An Improved Quantum Algorithm for 3-Tuple Lattice Sieving

Lynn Engelberts* Yanlin Chen† Amin Shiraz Gilani‡ Maya-Iggy van Hoof§
Stacey Jeffery¶ Ronald de Wolf‖

July 7, 2026

Abstract

The assumed hardness of the Shortest Vector Problem (SVP) in high-dimensional lattices is one of the cornerstones of post-quantum cryptography. The fastest known heuristic attacks on SVP are via so-called sieving methods. While these still take exponential time in the dimension d, they are significantly faster than non-heuristic approaches and their heuristic assumptions are verified by extensive experiments. k-Tuple sieving is an iterative method where each iteration takes as input a large number of lattice vectors of a certain norm, and produces an equal number of lattice vectors of slightly smaller norm, by taking sums and differences of k of the input vectors. Iterating these "sieving steps" sufficiently many times produces a short lattice vector. The fastest attacks (both classical and quantum) are for $k = 2$, but taking larger k reduces the amount of memory required for the attack. In this paper we improve the quantum time complexity of 3-tuple sieving from $2^{0.3098d}$ to $2^{0.2846d}$, using a two-level amplitude amplification aided by a preprocessing step that associates the given lattice vectors with nearby "center points" to focus the search on the neighborhoods of these center points. Our algorithm uses $2^{0.1887d}$ classical bits and QCRAM bits, and $2^{o(d)}$ qubits. This is the fastest known quantum algorithm for SVP when total memory is limited to $2^{0.1887d}$.

*QuSoft and CWI, the Netherlands. Supported by the Dutch National Growth Fund (NGF), as part of the Quantum Delta NL program. lynn.engelberts@cwi.nl

†QuICS and University of Maryland, United States. Most of this work was completed while the author was at QuSoft and CWI. yanlin@umd.edu

‡QuICS and University of Maryland, United States. Partially supported by the DOE ASCR Quantum Testbed Pathfinder program (awards DE-SC0019040 and DE-SC0024220). asgilani@umd.edu

§Horst Görtz Institute for IT-Security, Ruhr University Bochum, Bochum, Germany. Partially supported by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy-EXC 2092 CASA-390781972 "Cyber Security in the Age of Large-Scale Adversaries". iggy.hoof@ruhr-uni-bochum.de

¶QuSoft, CWI and University of Amsterdam, the Netherlands. Partially supported by the European Union (ERC, ASC-Q, 101040624). jeffery@cwi.nl

‖QuSoft, CWI and University of Amsterdam, the Netherlands. Partially supported by the Dutch Research Council (NWO) through Gravitation-grant Quantum Software Consortium, 024.003.037. rdewolf@cwi.nl

1 Introduction

1.1 Classical and quantum sieving approaches to the Shortest Vector Problem

The Shortest Vector Problem (SVP) is the following: given linearly independent vectors $\mathbf{b}_1, \ldots, \mathbf{b}_d \in \mathbb{R}^d$, find a shortest nonzero vector in the lattice obtained by taking integer linear combinations of these vectors, that is, in

$$\Lambda = \left\{ \sum_{i=1}^{d} z_i \mathbf{b}_i \colon z_1, \ldots, z_d \in \mathbb{Z} \right\}.$$

The geometry and combinatorics of SVP are interesting mathematical problems in their own right, but SVP also underlies the most promising alternatives to the old favorites of integer factorization and discrete logarithm (which are both broken by quantum computers [Sho97]) as a basis for public-key cryptography; see [MR08] for a survey on lattice-based cryptography. Indeed, much of our future cryptography is premised on the assumption that there is no efficient algorithm for SVP, in fact not even for *approximate* SVP, which is the task of finding a nonzero vector of length at most some small factor γ (say some polynomial in d) larger than the shortest length.

The fastest provable classical algorithm for SVP has runtime essentially 2^d [ADRS15] on worst-case instances. This was improved to runtime roughly $2^{0.95d}$ on a quantum computer, and even to $2^{0.835d}$ on a quantum computer with a large QCRAM [ACKS25]. For meaningfully breaking cryptosystems, worst-case algorithms are more than needed and it suffices to have a fast *heuristic* algorithm, one that works for most (practical) instances under some plausible, though not quite rigorous, assumptions. After all, no one would use a cryptographic system that is known to be breakable with a significant probability. The fastest heuristic methods we know for solving approximate SVP still take exponential time 2^{cd}, but with a constant $c < 1$ that is much smaller than for the best known worst-case algorithms. These heuristic methods are based on sieving ideas, which were first introduced in the context of SVP by Ajtai, Kumar, and Sivakumar [AKS01] and made more practical by Nguyen and Vidick [NV08]. (In fact, the best non-heuristic, worst-case algorithms such as [ADRS15] can also be viewed as "sieving" algorithms.)

The most basic type of sieving is *2-tuple* sieving. The idea here is to begin by sampling a large list L of m random vectors from the lattice Λ, all of roughly the same norm R, with R chosen large enough to allow efficient sampling (in fact we may assume $R = 1$ by scaling the lattice at the start). We then try to find slightly shorter vectors that are sums or differences of pairs of vectors $\mathbf{x}, \mathbf{y} \in L$ (note that $\mathbf{x} + \mathbf{y}$ and $\mathbf{x} - \mathbf{y}$ are still in Λ because a lattice is closed under taking integer linear combinations). The use of heuristics in the analysis of this approach can be explained as follows. Given two uniformly random $\mathbf{x}, \mathbf{y} \in \mathbb{R}^d$ of the same norm R, the probability that $\mathbf{x}+\mathbf{y}$ or $\mathbf{x}-\mathbf{y}$ has norm $< R$ can be seen to be $p \approx 2^{-0.2075d}$ from basic estimates of the area of a cap on a d-dimensional sphere. With m initial vectors, we have $\binom{m}{2}$ 2-tuples of vectors, so if these vectors behave like i.i.d. uniformly random unit vectors then the expected number of 2-tuples inducing a shorter vector is $p\binom{m}{2}$. Hence, choosing $m \approx 2/p \approx 2^{0.2075d}$ ensures there will be roughly m 2-tuples that each give a shorter vector. If we can efficiently *find* those m good 2-tuples, then we have generated a new list L' of m shorter lattice vectors. The heuristic assumption is that, once normalized, the vectors in L' again behave like i.i.d. uniformly random unit vectors (this is a common assumption in lattice-sieving algorithms, and has been extensively verified numerically [NV08, BLS16, ADH+19]). This list L' will form the starting point of the next sieving iteration. Under the heuristic assumption, we can then form a new list L'' of roughly m lattice vectors of even shorter norm by taking sums or

differences of 2-tuples of vectors from L', and so on.

Usually a relatively small (polynomial in d) number of such iterations suffices to find quite short vectors (see Section 4.2), so the cost of the overall procedure for solving approximate SVP will be dominated by the cost of one sieving iteration, which is the cost of finding m good 2-tuples among the $\binom{m}{2}$ 2-tuples. How much time does this take? Nguyen and Vidick [NV08] used brute-force search over all $\binom{m}{2}$ 2-tuples, which takes time $O(m^2) = O(2^{0.416d})$. This time has since been improved using nearest-neighbor data structures that allow us to quickly find, for a given $\mathbf{x} \in L$, a small number of $\mathbf{y} \in L$ that are somewhat close to $\mathbf{x}$, and in particular using locality-sensitive filtering techniques that allow us to focus the search for a close $\mathbf{y}$ to the neighborhoods of shared "center points". These algorithms have been further improved by using various *quantum* subroutines to speed up the search for good 2-tuples, in particular Grover's search algorithm [LMP15, Laa15], quantum walks [CL21], and reusable quantum walks [BCSS23]. Currently, the best classical and quantum runtimes are $2^{0.2925d+o(d)}$ [BDGL16] and $2^{0.2563d+o(d)}$ [BCSS23], respectively, which are the fastest known heuristic attacks on SVP. These attacks, however, do require at least $2^{0.2075d}$ bits of memory.

The 2-tuple-sieving approach can be generalized to k-tuple sieving for some larger constant $k \geq 3$ in the natural way [BLS16]: by looking for k-tuples $\mathbf{x}_1, \ldots, \mathbf{x}_k$ from our current list L such that $\|\mathbf{x}_1 \pm \mathbf{x}_2 \pm \cdots \pm \mathbf{x}_k\| \leq 1$ for some choice of the coefficients ± 1 (note that the number of sign patterns for the coefficients is just 2^{k-1}, which disappears in the big-O because k is a constant). The advantage of going to $k > 2$ is that the initial list size m can be smaller while still giving roughly m good k-tuples (and hence roughly m shorter lattice vectors for the next sieving iteration), meaning the algorithm requires less memory. The disadvantage, on the other hand, is that the time to find m good k-tuples increases with k, because the set of k-tuples has $\binom{m}{k}$ elements, which grows with k, even when we take into account that the minimal required list size m itself goes down with k. For example, for 2-tuple sieving and 3-tuple sieving, the required list sizes are $m_2 \approx 2^{0.2075d}$ and $m_3 \approx 2^{0.1887d}$, respectively, yet $\binom{m_3}{3} \gg \binom{m_2}{2}$ despite the fact that $m_3 \ll m_2$.

As in 2-tuple sieving, a relatively small (polynomial in d) number of iterations suffices for finding short lattice vectors. Hence, the overall cost of k-tuple sieving will be dominated by the cost of finding m good k-tuples among the set of all $\binom{m}{k}$ k-tuples from the given list of m vectors. In Table 1, we give the best known classical and quantum upper bounds on the time complexity of k-tuple sieving for $k = 2, 3, 4$, as established in previous work. These algorithms use time and memory that is exponential in d, so the table just gives the constant in the exponent of the time complexity, suppressing $o(1)$ terms.

k	Memory	Classical time	Quantum time
2	0.2075	0.2925 [BDGL16]	0.2563 [BCSS23]
3	0.1887	0.3383 [CL23]	0.3098 [CL23], **0.2846 (this work)**
4	0.1724	0.3766 [HKL18]	0.3178 [KMPM19, App. B.2]

Table 1: Exponents of the best known classical and quantum time complexities for k-tuple sieving with minimal memory usage, for small k. Our main result improves the quantum time exponent for $k = 3$ to 0.2846, while keeping the memory exponent at 0.1887.

1.2 Our results

Our main result is an improvement of the quantum time complexity of 3-tuple sieving, from $2^{0.3098d+o(d)}$ to $2^{0.2846d+o(d)}$. Interestingly, this means quantum 3-tuple sieving is now faster than classical 2-tuple sieving which is still the best classical heuristic algorithm for SVP (and of course 3-tuple sieving uses less memory than 2-tuple sieving, which is the main advantage of considering $k > 2$). Our improvement is relatively small and will not scare cryptographers who base their constructions on the assumed hardness of SVP. Often when deciding on an appropriately large key size to guarantee a certain level of security for their cryptosystem, they already allow for a quadratic quantum speedup over the best known classical attack (the best speedup one can hope for just using Grover's algorithm or amplitude amplification without exploiting further specific structure of the problem), which is already better than all quantum speedups that we actually know how to achieve. In addition, our results include factors of the form $2^{o(d)}$ in the runtime, which are insignificant for asymptotic analysis but may be quite large for the dimensions used in practice. However, we remark that our improvement is significantly larger than recent progress on this front. For comparison, the most recent improvement in the exponent for quantum 2-tuple sieving was from 0.2570 [CL21] to 0.2563 [BCSS23].

The core computational problem that we would like to solve, and which dominates the cost of 3-tuple sieving, is the following:

Problem 1 (Finding many solutions)**.** Given a list L of m i.i.d. uniform samples from the unit sphere in $\mathbb{R}^d$, find m 3-tuples of distinct $\mathbf{x}, \mathbf{y}, \mathbf{z} \in L$ such that $\|\mathbf{x}-\mathbf{y}-\mathbf{z}\| \leq 1$. We will refer to such a tuple $(\mathbf{x}, \mathbf{y}, \mathbf{z})$ as a *3-tuple solution*.

As argued before, we could also allow all sign patterns $\|\mathbf{x}\pm\mathbf{y}\pm\mathbf{z}\|$, but asymptotically this does not matter because there are only 4 such patterns. We assumed here for simplicity that the m initial vectors all have norm exactly 1.[1] To ensure that m such tuples exist with high probability over the choice of L, the list size m has to be at least roughly $(27/16)^{d/4+o(d)} \approx 2^{0.1887d}$ [HK17], providing a memory lower bound for 3-tuple sieving.

