2^{o(n)}$, there is a classical (resp. quantum) algorithm that solves the SVP on $\mathcal{L}_n$ in time $2^{1.292n+o(n)}$ (resp. $2^{0.750n+o(n)}$), in space $2^{0.5n}$ (plus $\mathrm{poly}(n)$ qubits in the quantum case).*

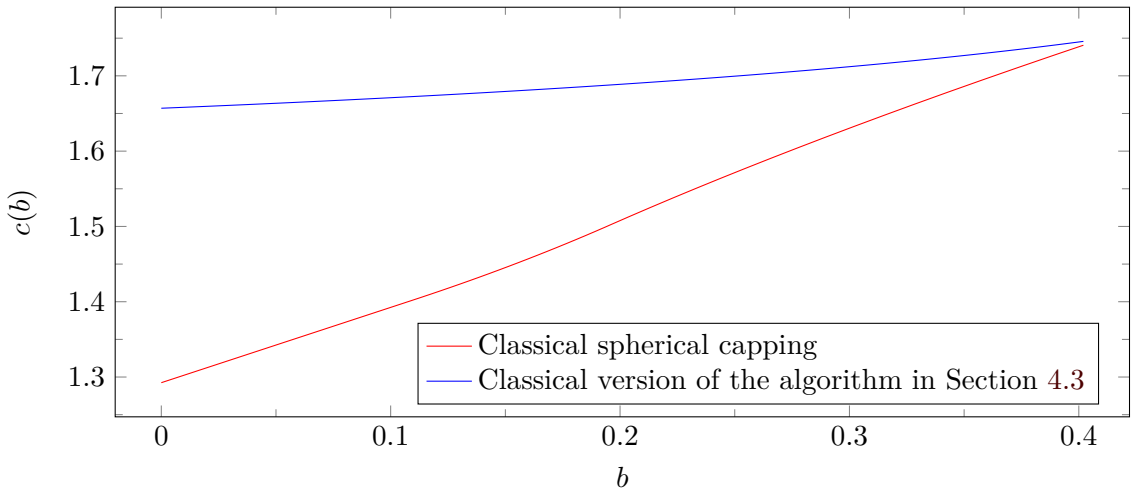


Figure 4.2: (Exponent $c(b)$ of the) time complexity of the classical version of the quantum algorithm in Section 4.3 and the spherical capping algorithm, plotted against $b = \log_2 \beta(\mathcal{L})$. The complexity of the algorithms is $2^{c(b)n+o(n)}$.

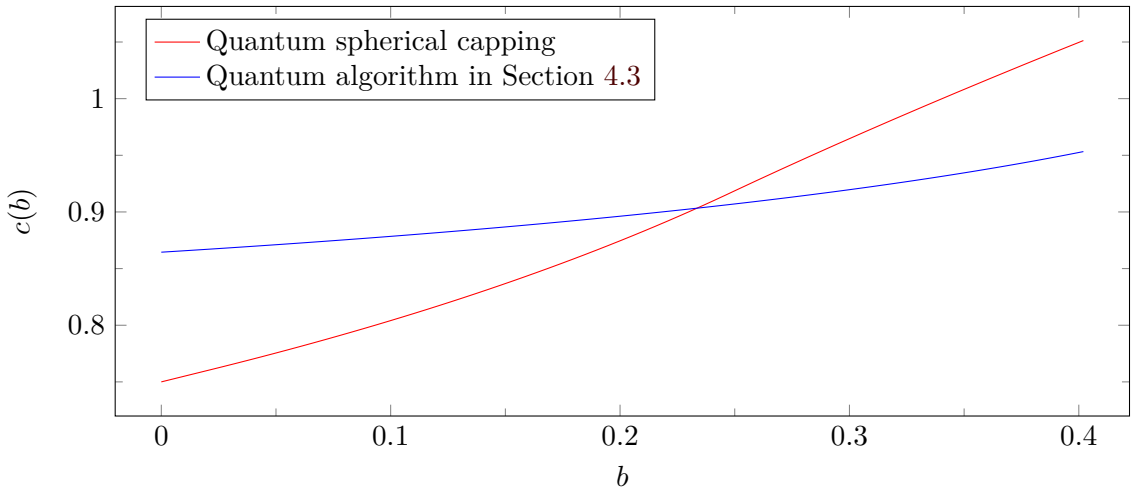


Figure 4.3: (Exponent $c(b)$ of the) time complexity of the quantum algorithm in Section 4.3 and the quantum spherical capping algorithm, plotted against $b = \log_2 \beta(\mathcal{L})$. The complexity of the algorithms is $2^{c(b)n+o(n)}$.

Chapter 5
Enumeration Algorithms for the Shortest Vector Problem

The work in this chapter has been published at ASIACRYPT 2018 [ANS18] and is a joint work with Yoshinori Aono and Phong Nguyen.

5.1 Introduction

Enumeration is the simplest algorithm to solve SVP/CVP: it outputs $\mathcal{L} \cap B$, given a lattice $\mathcal{L}$ and an n-dimensional ball $B \subseteq \mathbb{R}^n$. Dating back to the early 1980s [Poh81, Kan83], it has been significantly improved in practice in the past twenty years, thanks to pruning methods introduced by Schnorr *et al.* [SE94b, SH95, Sch03], and later revisited and generalized as cylinder pruning [GNR10] and discrete pruning [AN17]: these methods offer a trade-off by enumerating over a subset $S \subseteq B$, at the expense of missing solutions. In cylinder pruning [GNR10], S is the intersection of n cylinders. In discrete pruning [AN17], S is the intersection of B with the union P of many disjoint boxes.

One may only be interested in finding one point in $\mathcal{L} \cap S$ (provided it exists), or the 'best' point in $\mathcal{L} \cap S$, *i.e.* a point minimizing the distance to a target (zero for SVP or given as input for CVP). Enumeration and cylinder pruning compute $\mathcal{L} \cap S$ by a depth-first search of a tree with super-exponentially many nodes. Discrete pruning is different, but the computation of S also relies on a depth-first search in a tree.

Discrete Pruning. This technique relies on the selection of finite number of disjoint boxes that define a set P which is called a pruning set. One then enumerates points in $\mathcal{L} \cap B \cap P$. The particular choice of P is very important for the complexity of this method. It is common to define P from a partition $\mathcal{C}$: given a set of *tags* $T \subseteq \mathbb{Z}^n$, each tag $\boldsymbol{t} \in T$ defines a set $\mathcal{C}(\boldsymbol{t})$ that contains exactly one lattice point (that is easy to compute from $\boldsymbol{t}$). We then select a finite subset $U \subset T$ of tags that define $P = \bigcup_{\boldsymbol{t} \in U} \mathcal{C}(\boldsymbol{t})$. Babai's partition and the natural partition are the most natural choices for $\mathcal{C}$ because they define what is called *universal* partition, a property that makes them suitable to solve CVP using this technique. Once the partition has been fixed, it remains to choose U in such way to maximize the success probability of the algorithm. A typical strategy is to fix an integer m, quantifying the effort one is willing to spend on the method, and to search for the best m boxes, where the "quality" of each box is quantified by a function $g(\boldsymbol{t})$ that evaluates its success probability. Since that cost can be difficult to compute, it is sometimes replaced by simpler functions that have experimentally been confirmed to corrolate well with the exact cost. To the best of our knowledge, no formal analysis on how to choose m has been conducted. Hence, in practice, the entire complexity

of the algorithm is expressed as a function of m, which is taken as big as possible given the available computing resources.

Quadratic Quantum Speed-up. Enumeration algorithms can naturally be viewed as doing a depth-first search on a (super-exponential in n) tree. Some leaves are marked when they contain a lattice point and the goal is either to find one marked node or all of them. Note that the tree is usually only defined locally: given a node, the algorithm can determine its children. Given such an algorithm, we can apply quantum algorithms such as Montanaro's [Mon15] to find a marked one in time proportional to $\sqrt{t}$ where t is the size of the tree. Since a classical algorithm might have to search entire tree, this can provide a quantum quadratic speed-up. In reality, one cannot just apply naively Montanaro's algorithm because some important technical details such as the degree of the tree and estimates on size of the tree need to be examined first. Furthermore, in the case of extreme pruning [GNR10], the enumeration process usually only succeeds with small probability and will be repeated a large number of times. A naive approach would be to repeat the process k times to obtain a $k\sqrt{t}$ complexity, but we will see that we can obtain a much better speedup.

Contributions. We show that lattice enumeration and its cylinder and discrete pruning variants can all be quadratically sped up on a quantum computer. Interestingly, we show that this speedup also applies to extreme pruning where one repeats enumeration over many reduced bases: a naive approach would only decrease to $k\sqrt{t}$ quantum operations, but we bring it down to $\sqrt{kt}$. In doing so, we also clarify the application of Montanaro's algorithm [Mon15] to enumeration with cylinder pruning, which was incomplete. We also develop two tweaks to discrete pruning. The first one enables to solve CVP and BDD (and in particular LWE-type instances). The second deals with the selection of optimal discrete pruning parameters, and is crucial for our quantum variant: We design a provably efficient (and even optimal in some cases) algorithm to find the best m tags when $g(\boldsymbol{t})$ satisfies certain conditions. Our theoretical analysis has also been validated by experiments. A detailed description of our contributions is available in Section 1.1.3.

5.2 Enumeration Algorithms and Pruning

Let $\mathcal{L}$ be a full-rank lattice in $\mathbb{R}^n$. Given a target $\mathbf{u} \in \mathbb{Q}^n$, a basis $B = (\mathbf{b}_1, \ldots, \mathbf{b}_n)$ of $\mathcal{L}$ and a radius $R > 0$, enumeration [Poh81, Kan83] outputs $\mathcal{L} \cap S$ where $S = B_n(\mathbf{u}, R)$: by comparing all the distances to $\mathbf{u}$, one extracts a lattice vector closest to $\mathbf{u}$. The enumeration requires to generalize the problem slightly as follows: given a (shifted) lattice $\boldsymbol{u} + \mathcal{L}$ and a ball S, output $(\boldsymbol{u} + \mathcal{L}) \cap S$. The algorithm performs a recursive search using projections, to reduce the dimension of the lattice. First we compute the projection on the last coordinate of the Gram-Schmidt basis: $\pi_n(\boldsymbol{u} + \mathcal{L})$ is an one-dimensional shifted lattice and $\pi_n(S)$ is an one-dimensional ball, and we can easily enumerate $\pi_n(\boldsymbol{u} + \mathcal{L}) \cap \pi_n(S)$. Then, for each $\boldsymbol{z} \in \pi_n((\boldsymbol{u} + \mathcal{L}) \cap S)$, we consider the hyperplane $H_z = \{\boldsymbol{x} \in \mathbb{R}^n : \pi_n(\boldsymbol{x}) = z\}$ and we observe that $(\boldsymbol{u} + \mathcal{L}) \cap H_z$ is a $(n-1)-$dimensional shifted lattice and $S \cap H_z$ is a $(n-1)-$dimensional ball, so it suffices to recursively enumerate the intersection. Concretely, it can be viewed as a depth-first search of the enumeration tree $\mathcal{T}$: the nodes at depth $n + 1 - k$ are the points of $\pi_k(\mathcal{L}) \cap S$. Figure 5.1 illustrates this process on an example.

The running-time of enumeration depends on R and B, but is typically super-exponential in n, even if $\mathcal{L} \cap S$ is small. Indeed, at the i^{th} recursion step the one-dimensional intersection will yield $1 + \lfloor 2r/\|\boldsymbol{b}_i^\star\| \rfloor$ points, where r is the radius of the ball (which depends on the previous choices). Since the radius of the ball only decreases during the recursion, we have $r \leqslant R$, so in particular, the tree will have at most $\prod_{i=1}^{n}(1 + \lfloor 2R/\|\boldsymbol{b}_i^\star\| \rfloor)$ nodes. A rigorous bound on the number of nodes exists. For instance, if the basis is LLL-reduced, and $R = \|\boldsymbol{b}_1\|$, then it is well-known that the number of nodes is at most $2^{O(n^2)}$. Also [HS07] showed that if the basis is so-called quasi-HKZ-reduced, and $R = \|\boldsymbol{b}_1\|$, then the number of nodes is at most $n^{n/2e+o(n)}$.

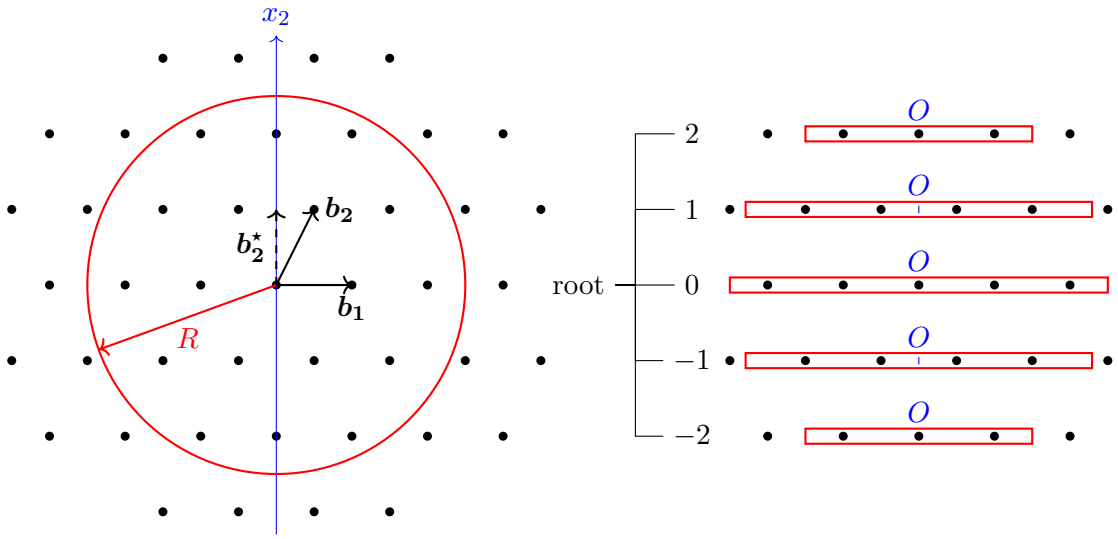


Figure 5.1: Example of enumeration in a lattice spanned by $\boldsymbol{b}_1, \boldsymbol{b}_2$ with a ball of radius $R = 5$, and the corresponding enumeration tree $\mathcal{T}$. The first level of the tree corresponds to the possible choices of $x_2 = \pi_2(\boldsymbol{v})$, the projection on $\mathbb{R}\boldsymbol{b}_2^\star$, for any $\boldsymbol{v} \in \mathcal{L} \cap B_2(R)$. The enumeration algorithm then proceeds recursively: for each choice z of x_2, one can consider the intersection of $\mathcal{L}$ with the hyperplane $x_2 = z$: this is a (possibly shifted) one-dimensional lattice. The intersection of the ball of radius R with this hyperplane also produces a one-dimensional ball of smaller radius (which depends on z). Hence for each x_2, we have a new one-dimensional enumeration sub-problem on a shifted lattice. The second level of the tree corresponds to the enumerated lattice points in each new sub-problem.

5.2.1 Pruned Enumeration

Pruned enumeration, introduced by [SE94b, SH95] and formalized by [GNR10, AN17], uses a pruning set $P \subseteq \mathbb{R}^n$ and outputs $\mathcal{L} \cap (\mathbf{u} + P)$. The advantage is that for suitable choices of P, enumerating $\mathcal{L} \cap (\mathbf{u} + P)$ is much cheaper than $\mathcal{L} \cap S$, and if we further intersect $\mathcal{L} \cap (\mathbf{u} + P)$ with S, we may have found non-trivial points of $\mathcal{L} \cap S$. Note that we use $\mathbf{u} + P$ rather than P, because it is natural to make P independent of $\mathbf{u}$, and it is what happens when one uses the pruning of [GNR10] to search for close vectors. Following [GNR10], we view the pruning set P as a random variable: it depends on the choice of basis B. When the pruned enumeration

fails, we can simply repeat the process with many different P's until we solve the problem. We distinguish two cases, which were considered separately in [AN17, GNR10]:

Approximation setting: Here, we are interested in finding any point in $\mathcal{L} \cap S \cap (\mathbf{u} + P)$ by enumerating $\mathcal{L} \cap (\mathbf{u} + P)$ then intersecting it with the ball S, so we define the success probability as:

$$\Pr_{\text{succ}} = \Pr_{P,\mathbf{u}}(\mathcal{L} \cap S \cap (\mathbf{u} + P) \neq \varnothing), \tag{5.1}$$

which is the probability that it outputs at least one point in $\mathcal{L} \cap S$. By (slightly) adapting the reasoning of [AN17] based on the Gaussian heuristic, we estimate that (5.1) is heuristically

$$\Pr_{\text{succ}} \approx \min(1, \text{vol}(S \cap (\mathbf{u} + P))/\text{covol}(\mathcal{L})), \tag{5.2}$$

and that the number of elements of $\mathcal{L} \cap S \cap (\mathbf{u} + P)$ is roughly $\text{vol}(S \cap (\mathbf{u} + P))/\text{covol}(\mathcal{L})$.

Unique setting: Here, we know that the target $\mathbf{u}$ is unusually close to the lattice, that is $\mathcal{L} \cap S$ is a singleton, and we want to find the closest lattice point to $\mathbf{u}$: this is the so-called *Bounded Distance Decoding* problem (BDD), whose hardness is used in most lattice-based encryption schemes. Thus, $\mathbf{u}$ is of the form $\mathbf{u} = \mathbf{v} + \mathbf{e}$ where $\mathbf{v} \in \mathcal{L}$ and $\mathbf{e} \in \mathbb{R}^n$ is very short, and we want to recover $\mathbf{v}$. [GNR10] studied the exact SVP case, where one wants to recover a shortest lattice vector (in our setting, if the target $\mathbf{u} \in \mathcal{L}$, the BDD solution would be $\mathbf{u}$, but one could alternatively ask for the closest distinct lattice point, which can be reduced to finding a shortest lattice vector). We are only interested in finding the closest lattice point $\mathbf{v} \in \mathcal{L}$, so we define the success probability as:

$$\Pr_{\text{succ}} = \Pr_{P,\mathbf{u}}(\mathbf{v} \in \mathcal{L} \cap (\mathbf{u} + P)), \tag{5.3}$$

because we are considering the probability that the solution $\mathbf{v}$ belongs to the enumerated set $\mathcal{L} \cap (\mathbf{u} + P)$. Usually, the target $\mathbf{u}$ is derived from the noise $\mathbf{e}$, which has a known distribution, then we can rewrite (5.3) as:

$$\Pr_{\text{succ}} = \Pr_{P,\mathbf{e}}(0 \in \mathbf{e} + P) = \Pr_{P,\mathbf{e}}(-\mathbf{e} \in P). \tag{5.4}$$

In the context of SVP, we would instead define $\Pr_{\text{succ}} = \Pr_P(\mathbf{v} \in P)$ where $\mathbf{v}$ is a shortest lattice vector. In general, it is always possible to make $\mathbf{u}$ depend solely on $\mathbf{e}$: one can take a canonical basis of $\mathcal{L}$, like the HNF, and use it to reduce $\mathbf{u}$ modulo $\mathcal{L}$, which only depends on $\mathbf{e}$. Whether $\Pr_{P,\mathbf{e}}(-\mathbf{e} \in P)$ can be evaluated depends on the choice of P and the distribution of the noise $\mathbf{e}$. For instance, if the distribution of $-\mathbf{e}$ is uniform over some measurable set E, then:

$$\Pr_{P,\mathbf{e}}(-\mathbf{e} \in P) = \frac{\text{vol}(E \cap P)}{\text{vol}(E)}.$$

We analyze the most important noise distributions in Section 5.2.6.

We have discussed ways to estimate the success probability of pruned enumeration. In summary, to estimate the running time of the full algorithm, we need the following information, which depends on the choice of pruning:

- an estimate of the cost of enumerating $\mathcal{L} \cap S \cap (\mathbf{u} + P)$,
- an estimate of the cost of computing the (random) reduced basis B,
- in the approximate setting: an estimate of $\mathrm{vol}(S \cap (\mathbf{u} + P))$,
- in the unique setting: an estimate of $\mathrm{vol}(E \cap P)$.

5.2.2 Cylinder Pruning

The first pruning set P ever used is the following generalization [GNR10] of [SE94b, SH95]. Fix a *bounding function* $f : \{1, \ldots, n\} \to [0, 1]$, a radius $R > 0$ a lattice basis $B = (\mathbf{b}_1, \ldots, \mathbf{b}_n)$ and define

$$P_f(B, R) = \{\mathbf{x} \in \mathbb{R}^n \text{ s.t. } \|\pi_{n+1-i}(\mathbf{x} - \mathbf{u})\| \leqslant f(i)R \text{ for all } 1 \leqslant i \leqslant n\}, \tag{5.5}$$

where the π_i's are the Gram-Schmidt projections defined by B. We call *cylinder pruning* this form of enumeration, because $P_f(B, R)$ is an intersection of cylinders: each inequality $\|\pi_{n+1-i}(\mathbf{x} - \mathbf{u})\| \leqslant f(i)R$ defines a cylinder.

Gama *et al.* [GNR10] showed how to efficiently compute tight lower and upper bounds for $\mathrm{vol}(P_f(B, R))$, thanks to the Dirichlet distribution and special integrals. Then we can do the same for $\mathrm{vol}(P_f(B, R) \cap S)$ if S is any zero-centered ball. Using the shape of $P_f(B, R)$, [GNR10] also estimated of the cost of enumerating $\mathcal{L} \cap S \cap P_f(B, R)$, using the Gaussian heuristic on projected lattices $\pi_i(\mathcal{L})$: these estimates are usually accurate in practice, and they can also be used in the CVP case [LN13]. To optimize the whole selection of parameters, one finally needs to take into account the cost of computing the (random) reduced basis B: for instance, this is done in [CN11, AWHT16]. In Section 5.3, we show a quantum quadratic speed-up of cylinder pruning.

5.2.3 Discrete Pruning

Discrete pruning is based on lattice partitions defined as follows. An $\mathcal{L}$-partition is a partition $\mathcal{C}$ of $\mathbb{R}^n$ such that:

- The partition is countable: $\mathbb{R}^n = \bigcup_{t \in T} \mathcal{C}(\boldsymbol{t})$ where T is a countable set, and $\mathcal{C}(\boldsymbol{t}) \cap \mathcal{C}(\boldsymbol{t}') = \varnothing$ whenever $\boldsymbol{t} \neq \boldsymbol{t}'$.
- Each cell $\mathcal{C}(\boldsymbol{t})$ contains a single lattice point, which can be found efficiently: given any $\boldsymbol{t} \in T$, one can "open" the cell $\mathcal{C}(\boldsymbol{t})$, *i.e.* compute $\mathcal{C}(\boldsymbol{t}) \cap \mathcal{L}$ in polynomial time. In other words, the partition defines a function $g : T \to \mathcal{L}$ where $\mathcal{C}(\boldsymbol{t}) \cap \mathcal{L} = \{g(\boldsymbol{t})\}$, and one can compute g in polynomial time.

Two useful $\mathcal{L}$-partitions, illustrated in Figure 5.2 and defined below, were presented in [AN17]: Babai's partition where $T = \mathbb{Z}^n$ and each cell $\mathcal{C}(\boldsymbol{t})$ is a box of volume $\mathrm{covol}(\mathcal{L})$; and the natural partition where $T = \mathbb{N}^n$ and each cell $\mathcal{C}(\boldsymbol{t})$ is a union of non-overlapping boxes, with total volume $\mathrm{covol}(\mathcal{L})$.

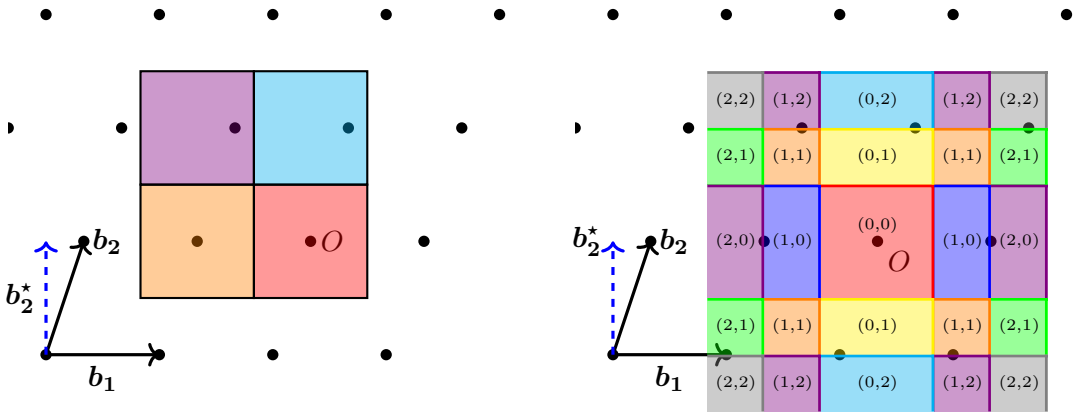


Figure 5.2: Example lattice with its corresponding Babai's and natural partition. The different cells have different colors and are tagged.

Universal Lattice Partitions. While sufficient for the SVP, $\mathcal{L}$-partitions are not sufficient in general for our framework: if $P = \bigcup_{t\in U} \mathcal{C}(\boldsymbol{t})$, then $\mathcal{L} \cap (P + \mathbf{u}) = \bigcup_{t\in U}(\mathcal{L} \cap (\mathcal{C}(\boldsymbol{t}) + \mathbf{u}))$ but the number of elements in $\mathcal{L} \cap (\mathcal{C}(\boldsymbol{t}) + \mathbf{u})$ is unclear, and it is also unclear how to compute $\mathcal{L} \cap (\mathcal{C}(\boldsymbol{t}) + \mathbf{u})$ efficiently. To fix this, we could compute instead $\mathcal{L} \cap P \cap S = \bigcup_{t\in U}(\mathcal{L} \cap \mathcal{C}(\boldsymbol{t})) \cap S$, but that creates two issues:

- In the unique setting, it is unclear how we would evaluate success probabilities. Given a tag $\boldsymbol{t}$ and a target $\mathbf{u} = \mathbf{v} + \mathbf{e}$ where $\mathbf{e}$ is the noise and $\mathbf{v} \in \mathcal{L}$, we would need to estimate the probability that $\mathbf{v} \in \mathcal{C}(\boldsymbol{t})$, *i.e.* $\mathbf{u} - \mathbf{e} \in \mathcal{C}(\boldsymbol{t})$.
- We would need to select the tag set U depending on the target $\mathbf{u}$, without knowing how to evaluate success probabilities.

The most natural solution is to subtract $\mathbf{u}$ to the lattice $\mathcal{L}$ as follows. We say that an $\mathcal{L}$-partition $\mathcal{C}$ is *universal* if for all $\mathbf{u} \in \mathbb{Q}^n$, the shifted partition $\mathcal{C} + \mathbf{u}$ is an $\mathcal{L}$-partition. In particular, this means that for any $\mathbf{u} \in \mathbb{Q}^n$, each cell $\mathcal{C}(\boldsymbol{t})$ contains a single point in $\mathcal{L} - \mathbf{u} = \{\mathbf{v} - \mathbf{u}, \mathbf{v} \in \mathcal{L}\}$, which can be found efficiently: given any $\boldsymbol{t} \in T$ and $\mathbf{u} \in \mathbb{Q}^n$, one can "open" the cell $\mathbf{u} + \mathcal{C}(\boldsymbol{t})$, *i.e.* compute $(\mathbf{u} + \mathcal{C}(\boldsymbol{t})) \cap \mathcal{L}$ in polynomial time.

Unfortunately, an $\mathcal{L}$-partition is not necessarily universal, even in dimension one. Indeed, consider the $\mathcal{L}$-partition $\mathcal{C}$ with $T = \mathbb{Z}$ defined as follows: $\mathcal{C}(0) = [-1/2, 1/2]$; For any $k > 0$, $\mathcal{C}(k) = (k - 1/2, k + 1/2]$; For any $k < 0$, $\mathcal{C}(k) = [k - 1/2, k + 1/2)$. It can be checked that $\mathcal{C}$ is not universal: the shifted cell $\mathcal{C}(0) + 1/2$ contains two lattice points, namely 0 and 1. Fortunately, we will show Babai's partition and the natural partition are actually universal, see Section 5.2.7 for the proofs.

Lemma 5.1 (Babai's partition)**.** *Let B be a basis of a full-rank lattice $\mathcal{L}$ in $\mathbb{Z}^n$. Let $T = \mathbb{Z}^n$ and for any $\mathbf{t} \in T$, $\mathcal{C}_{\mathbb{Z}}(\mathbf{t}) = \mathbf{t}B^\star + \mathcal{D}$ where $\mathcal{D} = \{\sum_{i=1}^n x_i \mathbf{b}_i^\star \text{ s.t. } -1/2 \leqslant x_i < 1/2\}$. Then Babai's partition $(\mathcal{C}_{\mathbb{Z}}, T)$ is universal.*

Lemma 5.2 (Natural partition)**.** *Let B be a basis of a full-rank lattice $\mathcal{L}$ in $\mathbb{Z}^n$. Let $T = \mathbb{N}^n$ and for any $\mathbf{t} = (t_1, \ldots, t_n) \in T$, $\mathcal{C}_{\mathbb{N}}(\mathbf{t}) = \{\sum_{i=1}^n x_i \mathbf{b}_i^\star \text{ s.t. } -(t_i+1)/2 < x_i \leqslant -t_i/2 \text{ or } t_i/2 < x_i \leqslant (t_i+1)/2\}$. Then the natural partition $(\mathcal{C}_{\mathbb{N}}, T)$ is universal.*

Discrete Pruning. Once a universal partition $(\mathcal{C}, T)$ has been selected, discrete pruning is obtained by selecting the pruning set P as the union of finitely many cells $\mathcal{C}(\boldsymbol{t})$, namely $P = \bigcup_{t \in U} \mathcal{C}(\boldsymbol{t})$ for some finite $U \subseteq T$. Then $\mathcal{L} \cap (\boldsymbol{u} + P) = \bigcup_{t \in U} (\mathcal{L} \cap (\boldsymbol{u} + \mathcal{C}(\boldsymbol{t})))$ can be enumerated by opening each cell $\boldsymbol{u} + \mathcal{C}(\boldsymbol{t})$ for $t \in U$: see Algorithm 5.1.

In theory one would like to select the cells $\mathcal{C}(\boldsymbol{t})$ which maximize $\mathrm{vol}(\mathcal{C}(\boldsymbol{t}) \cap S)$: [AN17] shows how to compute $\mathrm{vol}(\mathcal{C}(\boldsymbol{t}) \cap S)$, but an exhaustive search to derive the best $\mathrm{vol}(\mathcal{C}(\boldsymbol{t}) \cap S)$ exactly would be too expensive. We discuss the selection of tags in Section 5.2.5.

Algorithm 5.1 Closest Vector Discrete Pruning from Universal Lattice Partitions

Input: A target vector $\mathbf{u} \in \mathbb{Q}^n$, a universal lattice partition $(\mathcal{C}, T)$, a finite subset $U \subseteq T$ and if we are in the approximation setting, a radius R.

Output: $\mathcal{L} \cap (\mathbf{u} + (S \cap P))$ where $S = B_n(R)$ and $P = \bigcup_{t \in U} \mathcal{C}(t)$.

1: $\mathcal{R} = \varnothing$
2: **for** $t \in U$ **do**
3: Compute $\mathcal{L} \cap (\mathbf{u} + \mathcal{C}(t))$ by opening $\mathbf{u} + \mathcal{C}(t)$: in the approx setting, check if the output vector is within distance $\leqslant R$ to $\mathbf{u}$, then add the vector to the set $\mathcal{R}$. In the unique setting, check if the output vector is the solution.
4: **end for**
5: Return $\mathcal{R}$.

5.2.4 Success Probability

In Section 5.2.1, we saw that the probability of success needs to be evaluated to understand the complexity the pruned enumeration. In the case of discrete pruning, this probability depends on the selection of tags. In this section, we consider a general framework under which we can optimize the selection of tags for discrete pruning. We distinguish two cases:

Approximation setting: based on (5.2), the success probability can be derived from

$$\mathrm{vol}(S \cap (\mathbf{u} + P)) = \sum_{\mathbf{t} \in U} \mathrm{vol}(B_n(R) \cap \mathcal{C}(\mathbf{t})). \tag{5.6}$$

This is exactly the same situation as in the SVP case already tackled by [AN17]. They showed how to compute $\mathrm{vol}(B_n(R) \cap \mathcal{C}(\mathbf{t}))$ for Babai's partition and the natural partition by focusing on the intersection of a ball with a box $H = \{(x_1, \ldots, x_n) \in \mathbb{R}^n \text{ s.t. } \alpha_i \leqslant x_i \leqslant \beta_i\}$: In the case of Babai's partition, each cell $\mathcal{C}_{\mathbb{Z}}(\mathbf{t})$ is a box. In the case of the natural partition, each cell $\mathcal{C}_{\mathbb{N}}(\mathbf{t})$ is the union of 2^j symmetric (non-overlapping) boxes, where j is the number of non-zero coefficients of $\mathbf{t}$. It follows that

$\mathrm{vol}(\mathcal{C}_{\mathbb{N}}(\mathbf{t}) \cap B_n(\mathbb{R})) = 2^j \mathrm{vol}(H \cap S)$, where H is any of these 2^j boxes. They also showed how to approximate a sum $\sum_{\mathbf{t}\in U} \mathrm{vol}(B_n(R) \cap \mathcal{C}(\mathbf{t}))$ in practice, without having to compute separately each volume.

Unique setting: Based on (5.4), if the noise vector is $\mathbf{e}$, then the success probability is

$$\Pr_{\text{succ}} = \Pr_{P,\mathbf{e}}(-\mathbf{e} \in P) = \sum_{\mathbf{t}\in U} \Pr_{P,\mathbf{e}}(-\mathbf{e} \in \mathcal{C}(\mathbf{t})) \tag{5.7}$$

It therefore suffices to compute the cell probability $\Pr_{P,\mathbf{e}}(\mathbf{e} \in \mathcal{C}(\mathbf{t}))$, instead of an intersection volume. Similarly to the approximation setting, we might be able to approximate the sum $\sum_{\mathbf{t}\in U} \Pr_{P,\mathbf{e}}(\mathbf{e} \in \mathcal{C}(\mathbf{t}))$ without having to compute individually each probability. In Section 5.2.6, we focus on the natural partition: we discuss ways to compute the cell probability $\Pr_{P,\mathbf{e}}(\mathbf{e} \in \mathcal{C}(\mathbf{t}))$ depending on the distribution of the noise $\mathbf{e}$.

In both cases, we see that the success probability is of the form:

$$\Pr_{\text{succ}} = \sum_{\mathbf{t}\in U} f(\mathbf{t}), \tag{5.8}$$

for some function $f : T \to [0, 1]$ such that $\sum_{\mathbf{t}\in T} f(\mathbf{t}) = 1$, where (5.8) is rigorous for the unique setting, and heuristic for the approximation setting due to the Gaussian heuristic. If ever the computation of f is too slow to compute individually each term of $\sum_{\mathbf{t}\in U} f(\mathbf{t})$, we can use the statistical inference techniques of [AN17] to approximate (5.8) from the computation of a small number of $f(\mathbf{t})$. Note that if we know that the probability is reasonably large, say > 0.01, we can alternatively use Monte-Carlo sampling to approximate it.

5.2.5 Selecting Tags

We would like to select the finite set U of tags to maximize $\Pr_{\text{succ}}$ given by (5.8). Let us assume that we have a function $f : T \to \mathbb{R}^+$ such that $\sum_{\mathbf{t}\in T} f(\boldsymbol{t})$ converges. If (5.8) provably holds, then $\sum_{\mathbf{t}\in T} f(t) = 1$, so the sum indeed converges. Since T is infinite, this implies that for any $B > 0$, the set $\{\mathbf{t} \in T \text{ s.t. } f(\mathbf{t}) > B\}$ is finite, which proves the following elementary result:

Lemma 5.3. *Let T be an infinite countable set. Let $f : T \to \mathbb{R}^+$ be a function such that $\sum_{\mathbf{t}\in T} f(\boldsymbol{t})$ converges. Then for any integer $m > 0$, there is a finite subset $U \subseteq T$ of cardinal m such that $f(\mathbf{t}) \leqslant \min_{\mathbf{u}\in U} f(\mathbf{u})$ for all $\mathbf{t} \in T\backslash U$. Such a subset U maximizes $\sum_{\mathbf{u}\in U} f(\mathbf{u})$ among all m-size subsets of T.*

Any such subset U would maximize $\Pr_{\text{succ}}$ among all m-size subsets of T, so we would ideally want to select such a U for any given m. And m quantifies the effort we want to spend on discrete pruning, since the bit-complexity of discrete pruning is exactly m poly-time operations.