Expanding $\|\mathbf{x}-\mathbf{y}-\mathbf{z}\|^2$ shows that unit vectors $\mathbf{x}, \mathbf{y}, \mathbf{z}$ satisfy $\|\mathbf{x}-\mathbf{y}-\mathbf{z}\| \leq 1$ if and only if $\langle \mathbf{x}, \mathbf{y} \rangle + \langle \mathbf{x}-\mathbf{y}, \mathbf{z} \rangle \geq 1$. This fact lets us replace the condition $\|\mathbf{x}-\mathbf{y}-\mathbf{z}\| \leq 1$ by two conditions on the inner products $\langle \mathbf{x}, \mathbf{y} \rangle$ and $\langle \mathbf{x}-\mathbf{y}, \mathbf{z} \rangle$, motivating us to search for 3-tuple solutions $(\mathbf{x}, \mathbf{y}, \mathbf{z})$ by first searching for a pair $(\mathbf{x}, \mathbf{y})$ with sufficiently large inner product. Our main result is indeed a faster quantum algorithm for a mildly relaxed version of Problem 1. Slightly simplified (by omitting approximations in the inner products), this relaxed problem is:

Problem 2 (Finding many solutions with a stronger property)**.** Given a list L of m i.i.d. uniform samples from the unit sphere in $\mathbb{R}^d$, find m 3-tuples of distinct $\mathbf{x}, \mathbf{y}, \mathbf{z} \in L$ such that $\langle \mathbf{x}, \mathbf{y} \rangle \geq 1/3$ and $\langle \mathbf{x}-\mathbf{y}, \mathbf{z} \rangle \geq 2/3$ (implying $\|\mathbf{x}-\mathbf{y}-\mathbf{z}\| \leq 1$).

The latter problem demands something stronger than $\|\mathbf{x}-\mathbf{y}-\mathbf{z}\| \leq 1$. This slightly increases the required list size m that ensures m triples with the stronger property exist, but only by a factor $2^{o(d)}$

[1] It may seem odd to start with vectors $\mathbf{x}, \mathbf{y}, \mathbf{z}$ of norm 1 and then to find new vectors $\mathbf{x}-\mathbf{y}-\mathbf{z}$ whose norm is still (at most) 1. However, the way this is actually used in practice is to aim at finding vectors with norm $\leq 1-\mu$ for some μ that is inverse-polynomially small in d: big enough to ensure that a polynomially-small number of sieving iterations results in quite short vectors, and small enough to only affect the runtime up to subexponential factors (i.e., $o(1)$ in the exponent). Aiming at norm ≤ 1 rather than $\leq 1-\mu$ is just to simplify notation.

as a result of the chosen inner-product conditions (e.g., see [HK17] or Lemma 3.15 below). This factor is negligible for our asymptotic purposes, and allows us to solve Problem 1 with list size $m \approx 2^{0.1887d}$ using an algorithm for Problem 2.

We will find those m triples in Problem 2 one by one. To find one good triple, consider the following approach:

1. Create a uniform superposition over all $\mathbf{x} \in L$, and conditioned on $\mathbf{x}$ use amplitude amplification to create (in a second register) a superposition over all $\mathbf{y} \in L$ such that $\langle \mathbf{x}, \mathbf{y} \rangle \geq 1/3$.

2. Starting from the state of the previous step, conditioned on $\mathbf{x}, \mathbf{y}$ use amplitude amplification to create (in a third register) a uniform superposition over all $\mathbf{z} \in L$ such that $\langle \mathbf{x}-\mathbf{y}, \mathbf{z} \rangle \geq 2/3$, and set a "flag" qubit to 1 if such a $\mathbf{z}$ exists.

Then use amplitude amplification on top of this two-step procedure to amplify the part of the state where the flag is 1. Measuring the resulting state gives us one of the triples $(\mathbf{x}, \mathbf{y}, \mathbf{z})$ that we want. Repeating $\tilde{O}(m)$ times, we will obtain (except with negligibly small error probability) m triples satisfying the stronger property, and hence m new vectors with norm at most 1. This approach is already better than a basic amplitude amplification on all m^3 triples $(\mathbf{x}, \mathbf{y}, \mathbf{z})$, due to the somewhat surprising properties of "two-oracle search" [KLL15]. However, when executed as stated, this procedure can be shown to yield an overall runtime of roughly $2^{0.3350d}$ for $m \approx 2^{0.1887d}$, and therefore does not yet outperform the state-of-the-art quantum runtime $2^{0.3098d}$ from [CL23] (see Section 3.1). The costs of steps 1 and 2 can also be seen to be unbalanced, suggesting there may be room for improvement.

To get a faster algorithm, we improve the search for vectors satisfying the stronger property by locality-sensitive filtering using random product codes (RPCs, following [BDGL16]). Roughly speaking, the idea here is to choose a certain number of sufficiently random center points, and do a preprocessing step that, for each of the m vectors in L, writes down their closest center points. The properties of RPCs allow us to do this efficiently. Then for a given $\mathbf{x}$ in step 1, we can improve the search for a $\mathbf{y} \in L$ satisfying $\langle \mathbf{x}, \mathbf{y} \rangle \geq 1/3$ by restricting the search to those $\mathbf{y} \in L$ that share a close center point with $\mathbf{x}$. Specifically, using the data structure prepared during preprocessing, we can quickly create, for a given $\mathbf{x}$, a superposition over all center points $\mathbf{c}$ that are close to $\mathbf{x}$, and subsequently, for each such $\mathbf{c}$, a superposition over all $\mathbf{y} \in L$ that are close to $\mathbf{c}$. Since those $\mathbf{y}$ are relatively close to $\mathbf{x}$, they are more likely to satisfy $\langle \mathbf{x}, \mathbf{y} \rangle \geq 1/3$. Putting amplitude amplification on top of this then allows us to quickly prepare a superposition over pairs $(\mathbf{x}, \mathbf{y})$ with $\langle \mathbf{x}, \mathbf{y} \rangle \geq 1/3$ that share a close center point, improving over the cost of the original step 1.

Similarly, for step 2, it is easier to find a $\mathbf{z} \in L$ close to $\mathbf{x} - \mathbf{y}$ if we focus only on those $\mathbf{z} \in L$ that share a close center point with $\mathbf{x} - \mathbf{y}$. This approach may overlook some close $(\mathbf{x}, \mathbf{y})$ pairs or some $\mathbf{z} \in L$ close to $\mathbf{x} - \mathbf{y}$ (if $\mathbf{x}$ and $\mathbf{y}$, respectively $\mathbf{x} - \mathbf{y}$ and $\mathbf{z}$, do not share a close center point). However, we can show that a careful application of these ideas enables us to balance the costs of steps 1 and 2, and that this modified method (repeated $\tilde{O}(m)$ times and choosing a fresh RPC once in a while to ensure no good triple is overlooked all the time) will find m good triples. This results in a quantum algorithm that solves Problem 1 with list size $m \approx 2^{0.1887d}$ in time $2^{0.2846d}$, giving a speedup over the previous best known time complexity for this choice of m.

1.3 Discussion and future work

Various ingredients of our algorithm, including the use of RPCs and of course amplitude amplification, have been used before in quantum k-tuple sieving algorithms. What distinguishes our

algorithm, and enables our speedup, is first our nested use of amplitude amplification (to do two-oracle search in the vein of [KLL15]), and second the way we leverage the preprocessing that we do in advance, where we write down for each lattice vector in L its set of close center points from the RPC. Since we can afford relatively expensive preprocessing in terms of both time and memory, this allows us to balance different terms in the time complexity. Altogether, this yields a more sophisticated nested amplitude amplification that results in our speedup.

One of the main messages of our paper is that the toolbox of techniques that have been used for sieving is not exhausted yet, and can still give improved results when combined with a well-chosen preprocessing step. How much further can this be pushed? Surprisingly, we only used basic amplitude amplification here, not the more sophisticated quantum-walk approaches that are good at finding collisions (and which have been used for sieving [CL21, BCSS23]). In fact, the work that led to our algorithm for Problem 2 started from a quantum-walk approach, but optimizing the parameters suggested that the current version (which can be viewed as a quantum walk with vertex size 1) is optimal. This might be due to the fact that there are *many* 3-tuple solutions that need to be found — a setting in which quantum walks may be less helpful (consider that for *element distinctness*, the optimal algorithm uses quantum walks [Amb04], whereas for its many-solution variant, *collision finding*, there is an optimal quantum algorithm that uses only amplitude amplification [BHT98]). Are there other ways to improve the algorithm further by using quantum walks? Also, can we improve the best quantum algorithm for 2-tuple sieving? This would give the fastest known heuristic attack on SVP, rather than just a faster attack with moderate memory size. Finally, do our techniques lead to improvements for 4-tuple sieving?

1.4 Organization

The remainder of this paper is organized as follows. Section 2 provides preliminaries on the computational model, data structure, and quantum-algorithmic techniques used in this work, as well as on properties of RPCs and the unit sphere relevant to our analysis. In Section 3, we present our new quantum algorithm, and the details of its application to SVP are given in Section 4.

2 Preliminaries

2.1 Notation

Throughout the paper, d will always be the dimension of the ambient space $\mathbb{R}^d$, log without a base means the binary logarithm, $\ln = \log_e$ the natural logarithm, and $\exp(f) = e^f$. We write p_θ as shorthand for $(1-\cos^2(\theta))^{d/2}$ (as motivated by Section 2.4). The set of rotation matrices over $\mathbb{R}^d$ is denoted by $\mathrm{SO}(d)$.

We write $x \sim D$ to denote that x is a sample from the distribution D. For $M \in \mathbb{Z}^+$ and a set $S \subseteq \mathbb{R}^d$, we let $\mathcal{U}(S, M)$ denote the distribution that samples $C \subseteq S$ by selecting M elements from S independently and uniformly with replacement. Throughout this work, we refer to multisets simply as sets, so the cardinality (or "size") $|C|$ corresponds to the total number of sampled elements, counting repetitions. When an argument requires using independence explicitly, we view C as an ordered sequence of random variables. We write $x \sim \mathcal{U}(S)$ as shorthand for $\{x\} \sim \mathcal{U}(S, 1)$.

For $a, b \in \mathbb{R}$ and $\epsilon > 0$, we write $a \approx_\epsilon b$ if $|a-b| \leq \epsilon$. In particular, there is some dependency on ϵ in our analysis, so we will fix $\epsilon := 1/(\log d)^2$ to absorb this dependence into the $2^{o(d)}$ notation. We

also write $a =_d b$ as shorthand for $b2^{-o(d)} \leq a \leq b2^{o(d)}$, i.e., a and b are equal up to subexponential factors in d. We let $a \geq_d b$ denote $a \geq b2^{-o(d)}$, and $a \leq_d b$ denote $a \leq b2^{o(d)}$.

Whenever we write $|0\rangle$, we mean the all-zero computational basis state over the number of qubits implied by the context, not necessarily a single-qubit state. When considering unitaries U on quantum states in some Hilbert space H, we will often only be interested in its application on a subset $X \subseteq H$ of states, and we will sometimes (with slight abuse of notation) write "$U\colon |\psi\rangle \mapsto |\perp\rangle$ if $|\psi\rangle \in H \setminus X$", where $|\perp\rangle$ is a substitute for a quantum state that is irrelevant for our analysis.

2.2 Tail bounds

Lemma 2.1 (Chernoff bound [Che52], [MU05, Section 4.2.1])**.** *Let $X = \sum_{i=1}^{m} X_i$ be a sum of independent random variables $X_i \in \{0,1\}$ and define $\mu \coloneqq \mathbb{E}[X]$. Then*

(i) $\Pr[X \geq (1+\delta)\mu] \leq \left(\frac{e^\delta}{(1+\delta)^{1+\delta}}\right)^\mu \leq e^{-\frac{\delta^2}{2+\delta}\mu}$ *for all* $\delta \geq 0$.[2]

(ii) $\Pr[X \leq (1-\delta)\mu] \leq e^{-\frac{\delta^2}{2}\mu}$ *for all* $\delta \in (0,1)$.

(iii) $\Pr[|X-\mu| \geq \delta\mu] \leq 2e^{-\frac{\delta^2}{3}\mu}$ *for all* $\delta \in (0,1)$.

We will use the following immediate corollary.