Now that we know that optimal subsets U exist, we discuss how to find such subsets U efficiently. In the approximation setting of [AN17], the actual function f is related to volumes: we want to select the k cells which maximize $\mathrm{vol}(B_n(R) \cap \mathcal{C}(\mathbf{t}))$ among all the cells. This is too expensive to do exactly, but [AN17] provides a fast heuristic method for the natural

partition, by selecting the cells $\mathcal{C}(t)$ minimizing $\mathbb{E}\{\mathcal{C}_{\mathbb{N}}(\mathbf{t})\}$: given as input m, it is possible to compute efficiently in practice the tags of the m cells which minimize

$$\mathbb{E}\{\mathcal{C}_{\mathbb{N}}(\mathbf{t})\} = \sum_{i=1}^{n}\left(\frac{t_i^2}{4} + \frac{t_i}{4} + \frac{1}{12}\right)\|\mathbf{b}_i^\star\|^2.$$

In other words, this is the same as replacing the function f related to volumes by the function

$$h(\mathbf{t}) = e^{-\sum_{i=1}^{n}\left(\frac{t_i^2}{4}+\frac{t_i}{4}+\frac{1}{12}\right)\|\mathbf{b}_i^\star\|^2},$$

and it can be verified that $\sum_{\mathbf{t}\in\mathbb{N}^n} h(\mathbf{t})$ converges. In practice (see [AN17]), the m cells maximizing $h(\mathbf{t})$ (*i.e.* minimizing $\mathbb{E}\{\mathcal{C}_{\mathbb{N}}(\mathbf{t})\}$) are almost the same as the cells maximizing $\mathrm{vol}(B_n(R) \cap \mathcal{C}(\mathbf{t}))$. However, the method of [AN17] was only heuristic. In Section 5.4, we modify that method to make it fully provable: for any integer $m > 0$, we can provably find the best m cells in essentially m polynomial-time operations and polynomial space (the m solutions are output as a stream).

5.2.6 Noise Distributions in the Unique Setting

We discuss how to evaluate the success probability of BDD discrete pruning in the unique setting for the natural partition. This can easily be adapted to Babai's partition, because it also relies on boxes. Following (5.7), it suffices to evaluate:

$$p(\mathbf{t}) = \Pr_{P,\mathbf{e}}(\mathbf{e} \in -\mathcal{C}(\mathbf{t})), \tag{5.9}$$

where P is the (random) pruning set, $\mathbf{e}$ is the BDD noise and $\mathcal{C}(\mathbf{t})$ is the cell of the tag $\mathbf{t}$. We now analyze the most frequent distributions for $\mathbf{e}$.

5.2.6.1 LWE and Gaussian Noise.

The most important BDD case is LWE [Reg05]. However, there are many variants of LWE using different distributions of the noise $\mathbf{e}$. We will use the continuous Gaussian distribution over $\mathbb{R}^n$, like in [Reg05]. Many schemes actually use a discrete distribution, such as some discrete Gaussian distribution over $\mathbb{Z}^n$ (or something easier to implement): because this is harder to analyze, some cryptanalysis papers such as [LP11, LN13] prefer to ignore this difference, and perform experiments to check if it matches with the theoretical analysis. The main benefit of the Gaussian distribution over $\mathbb{R}^n$ is that for any basis, each coordinate is a one-dimensional Gaussian.

Lemma 5.4. *Let $\mathbf{t} = (t_1, \ldots, t_n) \in \mathbb{N}^n$ be a tag of the natural partition $\mathcal{C}_{\mathbb{N}}()$ with basis $B = (\mathbf{b}_1, \ldots, \mathbf{b}_n)$. If the noise $\mathbf{e}$ follows the multivariate Gaussian distribution over $\mathbb{R}^n$ with parameter σ, then:*

$$p(\mathbf{t}) = \prod_{i=1}^{n}\left(\mathrm{erf}\left(\frac{1}{\sqrt{2}\sigma}\cdot\frac{t_i+1}{2}\cdot\|\mathbf{b}_i^\star\|\right) - \mathrm{erf}\left(\frac{1}{\sqrt{2}\sigma}\cdot\frac{t_i}{2}\cdot\|\mathbf{b}_i^\star\|\right)\right) \tag{5.10}$$

Proof. Each cell is a product of 2^n boxes, so the CDF of the Gaussian distribution gives:

$$p(\mathbf{t}) = 2^n \prod_{i=1}^{n} \frac{1}{2}\left(\operatorname{erf}\left(\frac{1}{\sqrt{2}\sigma}\cdot\frac{t_i+1}{2}\cdot\|\mathbf{b}_i^\star\|\right) - \operatorname{erf}\left(\frac{1}{\sqrt{2}\sigma}\cdot\frac{t_i}{2}\cdot\|\mathbf{b}_i^\star\|\right)\right)$$
$$= \prod_{i=1}^{n}\left(\operatorname{erf}\left(\frac{1}{\sqrt{2}\sigma}\cdot\frac{t_i+1}{2}\cdot\|\mathbf{b}_i^\star\|\right) - \operatorname{erf}\left(\frac{1}{\sqrt{2}\sigma}\cdot\frac{t_i}{2}\cdot\|\mathbf{b}_i^\star\|\right)\right)$$

□

5.2.6.2 Spherical Noise.

If the noise $\mathbf{e}$ is uniformly distributed over a centered ball, we can reuse the analysis of [AN17]:

Lemma 5.5. *Let $(\mathcal{C}, T)$ be a universal $\mathcal{L}$-partition. Let $\mathbf{t} \in T$ be a tag. If the noise $\mathbf{e}$ is uniformly distributed over the n-dimensional centered ball of radius R, then:*

$$p(\mathbf{t}) = \frac{\operatorname{vol}(\mathcal{C}(\mathbf{t}) \cap B_n(R))}{\operatorname{vol}(B_n(R))} \tag{5.11}$$

For both Babai's partition $\mathcal{C}_{\mathbb{Z}}$ and the natural partition $\mathcal{C}_{\mathbb{N}}$, $\mathcal{C}(\mathbf{t})$ is the union of disjoint symmetric boxes, so the evaluation of (5.11) is reduced to the computation of the volume of a ball-box intersection, which was done in [AN17].

5.2.6.3 Product of Finite Distributions.

We now consider a general distribution $\mathcal{D}$ for the noise $\mathbf{e}$ where each coordinate e_i is independently sampled from the uniform distribution over some finite set. This includes the box distribution, which is the uniform distribution over a set of the form $\prod_{i=1}^{n}\{a_i, \ldots, b_i\}$. The continuous Gaussian distribution and the uniform distribution over a ball are both invariant by rotation. But if the noise distribution $\mathcal{D}$ is not invariant by rotation, the tag probability $p(\mathbf{t})$ may take different values for the same $(\|\mathbf{b}_1^\star\|, \ldots, \|\mathbf{b}_n^\star\|)$, which is problematic for analysing the success probability. To tackle this issue, we reuse the following heuristic assumption introduced in [GNR10]:

Heuristic 5.6 ([GNR10, Heuristic 3])**.** *The distribution of the normalized Gram-Schmidt orthogonalization $(\mathbf{b}_1^\star/||\mathbf{b}_1^\star||, \ldots, \mathbf{b}_n^\star/||\mathbf{b}_n^\star||)$ of a random reduced basis $(\mathbf{b}_1, \ldots, \mathbf{b}_n)$ looks like that of a uniformly distributed orthogonal matrix.*

We obtain:

Lemma 5.7. *Let $\mathcal{C}_{\mathbb{N}}$ be the natural partition. Let $\mathbf{t} \in \mathbb{N}^n$ be a tag. If the distribution of the noise $\mathbf{e}$ has finite support, then under Heuristic 5.6:*

$$p(\mathbf{t}) = \sum_{r \in E} \Pr_{\mathbf{e}}(\|\mathbf{e}\| = r) \times \Pr_{\mathbf{x} \leftarrow S_n}(\mathbf{x} \in \mathcal{C}(\mathbf{t})/r) \tag{5.12}$$

where $E \subseteq \mathbb{R}_{\geqslant 0}$ denotes the finite set formed by all possible values of $\|\mathbf{e}\|$ and S_n denotes the n-dimensional unit sphere.

Proof. It suffices to decompose the noise $\mathbf{e}$ as $\mathbf{e} = \|\mathbf{e}\| \times \frac{\mathbf{e}}{\|\mathbf{e}\|}$. The variable $\|\mathbf{e}\|$ only takes finitely many values, and because of Heuristic 5.6,

$$\Pr_{\mathbf{e}}\left(-r\frac{\mathbf{e}}{\|\mathbf{e}\|} \in \mathcal{C}(\mathbf{t})\right) = \Pr_{\mathbf{x}\leftarrow S_n}(\mathbf{x} \in \mathcal{C}(\mathbf{t})/r).$$

□

5.2.7 Universality proof of Babai's and the natural partition

In this section, we show that Babai's partition and the natural partition are universal.

Proof of universality of Babai's partition (Lemma 5.1). We already know from [AN17] that $(\mathcal{C}_{\mathbb{Z}}, T)$ is a $\mathcal{L}$-partition. To show that it is actually universal, it suffices to show that for all $\mathbf{u} \in \mathbb{Q}^n$, $(\mathbf{u} + \mathcal{C}_{\mathbb{Z}}(\boldsymbol{t})) \cap \mathcal{L}$ is always a singleton, which can be found in polynomial time. To see this, note that Babai's nearest plane algorithm [Bab85] implies that for any $\mathbf{t} \in \mathbb{Z}^n$ and any $\mathbf{u} \in \mathbb{R}^n$, there is a unique $\mathbf{v} \in \mathcal{L}$ such that $\mathbf{v} - \mathbf{u} - \mathbf{t}B^\star \in \mathcal{D}$, and that $\mathbf{v}$ can be found in polynomial time when $\mathbf{u} \in \mathbb{Q}^n$. It follows that $(\mathbf{u} + \mathcal{C}_{\mathbb{Z}}(\boldsymbol{t})) \cap \mathcal{L} = \{\mathbf{v}\}$. □

Algorithm 5.2 Universal cell opening for Babai's partition from Babai's Nearest Plane algorithm [Bab85]

Input: A tag $\mathbf{t} \in \mathbb{Z}^n$, a target $\mathbf{u} \in \mathbb{Q}^n$, and a basis $B = (\mathbf{b}_1, \ldots, \mathbf{b}_n) \in \mathbb{Q}^n$ of a lattice $\mathcal{L}$, with Gram-Schmidt orthogonalization $B^\star$.
Output: $\mathbf{v} \in \mathcal{L}$ such that $\{\mathbf{v}\} = \mathcal{L} \cap (\mathbf{u} + \mathcal{C}_{\mathbb{Z}}(\mathbf{t}))$

1: $\mathbf{v} \leftarrow \mathbf{0}$ and $\mathbf{w} \leftarrow \mathbf{u} + \mathbf{t}B^\star$
2: **for** $i := n$ downto 1 **do**
3: Compute the integer c closest to $\langle \mathbf{b}_i^\star, \mathbf{w}\rangle / \langle \mathbf{b}_i^\star, \mathbf{b}_i^\star\rangle$
4: $\mathbf{w} \leftarrow \mathbf{w} - c\mathbf{b}_i$ and $\mathbf{v} \leftarrow \mathbf{v} + c\mathbf{b_i}$
5: **end for**
6: Return $\mathbf{v}$

Proof of universality of Babai's partition (Lemma 5.2). We already know that $(\mathcal{C}_{\mathbb{N}}, T)$ is an $\mathcal{L}$-partition. Let $\mathbf{u} \in \mathbb{Q}^n$: we can write $\mathbf{u} = \sum_{i=1}^n u_i \mathbf{b}_i^\star$. Then we only need to show that the shifted cell $\mathbf{u} + \mathcal{C}_{\mathbb{N}}(\mathbf{t}) = \{\sum_{i=1}^n (u_i + x_i)\mathbf{b}_i^\star$ s.t. $-(t_i+1)/2 < x_i \leqslant -t_i/2$ or $t_i/2 < x_i \leqslant (t_i+1)/2\}$ contains only one lattice point which can be found in polynomial time using Algorithm 5.3. Consider the projection π over the orthogonal supplement to the subspace spanned by $\mathbf{b}_1, \ldots, \mathbf{b}_{n-1}$. Then:

$$\pi(\mathbf{u} + \mathcal{C}_{\mathbb{N}}(\mathbf{t})) = \{(u_n + x_n)\mathbf{b}_n^\star \text{ s.t. } -(t_n+1)/2 < x_n \leqslant -t_n/2 \text{ or } t_n/2 < x_n \leqslant (t_n+1)/2\}.$$

Notice that the union $(u_n - (t_n+1)/2, u_n - t_n/2] \bigcup (u_n + t_n/2, u_n + (t_n+1)/2]$ only contains one integer: this is because the number of integers in an interval of the form $(x, y]$ is $\lfloor y \rfloor - \lfloor x \rfloor$ when $x \geqslant y$. And that integer can be found in polynomial time, as shown by Algorithm 5.3. This shows that $\pi(\mathbf{u} + \mathcal{C}_{\mathbb{N}}(\mathbf{t})) \cap \pi(\mathcal{L})$ is a singleton, which can be found in polynomial time. Algorithm 5.3 iterates this process using projections orthogonally to $\mathbf{b}_1, \ldots, \mathbf{b}_i$. □

Algorithm 5.3 Universal cell opening for the natural partition: adaptation of [AN17, Algorithm 3]

Input: A tag $\mathbf{t} \in \mathbb{N}^n$, a target $\mathbf{u} = \sum_{i=1}^n u_i \mathbf{b}_i^\star \in \mathbb{Q}^n$, and a basis $B = (\mathbf{b}_1, \ldots, \mathbf{b}_n) \in \mathbb{Q}^n$ of a lattice $\mathcal{L}$, with Gram-Schmidt matrix μ.
Output: $\mathbf{v} \in \mathcal{L}$ such that $\{\mathbf{v}\} = \mathcal{L} \cap (\mathbf{u} + \mathcal{C}_{\mathbb{N}}(\mathbf{t}))$

1: **for** $i := n$ downto 1 **do**
2: $y \leftarrow -\sum_{j=i+1}^n v_j \mu_{j,i}$ and $v_i \leftarrow \lfloor u_i + y + 0.5 \rfloor$
3: **if** $v_i < u_i + y$ **then**
4: $v_i \leftarrow v_i - (-1)^{t_i} \lfloor t_i/2 \rceil$
5: **else**
6: $v_i \leftarrow v_i + (-1)^{t_i} \lfloor t_i/2 \rceil$
7: **end if**
8: **end for**
9: Return $\sum_{i=1}^n v_i \mathbf{b}_i$

5.3 Quantum speed-up of Cylinder Pruning

5.3.1 Tools

The analysis of quantum tree algorithms requires the tree to have constant degree $d = O(1)$. Without this assumption, there is an extra $\mathrm{poly}(d)$ term in the complexity bound like in Theorem 2.12. Instead, it is more efficient to first convert the tree into a binary tree, so that the overhead is limited to $\mathrm{poly}(\log d)$. We will use the following conversion (illustrated by Figure 5.3):

Theorem 5.8. *One can transform any tree $\mathcal{T}$ of depth n and degree d into a binary one $\mathcal{T}_2$ so that: $\mathcal{T}_2$ can be explored locally; $\mathcal{T}$ and $\mathcal{T}_2$ have roughly the same number of nodes, namely $\#\mathcal{T} \leqslant \#\mathcal{T}_2 \leqslant 2\#\mathcal{T}$; the leaves of $\mathcal{T}$ and $\mathcal{T}_2$ are identical; the depth of $\mathcal{T}_2$ is $\leqslant n \log d$. Moreover, a black-box function $\mathcal{P}$ over $\mathcal{T}$ can be adapted to a black box $\mathcal{P}_2$ for $\mathcal{T}_2$, so that the marked nodes of $\mathcal{T}$ and $\mathcal{T}_2$ are the same. One query to $\mathcal{P}_2$ requires at most one query to $\mathcal{P}$ with additional $O(\log d)$ auxiliary operations.*

Proof. For any node $\mathcal{N} \in \mathcal{T}$ which is not a leaf, we want to transform the subtree $\mathcal{N}$ plus its children into a binary subtree in $\mathcal{T}$. By making queries to the black box, we know the number of children $d(\mathcal{N})$ of $\mathcal{N}$. We also know the i-th child of $\mathcal{N}$ for all $1 \leqslant i \leqslant d(\mathcal{N})$. There is thus a bijection $f_{\mathcal{N}}$ between $[|0, \cdots, d(\mathcal{N}) - 1|]$ and the children of $\mathcal{N}$. We define $l_{\mathcal{N}} = \lceil \log_2(d(\mathcal{N})) \rceil$. For each node of the tree $\mathcal{N}$, we encode the corresponding node in $\mathcal{T}_2$ in the same way.

For those nodes which are in the local binary sub-tree in $\mathcal{T}_2$ corresponding to the local sub-tree $\mathcal{N}$+its children, and which does not correspond to $\mathcal{N}$ or its children, we can encode them as: $E(\mathcal{N})|((x_1 \in \{0,1\}, \cdots, x_i \in \{0,1\}, *, \cdots, *))$ where $E(\mathcal{N})$ is the encoding of the node $\mathcal{N}$, where $i \leqslant l_{\mathcal{N}}$ and where the number in base 10 corresponding to $(x_1, \cdots, x_i, 0, \cdots, 0)$ in base 2 is smaller than $d(\mathcal{N}) - 1$. Note that in the representation in base 2 $(x_1, \cdots, x_i, 0, \cdots, 0)$, the heaviest bit is on the left.

Given an encoding of a node $\mathcal{N}_2$ in $\mathcal{T}_2$ which is in the local binary sub-tree in $\mathcal{T}_2$ corresponding to the local sub-tree $\mathcal{N}$ plus its children in $\mathcal{T}$, we can easily build a black box which

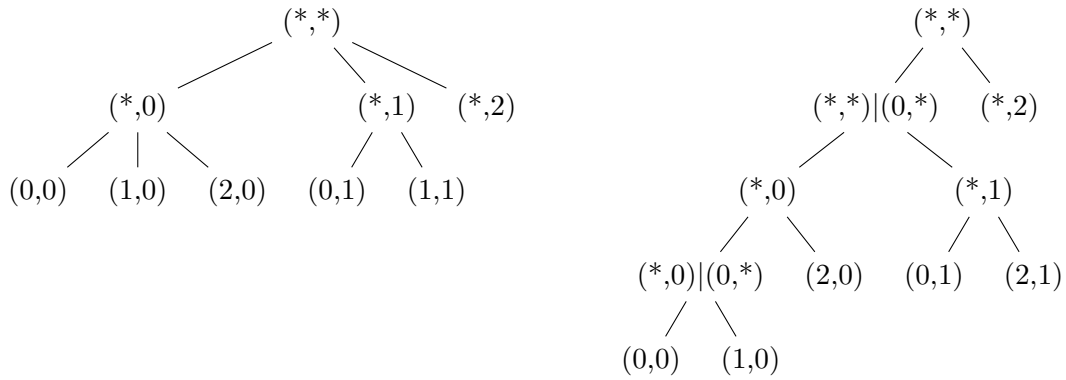


Figure 5.3: An example of the transformation in Theorem 5.8

gives the children of this node by using the function $f_{\mathcal{N}}$ and the value $d(\mathcal{N})$. We omit the details here. Depending on if the node $\mathcal{N}_2$ corresponds to a node in $\mathcal{T}$ or not, a query on the node $\mathcal{N}_2$ requires a query on $\mathcal{T}$ or not. If $\mathcal{N}_2$ does not correspond to a node in $\mathcal{T}$, we need $O(\log(d))$ auxiliary operations on the extra encoding to see if its children correspond to nodes in $\mathcal{T}$. These operations can then be quantized using Corollary 2.2 with potentially additional workspace of size at most polynomial of the initial memory space.

According to our construction, the leaves of both trees are identical.

We will now prove that: $\#\mathcal{T} \leqslant \#\mathcal{T}_2 \leqslant 2\#\mathcal{T}$. The left-hand inequality is obvious. If a node $\mathcal{N}$ of $\mathcal{T}$ is a leaf or has a single child, the subtree of $\mathcal{N}$ plus its child (in case that it exists) will not change in $\mathcal{T}_2$. If $\mathcal{N}$ has at least two children, the subtree of $\mathcal{N}$+its children will be transformed into a binary subtree in $\mathcal{T}_2$. Assume that $\mathcal{N}$ has $k \geqslant 2$ children, the corresponding subtree in $\mathcal{T}_2$ has $2k-1$ nodes. It has $2k-2 < 2k$ nodes if we do not count the root corrsponding to $\mathcal{N}$ itself. Thus the k children of $\mathcal{N}$ are transformed into $2k-2$ nodes in $\mathcal{T}_2$ if $k \geqslant 2$. By combining the previous two cases, we obtain that $\#\mathcal{T}_2 \leqslant 2\#\mathcal{T}$. □

In the context of enumeration with pruning, instead of enumerating the whole set $\mathcal{L} \cap S$, we may only be interested in the 'best' vector in $\mathcal{L} \cap S$, *i.e.* minimizing some distance. In terms of tree, this means that given a tree $\mathcal{T}$ with marked leafs defined by a predicate $\mathcal{P}$, we want to find a marked leaf minimizing an integral function g which is defined on the marked leaves of $\mathcal{T}$. We know that $L(\mathcal{T}) = L(\mathcal{T}_2)$. g is thus also defined on the marked leaves of $\mathcal{T}_2$. We denote by g_V the predicate which returns true on a node $\mathcal{N}$ if and only if it is a marked leaf and $g(\mathcal{N}) \leqslant V$. We first find a parameter R such that there is at least one marked leaf $\mathcal{N}$ such that $g(\mathcal{N}) \leqslant R$. Then we decrease R by dichotomy using Theorem 2.9 with different marking functions. We thus obtain **FindMin1**$(\mathcal{T}, \mathcal{P}, g, R, d, \varepsilon)$ (Algorithm 5.4), which is a general algorithm to find a leaf minimizing the function g with error probability ε, using the binary tree $\mathcal{T}_2$.

Theorem 5.9. *Let $\varepsilon > 0$. Let $\mathcal{T}$ be a tree with its marked leaves defined by a predicate $\mathcal{P}$. Let d be an upper-bound on the degree of $\mathcal{T}$. Let g be an integral function defined on the marked leaves such that $g(\mathcal{N}) \leqslant R$ has at least one solution over all of the marked leaves. Then Algorithm 5.4 outputs $\mathcal{N} \in \mathcal{T}$ such that g takes its minimum on $\mathcal{N}$ among all of the*

Algorithm 5.4 Finding a minimum: **FindMin1**$(\mathcal{T}, \mathcal{P}, g, R, d, \varepsilon)$

Input: A tree $\mathcal{T}$ with marked leaves defined by the predicate $\mathcal{P}$. An integral function g defined on the marked leaves of $\mathcal{T}$. A parameter R, such that $g(\mathcal{N}) \leqslant R$ has at least one solution over all of the marked leaves. An upper-bound d of the number of children of a node in $\mathcal{T}$.

Output: A marked leaf $\mathcal{N}$ such that g takes its minimum on $\mathcal{N}$ among all the marked leaves explored by the backtracking algorithm.

1: $\mathcal{T}_2 \leftarrow$ the corresponding binary tree of $\mathcal{T}$.[1]
2: $N \leftarrow R$, $N' \leftarrow 0$, $Round \leftarrow \lceil \log_2 R \rceil$, $\mathbf{v} \leftarrow (0, \cdots, 0)$
3: **while** $N' < N - 1$ **do**
4: Call **FindSolution**$(\mathcal{T}_2, g_{\lfloor (N+N')/2 \rceil}, n \log d, \varepsilon / Round)$
5: **if FindSolution**$(\mathcal{T}_2, g_{\lfloor (N+N')/2 \rceil}, n \log d, \varepsilon / Round)$ returns $\mathbf{x}$ **then**
6: $\mathbf{v} \leftarrow \mathbf{x}$, $N \leftarrow \lfloor (N + N')/2 \rceil$
7: **else**
8: $N' \leftarrow \lfloor (N + N')/2 \rceil$
9: **end if**
10: **end while**
11: **return v**

marked leaves of $\mathcal{T}$, with probability at least $1 - \varepsilon$. It requires

$$O(\sqrt{T}(n \log d)^{3/2} \log(n \log d) \log(\lceil \log_2 R \rceil / \varepsilon) \lceil \log_2 R \rceil)$$

queries on $\mathcal{T}$ and on $\mathcal{P}$, where $T = \#\mathcal{T}$. Each query on $\mathcal{T}$ requires $O(\log d)$ auxiliary operations. The algorithm needs $\mathrm{poly}(n \log d, \log R)$ *qubits.*

Proof. Correctness is trivial. Regarding the query complexity, there are in total $Round = \lceil \log_2 R \rceil$ calls to **FindSolution**. According to Theorem 2.9, each call requires

$$O(\sqrt{T}(n \log d)^{3/2} \log(n \log d) \log(Round/\varepsilon))$$

queries on the local structure of $\mathcal{T}_2$ and on g. Thus according to Theorem 5.8, in total, we need

$$O(\sqrt{T}(n \log d)^{3/2} \log(n \log d) \log(\lceil \log_2 R \rceil / \varepsilon) \lceil \log_2 R \rceil)$$

queries on the local structure of $\mathcal{T}$ and on g. Each query on $\mathcal{T}$ requires $O(\log d)$ auxiliary operations. For each call, we need $\mathrm{poly}(n \log d)$ qubits. In total, we need $\mathrm{poly}(n \log d, \log R)$ qubits. □

If we know an upper-bound T on the number of nodes in the tree $\mathcal{T}$, we can speed up the algorithm by replacing **FindSolution** by **ExistSolution** in lines 4, 5: the new algorithm **FindMin2**$(\mathcal{T}, \mathcal{P}, g, R, d, T, \varepsilon)$ is given and analyzed in the followings.

Theorem 5.10. *Let $\varepsilon > 0$. Let $\mathcal{T}$ be a tree with its marked leaves defined by a predicate $\mathcal{P}$ Let g be an integral function defined on the marked leaves such that $g(\mathcal{N}) \leqslant R$ has at least one solution over all of the marked leaves and an upper-bound d of the number of children of a*

[1] The access to $\mathcal{T}_2$ is guaranteed by Theorem 5.8 via the access to $\mathcal{T}$.

Algorithm 5.5 Finding a minimum, given an upper-bound of the tree-size: **FindMin2**$(\mathcal{T}, \mathcal{P}, g, R, d, T, \varepsilon)$

Input: A tree $\mathcal{T}$ with marked leaves defined by the predicate $\mathcal{P}$. An integral function g defined on the marked leaves of $\mathcal{T}$. A parameter R, such that $g(\mathcal{N}) \leqslant R$ has at least one solution over all of the marked leaves. An upper-bound d of the number of children of a node in $\mathcal{T}$.

Output: A marked leaf $\mathcal{N}$ such that g takes its minimum on $\mathcal{N}$ among all the marked leaves explored by the backtracking algorithm.

$\mathcal{T}_2 \leftarrow$ the corresponding binary tree of $\mathcal{T}$
$N \leftarrow R$, $N' \leftarrow 0$
$Round \leftarrow \lceil \log_2(R) \rceil + 1$
$\mathbf{v} \leftarrow (0, \cdots, 0)$
while $N' < N - 1$ **do**
  Call **ExistSolution**$(\mathcal{T}_2, T, g_{\lfloor (N+N')/2 \rceil}, n\log(d), \varepsilon/Round)$
  if ExistSolution$(\mathcal{T}_2, T, g_{\lfloor (N+N')/2 \rceil}, n\log(d), \varepsilon/Round)$ returns "marked node exists" **then**
    $N \leftarrow \lfloor (N + N')/2 \rceil$
  else
    $N' \leftarrow \lfloor (N + N')/2 \rceil$
  end if
end while
Call **FindSolution**$(\mathcal{T}_2, g_N, n\log(d), \varepsilon/Round)$
if FindSolution$(\mathcal{T}_2, g_N, n\log(d), \varepsilon/Round)$ returns **x then**
  $\mathbf{v} \leftarrow \mathbf{x}$
  return v
else
  return
end if

node in $\mathcal{T}$. Then **FindMin2**$(\mathcal{T}, \mathcal{P}, g, R, d, T, \varepsilon)$ *outputs a marked leaf $\mathcal{N}$, such that g takes its minimum on $\mathcal{N}$ among all of the marked leaves of $\mathcal{T}$, with probability at least $1 - \varepsilon$. It requires*

$$O(\sqrt{Tn\log(d)}\log((\lceil\log_2(R)\rceil)/\varepsilon)\lceil\log_2(R)\rceil + \sqrt{T}(n\log(d))^{3/2}\log(n\log(d))\log((\lceil\log_2(R)\rceil)/\varepsilon))$$

queries on the tree $\mathcal{T}$ and on g. Each query on $\mathcal{T}$ requires $O(\log(d))$ auxiliary operations. The algorithm needs $\mathrm{poly}(n\log(d), \log(R))$ *qubits.*

Proof. The correctness of the algorithm is easy to prove. We will compute the query complexity. There are in total $Round - 1 = \lceil \log_2(R) \rceil$ calls on **ExistSolution** and one call on **FindSolution**. According to Theorem 2.8, for each call of **ExistSolution**, we need $O(\sqrt{Tn\log(d)}\log(Round/\varepsilon))$ queries on $\mathcal{T}_2$ and on g. According to Theorem 2.9, the call on **FindSolution** requires $O(\sqrt{T}(n\log(d))^{3/2}\log(n\log(d))\log(Round/\varepsilon))$ queries on the local structure of the tree $\mathcal{T}_2$ and on g. In total, we need

$$\begin{aligned}&O(\sqrt{Tn\log(d)}\log(Round/\varepsilon) * (Round - 1)) + \sqrt{T}(n\log(d))^{3/2}\log(n\log(d))\log(Round/\varepsilon))\\ &= O(\sqrt{Tn\log(d)}\log((\lceil\log_2(R)\rceil + 1)/\varepsilon)\lceil\log_2(R)\rceil + \sqrt{T}(n\log(d))^{3/2}\log(n\log(d))\log((\lceil\log_2(R)\rceil + 1)/\varepsilon))\end{aligned}$$

$$= O(\sqrt{Tn\log(d)}\log((\lceil\log_2(R)\rceil)/\varepsilon)\lceil\log_2(R)\rceil + \sqrt{T}(n\log(d))^{3/2}\log(n\log(d))\log((\lceil\log_2(R)\rceil)/\varepsilon))$$

queries on $\mathcal{T}_2$ and on g. According to Theorem 5.8, in total, we need

$$O(\sqrt{Tn\log(d)}\log((\lceil\log_2(R)\rceil)/\varepsilon)\lceil\log_2(R)\rceil + \sqrt{T}(n\log(d))^{3/2}\log(n\log(d))\log((\lceil\log_2(R)\rceil)/\varepsilon))$$

queries on $\mathcal{T}$ and on g. Each query on $\mathcal{T}$ requires $O(\log(d))$ auxiliary operations.

Each call of **ExistSolution** and **FindSolution** requires $\mathrm{poly}(n\log(d))$ qubits. In total the algorithm needs $\mathrm{poly}(n\log(d), \log(R))$ qubits.

□

5.3.2 Application to Cylinder Pruning

Lemma 5.11. *Let $(\mathbf{b}_1, \cdots, \mathbf{b}_n)$ be an LLL-reduced basis. Let $\mathcal{T}$ be the backtracking tree corresponding to the cylinder pruning algorithm for SVP with radius $R \leqslant \|\mathbf{b}_1\|$ and bounding function f. Then the degree of the tree satisfies: $d(\mathcal{T}) \leqslant 2^n$.*

Proof. In $\mathcal{T}$, the number of children of a node $\mathcal{N}$ of depth k can be upper-bounded by $d_k = 2f(k)\frac{\|\mathbf{b}_1\|}{\|\mathbf{b}^\star_{n-k+1}\|} + 1 \leqslant 2^{(n-k)/2+1} + 1$. The result follows from the fact that an LLL-reduced basis satisfies: $\frac{\|\mathbf{b}_1\|^2}{\|\mathbf{b}_i^\star\|^2} \leqslant 2^{i-1}$ for all $1 \leqslant i \leqslant n$. □

Theorem 5.12. *There is a quantum algorithm which, given $\varepsilon > 0$, an LLL-reduced basis $B = (\mathbf{b}_1, \cdots, \mathbf{b}_n)$ of a lattice $\mathcal{L}$ in $\mathbb{Z}^n$, a radius $R \leqslant \|\mathbf{b}_1\|$ and a bounding function $f : \{1, \cdots, n\} \to [0, 1]$, outputs with correctness probability $\geqslant 1 - \varepsilon$:*

1. *a non-zero vector $\mathbf{v}$ in $\mathcal{L} \cap P_f(B, R)$, in time $O(\sqrt{T}n^3\,\mathrm{poly}(\log(n), \log(1/\varepsilon)))$, if $\mathcal{L} \cap P_f(B, R) \nsubseteq \{0\}$.*
2. *all vectors in $\mathcal{L} \cap P_f(B, R)$, in time $O(\#(\mathcal{L} \cap P_f(B, R))\sqrt{T}n^3\log(n)\,\mathrm{poly}(\log(\#(\mathcal{L} \cap P_f(B, R)), \log(1/\varepsilon)))$.*
3. *a shortest non-zero vector $\mathbf{v}$ in $\mathcal{L} \cap P_f(B, R)$, in time $O(\sqrt{T}n^3\beta\,\mathrm{poly}(\log(n), \log(1/\varepsilon), \log(\beta)))$, if $\mathcal{L} \cap P_f(B, R) \nsubseteq \{0\}$. Here β is the bitsize of the vectors of B.*

Here T is the total number of nodes in the enumeration tree $\mathcal{T}$ searched by the cylinder pruning algorithm over $P_f(B, R)$.

Proof. Let $\mathcal{T}$ be the enumeration tree searched by the cylinder pruning algorithm in which a node of depth i, where $1 \leqslant i \leqslant n$, is encoded as $(*, \cdots, *, x_{n-i+1}, \cdots, \cdots, x_n)$ and where the root is encoded as $(*, \cdots, *)$. Let $\mathcal{T}_2$ be the corresponding binary tree. Let $\mathcal{P}$ be a predicate which returns true only on the nodes encoded as $(x_1, \cdots, x_n)$ in $\mathcal{T}_2$ (*i.e.* the leaves of $\mathcal{T}_2$, where all the variables are assigned), such that $\|\sum_{i=1}^n x_i\mathbf{b}_i\|^2 \leqslant R^2$ and $(x_1, \cdots, x_n) \neq (0, \cdots, 0)$.

For 1, if $\mathcal{L} \cap P_f(B, R) \neq \varnothing$, we apply **FindSolution**$(\mathcal{T}_2, \mathcal{P}, n\log d, \varepsilon)$. For 2, we find all marked nodes by simply repeating the algorithm **FindSolution**, modifying the oracle operator to strike out previously seen marked elements, which requires space complexity $O(\#(\mathcal{L} \cap P_f(B, R)))$.

For 3, if $\mathcal{L} \cap P_f(B, R) \neq \varnothing$, we apply Theorem 5.9 to **FindMin1**$(\mathcal{T}, \mathcal{P}, \|\cdot\|^2, R^2, 2^n + 1, \varepsilon)$. In $\mathcal{T}_2$, the height of the tree can be upper-bounded by $n\log d = O(n^2)$. We also have $Round = O(\beta)$. The time complexity is $O(\sqrt{T}n^3\beta\,\mathrm{poly}(\log(n), \log(1/\varepsilon), \log(\beta)))$. □

We can also apply the quantum tree algorithms to extreme pruning. If we run cylinder pruning over m trees, we can combine these trees into a global one and apply the quantum tree algorithms on it.