Corollary 2.2 (Simple application of the Chernoff bound)**.** *Let $m\colon \mathbb{N} \to \mathbb{N}$. For $d \in \mathbb{N}$, let $X(d) = \sum_{i=1}^{m(d)} X_i(d)$ be a sum of independent random variables $X_i(d) \in \{0,1\}$. Then $X(d) \leq_d \max\{1, \mathbb{E}[X(d)]\}$, except with probability $e^{-\omega(d)}$. Moreover, if $\mathbb{E}[X(d)] = \omega(d)$, then $X(d) =_d \mathbb{E}[X(d)]$, except with probability $e^{-\omega(d)}$.*

Proof. Let $\mu \coloneqq \mathbb{E}[X]$, where $X = X(d)$. Applying part (i) of Lemma 2.1 with $\delta = d^2 \max(1, 1/\mu)$ yields $\Pr[X \geq (1+\delta)\mu] \leq e^{-\delta\mu/2} \leq e^{-d^2/2} = e^{-\omega(d)}$, where the last inequality uses that $\delta \geq 2$ for $d \geq 2$. Note that $(1+\delta)\mu \leq_d \max\{1, \mu\}$ by definition of δ. (This can be seen by separating the case $\mu \leq 1$ and $\mu > 1$.) To prove the second part, assume that $\mu = \omega(d)$. Applying part (iii) of Lemma 2.1 with (say) arbitrary constant $\delta \in (0,1)$ yields $\Pr[|X-\mu| \geq \delta\mu] \leq 2e^{-\delta^2\mu/3} = e^{-\omega(d)}$ by assumption on μ, from which the claim follows. □

2.3 Computational model and quantum preliminaries

2.3.1 Computational model

Our computational model is a classical computer (a classical random-access machine) that can invoke a quantum computer as a subroutine. This classical computer can also write bits to a quantum-readable classical-writable classical memory (QCRAM). This memory stores an n-bit string $w = w_0 \dots w_{n-1}$, and supports *quantum random access queries* or *QCRAM queries*, which correspond to calls to the unitary $O_w\colon |i,b\rangle \mapsto |i, b \oplus w_i\rangle$ for $i \in \{0,\dots,n-1\}$ and $b \in \{0,1\}$. Note that QCRAM itself (as a memory) remains classical throughout the algorithm: it stores a classical string rather than a quantum superposition over strings. The notion of QCRAM is commonly used in quantum algorithm design for efficient quantum queries to classical data, allowing us to query (read) multiple

[2] The second inequality follows because $\delta - (1+\delta)\ln(1+\delta) + \delta^2/(2+\delta) < 0$ for all $\delta > 0$; for $\delta = 0$ it is trivial.

bits of the classical data in superposition. In contrast, write operations (i.e., changes to the stored string w) can only be done classically.

In classical algorithms, allowing random access memory queries with unit or logarithmic cost is standard practice. The intuition is that the n bits of memory can be, for example, arranged on the leaves of a binary tree with depth $\lceil \log_2 n \rceil$, and querying the i-th bit corresponds to traversing a $\lceil \log_2 n \rceil$-length path from the root to the i-th leaf of this binary tree. For similar reasons, QCRAM queries are often assumed "cheap" to execute (i.e., in time $O(\log n)$) once the classical memory is stored in QCRAM.

While the physical implementation of such a device is nontrivial and controversial due to the challenge and expense of doing quantum error-correction on the whole device (whose size will have to be proportional to the number of stored bits rather than to its logarithm, though its depth will still be logarithmic), QCRAM remains conceptually acceptable for theoretical purposes. This is particularly true in theoretical computer science contexts, where classical RAM is likewise assumed to be fast and error-free without a concern for error correction. One may hope that in the future, quantum hardware will be able to implement such memory with comparable reliability and efficiency. In cryptography it is also important to learn the runtime of the best quantum algorithms for breaking cryptography under fairly optimistic assumptions on the hardware, which is exactly what we do in this paper.

In our computational model, we will count one RAM operation or one QCRAM write operation in the classical machine, or one elementary gate in a quantum circuit, as one "step". When we refer to the "time" or "time complexity" of an algorithm or subroutine, we mean the total number of steps it takes on a worst-case input. We will typically count the number of QCRAM queries separately.

2.3.2 Amplitude amplification

Our main quantum algorithmic tool will be amplitude amplification [BHMT02]. The goal of amplitude amplification is to project an easily generated state $|\psi\rangle$ onto a "marked" subspace. *Fixed-point* amplitude amplification [YLC14, GSLW19] ensures that, with high probability, the output state is the desired state (assuming it is nonzero), even if we run for more steps than necessary. The version that we give below ensures that the algorithm always stops after a fixed number of steps, even if $|\psi\rangle$ does not overlap the marked space at all.

Lemma 2.3 (Fixed-point amplitude amplification (implicit in [YLC14, GSLW19]))**.** *Let* $\mathtt{Samp}$ *be a unitary implementable in time* S*, and let* $|\psi\rangle := \mathtt{Samp}\,|0\rangle$*. Let* Π *be a projector on the Hilbert space* H *of* $|\psi\rangle$*, and let* $\mathtt{Check}$ *be a unitary implementable in time* C *such that, for all* $|\phi\rangle \in H$*,* $\mathtt{Check}\,|\phi\rangle\,|0\rangle = \Pi\,|\phi\rangle\,|1\rangle + (I-\Pi)\,|\phi\rangle\,|0\rangle$*. For all* $r \geq 1$ *and* $\delta \in (0, 1/2]$*, there exists a quantum algorithm* $\mathtt{AA}_r(\mathtt{Samp}, \mathtt{Check})$ *that satisfies the following, except with probability* δ*:*

(i) If $\|\Pi\,|\psi\rangle\| \geq \frac{1}{r}$*, then it outputs the state* $|\psi_{\text{out}}\rangle = \frac{\Pi|\psi\rangle}{\|\Pi|\psi\rangle\|}\,|1\rangle$ *in time* $O(\log(\frac{1}{\delta})\|\Pi\,|\psi\rangle\|^{-1}(\mathsf{S}+\mathsf{C}))$*.*

(ii) If $\|\Pi\,|\psi\rangle\| = 0$*, then it outputs the state* $|\psi_{\text{out}}\rangle = |\psi\rangle\,|0\rangle$ *in time* $O(\log(\frac{1}{\delta})r(\mathsf{S}+\mathsf{C}))$*.*

We call the auxiliary qubit in the output of amplitude amplification its *flag* qubit or *flag* register.

Proof. By [YLC14, GSLW19], there exists a universal constant $\eta > 0$ and a quantum algorithm $\mathcal{A}_{r'} = \mathcal{A}_{r'}(\mathtt{Samp}, \mathtt{Check})$ that generates a state $|\psi_{\text{out}}\rangle$ in time $O(r'(\mathsf{S}+\mathsf{C}))$, satisfying:

(1) If $\|\Pi|\psi\rangle\| \neq 0$ and $r' \geq \eta \log(\frac{1}{\delta})\|\Pi|\psi\rangle\|^{-1}$, then $\||\psi_{\text{out}}\rangle - \frac{\Pi|\psi\rangle}{\|\Pi|\psi\rangle\|}|1\rangle\| \leq \delta$.

(2) If $\|\Pi|\psi\rangle\| = 0$ and $r' \geq 1$, then $|\psi_{\text{out}}\rangle = |\psi\rangle|0\rangle$.

Our lemma extends this result slightly: in part (i), we require the runtime to be much smaller than in part (ii) whenever the actual (possibly unknown) value of $\|\Pi|\psi\rangle\|$ is significantly larger than the lower bound $\frac{1}{r}$ that our algorithm has for it.

Specifically, we define the quantum algorithm $\mathtt{AA}_r(\mathtt{Samp}, \mathtt{Check})$ as follows. It first runs $\mathcal{A}_{r'}$ with $r' = 1$ and measures the auxiliary qubit. If the measurement outcome is 1, then it stops and outputs the resulting state. Otherwise, it doubles r' (i.e., replaces r' by $2r'$) and repeats. This continues until $r' \geq 2\eta \log(\frac{2}{\delta})r$, at which point it outputs the current state.

By construction, $\mathtt{AA}_r(\mathtt{Samp}, \mathtt{Check})$ always terminates, and outputs a state $|\psi_{\text{out}}\rangle$. Note that if $\mathtt{AA}_r(\mathtt{Samp}, \mathtt{Check})$ terminates after ℓ repetitions, then the aforementioned result implies that the total runtime is $O(2^\ell(\mathsf{S} + \mathsf{C}))$, because $\sum_{i=1}^{\ell} 2^i = O(2^\ell)$.

Suppose $\|\Pi|\psi\rangle\| \geq \frac{1}{r}$. Let ℓ be the smallest integer such that $2^\ell \geq \eta \log(\frac{2}{\delta})\|\Pi|\psi\rangle\|^{-1}$. By statement (1), the inner product α between $\frac{\Pi|\psi\rangle}{\|\Pi|\psi\rangle\|}|1\rangle$ and the output state $|\psi_{\text{out}}\rangle$ of $\mathcal{A}_{2^\ell}$ satisfies $|\alpha| \geq 1 - \delta/2$, so the probability that measuring the ancilla qubit yields outcome 1 is at least $(1 - \delta/2)^2 \geq 1 - \delta$. In other words, with probability at least $1 - \delta$, $\mathtt{AA}_r(\mathtt{Samp}, \mathtt{Check})$ generates the state $|\psi_{\text{out}}\rangle = \frac{\Pi|\psi\rangle}{\|\Pi|\psi\rangle\|}|1\rangle$ after at most ℓ repetitions, from which part (i) follows.

On the other hand, if $\|\Pi|\psi\rangle\| = 0$, statement (2) implies that $\mathtt{AA}_r(\mathtt{Samp}, \mathtt{Check})$ always generates $|\psi_{\text{out}}\rangle = |\psi\rangle|0\rangle$. The complexity claim follows because, by construction of the algorithm, it terminates after ℓ repetitions, for some ℓ satisfying $2^\ell = O(\log(\frac{1}{\delta})r)$. □

Remark 1 (Choice of δ). When applying Lemma 2.3, we will always consider a superexponentially small δ, say $\delta = \exp(-2^{\sqrt{d}})$, and will not state its value explicitly. This implicit choice ensures that the error probability of $\mathtt{AA}_r(\mathtt{Samp}, \mathtt{Check})$ is $2^{-\omega(d)}$, and incurs a cost of only a factor $\log(1/\delta) \leq 2^{o(d)}$ in the overall complexity of $\mathtt{AA}_r(\mathtt{Samp}, \mathtt{Check})$, a cost that is negligible for our purposes.

Remark 2 (Unstructured search as a special case). Amplitude amplification allows us to search in a finite set M_0 for elements of a (possibly empty) subset M_1, and create a uniform superposition $|M_1\rangle$ over these elements if they exist. Namely, let $\mathtt{Samp}$ be a quantum algorithm that maps $|0\rangle$ to a uniform superposition $|\psi\rangle$ over the elements of M_0, and $\mathtt{Check}$ a quantum algorithm that maps $|x\rangle|0\rangle \mapsto |x\rangle|1\rangle$ if $x \in M_1$ and that acts as identity for $x \in M_0 \setminus M_1$. Taking $r \coloneqq |M_0|$ (assuming $|M_0|$ is known or encoded in $\mathtt{Samp}$), Lemma 2.3 implies that, with probability at least $1-\delta$, $\mathtt{AA}_r(\mathtt{Samp}, \mathtt{Check})$ generates a state $|\psi_{\text{out}}\rangle$ in time $O(\sqrt{|M_0|}\log\frac{1}{\delta})$, which satisfies $|\psi_{\text{out}}\rangle = |M_1\rangle|1\rangle$ if $M_1 \neq \emptyset$, and $|\psi_{\text{out}}\rangle = |\psi\rangle|0\rangle$ otherwise.

2.3.3 Two-oracle search

One of the most important quantum algorithmic primitives is Grover's algorithm [Gro96], which solves the problem of *oracle search*. Consider a set M_0 of elements that are easy to "sample" — meaning that we can generate some superposition over the elements of M_0, where we think of the squared amplitudes as the sampling probabilities. For a marked subset $M_1 \subseteq M_0$, consider the problem of searching over M_0 for an element of M_1. If we can sample an element of M_0 in complexity S and we can check membership in M_1 in complexity C, then there is a quantum algorithm [BHMT02] (see Lemma 2.3 for a slightly different "fixed-point" version) that finds an

element of M_1 in complexity (neglecting constants)

$$\frac{1}{\sqrt{\varepsilon}}\,(\mathsf{S}+\mathsf{C})$$

where ε is the probability that an element sampled from M_0 is in M_1. In our applications, the sampling complexity S is negligible, resulting in complexity $\frac{1}{\sqrt{\varepsilon}}\mathsf{C}$.