Theorem 5.13 (Quantum speed-up for SVP extreme pruning)**.** *There is a quantum algorithm which, given $\varepsilon > 0$, m LLL-reduced bases $B_1, \cdots B_m$ of a lattice $\mathcal{L}$ in $\mathbb{Z}^n$,a radius $R \leqslant \min_i \|\mathbf{b}_{1,i}\|$ where $\mathbf{b}_{1,i}$ is the first vector of B_i and a bounding function $f : \{1, \cdots, n\} \to [0, 1]$, outputs with correctness probability $\geqslant 1 - \varepsilon$ a shortest non-zero vector $\mathbf{v}$ in $\mathcal{L} \cap (\cup P_f(B_i, R))$, in time $O(\sqrt{T}n^3\beta \operatorname{poly}(\log(n), \log(1/\varepsilon), \log(\beta), \log(m)))$, if $\mathcal{L} \cap (\cup P_f(B_i, R) \nsubseteq \{0\}$. Here β is a bound on the bitsize of vectors of B_i's, T is the sum of number of nodes in the enumeration trees $\mathcal{T}_i$ searched by cylinder pruning over $P_f(B_i, R)$ for all $1 \leqslant i \leqslant m$.*

In the case of CVP with target vector $\mathbf{u}$, we use the cylinder pruning algorithm with radius $R \leqslant \sqrt{\sum_{i=1}^n \|\mathbf{b}_i^\star\|^2}/2$ and bounding function f. The degree of the tree is now upper-bounded by $d = \max \sqrt{\sum_{i=1}^n \|\mathbf{b}_i^\star\|^2}/\|\mathbf{b}_j^\star\| + 1$. We have $\log d = O(\beta + n)$ where β is the bitsize of the vectors of the basis B. We can obtain a similar theorem as Theorem 5.12 with different overheads. For exemple for the first case, the time complexity becomes $O(\sqrt{T}n^{3/2}(n + \beta)^{3/2} \operatorname{poly}(\log(n), \log(1/\varepsilon), \log(\beta))))$.

For the extreme pruning for CVP the time complexity is

$$O(\sqrt{T}n^{3/2}(n + \beta)^{3/2}\beta \operatorname{poly}(\log(n), \log(1/\varepsilon), \log(\beta), \log(m)))$$

5.4 Linear Optimization for Discrete Pruning

We saw in Section 5.2.6 how to compute or approximate the probability $p(\mathbf{t})$ that the cell of the tag $\mathbf{t}$ contains the BDD solution. From Lemma 5.3, we know that for any integer $m > 0$, there are m tags which maximize $p(\mathbf{t})$ in the sense that any other tag must have a lower $p(\mathbf{t})$. To select optimal parameters for BDD discrete pruning, we want to find these m tags as fast as possible, possibly in m operations and polynomial-space (by outputting the result as a stream).

5.4.1 Reduction to Linear Optimization

We distinguish two cases:

- Selection based on expectation. Experiments performed in [AN17] show that in practice, the m tags $\mathbf{t}$ which maximize $\operatorname{vol}(\mathcal{C}_\mathbb{N}(\mathbf{t}) \cap B_n(R))$ are essentially the ones which minimize the expectation $\mathbb{E}\{\mathcal{C}_\mathbb{N}(\mathbf{t})\}$ where $\mathbb{E}\{C\} := \mathbb{E}_{\mathbf{x}\in C}(\|\mathbf{x}\|^2)$ over the uniform distribution. Corollary 3 in [AN17] shows that this expectation is:

$$\mathbb{E}\{\mathcal{C}_\mathbb{N}(\mathbf{t})\} = \sum_{i=1}^n \left(\frac{t_i^2}{4} + \frac{t_i}{4} + \frac{1}{12}\right)\|\mathbf{b}_i^\star\|^2.$$

  So we can assume that for a noise uniformly distributed over a ball (see Lemma 5.5), the m tags $\mathbf{t}$ maximizing $p(\mathbf{t})$ are the tags minimizing $\mathbb{E}\{\mathcal{C}_\mathbb{N}(\mathbf{t})\}$.

- Gaussian noise. If the noise distribution is the continuous multivariate Gaussian distribution, Lemma 5.4 shows that $p(\mathbf{t})$ is given by (5.10). This implies that the m tags $\mathbf{t}$ which maximize $p(\mathbf{t})$ are the ones which minimize $-\log p(\mathbf{t})$

In both cases, we want to find the m tags $\mathbf{t} \in \mathbb{N}^n$ which minimize an objective function g of the form $g(\mathbf{t}) = \sum_{i=1}^{n} f(i, t_i)$, where $f(i, t_i) \geqslant 0$. The fact that the objective function can be decomposed as a sum of individual positive functions in each coordinate allows us to view this problem as a *linear optimization*. We will see that in the case that g has integral outputs, it is possible to provably find the best m tags which minimize such a function g in essentially m operations. If g is not integral, it is nevertheless possible to enumerate all solutions such that $g(\mathbf{t}) \leqslant R$ where R is an input, in time linear in the number of solutions. A special case is the problem of enumerating smooth numbers below a given number.

In practice, it is more efficient to rely on the expectation, because it is faster to evaluate. Figure 5.4 shows how similar are the best tags with respect to one indicator compared to another: to compare two sets A and B formed by the best M tags, the graph displays $\#(A \cap B)/M$. For instance, the top curve confirms the experimental result of [AN17] that the

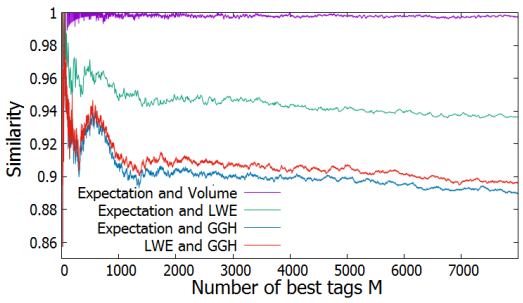


Figure 5.4: Similarity between optimal sets of tags, depending on the objective function.

m tags $\mathbf{t}$ which maximize $\mathrm{vol}(\mathcal{C}_{\mathbb{N}}(\mathbf{t}) \cap B_n(R))$ are almost the same as the ones which minimize the expectation $\mathbb{E}\{\mathcal{C}_{\mathbb{N}}(\mathbf{t})\}$. The top second curve shows that the best tags that maximize the LWE probability are very close to those minimizing the expectation. The bottom two curves compare with the finite noise distribution arising in GGH challenges [GGH97]. In all cases, at most 10% of the best tags are different, and more importantly, we report that the global success probabilities are always very close, with a relative error typically $\leqslant 1\%$.

We conclude that in practice, the expectation is a very good indicator to select the best tags for the distributions studied in Section 5.2.6.

5.4.2 Limits of Orthogonal Enumeration

Aono and Nguyen [AN17, Section 6] presented a heuristic method to solve this linear optimization problem in the special case: $g(\mathbf{t}) = \mathbb{E}\{\mathcal{C}_{\mathbb{N}}(\mathbf{t})\} = \sum_{i=1}^{n} \left(\frac{t_i^2}{4} + \frac{t_i}{4} + \frac{1}{12}\right) \|\mathbf{b}_i^\star\|^2$, by noticing that $\mathbb{E}\{\mathcal{C}_{\mathbb{N}}(\mathbf{t})\}$ was the squared distance between a target point and a special lattice with a known orthogonal basis. This allowed to find all $\mathbf{t} \in \mathbb{N}^n$ such that $\mathbb{E}\{\mathcal{C}_{\mathbb{N}}(\mathbf{t})\} \leqslant R$ for any R, using a variant [AN17, Algorithm 6] of enumeration. And by using a binary search based on an early-abort variant, it was also possible to find an R yielding slightly more than m solutions.

[AN17, Section 6] reported that this algorithm worked very well in practice: if ℓ is the number of $\mathbf{t} \in \mathbb{N}^n$ such that $\mathbb{E}\{\mathcal{C}_{\mathbb{N}}(\mathbf{t})\} \leqslant R$, the number of nodes $\mathcal{L}$ of the enumeration algorithm [AN17, Algorithm 6] seemed to be bounded by $O(\ell n)$, perhaps even $\ell \times n$. This was in contrast with the usual situation where the number of nodes of the enumeration tree is exponentially larger than the number of solutions. However, no rigorous result could be proved in [AN17], leaving it as an open problem to show the efficiency of [AN17, Algorithm 6].

Surprisingly, we solve this open problem of [AN17] in the negative. More precisely, we show that there are cases where the number of nodes $\mathcal{L}$ of enumeration [AN17, Algorithm 6] is exponentially larger than the number of solutions ℓ. To see this, consider the orthogonal lattice $\mathbb{Z}^n$ with the canonical basis. Then: $\mathbb{E}\{\mathcal{C}_{\mathbb{N}}(\mathbf{t})\} = \sum_{i=1}^{n}\left(\frac{t_i^2}{4} + \frac{t_i}{4} + \frac{1}{12}\right)$. But we have:

Lemma 5.14. *Let $R = \frac{n}{12} + \frac{1}{2}$ and $n' = \lfloor n/10 \rfloor$. Then the number ℓ of $\mathbf{t} \in \mathbb{N}^n$ such that $\sum_{i=1}^{n}\left(\frac{t_i^2}{4} + \frac{t_i}{4} + \frac{1}{12}\right) \leqslant R$ is exactly $n+1$. But the number ℓ' of $(x_{n-n'+1}, \ldots, x_n) \in \mathbb{N}^{n'}$ such that $\sum_{i=n-n'+1}^{n}\left(\frac{x_i^2}{4} + \frac{x_i}{4} + \frac{1}{12}\right) \leqslant R$ is $\geqslant 2^{n'}$.*

Proof. For the choice $R = \frac{n}{12} + \frac{1}{2}$, we have $\sum_{i=1}^{n}\left(\frac{t_i^2}{4} + \frac{t_i}{4} + \frac{1}{12}\right) \leqslant R$ if and only if all the t_i's are equal to zero, except at most one, which must be equal to one.

Furthermore, for any $(x_{n-n'+1}, \ldots, x_n) \in \{0,1\}^{n'}$, we have:

$$\sum_{i=n-n'+1}^{n}\left(\frac{x_i^2}{4} + \frac{x_i}{4} + \frac{1}{12}\right) \leqslant n'\left(\frac{1}{2} + \frac{1}{12}\right) \leqslant \frac{n}{10}\frac{7}{12} = \frac{7n}{120} < R.$$

□

It follows in this case that the number of nodes $\mathcal{L}$ of the enumeration algorithm [AN17, Algorithm 6] for that R is at least exponential in n, though the number of solutions is linear in n.

5.4.3 Solving Linear Optimization

We show that a slight modification of orthogonal enumeration can solve the more general problem of linear optimization essentially optimally. This is based on two key ideas. The first idea is that when solving linear optimization, we may assume without loss of generality that each function $f(i,)$ is sorted by increasing value, with a starting value equal to zero, which changes the tree: $f(i,0) = 0$ and $f(i,j) \leqslant f(i,j')$ whenever $j \leqslant j'$. Indeed, it suffices to sort the values of $f(i,)$ if necessary and subtract the minimal value: however, note that for both the expectation $\mathbb{E}\{\mathcal{C}_{\mathbb{N}}(\mathbf{t})\} = \sum_{i=1}^{n}\left(\frac{t_i^2}{4} + \frac{t_i}{4} + \frac{1}{12}\right)\|\mathbf{b}_i^\star\|^2$ and for $-\sum_{i=1}^{n}\log\left(\operatorname{erf}\left(\frac{1}{\sqrt{2}\sigma}\cdot\frac{t_i+1}{2}\cdot\|\mathbf{b}_i^\star\|\right) - \operatorname{erf}\left(\frac{1}{\sqrt{2}\sigma}\cdot\frac{t_i}{2}\cdot\|\mathbf{b}_i^\star\|\right)\right)$, the values of $f(i,)$ are already sorted. For instance, $\frac{t_i^2}{4} + \frac{t_i}{4} + \frac{1}{12}$ is an increasing function of t_i.

The second idea is that we may assume to simplify that f has integral values, which allows us to bound the running time of dichotomy. This is not directly true for the expectation $\mathbb{E}\{\mathcal{C}_{\mathbb{N}}(\mathbf{t})\} = \sum_{i=1}^{n}\left(\frac{t_i^2}{4} + \frac{t_i}{4} + \frac{1}{12}\right)\|\mathbf{b}_i^\star\|^2$. However, because we deal with integer lattices, the basis B is integral, the $\|\mathbf{b}_i^\star\|^2$'s are rational numbers with denominator $\operatorname{covol}(\mathcal{L}(\mathbf{b}_1, \ldots, \mathbf{b}_{i-1}))^2$, so we

can transform the expectation into an integer, by multiplying with a suitable polynomial-size integer.

First, we present a slight modification Algorithm 5.6 of [AN17, Algorithm 6], whose running time is provably essentially proportional to the number of solutions:

Theorem 5.15. *Assume that $f : \{1, \dots, n\} \times \mathbb{N} \to \mathbb{R}$ satisfies $f(i, 0) = 0$ and $f(i, j) \geqslant f(i, j')$ for all i and $j > j'$. Given as input a number $R > 0$, Algorithm 5.6 outputs all $(v_1, \dots, v_n) \in \mathbb{N}^n$ such that $\sum_{i=1}^{n} f(i, v_i) \leqslant R$ using $O(nN + 1)$ arithmetic operations and $\leqslant (2n - 1)N + 1$ calls to the function $f()$, where the number N is the number of $(v_1, \dots, v_n) \in \mathbb{N}^n$ such that $\sum_{i=1}^{n} f(i, v_i) \leqslant R$.*

Proof. To analyze the complexity of Algorithm 5.6, let n_k denote the number of times we enter Lines 3–18, depending on the value of k, which is $\geqslant 1$ and $\leqslant n$ at each Line 3. Then n_k can be decomposed as $n_k = a_k + b_k$, where a_k (resp. b_k) denotes the number of times we enter Lines 5–10 (resp. Lines 12–17). Notice that $a_{n+1} = 0$ and a_1 is exactly the number N of $(v_1, \dots, v_n) \in \mathbb{N}^n$ such that $\sum_{i=1}^{n} f(i, v_i) \leqslant R$. And if $1 < i \leqslant n$, then a_i is the number of times that the variable k is decremented from i to $i - 1$. Similarly, $b_n = 1$, and if $1 \leqslant i \leqslant n$, then b_i is the number of times that the variable k is incremented from i to $i + 1$. By Line 1 (resp. 14), the initial (resp. final) value of k is n (resp. $n + 1$). Therefore, for any $1 \leqslant i \leqslant n - 1$, the number of times k is incremented from i to $i + 1$ must be equal to the number of times k is decremented from $i + 1$ to i, in other words: $b_i = a_{i+1}$. Thus, the total number of loop iterations is:

$$\sum_{i=1}^{n} n_i = \sum_{i=1}^{n} (a_i + b_i) = N + 1 + 2 \sum_{i=2}^{n} a_i.$$

Note that because $f(i, 0) = 0$, any partial assignment $\sum_{i=i_0}^{n} f(i, v_i) \leqslant R$ can be extended to a larger partial assignment $\sum_{i=1}^{n} f(i, v_i) \leqslant R$, which implies that $a_1 \geqslant a_2 \geqslant \dots a_n$. It follows that the total number of loop iterations is:

$$\sum_{i=1}^{n+1} n_i \leqslant N + 1 + 2(n - 1)N = (2n - 1)N + 1.$$

For each loop iteration (Lines 3–18), the number of arithmetic operations performed is $O(1)$ and the number of calls to $f()$ is exactly one. It follows that the total number of arithmetic operations is $O(nN + 1)$ and the number of calls to $f()$ is $\leqslant (2n - 1)N + 1$. □

We showed that the number of nodes in the search tree is linear in the number of solutions. Next, we present Algorithm 5.7, which is a counting version of Algorithm 5.6:

Theorem 5.16. *Assume that $f : \{1, \dots, n\} \times \mathbb{N} \to \mathbb{R}$ satisfies $f(i, 0) = 0$ and $f(i, j) \geqslant f(i, j')$ for all i and $j > j'$. Given as input two numbers $R > 0$ and $M > 0$, Algorithm 5.7 decides if is $N \geqslant M$ or $N < M$, where N is the number of $(v_1, \dots, v_n) \in \mathbb{N}^n$ such that $\sum_{i=1}^{n} f(i, v_i) \leqslant R$. Furthermore, if $N \geqslant M$, the number of arithmetic operations is $O(N)$, and otherwise, the number of arithmetic operations is $O(nN + 1)$, and the algorithms outputs N.*

Proof. Similarly to the proof of Theorem 5.15, let n_k denote the number of times we enter Lines 3–17, depending on the value of k, which is $\geqslant 1$ and $\leqslant n$ at each Line 3. Then n_k can be decomposed as $n_k = a_k + b_k$, where a_k (resp. b_k) denotes the number of times we enter Lines 5–9 (resp. Lines 11–16).

Let M be the number of $(v_1, \ldots, v_n) \in \mathbb{N}^n$ such that $\sum_{i=1}^n f(i, v_i) \leqslant R$. If $M \leqslant N$, then Algorithm 5.7 will perform the same operations as Algorithm 5.6 (except Line 6), so the cost is $O(nM + 1) \leqslant O(nN + 1)$ arithmetic operations. Otherwise, $M > N$, which means that the while loop will stop after exactly N iterations, and the total number of operations is therefore $O(N)$. □

Our main result states that if the function f is integral, given any M, Algorithm 5.8 finds the best N assignments in time M where $M \leqslant N \leqslant (n+1)M$:

Theorem 5.17. *Assume that $f : \{1, \ldots, n\} \times \mathbb{N} \to \mathbb{N}$ satisfies $f(i, 0) = 0$ and $f(i, j) < f(i, j')$ for all i and $j > j'$. Assume that $f(i, j) \leqslant j^{O(1)} 2^{n^{O(1)}}$. Given as input a number $M > 1$, Algorithm 5.8 outputs the N assignments $(v_1, \ldots, v_n) \in \mathbb{N}^n$ which minimize $\sum_{i=1}^n f(i, v_i)$ in time $O(n(n+1)M) + n^{O(1)} + O(\log_2 M)$, where the number N satisfies: $M \leqslant N \leqslant (n+1)M$.*

Proof. We have the following invariant at the beginning of each loop iteration: the number of $(v_1, \ldots, v_n) \in \mathbb{N}^n$ such that $\sum_{i=1}^n f(i, v_i) \leqslant R_0$ is $< M$, and the the number of $(v_1, \ldots, v_n) \in \mathbb{N}^n$ such that $\sum_{i=1}^n f(i, v_i) \leqslant R_1$ is $\geqslant M$. Initially, this holds because the number of $(v_1, \ldots, v_n) \in \mathbb{N}^n$ such that $\sum_{i=1}^n f(i, v_i) \leqslant 0$ is 1 and the number of $(v_1, \ldots, v_n) \in \mathbb{N}^n$ such that $\sum_{i=1}^n f(i, v_i) \leqslant \sum_{i=1}^n f(i, \lceil M^{1/n} \rceil)$ is $\geqslant (M^{1/n})^n = M$. Furthermore, the loop preserves the invariant by definition of the loop. Since the length $R_1 - R_0$ decreases by a factor two, it follows that the number of loop iterations is $\leqslant \log_2(\sum_{i=1}^n f(i, \lceil M^{1/n} \rceil))$.

After the loop, we must have $R_0 = R_1 - 1$. Let N_1 (resp. N_0) be the number of $(v_1, \ldots, v_n) \in \mathbb{N}^n$ such that $\sum_{i=1}^n f(i, v_i) \leqslant R_1$ (resp. R_0) after the loop. By the invariant, we know that $N_0 < M \leqslant N_1$. We claim that $(N_1 - N_0) \leqslant nM$, which implies that $N_1 \leqslant (n+1)M$. Notice that $N_1 - N_0$ is the number of $(v_1, \ldots, v_n) \in \mathbb{N}^n$ such that $\sum_{i=1}^n f(i, v_i) = R_1$. For any such assignment, one of the v_i's must be $\geqslant 1$: if we decrement that v_i, we get a cost $< R_1$, so it must be $\leqslant R_0$ because $R_0 = R_1 - 1$, which means that this assignment is counted by N_0. Since we have at most n possibilities for i, it follows that $N_1 - N_0 \leqslant nM$. □

Furthermore, Algorithm 5.8 uses negligible space, except that the output is linear in M: the best tags are actually output as a stream. If we sort the N tags, which requires space, we could output exactly the best M tags.

5.5 Quantum Speed-up of Discrete Pruning

We present a quadratic quantum speed-up for discrete pruning, namely:

Theorem 5.18. *There is a quantum algorithm which, given $\varepsilon > 0$, a number $M > 0$, and an LLL-reduced basis B of a full-rank lattice $\mathcal{L}$ in $\mathbb{Z}^n$, outputs the shortest non-zero vector in $\mathcal{L} \cap P$ in time $O(n^2\sqrt{M})\,\mathrm{poly}(\log(n), \log(M), \log(1/\varepsilon), \beta)$ with error probability ε. Here, β denotes the bitsize of the vectors of B, $P = \bigcup_{\mathbf{t} \in U} \mathcal{C}_{\mathbb{N}}(\mathbf{t})$ where $\mathcal{C}_{\mathbb{N}}$ is the natural partition with respect to B, U is formed by the N tags $\mathbf{t}$ minimizing $\mathbb{E}\{\mathcal{C}_{\mathbb{N}}(\mathbf{t})\}$, for some $M \leqslant N \leqslant 32n^2M$ with probability at least $1 - \varepsilon/2$. If the algorithm is further given a target $\mathbf{u} \in \mathbb{Z}^n$, it also outputs the shortest vector in $(\mathcal{L} - \mathbf{u}) \cap P$.*

By comparison, opening all the cells returned by Algorithm 5.8 of Section 5.4 does the same in $O(M)$ poly-time operations, except that the upper bound on N is slightly lower.

Algorithm 5.6 Enumeration of low-cost assignments

Input: A function $f : \{1, \ldots, n\} \times \mathbb{N} \to \mathbb{R}_{\geqslant 0}$ such that $f(i, 0) = 0$ and $f(i, j) \geqslant f(i, j')$ for all i and $j > j'$; a bound $R > 0$.

Output: All $(v_1, \ldots, v_n) \in \mathbb{N}^n$ such that $\sum_{i=1}^{n} f(i, v_i) \leqslant R$.

$v_1 = v_2 = \cdots = v_n = 0$ and $\rho_{n+1} = 0$ and $k = n$
while true **do**
  $\rho_k = \rho_{k+1} + f(k, v_k)$ *// cost of the tag* $(0, \ldots, 0, v_k, \ldots, v_n)$
  if $\rho_k \leqslant R$ **then**
    if $k = 1$ **then**
      return $(v_1, \ldots, v_n)$; (*solution found*)
      $v_k \leftarrow v_k + 1$
    else
      $k \leftarrow k - 1$ and $v_k \leftarrow 0$ *// going down the tree*
    end if
  else
    $k \leftarrow k + 1$ *// going up the tree*
    if $k = n + 1$ **then**
      exit (*no more solutions)*
    else
      $v_k \leftarrow v_k + 1$
    end if
  end if
end while

The proof of Theorem 5.18 has two parts: first, we show how to determine the best N cells without computing them, for some N close to M, with high probability; then we find the best candidate inside these N cells. Both rely on a tree interpretation. Algorithm 5.6 can be seen as a backtracking algorithm on a tree $\mathcal{T}(R)$, where each node can be encoded as $(*, \cdots, *, v_k, \cdots, v_n)$. The root is encoded as $(*, \cdots, *)$. Given a node $(*, \cdots, *, v_k, \cdots, v_n)$, if $k = 1$, then it is a leaf. If $\sum_{i=k}^{n} f(i, v_i) > R$, then it is also a leaf. If $\sum_{i=k}^{n} f(i, v_i) \leqslant R$, then its children are $(*, \cdots, *, v_{k-1}, v_k, \cdots, v_n)$, where v_{k-1} can take all integer values between 0 and $\rho_{v_k, \cdots, v_n}$. Here $\rho_{v_k, \cdots, v_n}$ is the smallest integer such that $f(i-1, \rho_{v_k, \cdots, v_n}) + \sum_{i=k}^{n} f(i, v_i) > R$. In case of discrete pruning, f is quadratic. We can compute $\rho_{v_k, \cdots, v_n}$ and build the black-box on $\mathcal{T}(R)$.

5.5.1 Determining the best cells implicitly

Given a number $M > 0$, Algorithm 5.8 finds (in time essentially M) the best N vectors $\mathbf{t} \in \mathbb{N}^n$ (for some N close to M) minimizing $\mathbb{E}\{\mathcal{C}_{\mathbb{N}}(\mathbf{t})\} = \sum_{i=1}^{n} \left(\frac{t_i^2}{4} + \frac{t_i}{4} + \frac{1}{12}\right) \|\mathbf{b}_i^\star\|^2$ by minimizing instead the function:

$$g(v_1, \cdots, v_n) = \sum_{i=1}^{n} f(i, v_i) = \sum_{i=1}^{n} v_i(v_i + 1)\|\mathbf{b}_i^\star\|^2 = \sum_{i=1}^{n} \alpha_i v_i(v_i + 1).$$

This is done by finding a suitable radius R by dichotomy, based on logarithmically many calls to Algorithm 5.7 until the number of solutions is close to M, and eventually enumerating the

Algorithm 5.7 Counting low-cost assignments

Input: A function $f : \{1, \dots, n\} \times \mathbb{N} \to \mathbb{R}_{\geqslant 0}$ such that $f(i, 0) = 0$ and $f(i, j) \geqslant f(i, j')$ for all i and $j > j'$; a bound $R > 0$ and a number $M \geqslant 0$.

Output: Decide if the number of $(v_1, \dots, v_n) \in \mathbb{N}^n$ such that $\sum_{i=1}^{n} f(i, v_i) \leqslant R$ is $\geqslant M$ or $< M$.

$v_1 = v_2 = \cdots = v_n = 0$ and $\rho_{n+1} = 0$ and $k = n$ and $m = 0$
while $m < M$ **do**
  $\rho_k = \rho_{k+1} + f(k, v_k)$ // *cost of the tag* $(0, \dots, 0, v_k, \dots, v_n)$
  if $\rho_k \leqslant R$ **then**
    if $k = 1$ **then**
      $m \leftarrow m + 1$ and $v_k \leftarrow v_k + 1$ (*one more solution*)
    else
      $k \leftarrow k - 1$ and $v_k \leftarrow 0$ // *going down the tree*
    end if
  else
    $k \leftarrow k + 1$ // *going up the tree*
    if $k = n + 1$ **then**
      return $m < M$ // *no more solutions*
    else
      $v_k \leftarrow v_k + 1$
    end if
  end if
end while
return $m \geqslant M$

Algorithm 5.8 Enumeration of lowest-cost assignments

Input: A function $f : \{1, \dots, n\} \times \mathbb{N} \to \mathbb{R}_{\geqslant 0}$ such that $f(i, 0) = 0$ and $f(i, j) \geqslant f(i, j')$ for all i and $j > j'$; a number $M > 0$.

Output: Output the N assignments $(v_1, \dots, v_n) \in \mathbb{N}^n$ that minimize $\sum_{i=1}^{n} f(i, v_i)$, where $M \leqslant N \leqslant nM$.

$R_0 \leftarrow 0$ and $R_1 \leftarrow \sum_{i=1}^{n} f(i, \lceil M^{1/n} \rceil)$;
while $R_0 < R_1 - 1$ **do**
  Call Algorithm 5.7 with $R = \lfloor (R_0 + R_1)/2 \rceil$ and M
  if number of solutions $\geqslant M$ **then**
    $R_1 \leftarrow R$
  else
    $R_0 \leftarrow R$
  end if
end while
Call Algorithm 5.6 with R_1.

marked leaves of a search tree by Algorithm 5.6. Both Algorithm 5.6 and Algorithm 5.7 can be viewed as algorithms exploring a tree $\mathcal{T}(R)$ depending on a radius $R > 0$: Algorithm 5.7 decides if the number $\#S(\mathcal{T}(R))$ of marked leaves (*i.e.* the number of outputs returned by

Algorithm 5.6) is $\geqslant$ or $<$ than an input number; Algorithm 5.6 returns all the marked leaves.

This tree interpretation gives rise to Algorithm 5.9, which is our quantum analogue of Algorithm 5.8 with the following differences: we are only interested in finding a suitable radius R such that $N = \#S(\mathcal{T}(R))$ is close to M up to a factor of $32n^2$, with correctness probability at least $1-\varepsilon/2$, because enumerating all the marked leaves would prevent any quadratic speed up. We replace Algorithm 5.7 by the quantum tree size estimation algorithm of [AK17]: this gives a quadratic speed up, but approximation errors slightly worsens the upper bound on N. The input $(\alpha_1, \cdots, \alpha_n)$ of Algorithm 5.9 corresponds to $(\|\mathbf{b}_1^\star\|^2, \cdots, \|\mathbf{b}_n^\star\|^2)$, where $(\mathbf{b}_1, \cdots, \mathbf{b}_n)$ is an integer basis. We know that $(\|\mathbf{b}_1^\star\|^2, \cdots, \|\mathbf{b}_n^\star\|^2) \in \mathbb{Q}^n$, but by suitable multiplication preserving polynomial sizes, we may assume that $(\|\mathbf{b}_1^\star\|^2, \cdots, \|\mathbf{b}_n^\star\|^2) \in \mathbb{N}^n$. The order between the $\|\mathbf{b}_i^\star\|^2$'s doesn't matter in our analysis. We can assume that $\|\mathbf{b}_1^\star\|^2 \leqslant \cdots \leqslant \|\mathbf{b}_n^\star\|^2$. We show

Algorithm 5.9 Computing implicitly the best cells quantumly

Input: $\varepsilon, M > 0$ and $(\alpha_1, \cdots, \alpha_n) \in \mathbb{N}^n$ with $\alpha_1 \leqslant \cdots \leqslant \alpha_n$ such that the input $f : \{1, \cdots, n\} \times \mathbb{N} \to \mathbb{N}$ of Algorithm 5.6 satisfies $f(i, x) = \alpha_i x(x+1)$

Output: R such that $M \leqslant \#S(\mathcal{T}(R)) \leqslant 32n^2M$ with probability $\geqslant 1-\varepsilon$

1: $r \leftarrow \lceil \log_2(\sum_{i=1}^n f(i, \lceil (4nM)^{1/n} \rceil)) \rceil$ and $R \leftarrow \sum_{i=1}^n f(i, \lceil (4nM)^{1/n} \rceil)$ and $R_0 \leftarrow 0$ and $R_1 \leftarrow R$
2: **while** $R_1 - R_0 > 1$ **do**
3: Call **TreeSizeEstimation**$(\mathcal{T}_2(R), 16n^2M, 1/2, \varepsilon r/2, 2)$
4: **if** the answer is "$\mathcal{T}_2(R)$ contains more than $16n^2M$ vertices" **then**
5: $R_1 \leftarrow R$ and $R \leftarrow \lfloor (R_0 + R_1)/2 \rceil$
6: **else if** the answer is "$\mathcal{T}_2(R)$ contains $\hat{T}$ vertices" with $\hat{T} < 3(2n-1)M$ **then**
7: $R_0 \leftarrow R$ and $R \leftarrow \lfloor (R_0 + R_1)/2 \rceil$
8: **else**
9: Return R
10: **end if**
11: **end while**
12: Return R_0

that Algorithm 5.9 finds a radius R corresponding to the best M cells in approximately $\sqrt{M}$ quantum operations:

Theorem 5.19. *The output R of Algorithm 5.9 satisfies $M \leqslant \#S(\mathcal{T}(R)) \leqslant 32n^2M$ with probability $\geqslant 1-\varepsilon/2$. Algorithm 5.9 runs in quantum time*

$$O(n^2\sqrt{M}\,\mathrm{poly}(\log(n), \log(M), \log(1/\varepsilon), \beta))$$

where β is the bitsize of the basis vectors $(\mathbf{b}_1, \cdots, \mathbf{b}_n)$. The algorithm needs

$$O(\mathrm{poly}(n, \log(M), \log(1/\varepsilon)))$$

qubits.

In order to prove the theorem, we will need the two following lemmas:

Lemma 5.20. *For all $R \in \mathbb{N}$, we have $\frac{\#\mathcal{T}_2(R)-2}{2(2n-1)} \leqslant \#S(\mathcal{T}_2(R)) \leqslant \#\mathcal{T}_2(R)$. We also have $\#\mathcal{T}_2(R+1) \leqslant 2n\#\mathcal{T}_2(R)$.*

Proof of Lemma 5.20. Under the transformation, the number of tags that we find in the tree with the parameter R won't change, *i.e.* $\#S(\mathcal{T}(R)) = \#S(\mathcal{T}_2(R))$.

Since we have: $\frac{\#\mathcal{T}(R)-1}{2n-1} \leqslant \#S(\mathcal{T}(R)) \leqslant \#\mathcal{T}(R)$ and we also know: $\#\mathcal{T}(R) \leqslant \#\mathcal{T}_2(R) \leqslant 2\#\mathcal{T}(R)$

We thus have $\frac{\#\mathcal{T}_2(R)-2}{2(2n-1)} \leqslant \#S(\mathcal{T}_2(R)) \leqslant \#\mathcal{T}_2(R)$

Now we will prove the second inequality. If there exists $(*, \cdots, *, v_k, \cdots, v_n) \in \mathcal{T}(R)$ where $v_k \neq 0$ such that $\sum_{j=k}^{n} f(j, v_j) = R+1$, then $(*, \cdots, *, v_k+1, \cdots, v_n) \in \mathcal{T}(R+1)\backslash\mathcal{T}(R)$. And for all $i \in [|1, k-1|]$, $(*, \cdots, *, v_i \in \{0,1\}, 0, \cdots, 0, v_k, \cdots, v_n) \in \mathcal{T}(R+1)\backslash\mathcal{T}(R)$.

$(*, \cdots, *, v_k+1, \cdots, v_n)$ generates two nodes in $\mathcal{T}_2(R+1)\backslash\mathcal{T}_2(R)$.

Each $(*, \cdots, *, v_i \in \{0,1\}, 0, \cdots, 0, v_k, \cdots, v_n)$ generates one node in $\mathcal{T}_2(R+1)\backslash\mathcal{T}_2(R)$.

On the other hand, a node in $\mathcal{T}_2(R+1)\backslash\mathcal{T}_2(R)$ can only be derived from a node $(*, \cdots, *, v_k, \cdots, v_n) \in \mathcal{T}(R)$ (and thus from the equivalent node in $\mathcal{T}_2(R)$) such that $\sum_{j=k}^{n} f(j, v_j) = R+1$ and $v_k \neq 0$, by using the above processus.

Therefore, $\#\mathcal{T}_2(R+1) \leqslant 2n\#\mathcal{T}_2(R)$. □

Lemma 5.21. *d can be upper-bounded by $\frac{\sum_{i=1}^{n}\alpha_i}{\alpha_1}\lceil(4nM)^{1/n}\rceil)$.*

Proof of Lemma 5.21. At the beginning,

$$\begin{aligned} R = \sum_{i=1}^{n} f(i, \lceil(4nM)^{1/n}\rceil) &= (\sum_{i=1}^{n}\alpha_i)\lceil(4nM)^{1/n}\rceil)(\lceil(4nM)^{1/n}\rceil)+1) \\ &< \sum_{i=1}^{n}\alpha_i\lceil(4nM)^{1/n}\rceil)(\frac{\sum_{i=1}^{n}\alpha_i}{\alpha_1}\lceil(4nM)^{1/n}\rceil))+1) \\ &= f(1, \frac{\sum_{i=1}^{n}\alpha_i}{\alpha_1}\lceil(4nM)^{1/n}\rceil)) = g(\frac{\sum_{i=1}^{n}\alpha_i}{\alpha_1}\lceil(4nM)^{1/n}\rceil)), 0, \cdots, 0) \end{aligned}$$

Since $\alpha_1 \leqslant \cdots \leqslant \alpha_n$, we also have: for all $1 \leqslant j \leqslant n$,
$R < g(0, \cdots, 0, \frac{\sum_{i=1}^{n}\alpha_i}{\alpha_1}\lceil(4nM)^{1/n}\rceil), 0, \cdots, 0)$ where $\frac{\sum_{i=1}^{n}\alpha_i}{\alpha_1}\lceil(4nM)^{1/n}\rceil)$ is on the j^{th} position.