Oracle search is often called *unstructured search*, since the oracle abstracts away any potential structure of the problem that an algorithm might be able to take advantage of. Although this makes it quite general, we can in some cases take advantage of a small amount of structure. A simple case is *two-oracle search*, first studied in [KLL15] for the case of a unique marked element. In this problem, one wants to find an element of $M_2 \subseteq M_0$, but in addition to having access to an oracle for checking membership in M_2 (at cost C_2), one also has access to an oracle for checking membership in a set M_1 such that $M_2 \subseteq M_1 \subseteq M_0$ (at cost C_1). See Figure 1 for a depiction. Assuming $\mathsf{C}_1 \ll \mathsf{C}_2$, this additional structure gives an advantage, intuitively because one can always first cheaply check membership in M_1, and for nonmembers, not waste time on a more expensive membership check in M_2. This significantly reduces the number of times we check membership in M_2. By variable-time search [Amb10], if ε_1 is the probability that an element sampled from M_0 is in M_1, and $\varepsilon_2 \leq \varepsilon_1$ is the probability that an element sampled from M_0 is in M_2, then a quantum algorithm can find an element of M_2 in complexity

$$\sqrt{\frac{\varepsilon_1}{\varepsilon_2}}\left(\frac{1}{\sqrt{\varepsilon_1}}\mathsf{C}_1+\mathsf{C}_2\right)=\frac{1}{\sqrt{\varepsilon_2}}\mathsf{C}_1+\sqrt{\frac{\varepsilon_1}{\varepsilon_2}}\mathsf{C}_2$$

where we neglect logarithmic factors and assume the cost S of sampling from M_0 is negligible.

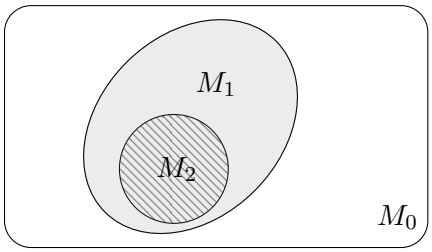

Figure 1: Illustration of the two-oracle search setup, where the task is to search for elements in the search space M_0 that belong to a marked subset $M_2 \subseteq M_0$, given the ability to check membership in both M_2 and some subset M_1 satisfying $M_2 \subseteq M_1 \subseteq M_0$.

2.3.4 Finding all elements from a set with unknown size

Suppose you aim to find all elements of a finite set X by querying an algorithm `Samp` that samples (say) uniformly from X. By the coupon collector's argument, $|X|^{1+o(d)}$ calls to `Samp` suffice to succeed with overwhelming probability. While this might seem to require knowing $|X|$ to decide when to stop, it actually suffices to keep sampling until no new element has been observed for some time — provided a suitable (possibly loose) upper bound on $|X|$ is known.

Lemma 2.4 (Folklore)**.** *Let X be a nonempty finite set and let* `Samp` *be a (classical, resp. quantum) algorithm that samples from X in time S. Suppose there exists $0 < \varepsilon \leq 1$ such that, for all $x \in X$,*

$$\Pr[\texttt{Samp} \text{ outputs } x] \geq \frac{\varepsilon}{|X|}$$

where the probability is over the internal randomness of `Samp`*. For all $\delta \in (0,1)$, there exists a (classical, resp. quantum) algorithm that, given a positive integer $t \geq \frac{1}{\varepsilon}\ln(|X|/\delta)$, outputs all elements of X in time $O(t|X|\ln(|X|)\mathsf{S})$ using $O(t|X|\ln(|X|))$ applications of* `Samp`*, except with probability δ. The algorithm uses $O(|X|)$ classical bits beyond the memory used by* `Samp`*. (The algorithm does not require prior knowledge of $|X|$.)*

Several arguments for this lemma are possible; the one presented below is inspired by [Iir20].

Proof. Let $\mathcal{A}(\texttt{Samp}, t)$ be the algorithm that repeatedly calls `Samp` and maintains the set $S \subseteq X$ of distinct elements seen so far, and that outputs S once no new element has been observed in the last $(|S|+1)t$ calls. (This stopping rule is inspired by [Iir20].) The runtime of $\mathcal{A}(\texttt{Samp}, t)$ is determined by the number of calls to `Samp`, and its additional memory usage beyond that of `Samp` is $O(|X|)$.

We claim that, except with probability δ, the following holds for all $s \in \{0, \ldots, |X|-1\}$: immediately after the set S constructed during $\mathcal{A}(\texttt{Samp}, t)$ is of size s, a new element of X is found within the next $\lceil |X|t/(|X|-s)\rceil$ calls to `Samp`. Since $\lceil |X|t/(|X|-s)\rceil \leq (s+1)t$, this implies that the stopping rule is satisfied if and only if all elements of X have been found, meaning that $\mathcal{A}(\texttt{Samp}, t)$ outputs all elements of X. Moreover, it implies that the total number of calls to `Samp` is upper bounded by

$$\sum_{s=0}^{|X|-1} \left\lceil \frac{|X|t}{|X|-s} \right\rceil + (|X|+1)t \leq |X|t \sum_{j=1}^{|X|} \frac{1}{j} + O(|X|t) = O(t|X|\ln(|X|))$$

since $\sum_{j=1}^{|X|} \frac{1}{j} \leq \ln(|X|) + 1$.

It remains to prove the claim. Fix $s \in \{0, \ldots, |X|-1\}$ and consider the call of `Samp` (during $\mathcal{A}(\texttt{Samp}, t)$) that returns the s-th new element of X. By assumption on ε, the probability that a single run of `Samp` returns a new element of X is at least $(|X|-s)\varepsilon/|X|$, so the probability that the next $\lceil |X|t/(|X|-s)\rceil$ (independent) calls to `Samp` yield no new element is at most

$$\left(1 - (|X|-s)\frac{\varepsilon}{|X|}\right)^{\left\lceil \frac{|X|t}{|X|-s} \right\rceil} \leq e^{-\varepsilon t}.$$

The claim now follows from a union bound over all $s \in \{0, \ldots, |X|-1\}$, because $|X|e^{-\varepsilon t} \leq \delta$ by assumption on t. □

2.3.5 Relations and associated data structures

We will facilitate the application of two-oracle search using carefully constructed relations and associated data structures.

Definition 2.5. A *relation* R on $X \times Y$ is a subset $R \subseteq X \times Y$, equivalently viewed as a function $R : X \times Y \to \{0,1\}$. For every $x \in X$, we let $R(x) = \{y \in Y : (x,y) \in R\}$. We let $R^{\mathsf{T}} \subseteq Y \times X$ be the relation defined by $(y,x) \in R^{\mathsf{T}}$ if and only if $(x,y) \in R$. A pair $x, x' \in X$ such that $R(x) \cap R(x') \neq \emptyset$ is called an *R-collision*.

Note that the latter is a natural generalization (from functions to relations) of the concept of a collision.

The standard way of accessing a function $f : X \to Y$ is through queries, meaning an algorithm is given a description of f that allows the ability to efficiently compute, for any $x \in X$, the value $f(x)$. If f is a 1-to-1 function, then a more powerful type of access allows the efficient computation of $f^{-1}(y)$ for any $y \in Y$. This is not always possible from a simple description of f, but is, for example, possible if all function values of f, $(x, f(x))$, are stored in a data structure, perhaps constructed during some preprocessing step.

For relations, we can distinguish three types of (quantum) access. First, the simplest type, *query access*, means that for any $(x, y) \in X \times Y$, it is possible to efficiently check if $(x, y) \in R$. A second type that is more analogous to evaluation of a function f is *forward superposition query access* to R, meaning we can query an oracle $\mathcal{O}_R$ that acts, for all $x \in X$, as:

$$|x\rangle |0\rangle \mapsto \begin{cases} |x\rangle \sum_{y \in R(x)} \frac{1}{\sqrt{|R(x)|}} |y\rangle & \text{if } R(x) \neq \emptyset \\ |x\rangle |\perp\rangle & \text{otherwise.} \end{cases}$$

The third type of access to R, analogous to having standard and inverse query access to f, is to have query access to both $\mathcal{O}_R$ and $\mathcal{O}_{R^\intercal}$. That is, one can not only implement a forward superposition query, but also its inverse:

$$|y\rangle |0\rangle \mapsto \begin{cases} |y\rangle \sum_{x \in R^\intercal(y)} \frac{1}{\sqrt{|R^\intercal(y)|}} |x\rangle & \text{if } R^\intercal(y) \neq \emptyset \\ |y\rangle |\perp\rangle & \text{otherwise.} \end{cases}$$

We will always implement this third type of access by working with a data structure $D(R)$ that stores R in QCRAM.

Data structures for relations. Specifically, we will store a relation $R \subseteq X \times Y$ in a classical data structure, denoted $D(R)$, in a way such that the following operations can be performed using $O(\log |X| + \log |Y|)$ time and classical memory:

- **Insert:** For any $(x, y) \in (X \times Y) \setminus R$, add (x, y) to $D(R)$.
- **Lookup by x:** For any $x \in X$, return a pointer to an array containing all y such that $(x, y) \in R$, and its size $|R(x)|$.
- **Lookup by y:** For any $y \in Y$, return a pointer to an array containing all x such that $(x, y) \in R$, and its size $|R^\intercal(y)|$.

To accomplish this, we store the elements $(x, y) \in R$ in two different ways: once in a keyed data structure with x as the "key" and y as an associated "value", and once in another keyed data structure, with y as the key, and x as the value.

By storing $D(R)$ in QCRAM, we can use the ability to access the QCRAM in superposition to perform lookups in superposition. We then call $D(R)$ a *QCRAM data structure*.

Lemma 2.6. *Let $D(R)$ be a QCRAM data structure for a relation $R \subseteq X \times Y$. Then the following operations can be performed using $O(\log|X| + \log|Y|)$ time and QCRAM queries:*

- **Insert:** *For any $(x, y) \in (X \times Y) \setminus R$, add (x, y) to $D(R)$.*
- **Lookup by x in superposition:** *For any $x \in X$, map*

$$|x\rangle \mapsto \begin{cases} |x\rangle \sum_{i=1}^{|R(x)|} \frac{1}{\sqrt{|R(x)|}} |i\rangle |y_i\rangle & \text{if } R(x) \neq \emptyset \\ |x\rangle |\perp\rangle & \text{otherwise} \end{cases}$$

  where $\{y_1, \ldots, y_{|R(x)|}\} = R(x)$.

- **Lookup by y in superposition:** *For any $y \in Y$, map*

$$|y\rangle \mapsto \begin{cases} |y\rangle \sum_{i=1}^{|R^\intercal(y)|} \frac{1}{\sqrt{|R^\intercal(y)|}} |i\rangle |x_i\rangle & \text{if } R^\intercal(y) \neq \emptyset \\ |y\rangle |\perp\rangle & \text{otherwise} \end{cases}$$

  where $\{x_1, \ldots, x_{|R^\intercal(y)|}\} = R^\intercal(y)$.

In later sections, we will abuse notation by letting $|x, y\rangle$ denote $|x, i, y\rangle$, where i is the index of y in the array storing $R(x)$, and sometimes we will even let $|x, y, x'\rangle$ denote $|x, i, y, j, x'\rangle$ where i is as above, and j is the index of x' in the array storing $R^\intercal(y)$. This is not a problem, as all we require from the specific encoding of $|x, y\rangle$ is that we can do computations on both x and y — it does not matter if there is superfluous information in the encoding, as long as it is not too large.

Proof. The insertion is directly inherited from the data structure. We describe a superposition lookup of x (the case for y is virtually identical). The classical lookup by x returns a number $N = |R(x)|$, and a pointer to an array storing $R(x) = \{y_1, \ldots, y_N\}$. If $N \neq 0$, then generate $\sum_{i=1}^{N} \frac{1}{\sqrt{N}} |i\rangle |0\rangle$. Use a QCRAM query to access the entries of the array, to map $|i\rangle |0\rangle \mapsto |i\rangle |y_i\rangle$. The time and memory complexities follow immediately from the properties of $D(R)$. □

2.4 Geometric properties of the unit sphere

We denote the sphere of the d-dimensional unit ball by $\mathcal{S}^{d-1} \coloneqq \{\mathbf{x} \in \mathbb{R}^d \colon \|\mathbf{x}\| = 1\}$. A unit vector $\mathbf{x} \in \mathcal{S}^{d-1}$ and angle $\alpha \in [0, \pi/2]$ define the set $\mathcal{H}_{\mathbf{x},\alpha} \coloneqq \{\mathbf{c} \in \mathcal{S}^{d-1} \colon \langle \mathbf{x}, \mathbf{c} \rangle \geq \cos(\alpha)\}$, called the *spherical cap* of center $\mathbf{x}$ and angle α. Note that $\mathcal{H}_{\mathbf{x},0} = \{\mathbf{x}\}$ and that $\mathcal{H}_{\mathbf{x},\pi/2}$ is a hemisphere. The intersection of two spherical caps forms a *spherical wedge*, and is denoted by $\mathcal{W}_{\mathbf{x},\alpha,\mathbf{y},\beta} \coloneqq \mathcal{H}_{\mathbf{x},\alpha} \cap \mathcal{H}_{\mathbf{y},\beta}$ for $\mathbf{x}, \mathbf{y} \in \mathcal{S}^{d-1}$ and $\alpha, \beta \in [0, \pi/2]$. If $\langle \mathbf{x}, \mathbf{y} \rangle = \cos(\theta)$ for $\theta \in [0, \pi/2]$, then

$$\mathcal{W}_{\mathbf{x},\alpha,\mathbf{y},\beta} = \begin{cases} \mathcal{H}_{\mathbf{x},\alpha} & \text{if } \theta \leq \beta - \alpha \\ \mathcal{H}_{\mathbf{y},\beta} & \text{if } \theta \leq \alpha - \beta \\ \emptyset & \text{if } \theta > \alpha + \beta \end{cases} \tag{1}$$

and in the remaining case ($|\alpha - \beta| < \theta \leq \alpha + \beta$) the wedge is nonempty and properly contained in both $\mathcal{H}_{\mathbf{x},\alpha}$ and $\mathcal{H}_{\mathbf{y},\beta}$.

The volume (or hyperarea) of a spherical cap or wedge allows us to quantify the probability that a random unit vector is close to one or two given vectors, respectively. Given $\mathbf{x}, \mathbf{y} \in \mathcal{S}^{d-1}$ with

$\langle \mathbf{x}, \mathbf{y} \rangle = \cos(\theta)$, we denote the ratio of the volume of $\mathcal{W}_{\mathbf{x},\alpha,\mathbf{y},\beta}$ to that of $\mathcal{S}^{d-1}$ by $\mathcal{W}(\alpha, \beta \mid \theta)$, as it depends on $\mathbf{x}, \mathbf{y}$ only through their mutual angle θ. This normalized wedge volume equals the probability that, for a fixed pair $(\mathbf{x}, \mathbf{y})$ of unit vectors satisfying $\langle \mathbf{x}, \mathbf{y} \rangle = \cos(\theta)$, a uniformly random unit vector lies in the wedge $\mathcal{W}_{\mathbf{x},\alpha,\mathbf{y},\beta}$.

The normalized volume of a spherical cap is thus of the form $\mathcal{W}(\alpha, \alpha \mid 0)$, and can be estimated using the following lemma from [BDGL16].

Lemma 2.7 (Volume of a spherical cap [BDGL16, Lemma 2.1])**.** *Let $\alpha \in (0, \pi/2)$. For all $\mathbf{x} \in \mathcal{S}^{d-1}$,*

$$\mathcal{W}(\alpha, \alpha \mid 0) \coloneqq \Pr_{\mathbf{c} \sim \mathcal{U}(\mathcal{S}^{d-1})} [\langle \mathbf{x}, \mathbf{c} \rangle \geq \cos(\alpha)] = d^{\Theta(1)} \underbrace{\left(1 - \cos^2(\alpha)\right)^{d/2}}_{=: p_\alpha}.$$

In [BDGL16], an estimate for the normalized volume of a spherical wedge was also given, but it was missing the condition $|\alpha - \beta| \leq \theta \leq \alpha + \beta$, which is necessary for their formula to hold. Note that the other cases follow directly from Equation (1) and Lemma 2.7: $\mathcal{W}(\alpha, \beta \mid \theta) = 0$ if $\theta > \alpha + \beta$, and $\mathcal{W}(\alpha, \beta \mid \theta) =_d \min\{p_\alpha, p_\beta\}$ if $\theta \geq |\alpha - \beta|$. (When $\theta = |\alpha - \beta|$, Lemma 2.7 and Lemma 2.8 coincide.) More detailed formulas for the wedge volume can be found in [LK14].

Lemma 2.8 (Volume of a spherical wedge (correction of [BDGL16, Lemma 2.2]))**.** *Let $\alpha, \beta, \theta \in (0, \pi/2)$ satisfy $|\alpha - \beta| \leq \theta \leq \alpha + \beta$. For all $\mathbf{x}, \mathbf{y} \in \mathcal{S}^{d-1}$ with $\langle \mathbf{x}, \mathbf{y} \rangle = \cos(\theta)$,*

$$\mathcal{W}(\alpha, \beta \mid \theta) \coloneqq \Pr_{\mathbf{c} \sim \mathcal{U}(\mathcal{S}^{d-1})} [\langle \mathbf{x}, \mathbf{c} \rangle \geq \cos(\alpha) \text{ and } \langle \mathbf{y}, \mathbf{c} \rangle \geq \cos(\beta)] = d^{\Theta(1)} (1 - \gamma^2)^{d/2}$$

where $\gamma^2 \coloneqq \frac{\cos^2(\alpha) + \cos^2(\beta) - 2\cos(\alpha)\cos(\beta)\cos(\theta)}{\sin^2(\theta)}$. In particular, if $\alpha = \beta$,

$$\mathcal{W}(\alpha, \alpha \mid \theta) \coloneqq \Pr_{\mathbf{c} \sim \mathcal{U}(\mathcal{S}^{d-1})} [\langle \mathbf{x}, \mathbf{c} \rangle \geq \cos(\alpha) \text{ and } \langle \mathbf{y}, \mathbf{c} \rangle \geq \cos(\alpha)] = d^{\Theta(1)} \left(1 - \frac{2\cos^2(\alpha)}{1 + \cos(\theta)}\right)^{d/2}.$$

The previous two lemmas allow us to estimate the probability that a random unit vector $\mathbf{c}$ satisfies both $\langle \mathbf{x}, \mathbf{c} \rangle \geq \cos(\alpha)$ and $\langle \mathbf{y}, \mathbf{c} \rangle \geq \cos(\beta)$, for fixed $\mathbf{x}, \mathbf{y} \in \mathcal{S}^{d-1}$. In our analysis, we instead deal with events where those inner products are equal to $\cos(\alpha)$ (respectively, $\cos(\beta)$) up to an additive $\epsilon > 0$. We therefore consider the following variants of Lemma 2.7 and Lemma 2.8. These results are folklore and follow from the previous lemmas, provided that ϵ is chosen sufficiently small (see Section A.1 for the proofs). Throughout this work, we globally fix $\epsilon = 1/(\log d)^2$.

Lemma 2.9 (Volume of a spherical cap, approximate version)**.** *Let $\alpha \in (0, \pi/2)$ satisfy $\alpha = \Omega(1)$. For all $\mathbf{x} \in \mathcal{S}^{d-1}$,*

$$\Pr_{\mathbf{c} \sim \mathcal{U}(\mathcal{S}^{d-1})} [\langle \mathbf{x}, \mathbf{c} \rangle \approx_\epsilon \cos(\alpha)] =_d p_\alpha.$$

Lemma 2.10 (Volume of a spherical wedge, approximate version)**.** *Let $\alpha, \beta, \theta \in (0, \pi/2)$ satisfy $|\alpha - \beta| + \Omega(1) \leq \theta \leq \alpha + \beta - \Omega(1)$. For all $\mathbf{x}, \mathbf{y} \in \mathcal{S}^{d-1}$ with $\langle \mathbf{x}, \mathbf{y} \rangle \approx_\epsilon \cos(\theta)$,*

$$\Pr_{\mathbf{c} \sim \mathcal{U}(\mathcal{S}^{d-1})} [\langle \mathbf{x}, \mathbf{c} \rangle \approx_\epsilon \cos(\alpha) \text{ and } \langle \mathbf{y}, \mathbf{c} \rangle \approx_\epsilon \cos(\beta)] =_d \mathcal{W}(\alpha, \beta \mid \theta).$$

The $\Omega(1)$ terms in these lemmas hide a (small) absolute constant $c > 0$ that is independent of d. They ensure that the non-approximate lemmas remain applicable as $d \to \infty$, which we use to prove Lemma 2.9 and Lemma 2.10. In this work, we mainly consider normalized wedge volumes of the form $\mathcal{W}(\alpha, \alpha \mid \theta)$ or $\mathcal{W}(\theta, \alpha \mid \alpha)$, in which case the condition in Lemma 2.10 simplifies to $\min\{\theta, 2\alpha - \theta\} = \Omega(1)$.

2.5 Random product codes and their induced relations

The main tasks in our quantum algorithm for finding 3-tuple solutions can be formulated as searching for pairs $(\mathbf{x}, \mathbf{y})$ of unit vectors that are somewhat "close" to each other, in the sense that we can bound their inner product. We simplify these tasks by only searching for pairs that form an R-collision under carefully constructed relations of the form $R = R_{(C,\alpha)}$, where

$$R_{(C,\alpha)} \coloneqq \{(\mathbf{x}, \mathbf{c}) \in \mathcal{S}^{d-1} \times C \colon \langle \mathbf{x}, \mathbf{c} \rangle \approx_\epsilon \cos(\alpha)\}$$

for a subset $C \subseteq \mathcal{S}^{d-1}$ and $\alpha \in (0, \pi/2)$. Consequently, if $(\mathbf{x}, \mathbf{y})$ form an $R_{(C,\alpha)}$-collision (i.e., $R_{(C,\alpha)}(\mathbf{x}) \cap R_{(C,\alpha)}(\mathbf{y}) \neq \emptyset$), then there is a point $\mathbf{c} \in C$ that is close to both $\mathbf{x}$ and $\mathbf{y}$, implying that $\mathbf{x}$ and $\mathbf{y}$ are also close to each other, where closeness is quantified by the parameter α. When the dependencies on C and α are clear from context, we often just write R. Note also that those relations $R_{(C,\alpha)}$ are infinite objects, but we will usually restrict them to a finite subset in the first coordinate, for instance the vectors in our list L, making the relation finite.

The family of subsets C that we will work with are based on random product codes [BDGL16].

Definition 2.11 (Random product code)**.** We define $\mathcal{C}(d, b, M)$ as the family of sets $C \subseteq \mathcal{S}^{d-1}$ that can be written as

$$C = \mathbf{Q}(C^{(1)} \times \cdots \times C^{(b)})$$

where $\mathbf{Q} \in \mathrm{SO}(d)$, and $C^{(i)} \subseteq \frac{1}{\sqrt{b}}\mathcal{S}^{d/b-1}$ with $|C^{(i)}| = M^{1/b}$ for each $i \in \{1, \ldots, b\}$. Any such tuple $(\mathbf{Q}, C^{(1)}, \ldots, C^{(b)})$ is called a *description* of C.

A *random product code (RPC)* is a random set $C \in \mathcal{C}(d, b, M)$ obtained by sampling a uniformly random description $(\mathbf{Q}, C^{(1)}, \ldots, C^{(b)})$ from the set of all valid descriptions. We write $\mathrm{RPC}(d, b, M)$ for the resulting distribution over $\mathcal{C}(d, b, M)$.

Random product codes C have two very useful properties. For a parameter α and induced relation $R = R_{(C,\alpha)}$, we have:

(1) **Efficient decodability:** In certain parameter regimes (in particular, if b is not too small), there is an algorithm that, on input $\mathbf{x} \in \mathcal{S}^{d-1}$, computes the set $R(\mathbf{x})$ in time roughly equal to its size $|R(\mathbf{x})|$. See Lemma 2.12. Note that this algorithm gives forward superposition query access to R (see Section 2.3.5).