Since R decreases during the execution of the algorithm, d can be upper-bounded by $\frac{\sum_{i=1}^{n}\alpha_i}{\alpha_1}\lceil(4nM)^{1/n}\rceil)$. □

Proof of Theorem 5.19. We will prove that the output R satisfies $2(2n-1)M \leqslant \#\mathcal{T}_2(R) \leqslant 32n^2M$ with probability at least $1-\varepsilon/2$. Since we have $\frac{\#\mathcal{T}(R)-2}{2(2n-1)} \leqslant \#S(\mathcal{T}_2(R)) \leqslant \#\mathcal{T}_2(R)$, this proves that $M \leqslant \#S(\mathcal{T}(R)) \leqslant 32n^2M$ with probability at least $1-\varepsilon/2$.

Since the algorithm will end after at most $Round = \lceil\log_2(\sum_{i=1}^{n} f(i, \lceil(4nM)^{1/n}\rceil))\rceil$ calls of the tree size estimation algorithm with correctness probability at least
$1-\varepsilon/2(\lceil\log_2(\sum_{i=1}^{n} f(i, \lceil(4nM)^{1/n}\rceil))\rceil)$, by using the union bound, the correctness probability of our algorithm is at least $1-\varepsilon/2$. Now we can assume that all the answers of the tree size estimation algorithm are correct.

In the following we will omit the last four parameters in **TreeSizeEstimation** for the clarity of the proof.

In case that the algorithm returns R inside the **while** loop, the output of the first **TreeSizeEstimation**$(\mathcal{T}_2(R))$ is "$\mathcal{T}_2(R)$ contains $\hat{T}$ vertices" with
$3(2n-1)M \leqslant \hat{T} \leqslant 16n^2M$. Since the tree size estimation is up to precision $1 \pm 1/2$, the real tree size should be in the interval $[\frac{3(2n-1)N}{1+1/2}, \frac{16n^2M}{1-1/2}] \subset [2(2n-1)M, 32n^2M]$.

In case that the algorithm returns R after the **while** loop, we have $R_1 = R_0 + 1$. The estimation of **TreeSizeEstimation**$(\mathcal{T}(R_1))$ with the parameters as in the while loop is "$\mathcal{T}(R_1)$ contains more than $16n^2M$ vertices". Since the precision parameter is $1/2$, we have $\#\mathcal{T}_2(R_1) \geqslant \frac{16n^2M}{1+1/2} > 8n^2M$.

The estimation of **TreeSizeEstimation**$(\mathcal{T}(R_0))$ with the parameters as in the while loop is "$\mathcal{T}(R_0)$ contains $\hat{T}$ vertices" with $\hat{T} < \lceil 3(2n-1)M \rceil$. Since the precision parameter is $1/2$, we have $\#\mathcal{T}_2(R_0) \leqslant \frac{3(2n-1)M}{1-1/2} = 6(2n-1)M$.

By using Lemma 5.20, we know that for all $R \in \mathbb{N}$, $\#\mathcal{T}(R+1) \leqslant 2n\#\mathcal{T}(R)$.Thus, there exists $R > 0$ such that $2(2n-1)M \leqslant \#\mathcal{T}_2(R) \leqslant 4n(2n-1)M < 8n^2M$. This R should be R_0.

We proved that in each case, Algorithm 5.9 outputs R such that $2(2n-1)M \leqslant \#\mathcal{T}(R) \leqslant 32n^2M$. Therefore, R satisfies $M \leqslant \#S(\mathcal{T}(R)) \leqslant 32n^2M$ with probability at least $1 - \varepsilon/2$.

The number of queries to the trees is $O(\sqrt{n\log(d)16n^2M}\log^2(Round/\varepsilon) * Round) = O(n^2\sqrt{M}\,\mathrm{poly}(\log(n), \log(M), \log(1/\varepsilon), \beta))$. Since each query needs $O(\log(16n^2M)) = \mathrm{poly}(\log(n), \log(M))$ non-query transformations, the time complexity of Algorithm 5.9 is $O(n^2\sqrt{M}\,\mathrm{poly}(\log(n), \log(M), \log(1/\varepsilon), \beta))$.

The algorithm needs $O(\mathrm{poly}(n, \log(M), \log(1/\varepsilon)))$ qubits by using Theorem 2.12. □

5.5.2 Finding the best lattice vector

We now know R such that the number N of $(v_1, \cdots, v_n) \in \mathbb{N}^n$ which satisfies $\sum_{i=1}^n f(i, v_i) \leqslant R$ is in $[M, 32n^2M]$ with probability at least $1 - \varepsilon/2$. All these solutions are leaves of the tree $\mathcal{T}(R)$ and they form the set U of the best N tags minimizing $\mathbf{t}$ minimizing $\mathbb{E}\{\mathcal{C}_{\mathbb{N}}(\mathbf{t})\}$. Let $P = \bigcup_{\mathbf{t}\in U} \mathcal{C}_{\mathbb{N}}(\mathbf{t})$ where $\mathcal{C}_{\mathbb{N}}$ is the natural partition with respect to the input basis B. We would like to find a shortest non-zero vector in $\mathcal{L} \cap P$ for the SVP setting, or the shortest vector in $(\mathcal{L} - \mathbf{u}) \cap P$ in the CVP setting, when we are further given target $\mathbf{u} \in \mathbb{Z}^n$. To do this, we notice that it suffices to apply **FindMin2** (in App), provided that the basis $(\mathbf{b}_1, \cdots, \mathbf{b}_n)$ is LLL-reduced. More precisely, we call **FindMin2**$(\mathcal{T}(R), \mathcal{P}, h, \|\mathbf{b}_1\|^2, d, 32n^2M, \varepsilon/2)$. Here $\mathcal{P}$ is the predicate which returns true on a node iff it is a leaf encoded as $(x_1, \cdots, x_n)$ such that $g(x_1, \cdots, x_n) = \sum_{i=1}^n f(i, x_i) \leqslant R$. $h_V(x_1, \cdots, x_n)$ is the predicate which indicates if the square of the norm of the lattice vector in the cell of tag $(x_1, \cdots, x_n)$ is $\leqslant V$. The time complexity is $O(n^2\sqrt{M}\,\mathrm{poly}(\log(n), \log(M), \log(1/\varepsilon), \beta))$.

Since the subroutine of determining the best cells and the one of finding a shortest non-zero vector, both have an error probability $\varepsilon/2$, by union bound, the total error probability is ε. We thus have proved Theorem 5.18.

5.5.3 The Case of Extreme Pruning

In this section, we explain how to tackle the extreme pruning case, where one wants to run discrete pruning over many reduced bases. We only give a proof sketches since the main ideas are the same as previously.

Given m LLL-reduced bases $(\mathbf{B}_1, \cdots, \mathbf{B}_m)$ of the same integer lattice $\mathcal{L}$ of rank n, we define for each basis $\mathbf{B}_i$ a function $g_i : \mathbb{N}^n \to \mathbb{Q}$ such that $g_i(x_1, \cdots, x_n) = \sum_{j=1}^n \|\mathbf{b}^\star_{i,j}\|^2 x_i(x_i+1)$, where $(\mathbf{b}^\star_{i,1}, \cdots, \mathbf{b}^\star_{i,n})$ is the Gram-Schmidt orthogonalization of the basis $\mathbf{B}_i$. Here, we want to first find the $\mathrm{poly}(n)M$ best cells with respect to all of the functions g_i altogether, and then find the shortest vector in these cells. Both steps have complexity $O(\sqrt{M}\,\mathrm{poly}(n, \log M, \log 1/\varepsilon, \beta))$,

where ε is the total error probability and where β is the bitsize of the vectors of the input bases.

Theorem 5.22. *There is a quantum algorithm which, given $\varepsilon > 0$, a number $M > 0$, and m LLL-reduced bases $(\mathbf{B}_1, \cdots, \mathbf{B}_m)$ of an n-rank integer lattice $\mathcal{L}$, outputs the shortest non-zero vector in $\mathcal{L} \cap P$ in time $O(\sqrt{M}\,\mathrm{poly}(n, \log M, \log 1/\varepsilon, \beta))$ with error probability ε. Here, β denotes the maximum bitsize of the vectors of all given bases, $P = \bigcup_{(i,\mathbf{t})\in U} \mathcal{C}_{\mathbb{N}}(i, \mathbf{t})$ where $\mathcal{C}_{\mathbb{N}}(i, \cdot)$ is the natural partition with respect to B_i, U is formed by the N tuples $(i, \mathbf{t}) \in \{1, \cdots, m\} \times \mathbb{N}^n$ minimizing $g_i(\mathbf{t})$ among all tuples, for some $N = \mathrm{poly}(n) \cdot M$ with probability at least $1 - \varepsilon/2$. If the algorithm is further given a target $\mathbf{u} \in \mathbb{Z}^n$, it also outputs the shortest vector in $(\mathcal{L} - \mathbf{u}) \cap P$.*

The main idea of the proof is the following. For each basis $\mathbf{B}_i$, there is a backtracking tree with respect to the function g_i as we explained in the previous section. We put all these trees together and obtain one single tree. We first apply the **TreeSizeEstimation** algorithm several times to find a good common radius R for all functions g_i by dichotomy, such that the total number of good cells in all trees is $\mathrm{poly}(n) \cdot M$. After that, we apply **FindMin2** to find the shortest vector among all these cells. Remark that in the previous section, we required the function g to have integral values, and this was achieved by multiplying all $\|\mathbf{b}_i^\star\|^2$ by a common denominator. Instead, we here want to keep the output rational, which is proved sufficient by the following lemma:

Lemma 5.23. *Given a basis $(\mathbf{b}_1, \cdots, \mathbf{b}_n)$ of an integer lattice $\mathcal{L}$, $g : \mathbb{N}^n \to \mathbb{Q}$ such that $g(x_1, \cdots, x_n) = \sum_{i=1}^n \|\mathbf{b}_i^\star\|^2 x_i(x_i + 1)$, we denote $\mathcal{T}(R)$ the backtracking tree for finding all solutions of $g(x_1, \cdots, x_n) \leqslant R$, $\mathcal{T}_2(R)$ the corresponding binary tree. For all $R \in \mathbb{R}^+$, $\#S(\mathcal{T}_2(R+\delta)) \leqslant 2n\#S(\mathcal{T}_2(R))$, where $\delta = \frac{1}{\prod_{i=1}^n \Delta_i}$ and $\Delta_i = \mathrm{covol}(\mathbf{b}_1, \cdots, \mathbf{b}_i)^2 = \prod_{j=1}^i \|\mathbf{b}_i^\star\|^2$.*

The proof of this lemma is the same as the proof of Lemma 5.20 by noticing that $\prod_{i=1}^n \Delta_i$ is a common denominator of all $\|\mathbf{b}_i^\star\|^2$.

For each basis $\mathbf{B}_i$, we define δ_i as in Lemma 5.23. In the dichotomy step, we stop when the difference of the two terms is smaller than $\min_{j\in\{1,\cdots,m\}} \delta_j$. The other steps are the same as in the previous section.

5.6 Impact

In this section, we compare the asymptotical complexity of quantum enumeration with extreme cylinder pruning and quantum sieving. We stress that this is just a first step, since we omitted the polynomial overhead factors and only use the asymptotic formula. Note that the polynomial factors in quantum sieve have not been investigated either. Such factors play an important role in security estimates and further studies will be necessary to get a full picture. If one is interested in more precise estimates, such as the number of quantum gates, one would need to assess the quantum cost of the algorithm of Montanaro [Mon15] and that of Ambainis and Kokainis [AK17]. Furthermore, one needs to assess the explicit quantum circuit complexity of the oracle which gives local access to the tree used in the enumeration. Another point which is worth noticing is that quantum sieving algorithms use QRACM gates whereas quantum enumeration with extreme cylinder pruning is only in the plain quantum circuit model.

The cost of cylinder pruning depends on the quality of the basis used. [CN11, AWHT16] discussed two models of strongly reduced bases: the HKZ model which is closer to the state-of-the-art, and the Rankin model which provides conservative bounds by anticipating progress in lattice reduction. In Figure 5.5, the red and yellow curves show $\sqrt{\#bases * N}$ where N is an upper bound cost, i.e., number of nodes of enumeration with extreme pruning with probability $1/\#bases$. The upper bounds for HKZ/Rankin bases are computed by the method of [ANSS18]. The dotted blue curve corresponds to the asymptotical best known quantum sieving complexity without polynomial factor which is $2^{0.265n}$.

Quantum enumeration with extreme pruning for solving the shortest vector problem would be faster than quantum sieve up to higher dimensions than previously thought, around 300 if we assume that 10^{10} quasi-HKZ-bases can be obtained for a total cost similar to the total cost of quantum enumeration on the same number of quasi-HKZ-bases, or beyond 400 if 10^{10} Rankin-bases can be used instead.

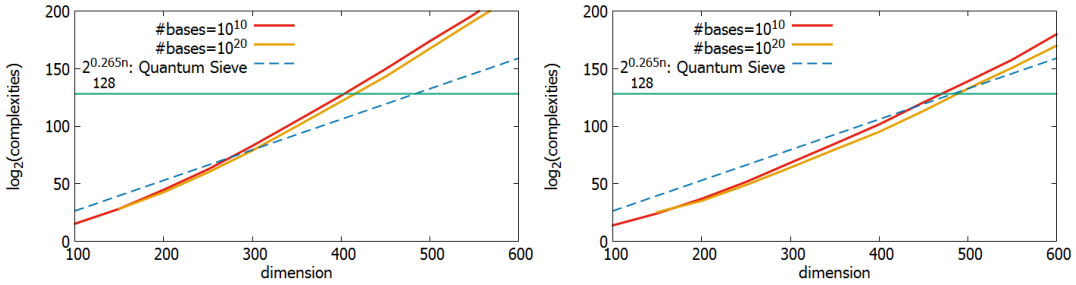


Figure 5.5: Q-sieve vs Q-enum: (Left) Using HKZ bases (Right) Using Rankin bases

In a recent work [ABLR20], using recent technical progress on speeding-up lattice enumeration in BKZ, ie, relaxing (the search radius of) enumeration and extended preprocessing which preprocesses in a larger rank than the enumeration rank k, the authors provide a faster BKZ algorithm using lattice enumeration which achieves the same root Hermite factor (RHF) with respect to k[2] as previous BKZ algorithms. Their updated extrapolation of the crossover rank between a square-root cost estimate for quantum enumeration using our algorithm and the Core-SVP cost estimate ($2^{0.265k}$) for quantum sieving is 547.

Furthermore, we note that our quantum speedup might actually be more than quadratic. Indeed, the number T of enumeration nodes is a random variable: the average quantum running time is $\mathbb{E}(\sqrt{T})$, which is $\leqslant \sqrt{\mathbb{E}(T)}$ and potentially much less (*e.g.* a log-normal distribution). It would be useful to identify the distribution of T: it cannot be log-normal for LLL bases (unlike what seems to be suggested in [YD17]), because it would violate the provable running time $2^{O(n^2)}$ of enumeration with LLL bases. Indeed, in [YD17], the authors showed that the complexity of enumeration on a random n-dimensional BKZ$_k$-reduced basis should be of the shape

$$\exp(n^2 x(k) + y(k) \pm n^{1.5} l \cdot z(k))$$

except a fraction at most $\exp(-l^2/2)$ of random bases, for some constants $x(k), y(k), z(k)$ depending on k. However there seems to be a mistake, because Chebyshev's inequality only guarantees the result for a fraction at most $1 - 1/l^2$ of random bases. By taking $l = O(\sqrt{n})$,

[2]The RHF essentially caracterizes how well a lattice basis is reduced.

we obtain that the enumaration on LLL-reduced basis ($k = 2$) has complexity $2^{O(n^2)}$ only for a guaranteed fraction $1 - 1/\Omega(n)$ of random basis.

Chapter 6
The Subset Sum Problem

The work in this chapter has been published at ASIACRYPT 2020 [BBSS20] and is a joint work with Xavier Bonnetain, Rémi Bricout and André Schrottenloher.

6.1 Introduction

In this chapter, we study the *subset-sum problem*, also known as *knapsack problem*: given n integers $\mathbf{a} = (a_1, \dots a_n)$ and a target integer S, find an n-bit vector $\mathbf{e} = (e_1, \dots e_n) \in \{0,1\}^n$ such that $\mathbf{e} \cdot \mathbf{a} = \sum_i e_i a_i = S$. The *density* of the knapsack instance is defined as $d = n/(\log_2 \max_i a_i)$ and, for a random instance $\mathbf{a}$, it is related to the number of solutions that one can expect. We focus on the case where $d = 1$, where expectedly a single solution exists. Instead of naively looking for the solution $\mathbf{e}$ via exhaustive search, in time 2^n, Horowitz and Sahni [HS74] proposed to use a meet-in-the-middle approach in $2^{n/2}$ time and memory. The idea is to find a collision between two lists of $2^{n/2}$ subknapsacks, *i.e.* to merge these two lists for a single solution. Schroeppel and Shamir [SS81] later improved this to a 4-list merge, in which the memory complexity can be reduced down to $2^{n/4}$.

The Representation Technique. At EUROCRYPT 2010, Howgrave-Graham and Joux [HJ10] (HGJ) proposed a heuristic algorithm solving *random* subset-sum instances in time $\widetilde{O}\left(2^{0.337n}\right)$, thereby breaking the $2^{n/2}$ bound. Their key idea was to represent the knapsack solution in many different ways, as a sum of vectors in $\{0,1\}^n$. This *representation technique* increases the search space size, allowing to merge more lists, with new arbitrary constraints, thereby allowing for a more time-efficient algorithm. The time complexity exponent is obtained by numerical optimization of the list sizes and constraints, assuming that the individual elements obtained in the merging steps are well-distributed. This is the standard heuristic of classical and quantum subset-sum algorithms. Later, Becker, Coron and Joux [BCJ11] (BCJ) improved the asymptotic runtime down to $\widetilde{O}\left(2^{0.291n}\right)$ by allowing even more representations, with vectors in $\{-1,0,1\}^n$.

The BCJ representation technique is not only a tool for subset-sums: it has been used to speed up generic decoding algorithms, classically [MMT11, BJMM12, MO15] and quantumly [KT17]. Therefore, the subset-sum problem serves as the simplest application of representations, and improving our understanding of the classical and quantum algorithms may have consequences on these other generic problems.

Quantum Algorithms for the Subset-Sum Problem. Cryptosystems based on hard subset-sums are natural candidates for post-quantum cryptography, but to understand precisely their security, we have to study the best generic algorithms for solving subset-sums. The first quantum time speedup for this problem was obtained in [BJLM13], with a quantum time $\widetilde{O}(2^{0.241n})$. The algorithm was based on the HGJ algorithm. Later on, [HM18] devised an algorithm based on BCJ, running in time $\widetilde{O}(2^{0.226n})$. Both algorithms use the corresponding classical merging structure, wrapped in a quantum walk on a Johnson graph, in the MNRS quantum walk framework [MNRS11]. However, they suffer from two limitations.

First, both use the model of *quantum memory with quantum random-access* (QRAQM), which is stronger than the standard quantum circuit model, as it allows fast lookups in superposition of all the qubits in the circuit. The QRAQM model is used in most quantum walk algorithms to date. With a more restrictive model, i.e. *classical memory with quantum random-access* (QRACM), no quantum time speedup over BCJ was previously known. This is not the case for some other hard problems in post-quantum cryptography, *e.g.* heuristic lattice sieving for the Shortest Vector Problem, where the best quantum algorithms to date require only QRACM [Laa15a].

Second, both use a conjecture (implicit in [BJLM13], made explicit in [HM18]) about quantum walk updates. In short, the quantum walk maintains a data structure, that contains a merging tree similar to HGJ (resp. BCJ), with lists of smaller size. A quantum walk step is made of updates that change an element in the lowest-level lists and requires to modify the upper levels accordingly, *i.e.* to track the partial collisions that must be removed or added. In order to be efficient, the update needs to run in polynomial time. Moreover, the resulting data structure shall be a function of the lowest-level list, and not depend on the path taken in the walk. The conjecture states that it should be possible to guarantee sound updates without impacting the time complexity exponent. However, it does not seem an easy task and the current literature on subset-sums lacks further justification or workarounds.

Contributions. We improve classical and quantum subset-sum algorithms based on representations. We write these algorithms as sequences of "merge-and-filter" operations, where lists of subknapsacks are first merged with respect to an arbitrary constraint, then *filtered* to remove the subknapsacks that cannot be part of a solution. In the classical setting, we propose a more time-efficient subset-sum algorithm based on representations. In the quantum setting, we introduce *quantum filtering*, which speeds up the filtering of representations with a quantum search. We obtain the first quantum time speedup that is *not* based on quantum walks and only requires the QRACM model. We also give an improved quantum walk based on quantum filtering and we show how to partially overcome the quantum walk update heuristic, by designing a new data structure for the vertices in the quantum walk. A detailed description of our contributions is available in Section 1.1.4.

6.2 List Merging and Classical Subset-sum Algorithms

In this section, we remain in the classical realm. We first introduce the HGJ $\{0,1\}$ representation technique, then the BCJ extended $\{-1,0,1\}$ representations and finally our extended $\{-1,0,1,2\}$ representations and detail our improvements over BCJ. In doing so, we will introduce the list filtering and merging technique that we use in the rest of this chapter.

Hereafter and in the rest of this chapter, all time and memory complexities, classical and quantum, are exponential in n. We use $\mathrm{negl}(n)$ for any function that vanishes inverse-exponentially in n. We often replace asymptotic exponential time and memory complexities (*e.g.* $\widetilde{O}(2^{\alpha n})$) by their exponents (*e.g.* α). We use capital letters (*e.g.* L) and corresponding letters (*e.g.* ℓ) to denote the same value, in $\log_2$ and relatively to n: $\ell = \log_2(L)/n$. The problem we will solve is defined as follows:

Definition 6.1 (Random subset-sum instance of weight $n/2$)**.** Let $\mathbf{a}$ be chosen uniformly at random from $(\mathbb{Z}_N)^n$, where $N \simeq 2^n$. Let $\mathbf{e}$ be chosen uniformly at random[1] from $\{0,1\}^n$ with Hamming weight $n/2$. Let $t = \mathbf{a} \cdot \mathbf{e} \pmod N$. Then $\mathbf{a}, t$ is a *random subset-sum instance*. A *solution* is a vector $\mathbf{e}'$ such that $\mathbf{a} \cdot \mathbf{e}' = t \pmod N$.

6.2.1 The HGJ algorithm

The representation technique of Howgrave-Graham and Joux [HJ10] consists in generalizing the problem to look for knapsacks with specific distributions of "0s" and "1s". The intuition is that if we have two random knapsacks with a proportion β of "1s" then the sum will have a proportion 2β of "1s", unless some *collisions* occur, *i.e.* the two vectors have a "1" at the same position. The final result must have a distribution $\beta = \frac{1}{2}$ of "1s" and the HGJ algorithm will obtain the result recursively from 8 lists with a proportion $\beta = \frac{1}{8}$ of "1s". At the same time, the algorithm will not only add two solutions but also change the modulo constraint: the final result must give t modulo N but we will typically obtain it from a vector giving s modulo N' and another vector giving $t - s$ modulo N', where N/N' is well-chosen and s is chosen at random. Because of collisions (and to avoid bad cases), it is necessary to generalize the problem from "looking for one solution" to "looking for many solutions". In summary, during intermediate steps of the computations, the HGJ algorithm is solving the following more general problem: given n and β, a modular constraint N' and a target s, find many $\boldsymbol{e}$ with Hamming weight βn such that $\boldsymbol{e} \cdot \boldsymbol{a} = s \bmod N'$.

Definition 6.2 (Distributions of knapsacks (HGJ))**.** A *knapsack* or *subknapsack* is a vector $\mathbf{e} \in \{0,1\}^n$. The set of $\mathbf{e}$ with βn "1" and $(1-\beta)n$ "0" is denoted $D^n[\beta]$.

Note that we always add vectors *over the integers*, and thus, the sum of two vectors of $D^n[*]$ may contain unwanted symbols 2 and is not necessarily a knapsack. In what follows, we will assume that we can efficiently sample from $D^n[\beta]$ for any β, see Section 6.2.5 for more details. We will also need the following estimate on the number of knapsacks.

Property 6.3 (Size of knapsack sets)**.** *We have*

$$\frac{1}{n}\log_2 |D^n[\beta]| \simeq h(\beta)$$

where $h(x) = -x\log_2 x - (1-x)\log_2(1-x)$ *is the* Hamming entropy.

[1] With the notations of this section, this is equivalent to sampling from from $D^n[0, 1/2, 0]$.

Merging and filtering. In the algorithm, we repeatedly sample vectors with certain distributions $D^n[*]$ and then combine them. Let $D_1 = D^n[\beta_1]$, $D_2 = D^n[\beta_2]$ be two input distributions and $D = D^n[\beta]$ a target. Given two lists $L_1 \in D_1^{|L_1|}$ and $L_2 \in D_2^{|L_2|}$, we define:

- the *merged list* $L = L_1 \bowtie_c L_2$ containing all vectors $\mathbf{e} = \mathbf{e_1} + \mathbf{e_2}$ such that: $\mathbf{e_1} \in L_1, \mathbf{e_2} \in L_2$, $(\mathbf{e}_1 + \mathbf{e}_2) \cdot \mathbf{a} = s \mod M$, $s \leqslant M$ is an arbitrary integer and $M \approx 2^{cn}$ (we write $L_1 \bowtie_c L_2$ because s is an arbitrary value, whose choice is without incidence on the algorithm)

- the *filtered list* $L^f = (L \cap \{0,1\}^n) \subseteq L$, containing the vectors with the target distribution of "0s" and "1s".

In general, L is exponentially bigger than L^f and does not need to be written down, as vectors can be filtered on the fly. The algorithms then repeat the merge-and-filter operation on multiple levels, moving towards the distribution $D^n[1/2]$ while increasing the bit-length of the modular constraint, until we satisfy $\mathbf{e} \cdot \mathbf{a} = t \mod 2^n$ and obtain a solution. Note that this merging-and-filtering view that we adopt, where the merged list is repeatedly sampled before an element passes the filter, has some similarities with the ideas developed in the withdrawn article [EM19].

The standard subset-sum heuristic assumes that vectors in L^f are drawn independently, uniformly at random from D. It simplifies the complexity analysis of both classical and quantum algorithms studied in this paper. Note that this heuristic, which is backed by experiments, actually leads to provable probabilistic algorithms in the classical setting (see [BCJ11, Theorem 2]). We adopt the version of [HM18].

Heuristic 6.4. *If input vectors are uniformly distributed in $D_1 \times D_2$, then the filtered pairs are uniformly distributed in D (more precisely, among the subset of vectors in D satisfying the modular condition).*

Filtering Representations. We let $\ell = (1/n)\log_2|L|$, and so on for ℓ_1, ℓ_2, ℓ^f. By Heuristic 6.4, the average sizes of L_1, L_2, L and L^f are related by:

- $\ell = \ell_1 + \ell_2 - c$

- $\ell^f = \ell + \mathsf{pf}$, where pf is negative and $2^{\mathsf{pf}n}$ is the probability that a pair $(\mathbf{e_1}, \mathbf{e_2})$, drawn uniformly at random from $D_1 \times D_2$, satisfies $(\mathbf{e_1} + \mathbf{e_2}) \in D$.

In particular, the occurrence of collisions in L^f is a negligible phenomenon, unless ℓ^f approaches $(\log_2|D|/n) - c$, which is the maximum number of vectors in D with constraint c. For a given random knapsack problem, with high probability, the size of any list built by sampling, merging and filtering remains very close to its average (by a Chernoff bound and a union bound on all lists).

Here, pf depends only on D_1, D_2 and D. Working with this *filtering probability* is especially useful for writing down our algorithm in Section 6.4. The formula for $\{0,1\}$ representations is given by the following lemma. Classically, the time complexity of the merge-and-filter operation is related to the size of the *merged list*. See Lemma 6.6 for more details.

Lemma 6.5 (Filtering HGJ-style representations)**.** *Let* $\mathbf{e_1} \in D^n[\alpha]$ *and* $\mathbf{e_2} \in D^n[\beta]$*. The probability that* $\mathbf{e_1} + \mathbf{e_2} \in D^n[\alpha + \beta]$ *is* $2^{\mathsf{pf}_1(\alpha,\beta)n}$*, with* $\mathrm{bin}(\omega, \alpha) = h(\alpha/\omega)\omega$ *and*

$$\mathsf{pf}_1(\alpha, \beta) = \mathrm{bin}(1 - \alpha, \beta) - h(\beta) = \mathrm{bin}(1 - \beta, \alpha) - h(\alpha)$$

if $\alpha + \beta \leqslant 1$*, and the probability is* 0 *otherwise.*

Proof. The probability that a $\mathbf{e}_1 + \mathbf{e}_2$ survives the filtering is:

$$\binom{n - \alpha n}{\beta n} / \binom{n}{\beta n} = \binom{n - \beta n}{\alpha n} / \binom{n}{\alpha n} \ .$$

Indeed, given a choice of αn bit positions among n, the other βn bit positions must be compatible, hence chosen among the $(1 - \alpha)n$ remaining positions. By taking the $\log_2$, we obtain the formula for the filtering probability. □

Lemma 6.6 (Classical merging with filtering)**.** *Let* L_1 *and* L_2 *be two sorted lists stored in classical memory with random access. In* $\log_2$*, relatively to* n*, and discarding logarithmic factors, merging and filtering* L_1 *and* L_2 *costs a time* $\max(\min(\ell_1, \ell_2), \ell_1 + \ell_2 - c)$ *and memory* $\max(\ell_1, \ell_2, \ell^f)$*, assuming that we must store the filtered output list.*

Proof. Assuming sorted lists, there are two symmetric ways to produce a stream of elements of $L_1 \bowtie_c L_2$: we can go through the elements of L_1, and for each one, find the matching elements in L_2 by dichotomy search (time $\ell_1 + \max(0, \ell_2 - c)$) or we can exchange the role of L_1 and L_2. Although we do not need to store $L_1 \bowtie_c L_2$, we need to examine all its elements in order to filter them. □

HGJ algorithm. We are now ready to recall the algorithm of Howgrave-Graham and Joux [HJ10], with the corrected time complexity pointed out in [BCJ11]. Recall that the basic operation in HGJ is to merge two lists of subknapsacks with a constraint t on cn bits, that is, from two lists L_1 and L_2, obtain the list $L' = \{\mathbf{e}_1 + \mathbf{e}_2, \mathbf{e}_1 \in L_1, \mathbf{e}_2 \in L_2, (\mathbf{e}_1 + \mathbf{e}_2) \cdot \mathbf{a} = t \bmod 2^{cn}\}$. Furthermore, we are interested in the *filtered* list of such representations, in which we remove duplicate representations and sums $\mathbf{e} = \mathbf{e}_1 + \mathbf{e}_2$ where $\mathbf{e}$ is not a binary vector. The length of the filtered list L determines how many representations will survive the constraint t. In HGJ, all lists at a given level have the same size.

The algorithm builds a merging tree of lists of subknapsacks, with four levels, numbered 3 down to 0. Level j contains 2^j lists. In total, 8 lists are merged together into one. This process is illustrated in Figure 6.1.

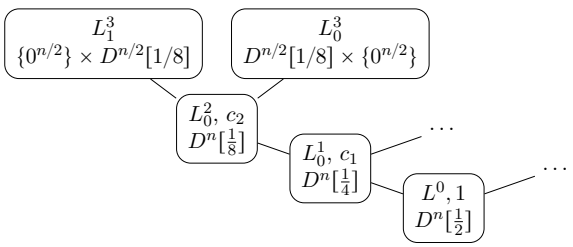


Figure 6.1: The HGJ algorithm (duplicate lists are omitted)

Level 3. We build 8 lists denoted $L_0^3 \dots L_7^3$. They contain *all* subknapsacks of weight $\frac{n}{16}$ on $\frac{n}{2}$ bits, either left or right:

$$\begin{cases} L_{2i}^3 &= D^{n/2}[1/8] \times \{0^{n/2}\} \\ L_{2i+1}^3 &= \{0^{n/2}\} \times D^{n/2}[1/8] \end{cases}$$

From Property 6.3, these level-3 lists have size $\ell_3 = h(1/8)/2$. As the positions set to 1 cannot interfere, these is no filtering when merging L_{2i}^3 and L_{2i+1}^3.

Level 2. We merge the lists pairwise with a (random) constraint on $c_2 n$ bits, and obtain 4 filtered lists. The size of the filtered lists plays a role in the memory complexity of the algorithm, but the time complexity depends on the size of the unfiltered lists.

In practice, when we say "with a constraint on $c_j n$ bits", we assume that given the subset-sum objective t modulo 2^n, random values r_i^j such that $\sum_i r_i^j = t \mod 2^{c_j n}$ are selected at level j, and the r_i^j have $c_j n$ bits only. Hence, at this step, we have selected 4 integers on $c_2 n$ bits $r_0^1, r_1^1, r_2^1, r_3^1$ such that $r_0^1 + r_1^1 + r_2^1 + r_3^1 = t \mod 2^{c_2 n}$. The 4 level-2 lists $L_0^2, L_1^2, L_2^2, L_3^2$ have size $\ell_2 = (h(1/8) - c_2)$, they contain subknapsacks of weight $\frac{n}{8}$ on n bits.

Remark 6.7. The precise values of these r_i are irrelevant, since they cancel out each other in the end. They are selected at random during a run of the algorithm, and although there could be "bad" values of them that affect significantly the computation, this is not expected to happen.

Level 1. We merge the lists pairwise with $(c_1 - c_2)n$ new bits of constraint, ensuring that the constraint is compatible with the previous ones. We obtain two filtered lists L_0^1, L_1^1, containing subknapsacks of weight $n/4$. They have size:

$$\ell_1 = 2\ell_2 - (c_1 - c_2) + \mathsf{pf}_1(1/8, 1/8)$$

where $\mathsf{pf}_1(1/8, 1/8)$ is given by Lemma 6.5.

Level 0. We find a solution to the subset-sum problem with the complete constraint on n bits. This means that the list L^0 must have expected length $2^{\ell_0 n} = 1$ so $\ell_0 = 0$. Note that there remains $(1 - c_1)n$ bits of constraint to satisfy, and the filtering term is similar as before, so:

$$\ell_0 = 2\ell_1 - (1 - c_1) + \mathsf{pf}_1(1/4, 1/4) \ .$$

By Lemma 6.6, the time complexity of this algorithm is determined by the sizes of the unfiltered lists: $\max(\ell_3, 2\ell_3 - c_2, 2\ell_2 - (c_1 - c_2), 2\ell_1 - (1 - c_1))$. The memory complexity depends on the sizes of the filtered lists: $\max(\ell_3, \ell_2, \ell_1)$. By a numerical optimization, one obtains a time exponent of $0.337n$ with the following parameters:

$$\ell_1 = 0.311, \ell_2 = 0.305, \ell_3 = 0.265$$
$$c_1 = 0.5, c_2 = 0.226$$

6.2.2 The BCJ Algorithm

The HGJ algorithm uses representations to increase artificially the search space. The algorithm of Becker, Coron and Joux [BCJ11] improves the runtime exponent down to 0.291 by allowing even more freedom in the representations, which can now contain "-1". The "-1" have to cancel out progressively, to ensure the validity of the final knapsack solution. We need to extend our notion of distribution of knapsack to present this algorithm.

Definition 6.8 (Distributions of knapsacks (BCJ))**.** A *knapsack* or *subknapsack* is a vector $\mathbf{e} \in \{-1, 0, 1, 2\}^n$. The set of $\mathbf{e}$ with αn "-1", $(\alpha + \beta)n$ "1" and $(1 - 2\alpha - \beta)n$ "0" is denoted $D^n[\alpha, \beta]$. This coincides with the notation $D^n[\alpha, \beta]$ from [HM18].

In what follows, we will again assume that we can efficiently sample from $D^n[\alpha, \beta]$ for any α and β, see Section 6.2.5 for more details. We also need the following estimate on the number of such representation

Property 6.9 (Size of knapsack sets)**.** *We have*

$$\frac{1}{n} \log_2 |D^n[\alpha, \beta]| \simeq g(\alpha, \alpha + \beta)$$

where $g(x, y) = -x \log_2 x - y \log_2 y - (1 - x - y) \log_2(1 - x - y)$ *is the* 2-way entropy.