(2) **Random behavior:** In certain parameter regimes (in particular, if b is not too large), a random product code C behaves like a uniformly random subset of $\mathcal{S}^{d-1}$ in the following sense: for all $\mathbf{x}, \mathbf{y} \in \mathcal{S}^{d-1}$ satisfying $\langle \mathbf{x}, \mathbf{y} \rangle \approx_\epsilon \cos(\theta)$, the probability that there exists $\mathbf{c} \in C$ satisfying $\langle \mathbf{x}, \mathbf{c} \rangle \approx_\epsilon \cos(\alpha)$ and $\langle \mathbf{y}, \mathbf{c} \rangle \approx_\epsilon \cos(\alpha)$ (meaning that $(\mathbf{x}, \mathbf{y})$ is an R-collision) is the same, up to subexponential factors, as in the case that each element of C was independently sampled from $\mathcal{U}(\mathcal{S}^{d-1})$. See Lemma 2.13.

In other words, RPCs give us sufficiently good random behavior, while still allowing for efficient decodability, which is the main reason we work with RPCs instead of uniformly random subsets.

The next two lemmas[3] show that it suffices to take $b = \log d$ for both properties to be satisfied, so we fix this choice of b in the remainder of this paper.

Lemma 2.12 (Efficient decodability (implicit in [BDGL16, Lemma 5.1]))**.** *There exists a classical algorithm that, given a description $(\mathbf{Q}, C^{(1)}, \dots, C^{(b)})$ of $C \in \mathcal{C}(d, b, M)$ and a target vector $\mathbf{x} \in \mathcal{S}^{d-1}$, returns the set $R_{(C,\alpha)}(\mathbf{x}) \coloneqq \{\mathbf{c} \in C \colon \langle \mathbf{x}, \mathbf{c}\rangle \approx_\epsilon \cos(\alpha)\}$ in time $O(d^2 M^{1/b} + dM^{1/b} \log M + bd|R_{(C,\alpha)}(\mathbf{x})|)$. In particular, if $b = \omega(1)$ and $M = 2^{O(d)}$, then the runtime is $O(bd|R_{(C,\alpha)}(\mathbf{x})|) + 2^{o(d)}$.*

Next, Lemma 2.13 provides sufficiently tight bounds on the probability that two unit vectors form a collision under the relation $R_{(C,\alpha)}$ induced by a sample $C \sim \mathrm{RPC}(d, b, M)$. Specifically, it identifies parameter regimes where RPCs behave similarly to uniformly random subsets with respect to collision probabilities, as $\Pr_{C \sim \mathcal{U}(\mathcal{S}^{d-1}, M)}[R_{(C,\alpha)}(\mathbf{x}) \cap R_{(C,\beta)}(\mathbf{y}) \neq \emptyset] = \Theta(\min\{1, M\mathcal{W}(\alpha, \beta \mid \theta)\})$.

Lemma 2.13 (Random behavior [BDGL16, Theorem 5.1])**.** *Let $\alpha, \beta \in (0, \pi/2)$ and $\theta \in [0, \pi/2)$ satisfy $|\alpha - \beta| + \Omega(1) \leq \theta \leq \alpha + \beta - \Omega(1)$ if $\theta \neq 0$, and $\alpha = \beta = \Omega(1)$ otherwise. Let $b = O(\log d)$ and let M be such that $M \cdot \mathcal{W}(\alpha, \beta \mid \theta) \geq 2^{-O(d)}$. For all $\mathbf{x}, \mathbf{y} \in \mathcal{S}^{d-1}$ with $\langle \mathbf{x}, \mathbf{y}\rangle \approx_\epsilon \cos(\theta)$,*

$$\Pr_{C \sim \mathrm{RPC}(d,b,M)}\left[R_{(C,\alpha)}(\mathbf{x}) \cap R_{(C,\beta)}(\mathbf{y}) \neq \emptyset\right] =_d \min\{1, M \cdot \mathcal{W}(\alpha, \beta \mid \theta)\}.$$

Note that the case $\theta = 0$ deals with the probability that $R_{(C,\alpha)}(\mathbf{x})$ is nonempty (meaning there exists $\mathbf{c} \in C$ that is "close" to $\mathbf{x} = \mathbf{y}$).

3 Quantum algorithm for finding many 3-tuple solutions

In this section, we present a quantum algorithm for Problem 1 from the introduction: given a list L of m i.i.d. uniform samples from $\mathcal{S}^{d-1}$, this algorithm returns m triples $(\mathbf{x}, \mathbf{y}, \mathbf{z}) \in L^3$ satisfying $\|\mathbf{x} - \mathbf{y} - \mathbf{z}\| \leq 1$. We are specifically interested in instances with $m = (27/16)^{d/4 + o(d)}$. For sufficiently large $o(d)$, this is the minimal list size to ensure that with high probability over the choice of L there exist m 3-tuple solutions [HK17, Theorem 3], and hence corresponds to the minimal memory regime of Problem 1. Our work is motivated by the observation that, for a list size m that is slightly larger, but still asymptotically $m = (27/16)^{d/4+o(d)}$, there exist in fact m 3-tuple solutions $(\mathbf{x}, \mathbf{y}, \mathbf{z}) \in L^3$ for which $\langle \mathbf{x}, \mathbf{y}\rangle$ is essentially $1/3$ and $\langle \mathbf{x} - \mathbf{y}, \mathbf{z}\rangle$ is essentially $2/3$. This allows us to reduce our search problem to (a less simplified version of) Problem 2 from the introduction. More precisely, as proven in Lemma 3.15, there exist $\theta, \theta' \in (0, \pi/2)$ such that

$$\mathcal{T}_{\mathrm{sol}}(L, \theta, \theta') \coloneqq \left\{(\mathbf{x}, \mathbf{y}, \mathbf{z}) \in L^3 \colon \langle \mathbf{x}, \mathbf{y}\rangle \approx_\epsilon \cos(\theta), \langle \tfrac{\mathbf{x}-\mathbf{y}}{\|\mathbf{x}-\mathbf{y}\|}, \mathbf{z}\rangle \approx_\epsilon \cos(\theta')\right\} \tag{2}$$

consists only of 3-tuple solutions and (with high probability over the choice of L) has size at least m. We therefore design a quantum algorithm that finds m elements of $\mathcal{T}_{\mathrm{sol}}(L, \theta, \theta')$, thereby

[3] The proofs in [BDGL16] consider $R_{(C,\alpha)}(\mathbf{x})$ defined as $\{\mathbf{c} \in C \colon \langle \mathbf{x}, \mathbf{c}\rangle \geq \cos(\alpha)\}$, but can easily be seen to work for our definition (with $\approx_\epsilon$ instead of $\geq$).

solving Problem 1. As we present the algorithm for a fixed choice[4] of (θ, θ'), we usually write $\mathcal{T}_{\mathrm{sol}}$ as shorthand for $\mathcal{T}_{\mathrm{sol}}(L, \theta, \theta')$.

Remark 3. There is a slight subtlety in the definition of $\mathcal{T}_{\mathrm{sol}}(L, \theta, \theta')$: it does not explicitly require each tuple $(\mathbf{x}, \mathbf{y}, \mathbf{z}) \in \mathcal{T}_{\mathrm{sol}}(L, \theta, \theta')$ to consist of distinct vectors, which is required by Problem 1. (Note that the latter problem is trivial otherwise: for instance, consider the $|L|$ 3-tuples $(\mathbf{x}, \mathbf{x}, \mathbf{x})$ for $\mathbf{x} \in L$.) While we could explicitly impose that $\mathbf{x}, \mathbf{y}, \mathbf{z}$ be distinct in Equation (2), this property is already implied by Equation (2) in our setting of interest. Namely, if $\theta = \Omega(1)$ and $\cos(\theta') = \epsilon + \sqrt{(1 - \cos(\theta) + \epsilon)/2}$ (as motivated by Lemma 3.15), then each $(\mathbf{x}, \mathbf{y}, \mathbf{z}) \in \mathcal{T}_{\mathrm{sol}}(L, \theta, \theta')$ that is not composed of distinct vectors must have $\langle \mathbf{x}, \mathbf{y} \rangle = \cos(\theta) - \epsilon$. As the latter event has measure zero for $\mathbf{y} \sim \mathcal{U}(\mathcal{S}^{d-1})$, we may, without loss of generality, assume that in the setting of interest each element of $\mathcal{T}_{\mathrm{sol}}(L, \theta, \theta')$ consists of distinct vectors.

3.1 High-level overview of our quantum algorithm and main result

Our quantum algorithm consists of several steps of amplitude amplification (Lemma 2.3), carefully nested together, and preceded by preprocessing the list L into a useful data structure. We describe the high-level ideas here, and defer the details to the following subsections. As outlined in the introduction, a natural strategy for finding an element of $\mathcal{T}_{\mathrm{sol}}$ is to search for a pair $(\mathbf{x}, \mathbf{y}) \in L^2$ satisfying $\langle \mathbf{x}, \mathbf{y} \rangle \approx_\epsilon \cos(\theta)$, and then attempt to find $\mathbf{z} \in L$ such that $(\mathbf{x}, \mathbf{y}, \mathbf{z}) \in \mathcal{T}_{\mathrm{sol}}$. This approach can be viewed as a two-oracle search (Section 2.3.3) with sets $\mathcal{T}_2 \subseteq \mathcal{T}_1 \subseteq \mathcal{T}_0$ defined by

$$\begin{aligned}
\mathcal{T}_0 &\coloneqq L^2, \\
\mathcal{T}_1 &\coloneqq \{(\mathbf{x}, \mathbf{y}) \in \mathcal{T}_0 \colon \langle \mathbf{x}, \mathbf{y} \rangle \approx_\epsilon \cos(\theta)\}, \\
\mathcal{T}_2 &\coloneqq \{(\mathbf{x}, \mathbf{y}) \in \mathcal{T}_1 \colon \exists \mathbf{z} \in L, \langle \tfrac{\mathbf{x}-\mathbf{y}}{\|\mathbf{x}-\mathbf{y}\|}, \mathbf{z} \rangle \approx_\epsilon \cos(\theta')\}.
\end{aligned}$$

Whereas checking membership in $\mathcal{T}_1$ has negligible time complexity $\mathsf{C}_1 = 2^{o(d)}$, checking membership in $\mathcal{T}_2$ is a more involved search problem with nontrivial time complexity C_2. For instance, using amplitude amplification we obtain $\mathsf{C}_2 = \sqrt{|L|}\, 2^{o(d)}$ when $|L| = 2^{O(d)}$. Assuming that, given $(\mathbf{x}, \mathbf{y}) \in \mathcal{T}_2$, we can also *find* $\mathbf{z}$ such that $(\mathbf{x}, \mathbf{y}, \mathbf{z}) \in \mathcal{T}_{\mathrm{sol}}$ in time C_2 (which is true for amplitude amplification), this results in a quantum algorithm for finding an element of $\mathcal{T}_{\mathrm{sol}}$ in time

$$\sqrt{\frac{\varepsilon_1}{\varepsilon_2}} \left(\frac{1}{\sqrt{\varepsilon_1}} \mathsf{C}_1 + \mathsf{C}_2 \right) 2^{o(d)}$$

where ε_1 and ε_2 are the probabilities that an element sampled uniformly at random from $\mathcal{T}_0$ lies in $\mathcal{T}_1$ and $\mathcal{T}_2$, respectively. Repeating the algorithm about $|L|$ times then hopefully solves Problem 1. Unfortunately, this naive strategy is too expensive.[5]

[4] For Problem 1, choosing $\cos(\theta) = 1/3$ and $\cos(\theta') = \epsilon + \sqrt{1/3 + \epsilon/2}$ is optimal in the sense that it yields the smallest possible list size $|L|$ for which $\mathcal{T}_{\mathrm{sol}}(L, \theta, \theta')$ consists of $|L|$ 3-tuple solutions. However, our quantum algorithm may also be of interest in settings where other choices are more suitable, such as the generalization of Problem 1 that searches for triples satisfying $\|\mathbf{x} - \mathbf{y} - \mathbf{z}\| \leq t$ for $t \neq 1$ (e.g., see [HK17]). We therefore present most results for a larger range of $\theta, \theta' \in (0, \pi/2)$.