The BCJ algorithm is also based on merging and filtering certain distributions $D^n[*, *]$. Let $D_1 = D^n[\alpha_1, \beta_1]$, $D_2 = D^n[\alpha_2, \beta_2]$ be two input distributions and $D = D^n[\alpha, \beta]$ a target. Given two lists $L_1 \in D_1^{|L_1|}$ and $L_2 \in D_2^{|L_2|}$, we denote by $L = L_1 \bowtie_c L_2$ the merged list and by $L^f = (L \cap \{-1, 0, 1\}^n) \subseteq L$ the filtered list. Note that the filtering is different from before because "-1" is now an acceptable value. We let $\ell = (1/n) \log_2 |L|$, and so on for ℓ_1, ℓ_2, ℓ^f. We assume that Heuristic 6.4 applies to this new representation and we obtain the same estimates $\ell = \ell_1 + \ell_2 - c$ and $\ell^f = \ell + \mathsf{pf}$ where pf is the filtering probability. In the following lemma we compute the filtering probabilty by assuming only symmetric input distributions for simplicity. Indeed, we only use symmetric input distributions in practice in the algorithms, since the formula are already complicated in this case. Thus the following lemma is not more general than Lemma 6.5.

Lemma 6.10 (Filtering BCJ-style representations)**.** *Let* $\mathbf{e_1}, \mathbf{e_2} \in D^n[\alpha, \beta]$ *and* $\gamma \leqslant 2\alpha$*. Then the logarithm of the probability that* $\mathbf{e_1} + \mathbf{e_2} \in D^n[\gamma, 2\beta]$ *is:*

$$\mathsf{pf}_2(\alpha, \beta, \gamma) = \mathrm{bin}(\beta + \alpha, \alpha - \gamma/2) + \mathrm{bin}(\alpha, \alpha - \gamma/2) \\ + \mathrm{trin}(1 - \beta - 2\alpha, \gamma/2, \beta + \gamma/2) - \mathrm{trin}(1, \beta + \alpha, \alpha)$$

where $\mathrm{trin}(\omega, \alpha, \beta) = g(\alpha/\omega, \beta/\omega)\omega$ *is the* trinomial*. Furthermore* $\mathrm{trin}(\omega, \alpha, \beta) \simeq \frac{1}{n} \log_2 \binom{\omega n}{\alpha n, \beta n}$.

Proof. In order to estimate the success probability, we need to estimate the number of well-formed representations, and how they can be decomposed. Given a fixed vector $\mathbf{e_1} \in D^n[\alpha, \beta]$, we count the number of compatible $\mathbf{e_2}$ such that xn positions with a -1 from $\mathbf{e_2}$ are cancelled by a 1 from $\mathbf{e_1}$. As $\mathbf{e_1} + \mathbf{e_2} \in D^n[\gamma, 2\beta]$, there are $\binom{(\beta+\alpha)n}{xn}\binom{\alpha n}{(2\alpha-\gamma-x)n)}\binom{(1-\beta-2\alpha)n}{(\alpha-x)n,(\beta+\gamma-\alpha+x)n}$ such vectors.

Taking the logarithm and the standard approximations, its derivative is $\log\left(\frac{\beta+\alpha-x}{x}\frac{2\alpha-\gamma-x}{\gamma-\alpha+x}\frac{\alpha-x}{\beta+\gamma-\alpha+x}\right)$. This term is strictly decreasing for $0 < x < \gamma - \alpha$, and equals 0 for $x = \alpha - \gamma/2$. Hence, this is the maximum, which correspond to the balanced case. It is equal, up to a polynomial loss, to the total number of compatible vectors. Hence, the log of the number of compatible vectors is $\mathrm{bin}(\beta+\alpha, \alpha-\gamma/2) + \mathrm{bin}(\alpha, \alpha-\gamma/2) + \mathrm{trin}(1-\beta-2\alpha, \gamma/2, \beta+\gamma/2)$. As there are $\mathrm{trin}(1, \beta+\alpha, \alpha)$ vectors in $D^n[\alpha, \beta]$, the lemma holds. □

The BCJ algorithm follows the same structure as the HGJ algorithm, represented in Figure 6.1, but the parameters of the distributions are different and not immediately clear. While the original construction had a complexity exponent of $0.291n$, we notice that it is possible to improve it slightly. We relax the constraints $\ell_j + c_j = g(\alpha_j, 1/2^{j+1})$ enforced in [BCJ11], as only the inequalities $\ell_j + c_j \leqslant g(\alpha_j, 1/2^{j+1})$ are necessary. This idea is implicitly used in [BCDL19], in the context of syndrome decoding. When optimizing the parameters under these new constraints, we bring the asymptotic time exponent down to $0.289n$ with the following parameters:

$$\begin{aligned}
&\alpha_1 = 0.0348, \alpha_2 = 0.0317, \alpha_3 = 0.0196\\
&\ell_1 = 0.2286, \ell_2 = 0.2753, \ell_3 = 0.2891, \ell_4 = 0.2730\\
&c_1 = 0.8074, c_2 = 0.5460, c_3 = 0.2568\\
&p_0 = -0.2647, p_1 = -0.0605, p_2 = -0.0138
\end{aligned}$$

We will use this improvement in the next section where we increase the representation further.

6.2.3 Our Extended Representation

We now introduce our new $\{-1, 0, 1, 2\}$ representation and detail our improvements our BCJ. The introduction of a new symbol requires us to extend again our notion of distribution of knapsacks.

Definition 6.11 (Distributions of knapsacks)**.** A *knapsack* or *subknapsack* is a vector $\mathbf{e} \in \{-1, 0, 1, 2\}^n$. The set of $\mathbf{e}$ with αn "-1", $(\alpha + \beta - 2\gamma)n$ "1", γn "2" and $(1 - 2\alpha - \beta + \gamma)n$ "0" is denoted $D^n[\alpha, \beta, \gamma]$. If $\gamma = 0$, we may omit the third parameter and it coincides with $D^n[\alpha, \beta]$.

In Section 6.2.5, we show how to efficiently sample classicaly and quantumly from $D^n[\alpha, \beta, \gamma]$ for any α, β and γ. We also need an the following estimate on the number of such representations.

Property 6.12 (Size of knapsack sets)**.** *We have*

$$\frac{1}{n}\log_2 |D^n[\alpha, \beta, \gamma]| \simeq f(\alpha, \alpha+\beta-2\gamma, \gamma)$$

where

$$f(x, y, z) = -x\log_2 x - y\log_2 y - z\log_2 z - (1-x-y-z)\log_2(1-x-y-z)$$

is the 3-way entropy.

We extend the merging and filtering of lists to this new representation and we assume that Heuristic 6.4 applies to this new representation. Since the filtering step includes one more symbol, the filtering probability needs to be re-computed. This becomes much more challenging than with the $\{-1, 0, 1\}$ representation and we again focus only on symmetric input distribution.

Lemma 6.13 (Filtering representations using "2"s)**.** *Let* $\mathbf{e_1}, \mathbf{e_2} \in D^n[\alpha_1, \beta, \gamma_1]$ *and* $\alpha_0, \gamma_0 \geqslant 0$. *Let us define :* $\begin{cases} x_{min} = \max(0, \alpha_1 + \beta - \frac{1-\alpha_0+\gamma_0}{2}, \gamma_1 - \gamma_0/2) \\ x_{max} = \min(\alpha_1 - \alpha_0/2, \alpha_0/2 + \beta_1 - \gamma_0, \gamma_1) \end{cases}$. *If* $x_{min} \leqslant x_{max}$, *then the logarithm of the probability that* $\mathbf{e_1} + \mathbf{e_2} \in D^n[\alpha_0, 2\beta, \gamma_0]$ *is at least:*

$$\begin{aligned}\mathsf{pf}_3(\alpha_0, \beta, \gamma_0, \alpha_1, \gamma_1) = \max_{x \in [x_{min}, x_{max}]} & \mathrm{trin}(\alpha_1, x, \alpha_0/2) + \\ & \mathrm{trin}(\alpha_1 + \beta - 2\gamma_1, \gamma_0 - 2\gamma_1 + 2x, \beta - \gamma_0 - x + \alpha_0/2) + \mathrm{bin}(\gamma_1, x) + \\ & \mathrm{quadrin}(1 - \beta - 2\alpha_1 + \gamma 1, \gamma_1 - x, \alpha_0/2, \beta - \gamma_0 - x + \alpha_0/2) - \\ & \mathrm{quadrin}(1, \alpha_1, \alpha_1 + \beta - 2\gamma_1, \gamma_1)\end{aligned}$$

where $\mathrm{quadrin}(\omega, \alpha, \beta, \gamma) = f(\alpha/\omega, \beta/\omega, \gamma/\omega)\omega$ *is the* quadrinomial. *Furthermore,* $\mathrm{quadrin}(\omega, \alpha, \beta, \gamma) \simeq \frac{1}{n} \log_2 \binom{\omega n}{\alpha n, \beta n, \gamma n}$.

Proof. To avoid the explosion of the number of variables, we restrain ourselves to the *symmetric* cases (for which there are as many 1s given by $0 + 1$ as 1s given by $1 + 0$, etc.) Given some $\mathbf{e_1} \in D^n[\alpha_1, \beta, \gamma_1]$, we compute the number of compatible $\mathbf{e_2} \in D^n[\alpha_1, \beta, \gamma_1]$. These vectors can be sorted according to the number xn of positions where a -1 from $\mathbf{e_1}$ cancels out a 2 from $\mathbf{e_2}$. For $\mathbf{e_1} + \mathbf{e_2}$ to be in $D^n[\alpha_0, 2\beta, \gamma_0]$, we must have :

- $(-1) + (0) | (0) + (-1)$: $\alpha_0 n/2$ times
- $(-1) + (1) | (1) + (-1)$: $(\alpha_1 - x - \alpha_0/2)n$ times
- $(-1) + (2) | (2) + (-1)$: xn times
- $(1) + (0) | (0) + (1)$: $(\alpha_0/2 + \beta - \gamma_0 - x)n$ times
- $(2) + (0) | (0) + (2)$: $(\gamma_1 - x)n$ times
- $(1) + (1)$: $(\gamma_0 - 2\gamma_1 + 2x)n$ times
- $(0) + (0)$: $(1 - \alpha_0 + \gamma_0 - 2\alpha_1 - 2\beta + 2x)n$ times (*i.e.* the remaining)

Thus, in $\mathbf{e_1}$ (which contains α_1 "-1"s), $\alpha_0 n/2$ of the "-1"s must match a "0", $(\alpha_1 - x - \alpha_0/2)n$ of the "-1"s must match a "1" and the remaining nx must match a "2". Therefore, there are $\binom{\alpha_1 n}{\alpha_0 n/2, xn}$ possible choices for the coordinates of $\mathbf{e_2}$ matching the "-1"s of $\mathbf{e_1}$. Similarly, there are $\binom{(\alpha_1+\beta-2\gamma_1)n}{(\gamma_0-2\gamma_1+2x)n, (\beta-\gamma_0-x+\alpha_0/2)n}$ possibilities for the coordinates of $\mathbf{e_2}$ matching the "1"s, $\binom{\gamma_1 n}{xn}$ for the coordinates matching the "2"s, and $\binom{(1-\beta-2\alpha_1+\gamma 1)n}{(\gamma_1-x)n, \alpha_0 n/2, (\beta-\gamma_0-x+\alpha_0/2)n}$ for the coordinates matching the "0"s.

The total number of possibilities is :

$$\sum_x \binom{\alpha_1 n}{\alpha_0 n/2, xn} \binom{(\alpha_1+\beta-2\gamma_1)n}{(\gamma_0-2\gamma_1+2x)n, (\beta-\gamma_0-x+\alpha_0/2)n} \binom{\gamma_1 n}{xn} \binom{(1-\beta-2\alpha_1+\gamma 1)n}{(\gamma_1-x)n, \alpha_0 n/2, (\beta-\gamma_0-x+\alpha_0/2)n}$$

This quantity is defined only for $x_{min} \leqslant x \leqslant x_{max}$. If x is outside of these bounds, one of these multinomials (at least) is zero and there is no compatible $\mathbf{e_2}$. As xn must be an integer, there are only a linear number of possible choices for x. Therefore the number of all possible $\mathbf{e_2}$ is given, up to a polynomial factor, by the number of $\mathbf{e_2}$s for the best x.

In order to obtain a probability, we divide the number of compatible $\mathbf{e_2}$ by $\binom{n}{\alpha_1 n,(\alpha_1+\beta-2\gamma_1)n,\gamma_1 n)}$, which is the size of $D^n[\alpha_1, \beta, \gamma_1]$. We observe here that the logarithm of the probability that $\mathbf{e_2}$ is compatible with $\mathbf{e_1}$ is exactly $\mathsf{pf}_3(\alpha_0, \beta, \gamma_0, \alpha_1, \gamma_1)$.

We only considered the symmetric cases, assuming that we can neglect the contribution of the asymmetric cases. If we cannot, it means that we underestimated the probability for $\mathbf{e_1}$ and $\mathbf{e_2}$ to be compatible, and we could improve further the parameters by taking into account the asymmetric cases as well. □

The algorithm. We build a merging tree with five levels, numbered 4 down to 0. Level j contains 2^j lists. In total, 16 lists are merged together into one. It is represented in Figure 6.2.

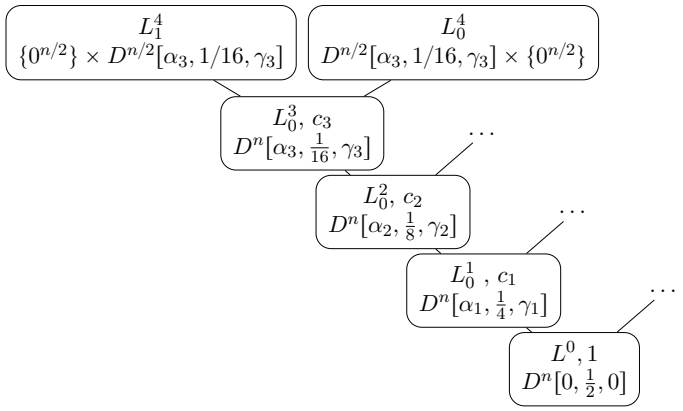


Figure 6.2: Our improved algorithm (duplicate lists are omitted).

Level 4. We build 16 lists $L_0^4 \dots L_{15}^4$. They contain complete distributions on $\frac{n}{2}$ bits, either left or right, with $\frac{n}{32} + \frac{\alpha_3 n}{2} - \gamma_3 n$ "1", $\frac{\alpha_3 n}{2}$ "-1" and $\frac{\gamma_3 n}{2}$ "2":

$$\begin{cases} L_{2i}^4 &= D^{n/2}[\alpha_3, 1/16, \gamma_3] \times \{0^{n/2}\} \\ L_{2i+1}^4 &= \{0^{n/2}\} \times D^{n/2}[\alpha_3, 1/16, \gamma_3] \end{cases}$$

As before, this avoids filtering at the first level. These lists have size: $\ell_4 = f(\alpha_3, 1/16 + \alpha_3 - 2\gamma_3, \gamma_3)/2$.

Level 3. We merge into 8 lists $L_0^3 \dots L_7^3$, with a constraint on c_3 bits. As there is no filtering, these lists have size: $\ell_3 = f(\alpha_3, 1/16 + \alpha_3 - 2\gamma_3, \gamma_3) - c_3$.

Level 2. We now merge and filter. We force a target distribution $D^n[\alpha_2, 1/8, \gamma_2]$, with α_2 and γ_2 to be optimized later. There is a first filtering probability p_2 given by Lemma 6.13. We have $\ell_2 = 2\ell_3 - (c_2 - c_3) + p_2$.

Level 1. Similarly, we have: $\ell_1 = 2\ell_2 - (c_1 - c_2) + p_1$.

Level 0. We have $\ell_0 = 2\ell_1 - (1 - c_1) + p_0 = 0$, since the goal is to obtain one solution in the list L_0.

With these constraints, we find a time $\widetilde{O}\left(2^{0.2830n}\right)$ (rounded upwards) with the following parameters:

$$\alpha_1 = 0.0340, \alpha_2 = 0.0311, \alpha_3 = 0.0202, \gamma_1 = 0.0041, \gamma_2 = 0.0006, \gamma_3 = 0.0001$$
$$c_1 = 0.8067, c_2 = 0.5509, c_3 = 0.2680, p_0 = -0.2829, p_1 = -0.0447, p_2 = -0.0135$$
$$\ell_1 = 0.2382, \ell_2 = 0.2694, \ell_3 = 0.2829, \ell_4 = 0.2755$$

6.2.4 Correctness of the Algorithms

While the operation of *merging and filtering* is the same as in previous works, our complexity analysis differs from [HJ10, BCJ11, BJLM13, HM18]. We enforce the constraint that the final list contains a single solution, hence if it is of size $2^{n\ell_0}$, we constrain $\ell_0 = 0$. Next, we limit the sizes of the lists so that they do not contain duplicate vectors: these are *saturation constraints*. A list of size $2^{n\ell}$, of vectors sampled from a distribution D, with a constraint of cn bits, has the constraint: $\ell \leqslant \frac{1}{n}\log_2|D| - c$. This says that there are not more than $|D|/2^{cn}$ vectors $\mathbf{e}$ such that $\mathbf{e}\cdot\mathbf{a} = r \pmod{2^{cn}}$ for the (randomly chosen) arbitrary constraint r.

Previous works focus on the solution vector $\mathbf{e}$ and compute the number of *representations* of $\mathbf{e}$, that is, the number of ways it can be decomposed as a sum: $\mathbf{e} = \mathbf{e}_1 + \ldots + \mathbf{e}_t$ of vectors satisfying the constraints on the distributions. Then, they compare this with the probability that a given representation passes the arbitrary constraints imposed by the algorithm. As their lists contains all the subknapsacks that fulfil the constraint, this really reflects the number of duplicates, and it suffices to enforce that the number of representations is equal to the inverse probability that a representation fulfils the constraint. For smaller lists, it suffices to add the probability that a representation compatible with the constraint is indeed in the list.

The two approaches are strictly equivalent, as the probability that the sum of two subknapsacks is valid is exactly the number of representations of the sum, divided by the number of pairs of subknapsacks. Therefore the correctness of our algorithm follows from the analysis from [HJ10, BCJ11, BJLM13, HM18].

6.2.5 Sampling from a distribution of knapsacks

Throughout this chapter, we assume that we can classically sample uniformly at random from $D^n[\alpha, \beta, \gamma]$ in time $\mathrm{poly}(n)$. Since αn, βn and γn will in general not be integers, we suppose to have them rounded to the nearest integer. This comes from the following efficient bijection between representations and integers.

It is well known that there exists a bijection between $\left[0, \binom{n}{m}\right)$ and n-bit vectors of Hamming weight m, and this bijection can be computed in polynomial time in n [Sch72]. In our case,

$m = \beta n$ and such vectors are subknapsacks from $D^n[0, \beta]$. If $i_1, \dots i_m$ are the bit-positions of the m "1" in this vector, we map it to the m-tuple of integers: $(i_1, \dots i_m)$, and define the bijection as:

$$\phi \ : (i_1, \dots i_m) \mapsto \binom{i_m - 1}{m} + \dots + \binom{i_1 - 1}{1}$$

where $\binom{i}{j}$ has supposedly been precomputed for all $i \leqslant n, j \leqslant m$. In order to compute the inverse ϕ^{-1}, we find for each $j \leqslant t$ the unique integer i_j such that $\binom{i_j - 1}{m} \leqslant x \leqslant \binom{i_j}{m}$. We can generalize this to an arbitrary number of nonzero symbols (3 in our extended algorithm: "1", "-1" and "2"), that we denote $1, 2, \dots t$. Let $k_1, \dots k_t$ be the counts of each symbol in the vector $\mathbf{v}$. We map it to a tuple of tuples: $(i_1^1, \dots i_{k_1}^1), \dots, (i_1^t, \dots i_{k_t}^t)$ where the first vector represents the positions of the "1" among the n bit positions, the second vector represents the positions of the "2" *after having removed the "1"*, and so on. Consequently, we have $0 \leqslant i_j^1 \leqslant n - 1$, $0 \leqslant i_j^2 \leqslant n - 1 - k_1$, *etc.*

Next, we map each of these t tuples individually to an integer, as was done above: $\phi^j(i_1^j, \dots i_{k_j}^j) = x^j$ where $0 \leqslant x^j \leqslant \binom{n - k_1 \dots - k_{j-1}}{k_j}$. Finally, we compute:

$$\begin{aligned}\phi(\mathbf{v}) &= x^1 + \binom{n}{k_1} x^2 + \binom{n}{k_1, k_2} x^3 + \dots + \binom{n}{k_1, \dots k_t} x^t \\ &= x^1 + \binom{n}{k_1} \left(x^2 + \binom{n - k_1}{k_2} \left(x^3 + \dots + \binom{n - k_1 \dots - k_{t-2}}{k_{t-1}} x^t \right) \dots \right) \ .\end{aligned}$$

By the bounds on the x^j, we remark that $0 \leqslant \phi(\mathbf{v}) \leqslant \binom{n}{k_1, \dots k_t} - 1$. Furthermore, we observe that one can easily retrieve the x^j by successive euclidean divisions, and use the bijections ϕ^j to finish the computation.

Lemma 6.14. *Let $n, k_1, \dots k_t$ be $t + 1$ integers such that $k_1 + \dots + k_t \leqslant n$. There exists a quantum unitary, realized with poly(n) gates, that on input a number j in $\left[0, \binom{n}{k_1, \dots k_t}\right)$, writes on its output register the j-th vector $\phi^{-1}(j) \in \{0, \dots, t\}^n$ having, for each $1 \leqslant i \leqslant t$, exactly k_i occurrences of the symbol "i". There exists another unitary which writes, on input $\mathbf{v}$, the integer value $\phi(\mathbf{v})$.*

Using this unitary in combination with a Quantum Fourier Transform, we can efficiently produce superpositions of subsets of $D^n[\alpha, \beta, \gamma]$, using poly$(n)$ quantum gates, by taking arbitrary integer intervals. We can thus perform a quantum search among representations.

6.3 Previous Quantum Algorithms for Subset-sum

In this section, we recall previous quantum algorithms for subset-sum. As we consider all our algorithms from the point of view of asymptotic complexities, and neglect polynomial factors in n, a high-level overview is often enough, and we will use quantum building blocks (quantum search and quantum walks) as black boxes.

6.3.1 Solving Subset-sum with Quantum Walks

In 2013, Bernstein, Jeffery, Lange and Meurer [BJLM13] constructed quantum subset sum algorithms inspired by Schroeppel-Shamir [SS81] and HGJ [HJ10]. We briefly explain the idea of their quantum walk for HGJ. The graph G that they consider is a product Johnson graph. We recall formal definitions from [KT17].

Definition 6.15 (Johnson graph)**.** A Johnson graph $J(N, R)$ is an undirected graph whose vertices are the subsets of R elements among a set of size N, and there is an edge between two vertices S and S' iff $|S \cap S'| = R - 1$, in other words, if S' can be obtained from S by replacing an element. Its spectral gap is given by $\delta = \frac{N}{R(N-R)}$.

Theorem 6.16 (Cartesian product of Johnson graphs [KT17])**.** *Let $J^m(N, R)$ be defined as the cartesian product of m Johnson graphs $J(N, R)$,* i.e.*, a vertex in $J^m(N, R)$ is a tuple of m subsets $S_1, \dots S_m$ and there is an edge between $S_1, \dots S_m$ and $S'_1, \dots S'_m$ iff all subsets are equal at all indices except one index i, which satisfies $|S_i \cap S'_i| = R - 1$. Then it has $\binom{N}{R}^m$ vertices and its spectral gap is greater than $\frac{1}{m}\frac{N}{R(N-R)}$.*

In [BJLM13], a vertex contains a product of 8 sublists $L'^3_0 \subset L^3_0, \dots, L'^3_7 \subset L^3_7$ of a smaller size than the classical lists: $\ell < \ell_3$. There is an edge between two vertices if we can transform one into the other by replacing only one element in one of the sublists. The spectral gap of such a graph is (in $\log_2$, relative to n) $-\ell$.

In addition, each vertex has an internal data structure which reproduces the HGJ merging tree, from level 3 to level 0. Since the initial lists are smaller, the list L^0 is now of expected size $8(\ell - \ell_3)$ (in $\log_2$, relative to n), *i.e.* the walk needs to run for $4(\ell_3 - \ell)$ steps. Each step requires $\ell/2$ updates.

In the `Setup` procedure, we simply start from all choices for the sublists and build the tree by merging and filtering. Assuming that the merged lists have decreasing sizes, the setup time is ℓ. The vertex is marked if it contains a solution at level 0. Hence, checking if a vertex is marked takes time $\mathsf{C} = 1$, but the update procedure needs to ensure the consistency of the data structure. Indeed, when updating, we remove an element $\mathbf{e}$ from one of the lists L'^3_i and replace it by a $\mathbf{e}'$ from L^3_i. We then have to track all subknapsacks in the upper levels where $\mathbf{e}$ intervened, to remove them, and to add the new collisions where $\mathbf{e}'$ intervenes.

Assuming that the update can run in $\mathrm{poly}(n)$, an optimization with the new parameter ℓ yields an exponent 0.241. In [BJLM13], the parameters are such that on average, a subknapsack intervenes only in a single sum at the next level. The authors propose to simply limit the number of elements to be updated at each level, in order to guarantee a constant update time.

Quantum Walk Based on BCJ. In [HM18], Helm and May quantize, in the same way, the BCJ algorithm. They add "-1" symbols and a new level in the merging tree data structure, reaching a time exponent of 0.226. But they remark that this result depends on a conjecture, or a heuristic, that was implicit in [BJLM13].

Heuristic 6.17 (Helm-May)**.** *In these quantum walk subset-sum algorithms, an update with expected constant time U can be replaced by an update with exact time U without affecting the runtime of the algorithm, up to a polynomial factor.*

Indeed, it is easy to construct "bad" vertices and edges for which an exact update, *i.e.* the complete reconstruction of the merging tree, will take exponential time: by adding a single new subknapsack $\mathbf{e}$, we find an exponential number of pairs $\mathbf{e} + \mathbf{e}'$ to include at the next level. So we would like to update only a few elements among them. But in the MNRS framework, the data structure of a vertex must depend solely on the vertex itself (*i.e.* on the lowest-level lists in the merging tree). And if we do as proposed in [BJLM13], we add a dependency on the path that leads to the vertex, and lose the consistency of the walk.

In a related context, the problem of "quantum search with variable times" was studied by Ambainis [Amb10]. In a quantum search for some x such that $f(x) = 1$, in a set of size N, if the time to evaluate f on x is always 1, then the search requires time $O(\sqrt{N})$. Ambainis showed that if the elements have different evaluation times $t_1, \dots t_N$, then the search now requires time $\widetilde{\mathcal{O}}(\sqrt{t_1^2 + \dots + t_N^2})$, the geometric mean of $t_1, \dots t_N$. As quantum search can be seen as a particular type of quantum walk, this shows that Heuristic 6.17 is wrong in general, as we can artificially create a gap between the geometric mean and expectation of the update time U; but also, that it may be difficult to actually overcome. In this paper, we will obtain different heuristic and non-heuristic times.

6.4 Quantum Asymmetric HGJ

In this section, we give the first quantum algorithm for the subset-sum problem, *in the QRACM model*, with an asymptotic complexity smaller than BCJ.

6.4.1 Quantum Match-and-Filter

We open this section with some technical lemmas that replace the classical merge-and-filter Lemma 6.6. In this section, we will consider a merging tree as in the HGJ algorithm, but this tree will be built using quantum search. The following lemmas bound the expected time of merge-and-filter *and* match-and-filter operations performed quantumly, in the QRACM model. This will have consequences both in this section and in the next one.

First, we remark that we can use a much more simple data structure than the ones in [Amb07, BJLM13]. In this data structure, we store pairs $\mathbf{e}, \mathbf{e} \cdot \mathbf{a}$ indexed by $\mathbf{e} \cdot \mathbf{a} \mod M$ for some $M \simeq 2^m$.

Definition 6.18 (Unique modulus list)**.** A *unique modulus list* is a qRAM data structure $\mathcal{L}(M)$ that stores at most M entries $(\mathbf{e}, \mathbf{e} \cdot \mathbf{a})$, indexed by $\mathbf{e} \cdot \mathbf{a} \mod M$, and supports the following operations:

- Insertion: inserts the entry $(\mathbf{e}, \mathbf{e} \cdot \mathbf{a})$ if the modulus is not already occupied;
- Deletion: deletes $(\mathbf{e}, \mathbf{e} \cdot \mathbf{a})$ (not necessary in this section)
- Query in superposition: returns the superposition of all entries $(\mathbf{e}, \mathbf{e} \cdot \mathbf{a})$ with some modular condition on $\mathbf{e} \cdot \mathbf{a}$, *e.g.* $\mathbf{e} \cdot \mathbf{a} = t \mod M'$ for some t and some modulus M'.

Note that all of these operations, *including* the query in superposition of all the entries with a given modulus, cost $O(1)$ qRAM gates only. For the latter, we need only some Hadamard

gates to prepare the adequate superposition of indices. Furthermore, the list remains sorted by design.

Next, we write a lemma for quantum *matching* with filtering, in which one of the lists is not written down. We start from a unitary that produces the uniform superposition of the elements of a list L_1, and we wrap it into an amplitude amplification, in order to obtain a unitary that produces the uniform superposition of the elements of the merged-and-filtered list.

Lemma 6.19 (Quantum matching with filtering)**.** *Let L_2 be a list stored in QRACM (with the* unique modulus list *data structure of Definition 6.18). Assume we are given a unitary U that produces in time t_{L_1} the uniform superposition of $L_1 = x_0, \dots x_{2^{m}-1}$ where $x_i = (\mathbf{e_i}, \mathbf{e_i} \cdot \mathbf{a})$. We merge L_1 and L_2 with a modular condition of cn bits and a filtering probability p. Let L be the merged list and L^f the filtered list. Assume $|L^f| \geqslant 1$. Then there exists a unitary U' producing the uniform superposition of L^f in time: $O\left(\frac{t_{L_1}}{\sqrt{p}} \max(\sqrt{2^{cn}/|L_2|}, 1)\right)$.*

Notice that this is also the time complexity to produce a single random element of L^f. If we want to produce and store the whole list L^f, it suffices to multiply this complexity by the number of elements in L^f (*i.e.* $p|L_1||L_2|/2^{cn}$). We would obtain: $O\left(t_{L_1}\sqrt{p}\max\left(|L_1|\sqrt{\frac{|L_2|}{2^{cn}}}, \frac{|L_1||L_2|}{2^{cn}}\right)\right)$.

Proof. Since L_2 is stored in a unique modulus list, all its elements have distinct moduli. Note that the *expected* sizes of L and L^f follow from Heuristic 6.4. Although the number of iterations of quantum search should depend on the *real* sizes of these lists, the concentration around the average is so high (given by Chernoff bounds) that the error remains negligible if we run the search with the expected number of iterations. We separate three cases.

- If $|L_2| < 2^{cn}$, then we have no choice but to make a quantum search on elements of L_1 that match the modular constraint and pass the filtering step, in time: $O\left(t_{L_1}\sqrt{\frac{2^{cn}}{L_2 p}}\right)$.
- If $|L_2| > 2^{cn}$ but $|L_2| < 2^{cn}/p$, an element of L_1 will always pass the modular constraint, with more than one candidate, but in general all these candidates will be filtered out. Given an element of L_1, producing the superposition of these candidates is done in time 1, so finding the one that passes the filter, if there is one, takes time $\sqrt{|L_2|/2^{cn}}$. Next, we wrap this in a quantum search to find the "good" elements of L_1 (passing the two conditions), with $O\left(\sqrt{2^{cn}/pL_2}\right)$ iterations. The total time is:

$$O\left(\sqrt{\frac{2^{cn}}{L_2 p}} \times \left(\sqrt{|L_2|/2^{cn}} \times t_{L_1}\right) = \frac{t_{L_1}}{\sqrt{p}}\right) .$$

- If $|L_2| > 2^{cn}/p$, an element of L_1 yields on average more than one filtered candidate. Producing the superposition of the modular candidates is done in time $O(1)$ thanks to the data structure, then finding the superposition of filtered candidates requires $1/\sqrt{p}$ iterations. The total time is: $O\left(t_{L_1}/\sqrt{p}\right)$.

The total time in all cases is: $O\left(\frac{t_{L_1}}{\sqrt{p}} \max(\sqrt{2^{cn}/|L_2|}, 1)\right)$. Note that classically, the coupon collector problem would have added a polynomial factor, but this is not the case here thanks to QRACM (Remark 2.5). □

In the QRACM model, we have the following corollary for merging and filtering two lists of equal size. This result will be helpful in Section 6.4.3 and 6.5.

Corollary 6.20. *Consider two lists L_1, L_2 of size $|L_1| = |L_2| = |L|$ exponential in n. We merge L_1 and L_2 with a modular condition of cn bits, and filter with a probability p. Assume that $2^{cn} < |L|$. Then L^f can be written down in quantum time: $O\left(\sqrt{p}\frac{|L|^2}{2^{cn}}\right)$.*

Proof. We do a quantum search to find each element of L^f. We have $t_{L_1} = O(1)$ since it is a mere QRACM query, and we use Lemma 6.19. □

6.4.2 Revisiting HGJ

We now introduce our new algorithm for subset-sum in the QRACM model.

Our starting point is the HGJ algorithm. Similarly to [NS20], we use a merging tree in which the lists at a given level may have different sizes. Classically, this does not improve the time complexity. However, quantumly, we will use quantum filtering. Since our algorithm does not require to write data in superposition, only to read from classical registers with quantum random access, we require only QRACM instead of QRAQM.

In the following, we consider that all lists, except L_0^3, L_0^2, L_0^1, L^0, are built with classical merges. The final list L^0, containing (expectedly) a single element, and a branch leading to it, are part of a nested quantum search. Each list L_0^3, L_0^2, L_0^1, L^0 corresponds either to a search space, the solutions of a search, or both. We represent this situation on Fig. 6.3. Our procedure runs as follows:

1. (Classical step): build the *intermediate lists* L_1^3, L_1^2, L_1^1 and store them using a *unique modulus list* data structure (Definition 6.18).
2. (Quantum step): do a quantum search on L_0^3. To test a vector $\mathbf{e} \in L_0^3$:
   - Find $\mathbf{e}_3 \in L_1^3$ such that $\mathbf{e} + \mathbf{e}_3$ passes the $c_0^2 n$-bit modular constraint (assume that there is at most one such solution). There is no filtering here.
   - Find $\mathbf{e}_2 \in L_1^2$ such that $(\mathbf{e} + \mathbf{e}_3) + \mathbf{e}_2$ passes the additional $(c^1 - c_0^2)n$-bit constraint.
   - If it also passes the filtering step, find $\mathbf{e}_1 \in L_1^1$ such that $(\mathbf{e} + \mathbf{e}_3 + \mathbf{e}_2) + \mathbf{e}_1$ is a solution to the knapsack problem (and passes the filter).

Structural constraints are imposed on the tree, in order to guarantee that there exists a knapsack solution. The only difference between the quantum and classical settings is in the optimization goal: the final time complexity.

Structural Constraints. We now introduce the variables and the structural constraints that determine the shape of the tree in Fig. 6.3. The asymmetry happens both in the weights at level 0 and at the constraints at level 1 and 2. We write $\ell_i^j = (\log_2 |L_i^j|)/n$. With the lists built classically, we expect a symmetry to be respected, so we have: $\ell_2^3 = \ell_3^3$, $\ell_4^3 = \ell_5^3 = \ell_6^3 = \ell_7^3$, $\ell_2^2 = \ell_3^2$. We also tweak the left-right split at level 0: lists from L_2^3 to L_7^3 have a standard balanced left-right split; however, we introduce a parameter r that determines the proportion of positions set to zero in list L_0^3: in L_0^3, the vectors weigh $cn(1-r)$ on a support of size

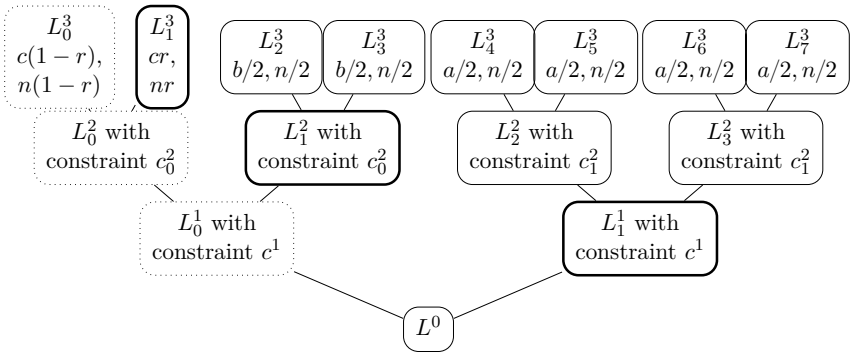


Figure 6.3: Quantum HGJ algorithm. Dotted lists are search spaces (they are not stored). Bold lists are stored in QRACM. In Section 6.4.3, L_2^2 and L_3^2 are also stored in QRACM.