[5] One can show $\varepsilon_1 =_d p_\theta$ and $\varepsilon_2 =_d p_\theta \min\{1, |L| p_{\theta'}\}$. For $|L| = (27/16)^{d/4 + o(d)}$, taking $\cos(\theta) \approx 1/3$ and $\cos(\theta') \approx 1/\sqrt{3}$ is necessary and sufficient to ensure $\mathcal{T}_{\mathrm{sol}}(L, \theta, \theta')$ has size $\geq |L|$ with high probability (see Lemma 3.15). This implies that the naive approach (using amplitude amplification to check membership in $\mathcal{T}_2$) takes time at least $2^{0.3350d + o(d)}$ to find $|L|$ solutions. Moreover, it yields $\frac{1}{\sqrt{\varepsilon_1}} \mathsf{C}_1 \ll \mathsf{C}_2$, suggesting that this approach is suboptimal.

Inspired by the locality-sensitive filtering technique, which has been used in state-of-the-art lattice sieving algorithms since [BDGL16], our key idea to improve this naive strategy is to search more *locally*. The sets $\mathcal{T}_0, \mathcal{T}_1, \mathcal{T}_2$ will be replaced by random subsets of them that only include those pairs of vectors that lie in the same "local" region of $\mathcal{S}^{d-1}$ (formally defined as forming a collision under some suitable relation), meaning that these vectors are not too far apart. While those subsets may not cover all of $\mathcal{T}_{\text{sol}}$, their randomness will ensure that repeating this approach sufficiently many times for different random subsets allows us to find all elements of $\mathcal{T}_{\text{sol}}$. Altogether, this modified strategy results in a better trade-off between the different cost components of two-oracle search.

Specifically, we restrict $\mathcal{T}_0 = L^2$ to those pairs of vectors in L that are both "close" to some vector in a fixed subset $C \subseteq \mathcal{S}^{d-1}$, where closeness is measured by a parameter α. Letting R be the relation on $\mathcal{S}^{d-1} \times C$ including exactly those pairs $(\mathbf{x}, \mathbf{c})$ that satisfy $\langle \mathbf{x}, \mathbf{c} \rangle \approx_\epsilon \cos(\alpha)$, we define

$$\mathcal{T}_0(R) \coloneqq \{(\mathbf{x}, \mathbf{y}) \in \mathcal{T}_0 \colon R(\mathbf{x}) \cap R(\mathbf{y}) \neq \emptyset\}$$

as the set of R-collisions in $\mathcal{T}_0 = L^2$. For each $(\mathbf{x}, \mathbf{y}) \in \mathcal{T}_0(R)$, we are now guaranteed that both $\mathbf{x}$ and $\mathbf{y}$ are close to some $\mathbf{c} \in C$, so they are also somewhat close to each other, and have a higher chance of satisfying $\langle \mathbf{x}, \mathbf{y} \rangle \approx_\epsilon \cos(\theta)$. We will therefore replace the role of $\mathcal{T}_1$ by a suitable subset

$$\mathcal{T}_1(R) \subseteq \{(\mathbf{x}, \mathbf{y}) \in \mathcal{T}_0(R) \colon \langle \mathbf{x}, \mathbf{y} \rangle \approx_\epsilon \cos(\theta)\}$$

consisting of R-collisions in $\mathcal{T}_1$. The probability ε_1' that an element sampled from $\mathcal{T}_0(R)$ is in $\mathcal{T}_1(R)$ could now be larger than ε_1, while the cost C_1' of checking membership in $\mathcal{T}_1(R)$ remains negligible, so we seem to have reduced one component of the two-oracle search cost.

However, there is a caveat: in order to project onto $\mathcal{T}_1(R)$ using amplitude amplification, we need a unitary that creates a superposition over this restricted subset $\mathcal{T}_0(R)$ of L^2, and it is not immediately clear that finding R-collisions in L^2 is easy. This will be resolved by adding a *pre-processing* phase during which the algorithm prepares a data structure in QCRAM that stores the finite relation $R_L \coloneqq R|_{L \times C}$, allowing us to efficiently construct a superposition over the elements of $\mathcal{T}_0(R)$ at any later stage of the algorithm. By taking C to be a random product code (RPC, Definition 2.11), we can prepare such a data structure at reasonable cost.

Next, given a superposition over $\mathcal{T}_1(R)$, the goal is to check whether there exists $\mathbf{z} \in L$ satisfying $\langle \frac{\mathbf{x}-\mathbf{y}}{\|\mathbf{x}-\mathbf{y}\|}, \mathbf{z} \rangle \approx_\epsilon \cos(\theta')$ for a given pair $(\mathbf{x}, \mathbf{y}) \in \mathcal{T}_1(R)$. Again, we will restrict our search to achieve this more efficiently than performing amplitude amplification over L: given $(\mathbf{x}, \mathbf{y}) \in \mathcal{T}_1(R)$, we only consider those $\mathbf{z} \in L$ that collide with the normalization of $\mathbf{x} - \mathbf{y}$ under the relation $R' \coloneqq \{(\mathbf{x}, \mathbf{c}) \in \mathcal{S}^{d-1} \times C' \colon \langle \mathbf{x}, \mathbf{c} \rangle \approx_\epsilon \cos(\alpha')\}$ defined by another subset $C' \subseteq \mathcal{S}^{d-1}$ and parameter α'. That is, the search is restricted to a subset

$$\mathcal{T}_2(R, R') \subseteq \{(\mathbf{x}, \mathbf{y}) \in \mathcal{T}_1(R) \colon \exists \mathbf{z} \in L, \langle \tfrac{\mathbf{x}-\mathbf{y}}{\|\mathbf{x}-\mathbf{y}\|}, \mathbf{z} \rangle \approx_\epsilon \cos(\theta'), R'(\tfrac{\mathbf{x}-\mathbf{y}}{\|\mathbf{x}-\mathbf{y}\|}) \cap R'(\mathbf{z}) \neq \emptyset\}.$$

This "local" search is again facilitated by letting C' be a random product code and by preparing a data structure for $R'_L \coloneqq R'|_{L \times C'}$. If restricting to R'-collisions reduces the search for $\mathbf{z}$ to a much smaller subset of L, then the cost C_2' of checking membership in $\mathcal{T}_2(R, R')$ could be significantly less than the cost C_2 in the naive approach, possibly resulting in an improved overall time complexity.

Indeed, this *local* two-oracle search algorithm finds elements of $\mathcal{T}_2(R, R') \subseteq \mathcal{T}_2$ more efficiently than the naive strategy. By sampling the subsets C, C' randomly, using the RPC distribution from Definition 2.11, we can ensure that this approach finds a sufficiently random subset of $\mathcal{T}_{\text{sol}}$, so

repeating the local two-oracle search algorithm for sufficiently many random pairs (C, C') allows us to find $|L|$ elements of $\mathcal{T}_{\text{sol}}$, thereby solving Problem 1.

We summarize the resulting quantum algorithm, called `3List`, in Algorithm 1.[6] In the following sections, we explain and analyze the individual phases of `3List` in more detail, allowing us to then prove our main result, Theorem 3.1, in Section 3.6. We remark that we cannot use the result of [KLL15] or [Amb10] directly: as our goal is to find all elements of $\mathcal{T}_2(R, R')$, we want to sample a single element with sufficient randomness so that many samples are likely to correspond to many distinct elements. We therefore give our own algorithm and analysis, for our particular setting.

Algorithm 1 `3List`

Input:	A list $L \subseteq \mathcal{S}^{d-1}$; Angles $\theta, \theta' \in (0, \pi/2)$
Parameters:	Angles $\alpha, \alpha' \in (0, \pi/2)$; Number of repetitions ℓ; Search threshold t
Output:	A list L' consisting of 3-tuple solutions in $\mathcal{T}_{\text{sol}}(L, \theta, \theta')$

1. $L' \leftarrow \emptyset$
2. Repeat for ℓ times:
   (a) **Sample:** Sample $C \sim \text{RPC}(d, \log d, 1/p_\alpha)$ and $C' \sim \text{RPC}(d, \log d, 1/p_{\alpha'})$
   (b) **Preprocess:** $(D, D') \leftarrow$ `Preprocess`(L, C, C') (Algorithm 2)
   (c) **Search:** Use (D, D') to find 3-tuple solutions:
      i. $S \leftarrow \emptyset$, $\text{COUNT} \leftarrow 0$
      ii. While $\text{COUNT} < (|S| + 1)t$:
         A. $\mathbf{s} \leftarrow$ `SolutionSearch`(D, D') (Algorithm 5)
         B. If $\mathbf{s} \in L^3 \setminus S$, then add $\mathbf{s}$ to S and set $\text{COUNT} \leftarrow 0$
         C. Else, set $\text{COUNT} \leftarrow \text{COUNT} + 1$
      iii. Add S to L'
3. Return L'

Theorem 3.1. *There exists a quantum algorithm that, with probability $1 - 2^{-\Omega(d)}$, solves Problem 1 with list size $m = (27/16)^{d/4+o(d)}$ in time $2^{0.284551d+o(d)}$. This algorithm uses $m2^{o(d)}$ classical memory and QCRAM bits, and $2^{o(d)}$ qubits.*

As explained in Section 4, Theorem 3.1 implies the existence of a quantum algorithm that *heuristically* solves the Shortest Vector Problem with the stated time and memory complexity.

[6] The algorithm `SolutionSearch` in the **Search** phase is essentially the aforementioned local two-oracle search algorithm, and samples from $\mathcal{T}_2(R, R')$ for fixed (R, R'). By choosing the parameter t carefully, the **Search** phase finds all elements of $\mathcal{T}_2(R, R')$ using $|\mathcal{T}_2(R, R')|2^{o(d)}$ calls to `SolutionSearch`, even if $|\mathcal{T}_2(R, R')|$ is not known a priori.

3.2 The Sampling phase

The **Sampling** phase of Algorithm 1 samples two RPCs (Definition 2.11) C and C', and stores their descriptions. This phase can be completed using $2^{o(d)}$ time and classical memory. The sizes of C and C' depend on the parameters $\alpha, \alpha' \in (0, \pi/2)$, which affect the complexity of `3List` and will be chosen to optimize performance. The sampled RPCs C, C' and angles α, α' induce the relations

$$R \coloneqq \{(\mathbf{x}, \mathbf{c}) \in \mathcal{S}^{d-1} \times C \colon \langle \mathbf{x}, \mathbf{c} \rangle \approx_\epsilon \cos(\alpha)\} \quad \text{and} \quad R' \coloneqq \{(\mathbf{x}, \mathbf{c}) \in \mathcal{S}^{d-1} \times C' \colon \langle \mathbf{x}, \mathbf{c} \rangle \approx_\epsilon \cos(\alpha')\}.$$

Remark 4 (On the RPC parameter setting). The parameter setting of the distribution of $C \sim \mathrm{RPC}(d, \log d, 1/p_\alpha)$ ensures that, for a given $\mathbf{x} \in \mathcal{S}^{d-1}$, the set $R(\mathbf{x}) = \{\mathbf{c} \in C \colon \langle \mathbf{x}, \mathbf{c} \rangle \approx_\epsilon \cos(\alpha)\}$ has expected size $2^{o(d)}$ (taken over the choice of C) by Lemma 2.9. The set $R(\mathbf{x})$ can therefore be computed in expected time $2^{o(d)}$ by Lemma 2.12. The same is true for C', and those properties are utilized during the **Preprocessing** and **Search** phases of `3List`.

To simplify our general analysis of `3List`, we impose the following conditions on the relations (R, R'). In the proof of Lemma 3.16 (the core lemma underlying Theorem 3.1), we show that with overwhelming probability over the input list $L \sim \mathcal{U}(\mathcal{S}^{d-1}, m)$, all relations encountered during `3List` satisfy these conditions.

Definition 3.2 (Well-balanced (R, R'))**.** For $L, C, C' \subseteq \mathcal{S}^{d-1}$, let $R \subseteq \mathcal{S}^{d-1} \times C$ and $R' \subseteq \mathcal{S}^{d-1} \times C'$ be relations. We say that (R, R') is *well-balanced on* L if the following conditions hold:

(*i*) $|R_L| =_d |L|$ and $|(R_L)^{\mathsf{T}}(\mathbf{c})| =_d \frac{|R_L|}{|C|}$ for all $\mathbf{c} \in C$, where $R_L \coloneqq R|_{L \times C}$.

(*ii*) $|R'_L| =_d |L|$ and $|(R'_L)^{\mathsf{T}}(\mathbf{c}')| =_d \frac{|R'_L|}{|C'|}$ for all $\mathbf{c}' \in C'$, where $R'_L \coloneqq R'|_{L \times C'}$.