$n(1-r)$, instead of $cn/2$ on a support of size $n/2$. In total we have $c + b + 2a = \frac{1}{2}$, as the weight of the solution is supposed to be exactly $n/2$.

Then we note that:

- The lists at level 3 have a maximal size depending on the corresponding weight of their vectors:
$$\ell_0^3 \leqslant h(c)(1-r), \quad \ell_1^3 \leqslant h(c)r, \quad \ell_2^3 = \ell_3^3 \leqslant h(b)/2, \quad \ell_4^3 \leqslant h(a)/2$$
- The lists at level 2 cannot contain more representations than the filtered list of all subknapsacks of corresponding weight:
$$\ell_0^2 \leqslant h(c) - c_0^2, \quad \ell_1^2 \leqslant h(b) - c_0^2, \quad \ell_2^2 = \ell_3^2 \leqslant h(a) - c_1^2$$
- Same at levels 1 and 0: $\ell_0^1 \leqslant h(c+b) - c^1, \quad \ell_1^1 \leqslant h(2a) - c^1$
- The merging at level 2 is exact (there is no filtering):
$$\ell_0^2 = \ell_0^3 + \ell_1^3 - c_0^2, \quad \ell_1^2 = \ell_2^3 + \ell_3^3 - c_0^2, \quad \ell_2^2 = \ell_4^3 + \ell_5^3 - c_1^2, \quad \ell_3^2 = \ell_6^3 + \ell_7^3 - c_1^2,$$
- At level 1, with a constraint $c^1 \geqslant c_0^2, c_1^2$ that subsumes the previous ones:
$$\ell_0^1 = \ell_0^2 + \ell_1^2 - c^1 + c_0^2 + \mathsf{pf}_1(b,c)\ , \qquad \ell_1^1 = \ell_2^2 + \ell_3^2 - c^1 + c_1^2 + \mathsf{pf}_1(a,a)$$
- And finally at level 0: $\ell^0 = 0 = \ell_0^1 + \ell_1^1 - (1-c^1) + \mathsf{pf}_1(b+c, 2a)$

Classical Optimization. All the previous constraints depend on the problem, not on the computation model. Now we can get to the time complexity in the classical setting, that we want to minimize:

$$\max\big(\ell_4^3, \ell_2^3, \ell_1^3, \ell_2^3 + \ell_3^3 - c_0^2, \ell_4^3 + \ell_5^3 - c_1^2, \ell_2^2 + \ell_3^2 - c^1 + c_1^2,$$
$$\ell_0^3 + \max(\ell_1^3 - c_0^2, 0) + \max(\ell_1^2 - c^1 + c_0^2, 0) + \max(\mathsf{pf}_1(b,c) + \ell_1^1 - (1-c^1), 0)\big)\ .$$

The last term corresponds to the exhaustive search on ℓ_0^3. In order to keep the same freedom as before, it is possible that an element of L_0^3 matches against *several* elements of L_1^3, all

of which yield a potential solution that has to be matched against L_1^2, *etc.* Hence for each element of L_0^3, we find the expected $\max(\ell_1^3 - c_0^2, 0)$ candidates matching the constraint c_0^1. For each of these candidates, we find the expected $\max(\ell_1^2 - c^1 + c_0^2, 0)$ candidates matching the constraint c^1. For each of these candidates, if it passes the filter, we search for a collision in L_1^1; this explains the $\max(\mathsf{pf}_1(b,c) + \ell_1^1 - (1-c^1), 0)$ term. In the end, we check if the final candidates pass the filter on the last level.

We verified that optimizing the classical time under our constraints gives the time complexity of HGJ.

Quantum Optimization. The time complexity for producing the intermediate lists is unchanged. The only difference is the way we find the element in L_0^3 that will lead to a solution, which is a nested sequence of quantum searches.

- We can produce the superposition of all elements in L_0^2 in time
$$t_2 = \frac{1}{2}\max(c_0^2 - \ell_1^3, 0)$$
- By Lemma 6.19, we can produce the superposition of all elements in L_0^1 in time
$$t_2 - \frac{1}{2}\mathsf{pf}_1(b,c) + \frac{1}{2}\max\big(c^1 - c_0^2 - \ell_1^2, 0\big)$$
- Finally, we expect that there are $\big(\ell_0^2 + \ell_1^2 - c^1 + c_0^2 + \mathsf{pf}_1(b,c)\big)$ elements in L_0^1, which gives the number of iterations of the quantum search.

The time of this search is:
$$\frac{1}{2}\big(\ell_0^1 + \max(c_0^2 - \ell_1^3, 0) - \mathsf{pf}_1(b,c) + \max\big(c^1 - c_0^2 - \ell_1^2, 0\big)\big)$$
and the total time complexity is:
$$\max\Big(\ell_4^3, \ell_2^3, \ell_1^3, \ell_2^3 + \ell_3^3 - c_0^2, \ell_4^3 + \ell_5^3 - c_1^2, \ell_2^2 + \ell_3^2 - c^1 + c_1^2, \frac{1}{2}\big(\ell_0^1 + \max(c_0^2 - \ell_1^3, 0) - \mathsf{pf}_1(b,c) + \max\big(c^1 - c_0^2 - \ell_1^2, 0\big)\big)\Big)$$

We obtain a quantum time complexity exponent of 0.2374 with this method (the detailed parameters are given in Table 6.1).

6.4.3 Improvement via Quantum Filtering

Let us keep the tree structure of Figure 6.3 and its structural constraints. The final quantum search step is already made efficient with respect to the filtering of representations, as we only pay half of the filtering term $\mathsf{pf}_1(b,c)$. However, we can look towards the *intermediate lists* in the tree, *i.e.* L_1^3, L_1^2, L_1^1. The merging at the first level is exact: due to the left-right split, there is no filtering of representations, hence the complexity is determined by the size of the output list. However, the construction of L_1^1 contains a filtering step. Thus, we can use Corollary 6.20 to produce the elements of L_1^1 faster and reduce the time complexity from: $\ell_2^2 + \ell_3^2 - c^1 + c_1^2$ to: $\ell_2^2 + \ell_3^2 - c^1 + c_1^2 + \frac{1}{2}\mathsf{pf}_1(a,a)$. By optimizing with this time complexity, we obtain a time exponent 0.2356 (the detailed parameters are given in Table 6.1). The corresponding memory is 0.2356 (given by the list L_1^3).

Table 6.1: Optimization results for the quantum asymmetric HGJ algorithm (in $\log_2$ and relative to n), rounded to four digits. The time complexity is an upper bound.

Variant	Time	a	b	c	ℓ_0^3	ℓ_1^3	ℓ_2^3	ℓ_4^3	ℓ_0^2	ℓ_0^1
Classical	0.3370	0.1249	0.11	0.1401	0.3026	0.2267	0.25	0.2598	0.3369	0.3114
Section 6.4.2	0.2374	0.0951	0.0951	0.2146	0.4621	0.2367	0.2267	0.2267	0.4746	0.4395
Section 6.4.3	0.2356	0.0969	0.0952	0.2110	0.4691	0.2356	0.2267	0.2296	0.4695	0.4368

Remark 6.21 (More improvements). We have tried increasing the tree depth or changing the tree structure, but it does not seem to bring any improvement. In theory, we could allow for more general representations involving "-1" and "2". However, computing the filtering probability, when merging two lists of subknapsacks in $D^n[\alpha, \beta, \gamma]$ *with different distributions* becomes much more technical. We managed to compute it for $D^n[\alpha, \beta]$, but the number of parameters was too high for our numerical optimizer, which failed to converge.

6.4.4 Quantum Time-Memory Tradeoff

In the original HGJ algorithm, the lists at level 3 contain full distributions $D^{n/2}[0, 1/8]$. By reducing their sizes to a smaller exponential, one can still run the merging steps, but the final list L^0 is of expected size exponentially small in n. Hence, one must redo the tree many times. This general time-memory tradeoff is outlined in [HJ10] and is also reminiscent of Schroeppel and Shamir's algorithm [SS81], which can actually be seen as repeating $2^{n/4}$ times a merge of lists of size $2^{n/4}$, that yields $2^{-n/4}$ solutions on average.

Asymmetric Tradeoff. The tradeoff that we propose is adapted to the QRACM model. It consists in increasing the asymmetry of the tree: we reduce the sizes of the intermediate lists L_1^3, L_1^2, L_1^1 in order to use less memory; this in turn increases the size of L_0^3, L_0^2 and L_0^1 in order to ensure that a solution exists. We find that this tradeoff is close to the time-memory product curve $TM = 2^{n/2}$, and actually slightly better (the optimal point when $m = 0.2356$ has $TM = 2^{0.4712n}$). This is shown on Figure 6.4. At $m = 0$, we start at $2^{n/2}$, where L_0^3 contains all vectors of Hamming weight $n/2$.

Fact 6.22. *For any memory constraint $m \leqslant 0.2356$ (in $\log_2$ and proportion of n), the optimal time complexity in the quantum asymmetric HGJ algorithm of Section 6.4.3 is lower than $\widetilde{O}\left(2^{n/2-m}\right)$.*

Improving the QRACM usage. In trying to reduce the quantum or quantum-accessible hardware used by our algorithm, it makes sense to draw a line between QRACM and classical RAM, *i.e.* between the part of the memory that is actually accessed quantumly, and the memory that is used only classically. We now try to enforce the constraint only on the QRACM, using possibly more RAM. In this context, we cannot produce the list L_1^1 via quantum filtering. The memory constraint on lists L_1^3, L_1^2, L_1^1 still holds; however, we can increase the size of lists $L_4^3, L_5^3, L_6^3, L_7^3, L_2^2, L_3^2$.

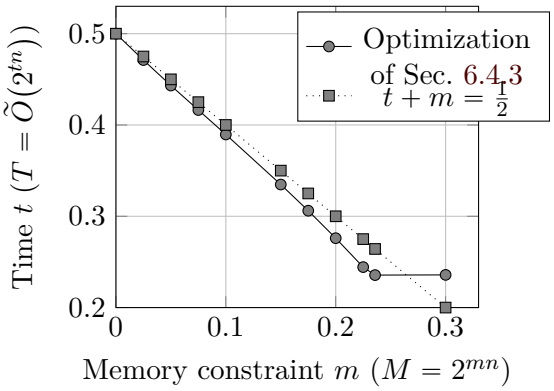


Figure 6.4: Quantum time-memory tradeoff of the asymmetric HGJ algorithm

Fact 6.23. *For any QRACM constraint $m \leqslant 0.2356$, the optimal time complexity obtained by using more RAM is always smaller than the best optimization of Section 6.4.3.*

The difference remains only marginal, as can be seen in Table 6.2, but it shows a tradeoff between quantum and classical resources.

Table 6.2: Time-memory tradeoffs (QRACM) for three variants of our asymmetric HGJ algorithm, obtained by numerical optimization, and rounded upwards. The last variant uses more classical RAM than the QRACM constraint.

QRACM	Section 6.4.2		Section 6.4.3		With more RAM	
bound	Time	Memory	Time	Memory	Time	Memory
0.0500	0.4433	0.0501	0.4433	0.0501	0.4412	0.0650
0.1000	0.3896	0.1000	0.3896	0.1000	0.3860	0.1259
0.1500	0.3348	0.1501	0.3348	0.1501	0.3301	0.1894
0.3000	0.2374	0.2373	0.2356	0.2356	0.2373	0.2373

6.5 New Algorithms Based on Quantum Walks

In this section, we improve the algorithm by Helm and May [HM18] based on BCJ and the MNRS quantum walk framework. Our algorithm is a quantum walk on a product Johnson graph, as in Section 6.3.1. There are two new ideas involved.

6.5.1 Asymmetric 5th level

In our new algorithm, we can afford one more level than BCJ. We then have a 6-level merging tree, with levels numbered 5 down to 0. Lists at level i all have the same size ℓ_i, *except at level 5*. Recall that the merging tree, and all its lists, is the additional data structure attached to a node in the Johnson graph. In the original algorithm of [HM18], there are 5 levels, and a

node is a collection of 16 lists, each list being a subset of size ℓ_4 among the $g(1/16+\alpha_3,\alpha_3)/2$ vectors having the right distribution.

In our new algorithm, at level 5, we separate the lists into "left" lists of size ℓ_5^l and "right" lists of size ℓ_5^r. The quantum walk will only be performed on the left lists, while the right ones are full enumerations. Each list at level 4 is obtained by merging a "left" and a "right" list. The left-right-split at level 5 is then asymmetric: vectors in one of the left lists L_5^l are sampled from $D^{\eta n}[\alpha_4,1/32,\gamma_4]\times\{0^{(1-\eta)n}\}$ and the right lists L_5^r contain *all* the vectors from $\{0^{\eta n}\}\times D^{(1-\eta)n}[\alpha_4,1/32,\gamma_4]$. This yields a new constraint: $\ell_5^r = f(1/32+\alpha_4-2\gamma_4,\alpha_4,\gamma_4)(1-\eta)$.

While this asymmetry does not bring any advantage classically, it helps in reducing the update time. We enforce the constraint $\ell_5^r = c_4$, so that for each element of L_5^l, there is on average one matching element in L_5^r. So updating the list L_4 at level 4 is done on average time 1. Then we also have $\ell_4 = \ell_5^l$.

With this construction, ℓ_5^r and ℓ_5^l are actually unneeded parameters. We only need the constraints $c_4(=\ell_5^r) = f(1/32+\alpha_4-2\gamma_4,\alpha_4,\gamma_4)(1-\eta)$ and $\ell_4(=\ell_5^l) \leqslant f(1/32+\alpha_4-2\gamma,\alpha_4,\gamma_4)\eta$. The total setup time is now:

$$\mathsf{S} = \max\Bigg(\underbrace{c_4,\ell_4}_{\text{Lv. 5 and 4}} \ , \ \underbrace{2\ell_4-(c_3-c_4)}_{\text{Level 3}}, \ \underbrace{2\ell_3-(c_2-c_3)}_{\text{Level 2}}, \ \underbrace{2\ell_2-(c_1-c_2)}_{\text{Level 1}}, \ \underbrace{\ell_1+\max(\ell_1-(1-c_1),0)}_{\text{Level 0}} \Bigg)$$

and the expected update time for level 5 (inserting a new element in a list L_5^l at the bottom of the tree) and at level 4 (inserting a new element in L_4) is 1. The spectral gap of the graph is $\delta = -\ell_5^l$ and the proportion of marked vertices is $\varepsilon = -\ell_0$.

Saturation Constraints. In the quantum walk, we have $\ell_0 < 0$, since we expect only some proportion of the nodes to be marked (to contain a solution). This proportion is hence ℓ_0. The saturation constraints are modified as follows:

$$\begin{array}{ll} \ell_5^l \leqslant \frac{\ell_0}{16} + f(\frac{1}{32}+\alpha_4-2\gamma_4,\alpha_4,\gamma_4)\eta, & \ell_4 \leqslant \frac{\ell_0}{16} + f(\frac{1}{32}+\alpha_4-2\gamma_4,\alpha_4,\gamma_4) - c_4 \\ \ell_3 \leqslant \frac{\ell_0}{8} + f(\frac{1}{16}+\alpha_3-2\gamma_3,\alpha_3,\gamma_3) - c_3, & \ell_2 \leqslant \frac{\ell_0}{4} + f(\frac{1}{8}+\alpha_2-2\gamma_2,\alpha_2,\gamma_2) - c_2 \\ \ell_1 \leqslant \frac{\ell_0}{2} + f(1/4+\alpha_1-2\gamma_1,\alpha_1,\gamma_1) - c_1 & \end{array}$$

Indeed, the *classical* walk will go through a total of $-\ell_0$ trees before finding a solution. Hence, it needs to go through $-\ell_0/16$ different lists at level 5 (and 4), which is why we need to introduce ℓ_0 in the saturation constraint: there must be enough elements, not only in L_5^l, but in the whole search space that will be spanned by the walk. These constraints ensure the existence of marked vertices in the walk.

6.5.2 Better Setup and Updates using quantum search

Along the lines of Lemma 6.19 and corollary 6.20, we now show how to use a quantum search to speed up the Setup and Update steps in the quantum walk. As the structure of the graph is unchanged, we still have $\varepsilon = -\ell_0$ and a spectral gap $\delta = -\ell_5^l$.

Setup. Let $p_i, (1 \leqslant i \leqslant 3)$ be the filtering probabilities at level i, *i.e.* the (logarithms of the) probabilities that an element that satisfies the modulo condition resp. at level i also has the desired distribution of 0s, 1s, -1s and 2s, and appears in list L_i. Notice that $p_i \leqslant 0$. Due to the left-right split, there is no filtering at level 4.

We use quantum filtering (Corollary 6.20) to speed up the computation of lists at levels 3, 2 and 1 in the setup, reducing in general a time $2\ell - c$ to $2\ell - c + \mathsf{pf}/2$. It does not apply for level 0, since L_0 has a negative expected size. At this level, we will simply perform a quantum search over L_1. If there is too much constraint, *i.e.* $(1 - c_1) > \ell_1$, then for a given element in L_1, there is on average less than one modular candidate. If $(1 - c_1) < \ell_1$, there is on average more than one (although less than one with the filter) and we have to do another quantum search on them all. This is why the setup time at level 0, in full generality, becomes $(\ell_1 + \max(\ell_1 - (1 - c_1), 0))/2$. The setup time can thus be improved to:

$$\mathsf{S} = \max\Bigg(\underbrace{c_4, \ell_4}_{\text{Lv. 5 and 4}}, \underbrace{2\ell_4 - (c_3 - c_4) + p_3/2}_{\text{Level 3}}, \underbrace{2\ell_3 - (c_2 - c_3) + p_2/2}_{\text{Level 2}}, \underbrace{2\ell_2 - (c_1 - c_2) + p_1/2}_{\text{Level 1}}, \underbrace{(\ell_1 + \max(\ell_1 - (1 - c_1), 0))/2}_{\text{Level 0}} \Bigg) .$$

Update. Our update will also use a quantum search. First of all, recall that the updates of levels 5 and 4 are performed in (expected) time 1. Having added an element in L_4, we need to update the upper level. There are on average $\ell_4 - (c_3 - c_4)$ candidates satisfying the modular condition. To avoid a blowup in the time complexity, we forbid to have more than one element inserted in L_3 on average, which means: $\ell_4 - (c_3 - c_4) + p_3 \leqslant 0 \iff \ell_3 \leqslant \ell_4$. We then find this element, if it exists, with a quantum search among the $\ell_4 - (c_3 - c_4)$ candidates.

Similarly, as at most one element is updated in L_3, we can move on to the upper levels 2, 1 and 0 and use the same argument. We forbid to have more than one element inserted in L_2 on average: $\ell_3 - (c_2 - c_3) + p_2 \leqslant 0 \iff \ell_2 \leqslant \ell_3$, and in L_1: $\ell_1 \leqslant \ell_2$. At level 0, a quantum search may not be needed, hence a time $\max(\ell_1 - (1 - c_1), 0)/2$. The expected update time becomes:

$$\mathsf{U} = \max\Bigg(0, \underbrace{(\ell_4 - (c_3 - c_4))/2}_{\text{Level 3}}, \underbrace{(\ell_3 - (c_2 - c_3))/2}_{\text{Level 2}}, \underbrace{(\ell_2 - (c_1 - c_2))/2}_{\text{Level 1}}, \underbrace{(\ell_1 - (1 - c_1))/2}_{\text{Level 0}} \Bigg) .$$

6.5.3 Parameters

Using the following parameters, we found an algorithm that runs in time $\widetilde{O}\big(2^{0.2156n}\big)$:

$$\ell_0 = -0.1916, \ell_1 = 0.1996, \ell_2 = 0.2030, \ell_3 = 0.2110, \ell_4 (= \ell_5^l) = 0.2110$$
$$c_1 = 0.6190, c2 = 0.4445, c3 = 0.2506, c_4 (= \ell_5^r) = 0.0487$$
$$\alpha_1 = 0.0176, \alpha_2 = 0.0153, \alpha_3 = 0.0131, \alpha_4 = 0.0087$$
$$\gamma_1 = 0.0019, \gamma_2 = \gamma_3 = \gamma_4 = 0, \eta = 0.8448$$

There are many different parameters that achieve the same time. The above set achieves the lowest memory that we found, at $\widetilde{O}\left(2^{0.2110n}\right)$. Note that time and memory complexities are different in this quantum walk, contrary to previous works, since the update procedure has now a (small) exponential cost.

Remark 6.24 (Time-memory tradeoffs). Quantum walks have a natural time-memory tradeoff which consists in reducing the vertex size. Smaller vertices have a smaller chance of being marked, and the walk goes on for a longer time. This is also applicable to our algorithms, but requires a re-optimization with a memory constraint.

6.6 Mitigating Quantum Walk Heuristics for Subset-Sum

In this section, we provide a modified quantum walk NEW-QW for any quantum walk subset-sum algorithm QW, including [BJLM13, HM18] and ours, that will no longer rely on Heuristic 6.17. In NEW-QW, the Johnson graph is the same, but the vertex data structure and the update procedure are different (Section 6.6.2). It allows us to guarantee the update time, at the expense of losing some marked vertices. In Section 6.6.4, we will show that most marked vertices in QW remain marked.

6.6.1 New Data Structure for Storing Lists

The main requirement of the vertex data structure is to store lists of subknapsacks with modular constraints in QRAQM. For each list, we will use two data structures. The first one is the combination of a hash table and a skip list given in [Amb07] (abbreviated *skip list* below) and the second one is a *Bucket-modulus list* data structure, adapted from Definition 6.18, that we define below.

Hash Table and Skip List. We use the data structure of [Amb07] to store lists of entries $(\mathbf{e}, \mathbf{e}\cdot\mathbf{a})$, sorted by knapsack value $\mathbf{e}\cdot\mathbf{a}$. The data structure for M entries, that we denote $\mathcal{SL}(M)$, uses $\widetilde{O}(M)$ qRAM memory cells and supports the following operations: inserting an entry in the list, deleting an entry from the list and producing the uniform superposition of entries in the list. All these operations require time $\mathsf{polylog}(M)$.

We resort to this data structure because the proposal of "radix trees" in [BJLM13] is less detailed. It is defined relatively to a choice of $\mathsf{polylog}(M) = \mathrm{poly}(n)$ hash functions selected from a family of independent hash functions of the entries (we refer to [Amb07] for more details). For a given choice of hash functions, the insertion or deletion operations can fail. Thus, the data structure is equipped with a superposition of such choices. Instead of storing $\mathcal{SL}(M)$, we store: $\sum_h |h\rangle|\mathcal{SL}_h(M)\rangle$ where $\mathcal{SL}_h$ is the data structure flavored with the choice of hash functions h. Insertions and deletions are performed depending on h. This allows for a globally negligible error: if sufficiently many hash functions are used, the insertion and deletion of *any element* add a global error vector of amplitude $o(2^{-n})$ *regardless of the current state of the data*. The standard "hybrid argument" from [BBBV97] and [Amb07, Lemma 5] can then be used in the context of an MNRS quantum walk.

Proposition 6.25 ([Amb07], Lemma 5, adapted)**.** *Consider an MNRS quantum walk with a "perfect" (theoretical) update unitary U, managing data structures, and an "imperfect" update unitary U' such that, for any basis state $|x\rangle$:*

$$U'|x\rangle = U|x\rangle + |\delta_x\rangle$$

where $|\delta_x\rangle$ is an error vector of amplitude bounded by $o(2^{-n})$ for any *x. Then running the walk with U' instead of U, after T steps, the final "imperfect" state $|\psi'\rangle$ deviates from the "perfect" state $|\psi\rangle$ by: $\||\psi'\rangle - |\psi\rangle\| \leqslant o(2^{-n}T)$.*

This holds as a general principle: in the update unitary, any perfect procedure can be replaced by an imperfect one as long as its error is negligible (with respect to the total number of updates) *and data-independent*. In contrast, the problem with Heuristic 6.17 is that a generic constant-time update induces data-dependent errors (bad cases) that do not seem easy to overcome.

Bucket-modulus List. Let $B = \text{poly}(n)$ be a "bucket size" that will be chosen later. The bucket-modulus list is a tool for making our update time data-independent: it limits the number of vectors that can have a given modulus (where moduli are of the same order as the list size).

Definition 6.26 (Bucket-modulus list)**.** A Bucket-modulus list $\mathcal{BL}(B, M)$ is a qRAM data structure that stores at most $B \times M$ entries $(\mathbf{e}, \mathbf{e}\cdot\mathbf{a})$, with at most B entries sharing the same modulus $\mathbf{e}\cdot\mathbf{a} \mod M$. Thus, $\mathcal{BL}(B, M)$ contains M "buckets". Buckets are indexed by moduli, and kept sorted. It supports the following operations:

- Insertion: insert $(\mathbf{e}, \mathbf{e}\cdot\mathbf{a})$. If the bucket at index $\mathbf{e}\cdot\mathbf{a} \mod M$ contains B elements, *empty the bucket*. Otherwise, sort it using a simple sorting circuit.
- Deletion: remove an entry from the corresponding bucket.
- Query in superposition: similar as in Definition 6.18.

In our new quantum walks, each list will be stored in a skip list $\mathcal{SL}(M)$ associated with a bucket-modulus $\mathcal{BL}(B, M)$. Each time we insert or delete an element from $\mathcal{SL}(M)$, we update the bucket-modulus list accordingly, according to the following rules.

Upon deletion of an element $\mathbf{e}$ in $\mathcal{SL}(M)$, let $\mathbf{e}\cdot\mathbf{a} = T \mod M$, there are three cases for $\mathcal{BL}(B, M)$:

- If $|\{\mathbf{e}' \in \mathcal{SL}(M), \mathbf{e}'\cdot\mathbf{a} = T\}| > B + 1$, then bucket number T in $\mathcal{BL}(B, M)$ stays empty;
- If $|\{\mathbf{e}' \in \mathcal{SL}(M), \mathbf{e}'\cdot\mathbf{a} = T\}| = B + 1$, then removing $\mathbf{e}$ makes the number of elements reach the bound B, so we add them all in the bucket at index T;
- If $|\{\mathbf{e}' \in \mathcal{SL}(M), \mathbf{e}'\cdot\mathbf{a} = T\}| \leqslant B$, then we remove $\mathbf{e}$ from its bucket.

Upon insertion of an element $\mathbf{e}$ in $\mathcal{SL}(M)$, there are also three cases for $\mathcal{BL}(B, M)$:

- If $|\{\mathbf{e}' \in \mathcal{SL}(M), \mathbf{e}'\cdot\mathbf{a} = T\}| = B$, then we empty the bucket at index T;
- If $|\{\mathbf{e}' \in \mathcal{SL}(M), \mathbf{e}'\cdot\mathbf{a} = T\}| < B$, then we add $\mathbf{e}$ to the bucket at index T in $\mathcal{BL}(B, M)$;

- If $|\{\mathbf{e}' \in \mathcal{SL}(M), \mathbf{e}' \cdot \mathbf{a} = T\}| > B$, then the bucket is empty and remains empty.

In all cases, there are at most B insertions or deletions in a single bucket. Note that $\mathcal{BL}(B, M) \subseteq \mathcal{SL}(M)$ but that some elements of $\mathcal{SL}(M)$ will be dropped.

Remark 6.27. The mapping from a skip list of size M (considered as perfect), which does not "forget" any of its elements, to a corresponding bucket-modulus list with M buckets, which forgets some of the previous elements, is deterministic. Given a skip list L, a corresponding bucket modulus list L' can be obtained by inserting all elements of L into an empty bucket modulus list.

6.6.2 New Data Structure for Vertices

The algorithms that we consider use multiple levels of merging. However, we will focus only on a single level. Our arguments can be generalized to any constant number of merges (with an increase in the polynomial factors involved). Recall that the product Johnson graph on which we run the quantum walk is unchanged, only the data structure is adapted.

In the following, we will consider the merging of two lists L_l and L_r of subknapsacks of respective sizes ℓ_l and ℓ_r, with a modular constraint c and a filtering probability pf. The merged list is denoted $L^c = L_l \bowtie_c L_r$ and the filtered list is denoted L^f. We assume that pairs $(\mathbf{e_1}, \mathbf{e_2})$ in L^c must satisfy $(\mathbf{e_1} + \mathbf{e_2}) \cdot \mathbf{a} = 0 \mod 2^{cn}$ (the generalization to any value modulo any moduli is straightforward).

On the positive side, our new data structure can be updated, *by design*, with a fixed time that is data-independent. On the negative side, we will not build the complete list L^f, and miss some of the solutions. As we drop a fraction of the vectors, some nodes that were previously marked will potentially appear unmarked, but this fraction is polynomial at most. We defer a formal proof of this fact to Section 6.6.4 and focus on the runtime.

We will focus on the case where $\ell_l = \ell_r$ and either L_l or L_r are updated, which happens at all levels in our quantum walk, except the first level. Because there is no filtering at the first level, it is actually much simpler to study with the same arguments. In previous quantum walks, we had $\ell^c = 2\ell - c \leqslant \ell$, *i.e.* $\ell \leqslant c$; now we will have $2\ell - c \geqslant \ell$ and $2\ell - c + \mathsf{pf} \leqslant \ell$.

Recall that our heuristic time complexity analysis assumes an update time $(\ell - c)/2$. Indeed, the update of an element in L_l or L_r modifies on average $(\ell - c)$ elements in $L_l \bowtie_c L_r$, among which we expect at most one filtered pair $(\mathbf{e_1}, \mathbf{e_2})$ (by the inequality $2\ell - c + \mathsf{pf} \leqslant \ell$). We find this solution with a quantum search. In the following, we modify the data structure of vertices in order to guarantee the best update time possible, up to additional polynomial factors. We will see however that it does not reach $(\ell - c)/2$. We now define our intermediate lists and sublists, before giving the update procedure and its time complexity.

Definitions. Both lists L_l, L_r are of size $M \simeq 2^{\ell n}$. We store them in skip lists. In both L_r and L_l, for each $T \leqslant M$, we expect on average only one element $\mathbf{e}$ such that $\mathbf{e} \cdot \mathbf{a} = T \mod M$. We introduce two *Bucket-modulus lists* (Definition 6.26) $L'_l(B, M)$ and $L'_r(B, M)$ that we will write as L'_l and L'_r for simplicity, indexed by $\mathbf{e} \cdot \mathbf{a} \mod M$, with an arbitrary bound $B = \mathrm{poly}(n)$ for the bucket sizes. They are attached to L_l and L_r as detailed in Section 6.6.1. When an element in L_l or L_r is modified, they are modified accordingly.

In L'_l and L'_r, we consider the sublists of subknapsacks having the same modulo $C \mod 2^{cn}$, and we denote by $L'_{l,C}$ and $L'_{r,C}$ these sublists. They can be easily considered separately since

the vectors are sorted by knapsack weight. By design of the bucket-modulus lists, $L'_{l,C}$ and $L'_{r,C}$ both have size at most $B2^{(\ell-c)n}$. We have:

$$L'_l \bowtie_c L'_r = \bigcup_{0 \leqslant C \leqslant 2^{cn}-1} L'_{l,C} \times L'_{r,C} \ .$$

Next, we have a case disjunction to make. The most complicated case is when $2\ell - 2c + \mathsf{pf} > 0$, that is, each product $L'_{l,C} \times L'_{r,C}$ for a given C yields more than one filtered pair on average. In that case, we define sublists $L'_{l,C,i}$ of $L'_{l,C}$ and sublists $L'_{r,C,j}$ of $L'_{r,C}$ using a new arbitrary modular constraint, so that each of these sublists is of size $-\mathsf{pf}/2$ (at most). There are $\ell - c + \mathsf{pf}/2$ sublists (exactly). The rationale of this cut is that a product $L'_{l,C,i} \times L'_{r,C,j}$ for a given i, j now yields on average a single filtered pair (or less). When $2\ell - 2c + \mathsf{pf} \leqslant 0$, we don't perform this last cut and consider the product $L'_{l,C} \times L'_{r,C}$ immediately. By a slight abuse of notation, we denote: $(L'_{l,C,i} \times L'_{r,C,j})^f$ the set of filtered pairs from $L'_{l,C,i} \times L'_{r,C,j}$, and we have:

$$L^f = \bigcup_{0 \leqslant C \leqslant 2^{cn}-1} \bigcup_{i,j} (L'_{l,C,i} \times L'_{r,C,j})^f \ .$$

Algorithm 6.1 Update algorithm: given L_l, L_r of size ℓ, we insert or delete an element in L_l and update the filtered list L^f accordingly. We focus here on the case $2\ell - 2c + \mathsf{pf} > 0$.

Data: skip lists for L_l, L_r, L^f, bucket-modulus lists L'_l, L'_r
▷ The bucket-modulus list for L^f will be updated later
Input: an insertion / deletion instruction for L_l
Output: updates L_l, L'_l, L^f accordingly
Insert or delete in L_l ▷ only one element to update
Update the bucket-modulus structure L'_l ▷ at most B elements to update
  each element $\mathbf{e}$ to insert / delete in L'_l ▷ $B = \mathrm{poly}(n)$ iterations
Select its corresponding sublist $L'_{l,C,i}$
Let $L''_{l,C,i} = L'_{l,C,i} \cup \{\mathbf{e}\}$ or $L'_{l,C,i} \backslash \{\mathbf{e}\}$ each sublist $L'_{r,C,j}$ ▷ $\ell - c + \mathsf{pf}/2$ iterations
Estimate $s = |(L'_{l,C,i} \times L'_{r,C,j})^f|$ ▷ time $\widetilde{O}\big(B \times 2^{-\mathsf{pf}n/2}\big)$
Estimate $s' = |(L''_{l,C,i} \times L'_{r,C,j})^f|$ ▷ time $\widetilde{O}\big(B \times 2^{-\mathsf{pf}n/2}\big)$
▷ In the case of an insertion, $s' \geqslant s$ and $s' \leqslant s$ for a deletion
if $s > B$ and $s' \leqslant B$
▷ The removal of $\mathbf{e}$ makes the number of filtered pairs acceptable
  then $L^f \leftarrow L^f \cup (L''_{l,C,i} \times L'_{r,C,j})^f$
if $s > B$ and $s' > B$
  then do nothing
if $s \leqslant B$ and $s' > B$
▷ The insertion of $\mathbf{e}$ overflows the filtered pairs
  then remove all $(L'_{l,C,i} \times L'_{r,C,j})^f$ from L^f
if $s \leqslant B$ and $s' \leqslant B$
  then update L^f with the (at most) B new or removed pairs

Algorithm and Complexity. Algorithm 6.1 details our update procedure. We now compute its time complexity and explain why it remains data-independent. Recall that we want to avoid the "bad cases" where an update goes on for too long: this is the case where an update in L_l (or L_r) creates too many updates in L^f. In Algorithm 6.1, we avoid this by deliberately limiting the number of elements that can be updated. We can see that L^f will be smaller than the "perfect" one for two reasons: • the bucket-modulus data structure loses some vectors, since the buckets are dropped when they overflow. • filtered pairs are lost. Indeed, the algorithm ensures that in L^f, at most B solutions $\mathbf{e}_l + \mathbf{e}_r$ come from a cross-product $L'_{l,C,i} \times L'_{r,C,j}$.

This makes the update procedure *history-independent* and its time complexity *data-independent*. Indeed:

Lemma 6.28. *The state of the data structures L_l, L_r, L^f after Algorithm 6.1 depends only on L_l, L_r, L^f before and on the element that was inserted / deleted.*

We omit a formal proof, as it follows from our definition of the bucket-modulus list and of Algorithm 6.1.

Lemma 6.29. *With a good choice of B, Algorithm 6.1 runs with a data-independent error in $o(2^n)$. The time complexity of Algorithm 6.1 is $\widetilde{O}\left(2^{(\ell-c)n}\right)$ and an update modifies $\widetilde{O}\left(2^{\max(\ell-c+\mathsf{pf}/2,0)n}\right)$ elements in the filtered list L^f at the next level (respectively $\ell - c$ and $\max(\ell - c + \mathsf{pf}/2, 0)$ in log scale).*

Proof. We check step by step the time complexity of Algorithm 6.1:

- Insertion and deletion from the skip list for L_l is done in $\mathrm{poly}(n)$, with a global error that can be omitted.
- The bucket-modulus list L'_l is updated in time $O(B) = \mathrm{poly}(n)$ without errors. At most B elements must be inserted or removed.
- For each insertion or removal in L'_l, we select the corresponding sublist $L'_{l,C,i}$ (or simply $L'_{l,C}$ if $2\ell - 2c + \mathsf{pf} \leqslant 0$). We look at the sublists $L'_{r,C,j}$ and we estimate the number of filtered pairs in the products $L'_{l,C,i} \times L'_{r,C,j}$ (of size $-\mathsf{pf}$), checking whether it is smaller or bigger than B. We explain in the next section (Section 6.6.3) how to do that reversibly in time $\widetilde{O}\left(B \times 2^{-\mathsf{pf}n/2}\right)$ ($-\mathsf{pf}/2$ in log scale). There are $\ell - c + \mathsf{pf}/2$ classical iterations, thus the total time is $\ell - c$.
- Depending whether we have found more or less than B filtered pairs, we will have to remove or to add all of them in L^f. This means that $B \times 2^{(\ell-c+\mathsf{pf}/2)n}$ insertion or deletion instructions will be passed over to L^f.

There are two sources of data-independent errors: first, the skip list data structure (see Section 6.6.1). Second, the procedure in the next section (Section 6.6.3). Both can be made exponentially small at the price of a polynomial overhead. Note that B will be set in order to get a sufficiently small probability of error (see the next section), and can be a global $O(n)$. However, the polynomial overhead of our update unitary grows with the number of levels. □

6.6.3 Estimating a Number of Solutions Reversibly

We give a reversible procedure that, given a search space X with good elements $G \subseteq X$, finds whether $|G| > B$ using $\widetilde{O}(B)$ independent Grover searches in X. This procedure uses a coupon collector instead of quantum counting [BHT98], because B is considered to be a constant, and we are interested in a good success probability rather than a quadratic speedup.

Lemma 6.30. *Let X be a search space of exact size $2^{\alpha n}$ and $G \subseteq X$ be a "good" subspace of exact size unknown. There exists a quantum algorithm that given superposition access to X, finds whether $|G| \geqslant B$ or not in time $\widetilde{O}\left(B\sqrt{X}\right)$ and with a negligible probability of error.*

Proof. Although $|G|$ is not known, we use the idea (see *e.g.* [BHMT02]) that we can perform quantum searches with an approximate number of iterations and still obtain solutions with a good probability.

More precisely, there exists a number t, depending on $|G|$ (unknown) and $|X|$ (known) such that after t iterations, the state will be exactly the uniform superposition of G. This ideal t is not an integer; it is between 1 and $\lceil \frac{\pi}{4}\sqrt{|X|} \rceil$ (we assume that there is at least a solution, otherwise we will also detect this). Since we don't know t, we will instead approximate it by a t' such that $\frac{t}{2} \leqslant t' \leqslant \frac{3t}{2}$. If we perform t' iterations with such a good t', we will fall on a state with a constant global amplitude b (roughly $\frac{1}{\sqrt{2}}$) on elements of G. Thus, we perform many different searches with different iteration numbers, ranging from $0, 1$ to $\lceil \frac{\pi}{4}\sqrt{|X|} \rceil$, and increasing exponentially. This ensures us that regardless of the value of t, one of these numbers will be an approximation sufficient for us.

Since we want the algorithm to work with a global error negligible and independent of $|G|$, we set $c = O\left(\ln \sqrt{|X|}\right)$ the number of different searches and perform $c'B$ copies of each. Thus, we have a total of $cc'B$ independent states $|\psi_i\rangle$ for $1 \leqslant i \leqslant c$. One of these packets approximates the good t at best, but we don't know which one. We see the state $\bigotimes_i (|\psi_i\rangle^{\otimes cB})$ as a superposition over tuples of $X^{cc'B}$:

$$\sum_{x_1,\dots x_{cc'B}} \alpha_{x_1,\dots x_{cc'B}} |x_1\rangle \dots |x_{cc'B}\rangle \ .$$

If $|G| < B$, then this qubit is always 0: we can immediately uncompute the quantum searches and we have obtained the result. If $|G| \geqslant B$, then some of these tuples contain B distinct solutions, but not all. We must ensure that their proportion is overwhelming, so that after uncomputing, the algorithm actually adds an error vector of negligible amplitude. To do that, we will only focus on the block of $c'B$ states that corresponds to the good t, since for them, we have a lower bound on the probability of finding a solution. The other states, that we dismiss, can only improve our success in finding B distinct solutions. So we now focus on $c'B$-tuples only.

Let us consider $|G| = B$ which is the worst case. First, we will look at the proportion of $c'B$-tuples that contain $(1-c'')c'bB$ solutions: this is the probability to succeed at least $(1-c'')c'bB$ times after $c'B$ independent trials of probability b each, and it is higher than $1-\exp(-c''c'Bb/2)$ by a Chernoff bound. Next, assuming that there are $(1-c'')c'bB$ independent solutions, we check the probability that they span all the B distinct solutions that there are in total. This is related to the coupon collector problem. The probability to miss at least a coupon among B after $c^{(3)}B\ln B$ trials is lower than $B^{-c^{(3)}+1}$. Thus, we may take $c^{(3)} = O(n)$, $c'' = \frac{1}{2}$ and $c' = O(n)$ for a total probability of failure in $o(2^{-n})$. □

6.6.4 Fraction of Marked Vertices

Now that we have computed the update time of NEW-QW, it remains to compute its fraction ε_{new} of marked vertices. We will show that $\varepsilon_{new} = \varepsilon\left(1 - \frac{1}{\mathrm{poly}(n)}\right)$ with overwhelming probability on the random subset-sum instance, where ε is the previous fraction in QW.

Consider a marked vertex in QW. There is a path in the data structure leading to the solution, hence a constant number of subknapsacks $\mathbf{e_1}, \ldots, \mathbf{e_t}$ such that the vertex will remain marked *if and only* if none of them is "accidentally" discarded by our new data structure. Thus, if G is the graph of the walk, we want to upper bound:

$$\Pr_{v\in G}\begin{pmatrix} v \text{ is marked in QW and} \\ \text{not marked in NEW-QW}\end{pmatrix} \leqslant \sum_{\mathbf{e_i}, 1\leqslant i\leqslant t} \Pr_{v\in G}\begin{pmatrix} \mathbf{e_i} \in v \text{ in QW} \\ \mathbf{e_i} \notin v \text{ in NEW-QW}\end{pmatrix} .$$

We focus on some level in the tree, on a list L of average size $2^{\ell n}$, and on a single vector $\mathbf{e_0}$ that must appear in L. Subknapsacks in L are taken from $\mathcal{B} \subseteq D^n[\alpha, \beta, \gamma]$. We study the event that $\mathbf{e_0}$ is accidentally discarded from L. This can happen for two reasons:

- we have $|\{\mathbf{e} \in L, \mathbf{e}\cdot\mathbf{a} = \mathbf{e_0}\cdot\mathbf{a} \mod 2^{\ell n}\}| > B$: the vector is dropped at the bucket-modulus level;
- at the next level, there are more than B pairs from some product of lists $L'_{l,C,i} \times L'_{r,C,j}$ to which the vector $\mathbf{e_0}$ belongs, that will pass the filter.

We remark the following to make our computations easier.

Fact 6.31. *We can replace the L from our new data structure* NEW-QW *by a list of exact size $2^{\ell n}$, which is a sublist from the list L in* QW*.*

At successive levels, our new data structure discards more and more vectors. Hence, the actual lists are smaller than in QW. However, removing a vector $\mathbf{e}$ from a list, if it does not unmark the vertex, does not increase the probability of unmarking it at the next level, since $\mathbf{e}$ does not belong to the unique solution.

Fact 6.32. *When a vertex in* NEW-QW *is sampled uniformly at random, given a list L at some merging level, we can assume that the elements of L are sampled uniformly at random from their distribution $\mathcal{B}$ (with a modular constraint).*

This fact translates Heuristic 6.4 as a global property of the Johnson graph. At the first level, nodes contain lists of exponential size which are sampled without replacement. However, when sampling with replacement, the probability of collisions is exponentially low. Thus, we can replace $\Pr_{v\in G}$ by $\Pr_{v\in G'}$ where G' is a "completed" graph containing all lists sampled uniformly at random with replacement. This adds only a negligible number of vertices and does not impact the probability of being discarded.

Number of Vectors Having the Same Modulus. Let $N \simeq 2^n$ and M be a divisor of N. Given a particular $\mathbf{e}_0 \in \mathcal{B}$ and a vector $\mathbf{a} \in \mathbb{Z}_N^n$,

$$\text{For } \mathbf{e} \in \mathcal{B}, \text{ define } X_{\mathbf{e}}(a) = \begin{cases} 1 & \text{if } \mathbf{e}\cdot\mathbf{a} = \mathbf{e}_0\cdot\mathbf{a} \pmod M \\ 0 & \text{otherwise}\end{cases}$$

We will first prove a result on the average number of vectors having the same modulus as $\mathbf{e}_0$, then we will use this in a Chernoff bound. Define

$$Y(\mathcal{B}, \mathbf{e}_0; \mathbf{a}) = \mathrm{card}\{\mathbf{e} \in \mathcal{B}, \mathbf{a} \cdot \mathbf{e} = \mathbf{a} \cdot \mathbf{e}_0 \pmod M\}.$$

where M divides $N \simeq 2^n$. For simplicity, we write $Y(\mathbf{a})$ for $Y(\mathcal{B}, \mathbf{e}_0; \mathbf{a})$ in the following. We are interested in $Y(\mathbf{a})$ as a random variable when $\mathbf{a}$ is drawn uniformly from $\mathbb{Z}_N^n$.

Lemma 6.33. *If $|\mathcal{B}| \gg M$, then with probability $1 - \mathrm{negl}(n)$,*

$$Y(\mathbf{a}) \leqslant 2\mathbb{E}_{\mathbf{a}}[Y(\mathbf{a})] \sim 2 \cdot \frac{|\mathcal{B}|}{M} \tag{6.1}$$

Proof. Following [NSS01], for any $z \in \mathbb{C}$, define $\mathcal{E}(z) = \exp(2\pi i z/M)$. It satisfies the identity

$$\forall k \in \mathbb{N}, \sum_{\lambda=0}^{kM-1} \mathcal{E}(\lambda u) = \begin{cases} 0 & \text{if } u \neq 0 \pmod M \\ kM & \text{if } u = 0 \pmod M \end{cases} \tag{6.2}$$

for any $u \in \mathbb{Z}$. We have

$$Y(\mathbf{a}) = \sum_{\mathbf{e} \in \mathcal{B}} X_{\mathbf{e}}(\mathbf{a}) \qquad \text{and} \qquad X_{\mathbf{e}}(\mathbf{a}) = \frac{1}{M} \sum_{\lambda=0}^{M-1} \mathcal{E}(\lambda \mathbf{a} \cdot (\mathbf{e} - \mathbf{e}_0)).$$

Therefore for any $\mathbf{e} \neq \mathbf{e}_0$,

$$\begin{aligned}
\mathbb{E}_{\mathbf{a}}[X_{\mathbf{e}}(\mathbf{a})] &= \frac{1}{N^n} \sum_{\mathbf{a} \in \mathbb{Z}_N^n} \frac{1}{M} \sum_{\lambda=0}^{M-1} \mathcal{E}(\lambda \mathbf{a} \cdot (\mathbf{e} - \mathbf{e}_0)) \\
&= \frac{1}{N^n} \sum_{\mathbf{a} \in \mathbb{Z}_N^n} \frac{1}{M} + \frac{1}{N^n} \sum_{\mathbf{a} \in \mathbb{Z}_N^n} \frac{1}{M} \sum_{\lambda=1}^{M-1} \mathcal{E}(\lambda \mathbf{a} \cdot (\mathbf{e} - \mathbf{e}_0)) \\
&= \frac{1}{M} + \frac{1}{MN^n} \sum_{\lambda=1}^{M-1} \sum_{\mathbf{a} \in \mathbb{Z}_N^n} \mathcal{E}(\lambda \mathbf{a} \cdot (\mathbf{e} - \mathbf{e}_0)) \\
&= \frac{1}{M} + \frac{1}{MN^n} \sum_{\lambda=1}^{M-1} \prod_{i=1}^{n} \sum_{a \in \mathbb{Z}_N} \mathcal{E}(\lambda a(\mathbf{e}^i - \mathbf{e}_0^i))) \\
&= \frac{1}{M} \qquad \text{(see below)}
\end{aligned}$$

where in the last step, we used that if $\mathbf{e} \neq \mathbf{e}_0$, there exists $i \in [1, n]$ such that $\mathbf{e}^i \neq \mathbf{e}_0^i$, where $\mathbf{e}^i$ is the ith component of $\mathbf{e}$ and hence of the i^{th} sum in the product is zero by (6.2) since $\lambda \neq 0 \pmod M$. It follows by linearity that

$$\mathbb{E}_{\mathbf{a}}[Y(\mathbf{a})] = \mathbb{E}_{\mathbf{a}}[X_{\mathbf{e}}(\mathbf{a})] + \sum_{\mathbf{e} \in \mathcal{B} \setminus \{\mathbf{e}_0\}} \mathbb{E}_{\mathbf{a}}[X_{\mathbf{e}}(\mathbf{a})] = 1 + \frac{|\mathcal{B}| - 1}{M}$$

since $\mathbf{e}_0 \in \mathcal{B}$. Similarly for any $\mathbf{e}, \mathbf{f} \in \mathcal{B} \setminus \{\mathbf{e}_0\}$,

$$\mathbb{E}_{\mathbf{a}}[X_{\mathbf{e}}(\mathbf{a}) X_{\mathbf{f}}(\mathbf{a})] = \frac{1}{N^n} \sum_{\mathbf{a} \in \mathbb{Z}_N^n} \left(\frac{1}{M} \sum_{\lambda=0}^{M-1} \mathcal{E}(\lambda \mathbf{a} \cdot (\mathbf{e} - \mathbf{e}_0)) \right) \left(\frac{1}{M} \sum_{\mu=0}^{M-1} \mathcal{E}(\mu \mathbf{a} \cdot (\mathbf{f} - \mathbf{e}_0)) \right)$$

$$
\begin{aligned}
&= \frac{1}{M^2 N^n} \sum_{\mathbf{a}\in\mathbb{Z}_N^n} \sum_{\lambda=0}^{M-1} \sum_{\mu=0}^{M-1} \mathcal{E}(\lambda\mathbf{a}\cdot(\mathbf{e}-\mathbf{e}_0))\mathcal{E}(\mu\mathbf{a}\cdot(\mathbf{f}-\mathbf{e}_0)) \\
&= \frac{1}{M^2 N^n} \sum_{\mathbf{a}\in\mathbb{Z}_N^n} \sum_{\lambda=0}^{M-1} \sum_{\mu=0}^{M-1} \mathcal{E}(\mathbf{a}\cdot(\lambda\mathbf{e}+\mu\mathbf{f}-(\lambda+\mu)\mathbf{e}_0)) \\
&= \frac{1}{M^2 N^n} \sum_{\lambda=0}^{M-1} \sum_{\mu=0}^{M-1} \sum_{\mathbf{a}\in\mathbb{Z}_N^n} \mathcal{E}(\mathbf{a}\cdot(\lambda\mathbf{e}+\mu\mathbf{f}-(\lambda+\mu)\mathbf{e}_0)) \\
&= \frac{1}{M^2} \sum_{\lambda=0}^{M-1} \sum_{\mu=0}^{M-1} \mathbb{1}\big[\lambda\mathbf{e}+\mu\mathbf{f} = (\lambda+\mu)\mathbf{e}_0 \mod M\big]
\end{aligned}
$$

by (6.2). If $\lambda = 0$ then the equation $\lambda\mathbf{e}+\mu\mathbf{f} = (\lambda+\mu)\mathbf{e}_0 \mod M$ becomes $\mu\mathbf{f} = \mu\mathbf{e}_0$ but since $\mathbf{f} \neq \mathbf{e}_0$, the only solution is $\mu = 0$. A symmetric reasoning shows that if $\mu = 0$ then $\lambda = 0$ is the only solution. Hence, given $\mathbf{e} \neq \mathbf{e}_0$, we have

$$
\sum_{\mathbf{f}\in\mathcal{B}\setminus\{\mathbf{e}_0\}} \mathbb{E}_{\mathbf{a}}[X_{\mathbf{e}}(\mathbf{a})X_{\mathbf{f}}(\mathbf{a})] = \frac{|\mathcal{B}| - 1 + |F_{\mathbf{e}}|}{M^2}
$$

where

$$
F_{\mathbf{e}} = \big\{(\lambda,\mu,\mathbf{f}) \in \{1,\dots,M-1\}^2 \times \mathcal{B}\setminus\{\mathbf{e}_0\} : \lambda\mathbf{e}+\mu\mathbf{f} = (\lambda+\mu)\mathbf{e}_0 \mod M\big\}.
$$

We now claim that this set is not too large. Assume that $(\lambda,\mu,\mathbf{f}) \in F_{\mathbf{e}}$, recall that $\lambda,\mu \neq 0$ and since $\mathbf{e} \neq \mathbf{e}_0$ then there exists i such that $\mathbf{e}^i \neq \mathbf{e}_0^i$ so in particular $\lambda(\mathbf{e}^i - \mathbf{e}_0^i) = \mu(\mathbf{f}^i - \mathbf{e}_0^i)$. But recall that $\mathbf{e},\mathbf{f},\mathbf{e}_0 \in \mathcal{B} \subseteq \{-1,0,1,2\}^n$ hence $\mathbf{e}^i - \mathbf{e}_0^i \in \{-3,-2,-1,1,2,3\}$. It follows that if we fix μ then there are at most 3 possible values[2] for λ. We note in passing that the constant 3 is not magical: if we had $\mathcal{B} \subseteq \{-a,\dots,a\}^n$ then it be bounded by $2a$. Now assume that $(\lambda,\mu,\mathbf{f}),(\lambda,\mu,\mathbf{g}) \in F_{\mathbf{e}}$ with $\mathbf{f} \neq \mathbf{g}$, then we must have

$$
\begin{aligned}
\mu(\mathbf{f}-\mathbf{g}) = 0 \mod M \quad &\Rightarrow \quad \mathbf{f}-\mathbf{g} = 0 \mod M/\mu \\
&\Rightarrow \quad \exists k \neq 0. \forall i, \mathbf{g}^i = \mathbf{f}^i + kM/\mu
\end{aligned}
$$

which is only possible if μ divides M. In particular, we must have $M/\mu \geqslant 2$, in other words all coordinates of $\mathbf{f}$ and $\mathbf{g}$ are at least at distance 2. This is clearly impossible because $\mathcal{B} \subseteq D^n[\alpha,\beta,\gamma]$: the distribution of "-1", "0", "1", "2" in one of $\mathbf{f}$ or $\mathbf{g}$ would be wrong. In summary, we have that:

- for every μ, $\mathbf{f}$, there are at most 3 possible values of λ such that $(\lambda,\mu,\mathbf{f}) \in F_{\mathbf{e}}$,
- for every λ and μ, there is at most one value of $\mathbf{f}$ such that $(\lambda,\mu,\mathbf{f}) \in F_{\mathbf{e}}$.

It follows that $F_{\mathbf{e}}$ has size at most $3M$. Then by linearity,

$$
\mathbb{E}_{\mathbf{a}}[Y(\mathbf{a})^2] = \sum_{\mathbf{e},\mathbf{f}\in\mathcal{B}} \mathbb{E}_{\mathbf{a}}[X_{\mathbf{e}}(\mathbf{a})X_{\mathbf{f}}(\mathbf{a})]
$$

[2] If we have, say, $3\lambda = x \pmod M$ then $\lambda = x/3 \pmod{M/3}$, which is only possible if x and M are divisible by 3, and then $\lambda \in \{x/3, (x+M)/3, (x+2M)/3\}$.

$$
\begin{aligned}
&= \sum_{\mathbf{f}\in\mathcal{B}} \mathbb{E}_{\mathbf{a}}[X_{\mathbf{f}}(\mathbf{a})] + \sum_{\mathbf{e}\in\mathcal{B}\setminus\{\mathbf{e}_0\}} \mathbb{E}_{\mathbf{a}}[X_{\mathbf{e}}(\mathbf{a})] + \sum_{\mathbf{e},\mathbf{f}\in\setminus\{\mathbf{e}_0\}} \mathbb{E}_{\mathbf{a}}[X_{\mathbf{e}}(\mathbf{a})X_{\mathbf{f}}(\mathbf{a})] \\
&\leqslant 2\mathbb{E}_{\mathbf{a}}[Y(\mathbf{a})] - 1 + (|\mathcal{B}|-1)\frac{|\mathcal{B}|-1+3M}{M^2} \\
&\leqslant 1 + 2\frac{|\mathcal{B}|-1}{M} + (|\mathcal{B}|-1)\frac{|\mathcal{B}|-1+3M}{M^2} \\
&\leqslant \frac{M^2 + (|\mathcal{B}|-1)(|\mathcal{B}|-1+5M)}{M^2}.
\end{aligned}
$$

Finally, we have

$$
\begin{aligned}
\mathbb{V}_{\mathbf{a}}(Y(\mathbf{a})) &= \mathbb{E}_{\mathbf{a}}[Y(\mathbf{a})^2] - \mathbb{E}_{\mathbf{a}}[Y(\mathbf{a})]^2 \\
&\leqslant \frac{M^2 + (|\mathcal{B}|-1)(|\mathcal{B}|-1+5M) - (|\mathcal{B}|+M-1)^2}{M^2} \\
&= \frac{3(|\mathcal{B}|-1)M}{M^2} \\
&= \frac{3(|\mathcal{B}|-1)}{M}.
\end{aligned}
$$

Thus $\mathbb{E}_{\mathbf{a}}[Y(\mathbf{a})] \approx \mathbb{V}_{\mathbf{a}}(Y(\mathbf{a}))$ when we look at their order of magnitude.

According to Tchebychev's inequality,

$$
\Pr_{\mathbf{a}}[|Y(\mathbf{a}) - \mathbb{E}_{\mathbf{a}}[Y(\mathbf{a})]| > \mathbb{E}_{\mathbf{a}}[Y(\mathbf{a})]] \leqslant \frac{\mathbb{V}_{\mathbf{a}}(Y(\mathbf{a}))}{\mathbb{E}_{\mathbf{a}}[Y(\mathbf{a})]^2} = \mathrm{negl}(n)
$$

which completes the proof. □

Lemma 6.34. *If $|\mathcal{B}| \gg M \simeq |L|$, then for a $1-\mathrm{negl}(n)$ proportion of $\mathbf{a} \in \mathbb{Z}_N^n$, and with an appropriate $B = O(n)$:*

$$
\Pr_{\mathbf{e}_1,\cdots,\mathbf{e}_{|L|}\sim Unif(\mathcal{B})}\left[\sum_{i=1}^{|L|} X_{\mathbf{e}_i}(\mathbf{a}) < B-1\right] > 1 - \frac{1}{\mathrm{poly}(n)} \tag{6.3}
$$

Proof.

$$
\begin{aligned}
&\Pr_{\mathbf{e}_1,\cdots,\mathbf{e}_{|L|}\sim Unif(\mathcal{B})}\left[\sum_{i=1}^{|L|} X_{\mathbf{e}_i}(\mathbf{a}) < B-1\right] \\
&\quad= \sum_{y=1}^{|\mathcal{B}|} \Pr_{\mathbf{e}_1,\cdots,\mathbf{e}_{|L|}\sim Unif(\mathcal{B})}\left[\sum_{i=1}^{|L|} X_{\mathbf{e}_i}(\mathbf{a}) < B-1 | Y(\mathbf{a}) = y\right] \Pr[Y(\mathbf{a}) = y]
\end{aligned}
$$

Under the condition of $Y(\mathbf{a}) = y$, for all $i \in [1, |L|]$, $X_{\mathbf{e}_i}(\mathbf{a})$ can be seen as a random variable following $\mathrm{Ber}(\frac{y}{|\mathcal{B}|})$, since $\mathbf{e}_i$s' are randomly chosen from $\mathcal{B}$. Here $\mathrm{Ber}(p)$ is a Bernoulli distribution of parameter p.

Using equation (6.1), for a $1 - \mathrm{negl}(n)$ portion of $\mathbf{a} \in \mathbb{Z}_N^n$, we have

$$\begin{aligned}
&\Pr_{\mathbf{e}_1,\cdots,\mathbf{e}_{|L|}\sim Unif(\mathcal{B})}\left[\sum_{i=1}^{|L|} X_{\mathbf{e}_i}(\mathbf{a}) < B-1\right] \\
&> \sum_{y=1}^{2\cdot\frac{|\mathcal{B}|}{M}} \Pr_{\mathbf{e}_1,\cdots,\mathbf{e}_{|L|}\sim Unif(\mathcal{B})}\left[\sum_{i=1}^{|L|} X_{\mathbf{e}_i}(\mathbf{a}) < B-1 | Y(\mathbf{a}) = y\right] \Pr[Y(\mathbf{a}) = y] \\
&= (1-\mathrm{negl}(n)) \sum_{y=1}^{2\cdot\frac{|\mathcal{B}|}{M}} \Pr_{\mathbf{e}_1,\cdots,\mathbf{e}_{|L|}\sim Unif(\mathcal{B})}\left[\sum_{i=1}^{|L|} X_{\mathbf{e}_i}(\mathbf{a}) < B-1 | Y(\mathbf{a}) = y\right] \\
&> (1-\mathrm{negl}(n)) \Pr_{\mathbf{e}_1,\cdots,\mathbf{e}_{|L|}\sim Unif(\mathcal{B})}\left[\sum_{i=1}^{|L|} X_{\mathbf{e}_i}(\mathbf{a}) < B-1 | Y(\mathbf{a}) = 2\cdot\frac{|\mathcal{B}|}{M}\right] \\
&= (1-\mathrm{negl}(n)) \Pr_{X_{\mathbf{e}_i}\sim \mathrm{Ber}(2\cdot\frac{|\mathcal{B}|}{M}\cdot\frac{1}{|\mathcal{B}|})=\mathrm{Ber}(\frac{2}{M})}\left[\sum_{i=1}^{|L|} X_{\mathbf{e}_i}(\mathbf{a}) < B-1\right]
\end{aligned}$$

Chernoff's inequality gives that for any $\delta \geqslant 1$:

$$\Pr\left[\sum_{i=1}^{|L|} X_{\mathbf{e}_i}(\mathbf{a}) \geqslant (1+\delta)\frac{2|L|}{M}\right] \leqslant e^{-\frac{\delta}{3}\frac{2|L|}{M}} \ .$$

Hence, when $M = |L|$, by taking B linear in n, we obtain that the probability of being unmarked due to this $\mathbf{e_0}$ is less than $\frac{1}{\mathrm{poly}(n)}$. □

For the number of filtered pairs, we use the fact that the vectors at each level are sampled uniformly at random from their distribution. If this is the case, then a Chernoff bound (similar to the proof of Lemma 6.34) limits the deviation of the number of filtered pairs in $L'_{l,C,i} \times L'_{r,C,j}$ from its expectation (which is 1 by construction): the probability that there are more than $B+1$ pairs is smaller than $e^{-(B+1)/3}$. By taking a sufficiently big $B = O(n)$, we can take a union bound over all products of lists $L'_{l,C,i} \times L'_{r,C,j}$ in which $\mathbf{e_0}$ intervenes. We also take a union bound over the intermediate subknapsacks that we are considering. The loss of vertices remains inverse polynomial.

6.6.5 Time Complexities without Heuristic 2

Previous quantum subset-sum algorithms [BJLM13, HM18] have the same time complexities without Heuristic 6.17, as they fall in parameter ranges where the bucket-modulus data structure is enough. However, this is not the case of our new quantum walk. We keep the same set of constraints and optimize with a new update time. Although using the extended $\{-1, 0, 1, 2\}$ representations brings an improvement, neither do the fifth level, nor the left-right split. This simplifies our constraints. Let $\widehat{\max}(\cdot) = \max(\cdot, 0)$. The guaranteed update time becomes:

$$\mathsf{U} = \widehat{\max}\Bigg(\underbrace{\ell_3 - (c_2 - c_3)}_{\text{Level 2}}, \quad \underbrace{\widehat{\max}(\ell_3 - (c_2 - c_3) + \frac{p_2}{2})}_{\text{Number of elements to update at level 1}} \quad + \widehat{\max}(\ell_2 - (c_1 - c_2)),$$
$$\underbrace{\frac{1}{2}\Big(\widehat{\max}\Big(\ell_3 - (c_2 - c_3) + \frac{p_2}{2}\Big) + \widehat{\max}\Big(\ell_2 - (c_1 - c_2) + \frac{p_1}{2}\Big) + \widehat{\max}(\ell_1 - (1 - c_1))\Big)}_{\text{Final quantum search among all updated elements}} \Bigg)$$

We obtain the time exponent 0.2182 (rounded upwards) with the following parameters (rounded). The memory exponent is 0.2182 as well.

$$\ell_0 = -0.2021, \ell_1 = 0.1883, \ell_2 = 0.2102, \ell_3 = 0.2182, \ell_4 = 0.2182$$
$$c_3 = 0.2182, c_2 = 0.4283, c_1 = 0.6305, p_0 = -0.2093, p_1 = -0.0298, p_2 = -0.0160$$
$$\alpha_1 = 0.0172, \alpha_2 = 0.0145, \alpha_3 = 0.0107, \gamma_1 = 0.0020$$

Chapter 7 Conclusion

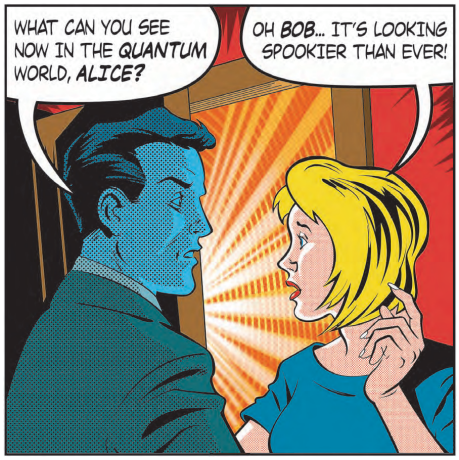
In this chapter, we summarize the main contributions of this thesis and present some future directions that could be explored.

An important aspect of all practical SVP algorithm is that they rely on heuristics to analyze their complexity. In particular, there is a significant gap between the best heuristic and provable algorithms for SVP. While provable algorithms are less of an interest for cryptography, since the choice of security parameters are based on heuristics that are much faster, they are still interesting from a theoretical perspective. The (provable) complexity of the SVP is still far from being well-understood. In Chapter 3 we managed to obtain a provable trade-off between time and memory for the SVP. However, we did not optimize the constants in the exponents since their origin in our proofs cannot be explained easily: They must satisfy nontrivial lower and upper bounds for the proofs to work. A more thorough analysis of our algorithm would be of interest to compare our trade-off with the best sieving algorithms and the best enumeration algorithms in the two extremes. Furthermore, even the best provable algorithm remains somewhat of a mystery [ADRS15, AS18b]. Its 2^n running time is based on Discrete Gaussian Sampling (DGS), a prominent tool in lattice-based cryptography. The algorithm works by sampling a Gaussian with parameter roughly $\frac{\lambda_1(\mathcal{L})}{\sqrt{n}}$, a task made difficult by the fact that this parameter can be quite below the smoothing parameter $\eta_\varepsilon(\mathcal{L})$ of the lattice. There are more efficient algorithms to sample discrete Gaussian above the smoothing parameter, notably a $2^{n/2}$ algorithm by the same authors, using towers of lattices [ADRS15]. While $\eta_\varepsilon(\mathcal{L})$ can be in the order of $\lambda_n(\mathcal{L})$ in the worst case, we expect a random lattice to have smoothing parameter in the order of $\frac{\lambda_1(\mathcal{L})}{\sqrt{n}}$, hence within the reach of more efficient algorithms. Therefore a natural question is whether DGS and hence SVP is easier for random lattices.

In Chapter 4, we obtained the fastest quantum algorithm without any heuristic assumptions for SVP and the fastest classical algorithm that has a space complexity $2^{0.5n+o(n)}$. An intriguing feature of our algorithms is that their complexities depends on a quantity related to kissing number of the lattice. While the only known upper bound on this number is exponential ($2^{0.401n+o(n)}$), very little is known about kissing numbers in general. So far it is only known that some family of lattices can have a barely exponential kissing number ($2^{0.0338n}$) and most practical lattices have a sub-exponential ($2^{o(n)}$) kissing number. This suggests that more algorithms could benefit from a similar analysis to ours and that some provable algorithms are more competitive in practice than imagined. One may also imagine a link between the lattice kissing number and the gap of a lattice defined as the quotient of successive minima. Lattices with a gap appear when solving the Learning With Errors (LWE) problem with the embedding technique and there have been some studies on the SVP problem in this setting [WLW]. Finally, the exact relationship between the kissing number and the quantity $\beta(\mathcal{L})$

that we introduced is unclear, more work will be needed to clarify the situation.

Enumeration is another important technique to solve SVP and CVP, and it has been significantly improved in practice thanks to cylinder pruning, discrete and extreme pruning methods. In Chapter 5, we obtained a quantum quadratic speed-up for both cylinder pruning and discrete pruning and developed several several tweaks to make discrete pruning more efficient and powerful. Whereas a lower bound for the extreme cylinder pruning is given in [ANSS18], no study of the limitations of discrete pruning exists. Recent work has shown that enumeration with cylinder pruning can be used to improve the complexity of lattice reduction algorithms [ABF+20]. It remains open whether further improvements can be obtained by using discrete pruning instead.



Another important feature of our results in Chapter 4 and Chapter 6 is the type of quantum memory used. There are three types of quantum memories, in increasing order of strength: plain qubit, quantum accessible classical memory (QRACM) and quantum accessible quantum memory (QRAQM). The QRAQM model is used in most quantum walk algorithms to date, but its practical time-efficient realizations are still unclear. The QRACM model is omnipresent in Quantum Machine Learning algorithms (see for example [Pra14]). All previously-known quantum algorithms for the SVP relied on QRACM whereas our algorithms in Chapter 4 only use plain qubits. We similarly improved algorithms for the random subset sum problem by demonstrating an algorithm using QRACM whose complexity is comparable with the state-of-art using QRAQM. We also obtained the fastest algorithm for this problem using QRAQM. A natural question is whether our improvements in SVP and subset-sum algorithms can be replicated in coding theory where only QRAQM algorithms exist for quantum information set decoding, due to the nested structure of the k-sum problem inside the search of the information set. Another interesting direction is to look at quantum algorithms for the Short Integer Solution problem (SIS) which shares some similar structure with the random subset sum problem.

In Chapter 6 we successfully removed a heuristic used in quantum walk in previous algorithms for decoding and subset sum, namely an update with expected constant time U can be replaced by an update with exact time U without affecting the runtime of the algorithm, up to a polynomial factor. We designed a new data structure for the vertices in the quantum walk,

[0]Image from Physics World march 2013.

and a new update procedure with guaranteed time to prove the correctness of this statement. However in our best QRAQM algorithm, we introduced a technique to speed up the filtering of representations with a quantum search. Our new data structure does not guarantee that the update can be done in constant time for algorithms using this technique. One can imagine another approach where we study the quantum walk with imperfect setup and updates and try to bound the amplitude of the bad states for each step of the walk and show than in the end the imperfect quantum walk returns a marked vertex with constant probability. Another possible way to resolve this issue is to use learning graphs [CLM20] for which the *expected* update time seems to appear naturally in the analyses without introducing heuristics.

Bibliography

[ABB+17] Erdem Alkim, Nina Bindel, Johannes A. Buchmann, Özgür Dagdelen, Edward Eaton, Gus Gutoski, Juliane Krämer, and Filip Pawlega. Revisiting TESLA in the quantum random oracle model. In *Proc. PQCrypto 2017*, volume 10346 of *Lecture Notes in Computer Science*, pages 143–162. Springer, 2017.

[ABF+20] Martin R. Albrecht, Shi Bai, Pierre-Alain Fouque, Paul Kirchner, Damien Stehlé, and Weiqiang Wen. Faster enumeration-based lattice reduction: Root hermite factor $k^{1/(2k)}$ time $k^{k/8+o(k)}$. In Daniele Micciancio and Thomas Ristenpart, editors, *Advances in Cryptology - CRYPTO 2020 - 40th Annual International Cryptology Conference, CRYPTO 2020, Santa Barbara, CA, USA, August 17-21, 2020, Proceedings, Part II*, volume 12171 of *Lecture Notes in Computer Science*, pages 186–212. Springer, 2020.

[ABI+20] Andris Ambainis, Kaspars Balodis, Janis Iraids, Kamil Khadiev, Vladislavs Klevickis, Krisjanis Prusis, Yixin Shen, Juris Smotrovs, and Jevgenijs Vihrovs. Quantum lower and upper bounds for 2d-grid and dyck language. In Javier Esparza and Daniel Král', editors, *45th International Symposium on Mathematical Foundations of Computer Science, MFCS 2020, August 24-28, 2020, Prague, Czech Republic*, volume 170 of *LIPIcs*, pages 8:1–8:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.

[ABLR20] Martin R. Albrecht, Shi Bai, Jianwei Li, and Joe Rowell. Lattice reduction with approximate enumeration oracles: Practical algorithms and concrete performance. *IACR Cryptol. ePrint Arch.*, 2020:1260, 2020.

[ACF+15] Martin R. Albrecht, Carlos Cid, Jean-Charles Faugère, Robert Fitzpatrick, and Ludovic Perret. On the complexity of the bkw algorithm on lwe. *Des. Codes Cryptography*, 74(2):325–354, February 2015.

[ACKS21] Divesh Aggarwal, Yanlin Chen, Rajendra Kumar, and Yixin Shen. Improved (provable) algorithms for the shortest vector problem via bounded distance decoding. In Markus Bläser and Benjamin Monmege, editors, *38th International Symposium on Theoretical Aspects of Computer Science, STACS 2021, March 16-19, 2021, Saarbrücken, Germany (Virtual Conference)*, volume 187 of *LIPIcs*, pages 4:1–4:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.

[ADH+19] Martin R. Albrecht, Léo Ducas, Gottfried Herold, Elena Kirshanova, Eamonn W. Postlethwaite, and Marc Stevens. The general sieve kernel and new records in lattice reduction. 11477:717–746, 2019.

[ADPS16] Erdem Alkim, Léo Ducas, Thomas Pöppelmann, and Peter Schwabe. Post-quantum key exchange - A new hope. In *Proc. 25th USENIX*, pages 327–343. USENIX, 2016.

[ADRS15] Divesh Aggarwal, Daniel Dadush, Oded Regev, and Noah Stephens-Davidowitz. Solving the shortest vector problem in 2^n time using discrete gaussian sampling: Extended abstract. In *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, June 14-17, 2015*, pages 733–742, 2015.

[AFV11] Shweta Agrawal, David Mandell Freeman, and Vinod Vaikuntanathan. Functional encryption for inner product predicates from learning with errors. In Dong Hoon Lee and Xiaoyun Wang, editors, *Advances in Cryptology – ASIACRYPT 2011*, pages 21–40, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[AGJ19] Simon Apers, András Gilyén, and Stacey Jeffery. A unified framework of quantum walk search, 2019.

[AGS18] Scott Aaronson, Daniel Grier, and Luke Schaeffer. A quantum query complexity trichotomy for regular languages. *Electronic Colloquium on Computational Complexity (ECCC)*, 26:61, 2018.

[Ajt96] Miklós Ajtai. Generating hard instances of lattice problems (extended abstract). In *Proceedings of the Twenty-Eighth Annual ACM Symposium on the Theory of Computing, Philadelphia, Pennsylvania, USA, May 22-24, 1996*, pages 99–108, 1996.

[Ajt98] Miklós Ajtai. The shortest vector problem in l2 is np-hard for randomized reductions (extended abstract). In *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing*, STOC '98, page 10–19, New York, NY, USA, 1998. Association for Computing Machinery.

[AK17] Andris Ambainis and Martins Kokainis. Quantum algorithm for tree size estimation, with applications to backtracking and 2-player games. In *Proc. STOC '17*. ACM, 2017.

[AKS01] Miklós Ajtai, Ravi Kumar, and D. Sivakumar. A sieve algorithm for the shortest lattice vector problem. In *Proceedings of the Thirty-third Annual ACM Symposium on Theory of Computing*, STOC '01, pages 601–610, New York, NY, USA, 2001. ACM.

[Ale03] M. Alekhnovich. More on average case vs approximation complexity. In *44th Annual IEEE Symposium on Foundations of Computer Science, 2003. Proceedings.*, pages 298–307, 2003.

[ALNS20] Divesh Aggarwal, Jianwei Li, Phong Q. Nguyen, and Noah Stephens-Davidowitz. Slide reduction, revisited—Filling the gaps in SVP approximation. In *CRYPTO*, 2020.

[Amb07] Andris Ambainis. Quantum Walk Algorithm for Element Distinctness. *SIAM J. Comput.*, 37(1):210–239, 2007.

[Amb10] Andris Ambainis. Quantum search with variable times. *Theory Comput. Syst.*, 47(3):786–807, 2010.

[AN17] Yoshinori Aono and Phong Q. Nguyen. Random sampling revisited: Lattice enumeration with discrete pruning. In *Advances in cryptology—EUROCRYPT 2017 Part II*, volume 10211 of *LNCS*, pages 65–102. Springer, 2017.

[ANS18] Yoshinori Aono, Phong Q. Nguyen, and Yixin Shen. Quantum lattice enumeration and tweaking discrete pruning. In Thomas Peyrin and Steven Galbraith, editors, *Advances in Cryptology – ASIACRYPT 2018*, pages 405–434, Cham, 2018. Springer International Publishing.

[ANSS18] Y. Aono, P. Q. Nguyen, T. Seito, and J. Shikata. Lower bounds on lattice enumeration with extreme pruning. In *Proc. of 38th CRYPTO, Part II*, volume 10992 of *LNCS*. Springer, 2018.

[AS18a] Divesh Aggarwal and Noah Stephens-Davidowitz. (gap/s) eth hardness of svp. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing*, pages 228–238, 2018.

[AS18b] Divesh Aggarwal and Noah Stephens-Davidowitz. Just take the average! an embarrassingly simple 2ˆn-time algorithm for SVP (and CVP). In *1st Symposium on Simplicity in Algorithms, SOSA 2018, January 7-10, 2018, New Orleans, LA, USA*, pages 12:1–12:19, 2018.

[AWHT16] Yoshinori Aono, Yuntao Wang, Takuya Hayashi, and Tsuyoshi Takagi. Improved progressive BKZ algorithms and their precise cost estimation by sharp simulator. *IACR Cryptology ePrint Archive*, 2016:146, 2016. Full version of EUROCRYPT 2016.

[Bab85] László Babai. On Lovász' lattice reduction and the nearest lattice point problem. In *Proc. STACS'85*, volume 182 of *LNCS*, pages 13–20. Springer, 1985.

[BBBV97] Charles H. Bennett, Ethan Bernstein, Gilles Brassard, and Umesh V. Vazirani. Strengths and weaknesses of quantum computing. *SIAM J. Comput.*, 26(5):1510–1523, 1997.

[BBHT05] Michel Boyer, Gilles Brassard, Peter Hoyer, and Alain Tappa. *Tight Bounds on Quantum Searching*, volume 46, pages 187 – 199. 01 2005.

[BBSS20] Xavier Bonnetain, Rémi Bricout, André Schrottenloher, and Yixin Shen. Improved classical and quantum algorithms for subset-sum. In Shiho Moriai and Huaxiong Wang, editors, *Advances in Cryptology - ASIACRYPT 2020 - 26th International Conference on the Theory and Application of Cryptology and Information Security, Daejeon, South Korea, December 7-11, 2020, Proceedings, Part II*, volume 12492 of *Lecture Notes in Computer Science*, pages 633–666. Springer, 2020.

[BCDL19] Rémi Bricout, André Chailloux, Thomas Debris-Alazard, and Matthieu Lequesne. Ternary syndrome decoding with large weight. In *SAC 2019*, volume 11959 of *LNCS*, pages 437–466. Springer, 2019.

[BCJ11] Anja Becker, Jean-Sébastien Coron, and Antoine Joux. Improved generic algorithms for hard knapsacks. In *EUROCRYPT*, volume 6632 of *LNCS*, pages 364–385. Springer, 2011.

[BDGL16] Anja Becker, Léo Ducas, Nicolas Gama, and Thijs Laarhoven. New directions in nearest neighbor searching with applications to lattice sieving. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 10–24, 2016.

[BEG+18] Mahdi Boroujeni, Soheil Ehsani, Mohammad Ghodsi, MohammadTaghi Hajiaghayi, and Saeed Seddighin. Approximating edit distance in truly subquadratic time: Quantum and mapreduce. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1170–1189. SIAM, 2018.

[Ben89] Charles H. Bennett. Time/space trade-offs for reversible computation. *SIAM J. Comput.*, 18(4):766–776, 1989.

[BGJ13] Anja Becker, Nicolas Gama, and Antoine Joux. Solving shortest and closest vector problems: The decomposition approach. *IACR Cryptol. ePrint Arch.*, 2013:685, 2013.

[BHMT02] Gilles Brassard, Peter Hoyer, Michele Mosca, and Alain Tapp. Quantum amplitude amplification and estimation. *Contemporary Mathematics*, 305:53–74, 2002.

[BHT98] Gilles Brassard, Peter Høyer, and Alain Tapp. Quantum counting. In *ICALP*, volume 1443 of *Lecture Notes in Computer Science*, pages 820–831. Springer, 1998.

[BI15] Arturs Backurs and Piotr Indyk. Edit distance cannot be computed in strongly subquadratic time (unless seth is false). In *Proceedings of the forty-seventh annual ACM symposium on Theory of computing*, pages 51–58. ACM, 2015.

[BJLM13] Daniel J. Bernstein, Stacey Jeffery, Tanja Lange, and Alexander Meurer. Quantum algorithms for the subset-sum problem. In *PQCrypto*, volume 7932 of *LNCS*, pages 16–33. Springer, 2013.

[BJMM12] Anja Becker, Antoine Joux, Alexander May, and Alexander Meurer. Decoding random binary linear codes in $2^{n/20}$: How $1 + 1 = 0$ improves information set decoding. In *EUROCRYPT*, volume 7237 of *LNCS*, pages 520–536. Springer, 2012.

[BKW03] Avrim Blum, Adam Kalai, and Hal Wasserman. Noise-tolerant learning, the parity problem, and the statistical query model. *J. ACM*, 50(4):506–519, July 2003.

[BLP+13] Zvika Brakerski, Adeline Langlois, Chris Peikert, Oded Regev, and Damien Stehlé. Classical hardness of learning with errors. In *Proceedings of the forty-fifth annual ACM symposium on Theory of computing*, pages 575–584. ACM, 2013.

[BLS16] Shi Bai, Thijs Laarhoven, and Damien Stehlé. Tuple lattice sieving. *IACR Cryptology ePrint Archive*, 2016:713, 2016.

[BM18] Leif Both and Alexander May. Decoding linear codes with high error rate and its impact for lpn security. In Tanja Lange and Rainer Steinwandt, editors, *Post-Quantum Cryptography*, pages 25–46, Cham, 2018. Springer International Publishing.

[BN18] Xavier Bonnetain and María Naya-Plasencia. Hidden shift quantum cryptanalysis and implications. In *ASIACRYPT (1)*, volume 11272 of *LNCS*, pages 560–592. Springer, 2018.

[Bon19] Xavier Bonnetain. Improved low-qubit hidden shift algorithms. *CoRR*, 2019.

[BPR12] Abhishek Banerjee, Chris Peikert, and Alon Rosen. Pseudorandom functions and lattices. In David Pointcheval and Thomas Johansson, editors, *Advances in Cryptology – EUROCRYPT 2012*, pages 719–737, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[BPS19] Harry Buhrman, Subhasree Patro, and Florian Speelman. The quantum strong exponential-time hypothesis, 2019.

[Bri84] Ernest F. Brickell. Breaking iterated knapsacks. In *Advances in Cryptology, Proceedings of CRYPTO '84, Santa Barbara, California, USA, August 19-22, 1984, Proceedings*, pages 342–358, 1984.

[BS20] Xavier Bonnetain and André Schrottenloher. Quantum security analysis of CSIDH. In *EUROCRYPT 2020*, LNCS. Springer, May 2020.

[Bö11] Elena Böhme. Verbesserte subset-sum algorithmen. Master's thesis, Ruhr Universität Bochum, 2011.

[CCL18] Yanlin Chen, Kai-Min Chung, and Ching-Yi Lai. Space-efficient classical and quantum algorithms for the shortest vector problem. *Quantum Information & Computation*, 18(3&4):285–306, 2018.

[CDG+18] Diptarka Chakraborty, Debarati Das, Elazar Goldenberg, Michal Koucký, and Michael E. Saks. Approximating edit distance within constant factor in truly sub-quadratic time. In *59th Annual IEEE Symposium on Foundations of Computer Science (FOCS), Paris, France, Oct 7-9, 2018*, pages 979–990, 2018.

[CDLP13] K. Chung, D. Dadush, F. Liu, and C. Peikert. On the lattice smoothing parameter problem. In *2013 IEEE Conference on Computational Complexity*, pages 230–241, 2013.

[Che13] Yuanmi Chen. *Réduction de réseau et sécurité concrète du chiffrement complètement homomorphe*. PhD thesis, Univ. Paris 7, 2013.

[CHKP10] David Cash, Dennis Hofheinz, Eike Kiltz, and Chris Peikert. Bonsai trees, or how to delegate a lattice basis. In Henri Gilbert, editor, *Advances in Cryptology – EUROCRYPT 2010*, pages 523–552, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[CLM20] Titouan Carette, Mathieu Laurière, and Frédéric Magniez. Extended learning graphs for triangle finding. *Algorithmica*, 82(4):980–1005, 2020.

[CN11] Yuanmi Chen and Phong Q. Nguyen. BKZ 2.0: better lattice security estimates. In *Proc. ASIACRYPT 2011*, volume 7073 of *LNCS*, pages 1–20. Springer, 2011.

[dB89] Rudi de Buda. Some optimal codes have structure. *IEEE Journal on Selected Areas in Communications*, 7(6):893–899, 1989.

[DH76] Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, November 1976.

[DH96] Christoph Dürr and Peter Høyer. A quantum algorithm for finding the minimum. *CoRR*, quant-ph/9607014, 1996.

[dPLP16] Rafaël del Pino, Vadim Lyubashevsky, and David Pointcheval. The whole is less than the sum of its parts: Constructing more efficient lattice-based AKEs. In *Proc. SCN 2016*, volume 9841 of *Lecture Notes in Computer Science*, pages 273–291. Springer, 2016.

[DRS14] Daniel Dadush, Oded Regev, and Noah Stephens-Davidowitz. On the closest vector problem with a distance guarantee. In *IEEE 29th Conference on Computational Complexity, CCC 2014, Vancouver, BC, Canada, June 11-13, 2014*, pages 98–109, 2014.

[Duc18] Léo Ducas. Shortest vector from lattice sieving: A few dimensions for free. In Jesper Buus Nielsen and Vincent Rijmen, editors, *Advances in Cryptology - EUROCRYPT 2018 - 37th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tel Aviv, Israel, April 29 - May 3, 2018 Proceedings, Part I*, volume 10820 of *Lecture Notes in Computer Science*, pages 125–145. Springer, 2018.

[EM19] Andre Esser and Alexander May. Better sample - random subset sum in $2^{0.255n}$ and its impact on decoding random linear codes. *CoRR*, abs/1907.04295, 2019.

[FT87] András Frank and Éva Tardos. An application of simultaneous diophantine approximation in combinatorial optimization. *Combinatorica*, 7(1):49–65, 1987.

[Gen09] Craig Gentry. Fully homomorphic encryption using ideal lattices. In *Proceedings of the 41st Annual ACM Symposium on Theory of Computing, STOC 2009, Bethesda, MD, USA, May 31 - June 2, 2009*, pages 169–178, 2009.

[GGH97] O. Goldreich, S. Goldwasser, and S. Halevi. Public-key cryptosystems from lattice reduction problems. In *Proc. CRYPTO 1997*, volume 1294 of *LNCS*, pages 112–131. Springer, 1997.

[GJ79] M. R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

[GKV10] S. Dov Gordon, Jonathan Katz, and Vinod Vaikuntanathan. A group signature scheme from lattice assumptions. In Masayuki Abe, editor, *Advances in Cryptology - ASIACRYPT 2010*, pages 395–412, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[GLM08] Vittorio Giovannetti, Seth Lloyd, and Lorenzo Maccone. Quantum random access memory. *Phys. Rev. Lett.*, 100:160501, Apr 2008.

[GN08] Nicolas Gama and Phong Q. Nguyen. Finding short lattice vectors within mordell's inequality. In *Proceedings of the 40th Annual ACM Symposium on Theory of Computing, Victoria, British Columbia, Canada, May 17-20, 2008*, pages 207–216, 2008.

[GNR10] Nicolas Gama, Phong Q. Nguyen, and Oded Regev. Lattice enumeration using extreme pruning. In Henri Gilbert, editor, *Advances in Cryptology – EUROCRYPT 2010*, pages 257–278, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[GOO] GOOGLE. Experimenting with post-quantum cryptography. Available at https://security.googleblog.com/2016/07/experimenting-with-post-quantum.html.

[Got09] Daniel Gottesman. An introduction to quantum error correction and fault-tolerant quantum computation, 2009.

[GPV08] Craig Gentry, Chris Peikert, and Vinod Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. In *Proceedings of the fortieth annual ACM symposium on Theory of computing*, pages 197–206. ACM, 2008.

[Gro96a] Lov K. Grover. A Fast Quantum Mechanical Algorithm for Database Search. In *Proceedings of the Twenty-Eighth Annual ACM Symposium on the Theory of Computing 1996*, pages 212–219. ACM, 1996.

[Gro96b] Lov K. Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the Twenty-Eighth Annual ACM Symposium on the Theory of Computing, Philadelphia, Pennsylvania, USA, May 22-24, 1996*, pages 212–219, 1996.

[Hel85] Bettina Helfrich. Algorithms to construct minkowski reduced and hermite reduced lattice bases. *Theor. Comput. Sci.*, 41(2–3):125–139, December 1985.

[HJ10] Nick Howgrave-Graham and Antoine Joux. New generic algorithms for hard knapsacks. In *EUROCRYPT*, volume 6110 of *LNCS*, pages 235–256. Springer, 2010.

[HK17] Gottfried Herold and Elena Kirshanova. Improved algorithms for the approximate k-list problem in euclidean norm. In Serge Fehr, editor, *Public-Key Cryptography – PKC 2017*, pages 16–40, Berlin, Heidelberg, 2017. Springer Berlin Heidelberg.

[HLGJ20] C. Hann, G. Lee, S. Girvin, and Liang Jiang. The resilience of quantum random access memory to generic noise. *arXiv: Quantum Physics*, 2020.

[HM18] Alexander Helm and Alexander May. Subset sum quantumly in 1.17^n. In *TQC*, volume 111 of *LIPIcs*, pages 5:1–5:15. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2018.

[Hoe63] Wassily Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58(301):13–30, 1963.

[HPS11] Guillaume Hanrot, Xavier Pujol, and Damien Stehlé. Analyzing blockwise lattice algorithms using dynamical systems. In Phillip Rogaway, editor, *Advances in Cryptology – CRYPTO 2011*, pages 447–464, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[HR07] Ishay Haviv and Oded Regev. Tensor-based hardness of the shortest vector problem to within almost polynomial factors. In *Proceedings of the thirty-ninth annual ACM symposium on Theory of computing*, pages 469–477, 2007.

[HS74] Ellis Horowitz and Sartaj Sahni. Computing partitions with applications to the knapsack problem. *J. ACM*, 21(2):277–292, 1974.

[HS07] Guillaume Hanrot and Damien Stehlé. Improved analysis of kannan's shortest lattice vector algorithm. In *Advances in Cryptology - CRYPTO 2007, 27th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2007, Proceedings*, pages 170–186, 2007.

[ILL89] Russell Impagliazzo, Leonid A Levin, and Michael Luby. Pseudo-random generation from one-way functions. In *Proceedings of the twenty-first annual ACM symposium on Theory of computing*, pages 12–24, 1989.

[Jr.83] Hendrik W. Lenstra Jr. Integer programming with a fixed number of variables. *Math. Oper. Res.*, 8(4):538–548, 1983.

[Kan83] R. Kannan. Improved algorithms for integer programming and related lattice problems. In *Proc. 15th ACM STOC*, pages 193–206, 1983.

[Kan87] Ravi Kannan. Minkowski's convex body theorem and integer programming. *Math. Oper. Res.*, 12(3):415–440, 1987.

[KF16] Paul Kirchner and Pierre-Alain Fouque. Time-memory trade-off for lattice enumeration in a ball. Cryptology ePrint Archive, Report 2016/222, 2016. https://eprint.iacr.org/2016/222.

[Kho05] Subhash Khot. Hardness of approximating the shortest vector problem in lattices. *J. ACM*, 52(5):789–808, 2005.

[KL78] Grigorii Anatol'evich Kabatiansky and Vladimir Iosifovich Levenshtein. On bounds for packings on a sphere and in space. *Problemy Peredachi Informatsii*, 14(1):3–25, 1978.

[Kle00] Philip Klein. Finding the closest lattice vector when it's unusually close. In *Proceedings of the Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '00, page 937–941, USA, 2000. Society for Industrial and Applied Mathematics.

[Kļe17] Vladislavs Kļevickis. Čaulu programmas ceļa atrašanai grafā (span programs for finding a path in a graph). Undergraduate 3rd year project, University of Latvia, 2017.

[KMPM19] Elena Kirshanova, Erik Mårtensson, Eamonn W. Postlethwaite, and Subhayan Roy Moulik. Quantum algorithms for the approximate k-list problem and their application to lattice sieving. In *ASIACRYPT*, volume 11921 of *LNCS*, pages 521–551. Springer, 2019.

[KP20] Iordanis Kerenidis and Anupam Prakash. Quantum gradient descent for linear systems and least squares. *Phys. Rev. A*, 101:022316, Feb 2020.

[KT17] Ghazal Kachigar and Jean-Pierre Tillich. Quantum information set decoding algorithms. In *PQCrypto*, volume 10346 of *LNCS*, pages 69–89. Springer, 2017.

[Kum21] Rajendra Kumar. *Exponential Time/Space Algorithms and Reductions for Lattice Problems*. Phd thesis, Indian Institute of Technology, Kanpur and National University of Singapore, 2021.

[Kup13] Greg Kuperberg. Another subexponential-time quantum algorithm for the dihedral hidden subgroup problem. In *TQC*, volume 22 of *LIPIcs*, pages 20–34. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2013.

[Laa15a] Thijs Laarhoven. *Search problems in cryptography*. PhD thesis, PhD thesis, Eindhoven University of Technology, 2015.

[Laa15b] Thijs Laarhoven. Sieving for shortest vectors in lattices using angular locality-sensitive hashing. In *Proc. CRYPTO 2015 - Part I*, volume 9215 of *LNCS*. Springer, 2015.

[LLL82] A. K. Lenstra, H. W. Lenstra, Jr., and L. Lovász. Factoring polynomials with rational coefficients. *Mathematische Ann.*, 261:513–534, 1982.

[LMPR08] Vadim Lyubashevsky, Daniele Micciancio, Chris Peikert, and Alon Rosen. Swifft: A modest proposal for fft hashing. In Kaisa Nyberg, editor, *Fast Software Encryption*, pages 54–72, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[LMvdP15a] Thijs Laarhoven, Michele Mosca, and Joop van de Pol. Finding shortest lattice vectors faster using quantum search. *Des. Codes Cryptography*, 77(2-3):375–400, 2015.

[LMvdP15b] Thijs Laarhoven, Michele Mosca, and Joop van de Pol. Finding shortest lattice vectors faster using quantum search. *Des. Codes Cryptogr.*, 77(2-3):375–400, 2015.

[LN13] Mingjie Liu and Phong Q. Nguyen. Solving BDD by enumeration: An update. In *Topics in Cryptology - Proc. CT-RSA 2013*, volume 7779 of *LNCS*. Springer, 2013.

[LN20] Jianwei Li and Phong Q. Nguyen. A complete analysis of the bkz lattice reduction algorithm. Cryptology ePrint Archive, Report 2020/1237, 2020. https://eprint.iacr.org/2020/1237.

[LO83] J. C. Lagarias and Andrew M. Odlyzko. Solving low-density subset sum problems. In *FOCS*, pages 1–10. IEEE Computer Society, 1983.

[LO85] J. C. Lagarias and Andrew M. Odlyzko. Solving low-density subset sum problems. *J. ACM*, 32(1):229–246, 1985.

[LP11] Richard Lindner and Chris Peikert. Better key sizes (and attacks) for lwe-based encryption. In *CT-RSA*, volume 6558 of *LNCS*, pages 319–339. Springer, 2011.

[LPS10] Vadim Lyubashevsky, Adriana Palacio, and Gil Segev. Public-key cryptographic primitives provably as secure as subset sum. In *TCC*, volume 5978 of *LNCS*, pages 382–400. Springer, 2010.

[LS90] Robert Y. Levin and Alan T. Sherman. A note on bennett's time-space tradeoff for reversible computation. *SIAM J. Comput.*, 19(4):673–677, 1990.

[Lyu05] Vadim Lyubashevsky. The parity problem in the presence of noise, decoding random linear codes, and the subset sum problem. In *APPROX-RANDOM*, volume 3624 of *LNCS*, pages 378–389. Springer, 2005.

[Lyu12] Vadim Lyubashevsky. Lattice signatures without trapdoors. In David Pointcheval and Thomas Johansson, editors, *Advances in Cryptology – EUROCRYPT 2012*, pages 738–755, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[MGM20] O. D. Matteo, V. Gheorghiu, and M. Mosca. Fault-tolerant resource estimation of quantum random-access memories. *IEEE Transactions on Quantum Engineering*, 1:1–13, 2020.

[Mic98] Daniele Micciancio. The shortest vector in a lattice is hard to approximate to within some constant. In *Proceedings of the 39th Annual Symposium on Foundations of Computer Science*, FOCS '98, page 92, USA, 1998. IEEE Computer Society.

[MMT11] Alexander May, Alexander Meurer, and Enrico Thomae. Decoding random linear codes in $\tilde{O}2^{0.054n}$. In *ASIACRYPT*, volume 7073 of *LNCS*, pages 107–124. Springer, 2011.

[MNRS11] F. Magniez, A. Nayak, J. Roland, and M. Santha. Search via quantum walk. *SIAM Journal on Computing*, 40(1):142–164, 2011.

[MO15] Alexander May and Ilya Ozerov. On computing nearest neighbors with applications to decoding of binary linear codes. In *EUROCRYPT (1)*, volume 9056 of *LNCS*, pages 203–228. Springer, 2015.

[Mon15] A Montanaro. Quantum walk speedup of backtracking algorithms. *ArXiv*, 2015.

[Mon20] Ashley Montanaro. Quantum speedup of branch-and-bound algorithms. *Phys. Rev. Research*, 2:013056, Jan 2020.

[MP13] Daniele Micciancio and Chris Peikert. Hardness of SIS and LWE with small parameters. In *Advances in Cryptology - CRYPTO 2013 - 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part I*, pages 21–39, 2013.

[MV10] Daniele Micciancio and Panagiotis Voulgaris. Faster exponential time algorithms for the shortest vector problem. In *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2010, Austin, Texas, USA, January 17-19, 2010*, pages 1468–1480, 2010.

[MW15] Daniele Micciancio and Michael Walter. Fast lattice point enumeration with minimal overhead. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, San Diego, CA, USA, January 4-6, 2015*, pages 276–294, 2015.

[NIS] NIST. Post-quantum cryptography standardization. Available at https://csrc.nist.gov/projects/post-quantum-cryptography/post-quantum-cryptography-standardization.

[NS60] Donald J. Newman and Lawrence Shepp. The double dixie cup problem. *The American Mathematical Monthly*, 67(1):58–61, 1960.

[NS02] Phong Q. Nguyen and Igor Shparlinski. The insecurity of the digital signature algorithm with partially known nonces. *J. Cryptology*, 15(3):151–176, 2002.

[NS20] María Naya-Plasencia and André Schrottenloher. Optimal merging in quantum k-xor and k-sum algorithms. In *EUROCRYPT 2020*, LNCS. Springer, May 2020.

[NSS01] Phong Q. Nguyen, Igor E. Shparlinski, and Jacques Stern. Distribution of modular sums and the security of the server aided exponentiation. In Kwok-Yan Lam, Igor Shparlinski, Huaxiong Wang, and Chaoping Xing, editors, *Cryptography and Computational Number Theory*, pages 331–342, Basel, 2001. Birkhäuser Basel.

[NV08] Phong Q. Nguyen and Thomas Vidick. Sieve algorithms for the shortest vector problem are practical. *J. Mathematical Cryptology*, 2(2):181–207, 2008.

[Oze16] Ilya Ozerov. *Combinatorial Algorithms for Subset Sum Problems*. PhD thesis, Ruhr Universität Bochum, 2016.

[Poh81] Michael Pohst. On the computation of lattice vectors of minimal length, successive minima and reduced bases with applications. *SIGSAM Bull.*, 15(1):37–44, 1981.

[Pra14] Anupam Prakash. *Quantum Algorithms for Linear Algebra and Machine Learning*. PhD thesis, EECS Department, University of California, Berkeley, Dec 2014.

[PS09] Xavier Pujol and Damien Stehlé. Solving the shortest lattice vector problem in time $2^{2.465n}$. *IACR Cryptology ePrint Archive*, 2009:605, 2009.

[Reg04] Oded Regev. Lattices in computer science, lecture 8, Fall 2004.

[Reg05] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In *Proceedings of the Thirty-Seventh Annual ACM Symposium on Theory of Computing*, STOC '05, page 84–93, New York, NY, USA, 2005. Association for Computing Machinery.

[Reg06] Oded Regev. Lattice-based cryptography. In *Advances in Cryptology - CRYPTO 2006, 26th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 2006, Proceedings*, pages 131–141, 2006.

[Sch72] J.P.M. Schalkwijk. An algorithm for source coding. *IEEE Transactions on Information Theory*, 18(3):395–399, May 1972.

[Sch87] Claus Peter Schnorr. A hierarchy of polynomial time lattice basis reduction algorithms. *Theor. Comput. Sci.*, 53:201–224, 1987.

[Sch03] Claus Peter Schnorr. Lattice reduction by random sampling and birthday methods. In *Proc. STACS 2003*, volume 2607 of *LNCS*, pages 145–156. Springer, 2003.

[SE94a] Claus-Peter Schnorr and M. Euchner. Lattice basis reduction: Improved practical algorithms and solving subset sum problems. *Math. Program.*, 66:181–199, 1994.

[SE94b] Claus-Peter Schnorr and M. Euchner. Lattice basis reduction: improved practical algorithms and solving subset sum problems. *Math. Programming*, 66:181–199, 1994.

[SH95] Claus-Peter Schnorr and H. H. Hörner. Attacking the Chor-Rivest cryptosystem by improved lattice reduction. In *Proc. of Eurocrypt '95*, volume 921 of *LNCS*, pages 1–12. IACR, Springer-Verlag, 1995.

[Sha84] Adi Shamir. A polynomial-time algorithm for breaking the basic merkle-hellman cryptosystem. *IEEE Trans. Information Theory*, 30(5):699–704, 1984.

[Sho97] Peter W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM J. Comput.*, 26(5):1484–1509, October 1997.

[SS81] Richard. Schroeppel and Adi. Shamir. A $T = O(2^{n/2})$, $S = O(2^{n/4})$ algorithm for certain np-complete problems. *SIAM Journal on Computing*, 10(3):456–464, 1981.

[SVP] SVP Challenges. https://www.latticechallenge.org/svp-challenge/.

[Vlă19] Serge Vlăduţ. Lattices with exponentially large kissing numbers. *Moscow Journal of Combinatorics and Number Theory*, 8(2):163–177, 2019.

[WF74] Robert A Wagner and Michael J Fischer. The string-to-string correction problem. *Journal of the ACM (JACM)*, 21(1):168–173, 1974.

[WLW] Wei Wei, Mingjie Liu, and Xiaoyun Wang. Finding shortest lattice vectors in the presence of gaps. In *Topics in Cryptology — CT-RSA 2015*.

[YD17] Yang Yu and Léo Ducas. Second order statistical behavior of LLL and BKZ. In *Proc. SAC 2017*, pages 3–22, 2017.

Notations and Acronyms

Notations (latin letters)

$\mathrm{bin}(\omega, \alpha)$	binomial, p. 97
$\mathbf{b}_i^\star$	Gram-Schmidt orthogonalization, p. 27
$\mathbf{B}^*$	basis dual, p. 27
$B_n(R)$	Euclidean ball, p. 17
$B^\star$	Gram-Schmidt orthogonalization, p. 27
$\mathrm{covol}(\mathcal{L})$	covolume of a lattice, p. 27
$\mathrm{d_{SD}}(X, Y)$	statistical distance, p. 26
$\mathrm{dist}(\mathbf{x}, S)$	distance, p. 17
$d(\mathcal{T})$	degree of a tree, p. 24
$D^n[\alpha, \beta]$	(sub)knapsack, p. 100
$D^n[\alpha, \beta, \gamma]$	(sub)knapsack, p. 100
$D_{\mathcal{L},s}$	discrete Gaussian distribution, p. 27
$f(x, y, z)$	3-way entropy, p. 100
$g(x, y)$	2-way entropy, p. 99
$h(x)$	hamming entropy of x, p. 95
$L_1 \bowtie_c L_2$	merged list, p. 96
L^f	filtered list, p. 96
$\mathcal{L}(\mathbf{b}_1, \ldots, \mathbf{b}_n)$	lattice span, p. 27
$\mathcal{L}^*$	lattice dual, p. 27
$L(\mathcal{T})$	set of leaves of a tree, p. 24
$\mathrm{negl}(n)$	negligible function, p. 95
$n(\mathcal{T})$	depth of a tree, p. 24
$o(f(n))$	"small o" notation, p. 17
$O(f(n))$	"big O" notation, p. 17
$\tilde{O}(f(n))$	"soft O" notation, p. 17
$\mathsf{pf}_1(\alpha, \beta)$	HGK filtering probability, p. 97
$\mathsf{pf}_2(\alpha, \beta, \gamma)$	BCJ filtering probability, p. 99
$\mathsf{pf}_3(\alpha_0, \beta, \gamma_0, \alpha_1, \gamma_1)$	new filtering probability, p. 101
$\mathrm{quadrin}(\omega, \alpha, \beta, \gamma)$	quadrinomial, p. 101
$\mathrm{trin}(\omega, \alpha, \beta)$	trinomial, p. 99
$\#\mathcal{T}$	number of nodes of the tree, p. 24
$U^\dagger$	conjugate transpose, p. 18
$\mathrm{vol}(\cdot)$	volume, p. 17

$V(\mathcal{T})$ set of nodes of a tree, p. 24

Notations (greek letters)

$\gamma(\mathcal{L})$ quantity related to the kissing number, p. 28
$\beta(\mathcal{L})$ quantity related to the kissing number, p. 28
$\eta_\varepsilon(\mathcal{L})$ smoothing parameter, p. 29
$\lambda_1(\mathcal{L})$ length of a shortest non-zero vector, p. 27
$\rho_s(\boldsymbol{x}), \rho_s(\mathcal{L})$ Gaussian mass function, p. 27
$\pi_i(\boldsymbol{v})$ orthogonal projection, p. 27
$\tau(\mathcal{L})$ kissing number, p. 28

Notations (blackboard letters)

$\mathbb{C}$ set of complex numbers, p. 17
$\mathbb{E}(X)$ expected value, p. 17
$\mathbb{E}\{C\}$ squared norm expected value, p. 17
$\mathbb{N}$ set of non-negative integers, p. 17
$\mathbb{R}$ set of real numbers, p. 17
$\mathbb{V}(X)$ variance, p. 17
$\mathbb{Z}$ set of integers, p. 17
$\mathbb{Z}_n$ ring of residue classes modulo n, p. 17

Acronyms

α-BDD Bounded Distance Decoding, p. 29
CVP Closest Vector Problem, p. 29
γ-approx-CVP γ-Approximate Closest Vector Problem, p. 29
δ-DGS_σ^m Discrete Gaussian Sampling problem, p. 28
QRACM classical memory with quantum random access, p. 21
QRAQM quantum memory with quantum random access, p. 21
SVP Shortest Vector Problem, p. 28
γ-approx-SVP γ-Approximate Shortest Vector Problem, p. 29