(*iii*) For all $(\mathbf{x}, \mathbf{y}, \mathbf{z}) \in \mathcal{T}_{\mathrm{sol}}(R, R')$ (Equation (3)), $|\{\mathbf{z}' \in L \colon (\mathbf{x}, \mathbf{y}, \mathbf{z}') \in \mathcal{T}_{\mathrm{sol}}(R, R')\}| =_d 1$.

3.3 The Preprocessing phase

After having sampled two subsets $C, C' \subseteq \mathcal{S}^{d-1}$, we proceed to the **Preprocessing** phase, which uses Algorithm 2 to construct QCRAM data structures D and D' (as defined in Section 2.3.5) for the relations induced by these subsets and the parameters α and α'.

By the end of this classical algorithm, D stores the relation $R_L \subseteq L \times C$, which relates each $\mathbf{x} \in L$ to all "close" vectors in C, quantified by the parameter α. Similarly, D' stores the relation $R'_L \subseteq L \times C'$, which relates each $\mathbf{x} \in L$ to all close vectors in C', this time quantified by α'.

By construction of C and C', the **Preprocessing** phase can be completed in time that is optimal up to subexponential factors.

Lemma 3.3 (Preprocessing cost)**.** *Given as input a list $L \subseteq \mathcal{S}^{d-1}$ and the descriptions of sets $C \in \mathcal{C}(d, \log d, 1/p_\alpha)$ and $C' \in \mathcal{C}(d, \log d, 1/p_{\alpha'})$,* `Preprocess`$(L, C, C')$ *(Algorithm 2) returns the data structures $D(R_L)$ and $D(R'_L)$ using $(|R_L| + |R'_L|)2^{o(d)}$ time and classical memory. In particular, if (R, R') is well-balanced on L (Definition 3.2), it uses $|L|2^{o(d)}$ time and classical memory.*

Proof. By Lemma 2.12, for each $\mathbf{x} \in L$, step 2(a) and step 2(c) take time $|R_L(\mathbf{x})|2^{o(d)}$ and $|R'_L(\mathbf{x})|2^{o(d)}$, respectively, so the cost of step 2 is $\sum_{\mathbf{x} \in L}(|R_L(\mathbf{x})| + |R'_L(\mathbf{x})|)2^{o(d)} = (|R_L| + |R'_L|)2^{o(d)}$.

As the complexity of `Preprocess` is determined by step 2, this proves the first part of the lemma. The second part follows from conditions (i) and (ii) in Definition 3.2. □

Algorithm 2 `Preprocess`(L, C, C')

Input: A list $L \subseteq \mathcal{S}^{d-1}$;
Descriptions of $C \in \mathcal{C}(d, \log d, 1/p_\alpha)$ and $C' \in \mathcal{C}(d, \log d, 1/p_{\alpha'})$

Output: QCRAM data structures $D = D(R_L)$ and $D' = D(R'_L)$ for the relations

$$R_L = \{(\mathbf{x}, \mathbf{c}) \in L \times C : \langle \mathbf{x}, \mathbf{c} \rangle \approx_\epsilon \cos(\alpha)\} \quad \text{and} \quad R'_L = \{(\mathbf{x}, \mathbf{c}) \in L \times C' : \langle \mathbf{x}, \mathbf{c} \rangle \approx_\epsilon \cos(\alpha')\}$$

---

1. Initialize a pair of empty data structures D and D'
2. For each $\mathbf{x} \in L$:
   (a) Compute $R_L(\mathbf{x})$ using Lemma 2.12
   (b) For each $\mathbf{c} \in R_L(\mathbf{x})$:
      i. Insert $(\mathbf{x}, \mathbf{c})$ into D
   (c) Compute $R'_L(\mathbf{x})$ using Lemma 2.12
   (d) For each $\mathbf{c} \in R'_L(\mathbf{x})$:
      i. Insert $(\mathbf{x}, \mathbf{c})$ into D'
3. Return D and D'

While most steps in the **Search** phase of Algorithm 1 can be achieved using the data structures $D(R_L)$ and $D(R'_L)$ that are prepared during the **Preprocessing** phase, we actually need one more tool. Namely, we would like to be able to efficiently create a uniform superposition over

$$R'(\mathbf{x}) = \{\mathbf{c}' \in C' \colon \langle \mathbf{x}, \mathbf{c}' \rangle \approx_\epsilon \cos(\alpha')\}$$

for a large number of vectors $\mathbf{x} \in \mathcal{S}^{d-1}$ that the algorithm will encounter (in superposition). That is, we want forward superposition query access to R' (recall Section 2.3.5) for those vectors. These $\mathbf{x}$ are not vectors from our list L itself, but rather (normalized) differences of two vectors from L. As the number of those $\mathbf{x}$ will be rather large ($\gg m$), it is too expensive for us to store each $R'(\mathbf{x})$ in a data structure. To ensure that we can create the superposition over $R'(\mathbf{x})$ in subexponential time (i.e., at negligible cost for us), we therefore consider a "decoding" subroutine `Dec`(C') that is guaranteed to run in time $2^{o(d)}$, and achieves the desired mapping for all $\mathbf{x} \in \mathcal{S}^{d-1}$ satisfying $|R'(\mathbf{x})| \leq 2^{d/\log d}$, which suffices for our purposes.

Lemma 3.4. *For $\alpha' \in (0, \pi/2)$ and $C' \in \mathcal{C}(d, \log d, 1/p_{\alpha'})$, define the relation*

$$R' := \{(\mathbf{x}, \mathbf{c}') \in \mathcal{S}^{d-1} \times C' : \langle \mathbf{x}, \mathbf{c}' \rangle \approx_\epsilon \cos(\alpha')\}.$$

There is a quantum algorithm $\texttt{Dec}(C')$ *that, given a description of* C'*, implements the map*

$$\widetilde{\mathcal{O}}_{R'}\colon |\mathbf{x}\rangle|0\rangle \mapsto \begin{cases} |\mathbf{x}\rangle \sum_{\mathbf{c}'\in R'(\mathbf{x})} \frac{1}{\sqrt{|R'(\mathbf{x})|}} |\mathbf{c}'\rangle & \text{if } |R'(\mathbf{x})| \in [1, 2^{d/\log d}] \\ |\mathbf{x}\rangle|\perp\rangle & \text{otherwise} \end{cases}$$

in time $2^{o(d)}$ *using an auxiliary register of* $2^{o(d)}$ *qubits. In particular,* $\texttt{Dec}(C')$ *correctly implements* $\mathcal{O}_{R'}$ *for all* $\mathbf{x} \in \mathcal{S}^{d-1}$ *satisfying* $|R'(\mathbf{x})| \le 2^{d/\log d}$*.*

Proof. The subroutine starts by enumerating the elements of $R'(\mathbf{x})$ using Lemma 2.12 in an auxiliary register, but stops as soon as it has found $2^{d/\log d}+1$ elements, unless it has already finished earlier. (Note that the algorithm from Lemma 2.12 finds elements of $R'(\mathbf{x})$ one by one.) If it finds either zero or $2^{d/\log d}+1$ elements, which means $|R'(\mathbf{x})| \notin [1, 2^{d/\log d}]$, then it undoes the computation and maps the second register to $|\perp\rangle$. Otherwise, when $|R'(\mathbf{x})| \in [1, 2^{d/\log d}]$, the subroutine prepares a uniform superposition over the found elements in the second register and then uncomputes the auxiliary register. Altogether this takes time $2^{o(d)}$ and uses $2^{o(d)}$ auxiliary qubits. (Note that, just as in Lemma 2.6, $|\mathbf{x}, \mathbf{c}'\rangle$ will be encoded by $|\mathbf{x}, i, \mathbf{c}'\rangle$ for some index i that depends on $\mathbf{x}$, $\mathbf{c}'$, and C', but this is not an issue.) □

3.4 The Search phase

The **Search** phase of Algorithm 1 repeatedly invokes a quantum algorithm called `SolutionSearch`, which will be presented in Algorithm 5. As mentioned before, this algorithm searches for elements of $\mathcal{T}_{\text{sol}}$ by carefully nesting two layers of amplitude amplification (Lemma 2.3) and by searching for collisions under the relations R, R' defined by the sets $C, C' \subseteq \mathcal{S}^{d-1}$ obtained during the **Sampling** phase (and by the angles α, α'). This is achieved by leveraging the data structures $D(R_L)$ and $D(R'_L)$ that were prepared during the **Preprocessing** phase for the finite relations $R_L \coloneqq R|_{L\times C}$ and $R'_L \coloneqq R'|_{L\times C'}$.

More precisely, `SolutionSearch` samples 3-tuple solutions from the set

$$\begin{aligned} \mathcal{T}_{\text{sol}}(R, R') \coloneqq \big\{(\mathbf{x},\mathbf{y},\mathbf{z}) \in \mathcal{T}_{\text{sol}}\colon\ & R(\mathbf{x}) \cap R(\mathbf{y}) \neq \emptyset, |R(\mathbf{x})| \le 2^{d/\log d}, \\ & R'\big(\tfrac{\mathbf{x}-\mathbf{y}}{\|\mathbf{x}-\mathbf{y}\|}\big) \cap R'(\mathbf{z}) \neq \emptyset, \big|R'\big(\tfrac{\mathbf{x}-\mathbf{y}}{\|\mathbf{x}-\mathbf{y}\|}\big)\big| \le 2^{d/\log d}\big\} \end{aligned} \tag{3}$$

by gradually refining the search through the sets $\mathcal{T}_0(R) \supseteq \mathcal{T}_1(R) \supseteq \mathcal{T}_2(R, R')$ defined as

$$\begin{aligned} \mathcal{T}_0(R) &\coloneqq \big\{(\mathbf{x},\mathbf{y}) \in L^2\colon R(\mathbf{x}) \cap R(\mathbf{y}) \neq \emptyset\big\}, \\ \mathcal{T}_1(R) &\coloneqq \big\{(\mathbf{x},\mathbf{y}) \in \mathcal{T}_0(R)\colon \langle \mathbf{x},\mathbf{y}\rangle \approx_\epsilon \cos(\theta), |R(\mathbf{x})| \le 2^{d/\log d}\big\}, \\ \mathcal{T}_2(R, R') &\coloneqq \big\{(\mathbf{x},\mathbf{y}) \in \mathcal{T}_1(R)\colon \exists \mathbf{z} \in L, (\mathbf{x},\mathbf{y},\mathbf{z}) \in \mathcal{T}_{\text{sol}}(R, R')\big\}. \end{aligned} \tag{4}$$

The constraints that $R(\mathbf{x})$ and $R'(\frac{\mathbf{x}-\mathbf{y}}{\|\mathbf{x}-\mathbf{y}\|})$ are of size $2^{o(d)}$ are imposed for technical reasons.[7]

The core of `SolutionSearch` is a subroutine, `TupleSamp` (presented in Algorithm 4), which generates a superposition over a subset of $\mathcal{T}_1(R) \times L$ and "flags" those tuples that belong to $\mathcal{T}_{\text{sol}}(R, R')$.

[7] In our applications, these sets will be of subexponential size for most $(\mathbf{x}, \mathbf{y})$. Excluding the rare cases where this is not the case ensures that the sampling probabilities over $\mathcal{T}_{\text{sol}}(R, R')$ remain approximately uniform.

SolutionSearch applies amplitude amplification on top of TupleSamp to keep only those flagged tuples, and then measures the resulting state, yielding an element of $\mathcal{T}_{\text{sol}}(R, R') \subseteq \mathcal{T}_{\text{sol}}$.

The sampling subroutine TupleSamp first generates a superposition over $(\mathbf{x}, \mathbf{y}) \in \mathcal{T}_1(R)$ by applying amplitude amplification to a subroutine RCollisionSamp (presented in Algorithm 3) that uses $D(R_L)$ to efficiently generate a superposition over $\mathcal{T}_0(R)$. TupleSamp then proceeds by searching for $\mathbf{z}$ such that $(\mathbf{x}, \mathbf{y}, \mathbf{z}) \in \mathcal{T}_{\text{sol}}$, setting a flag if such a $\mathbf{z}$ is found. Using $D(R'_L)$, we save computation effort by restricting this search to those $\mathbf{z}$ that form an R'-collision with $\frac{\mathbf{x}-\mathbf{y}}{\|\mathbf{x}-\mathbf{y}\|}$.

The structure of the main algorithm 3List and its subroutines, including the **Search** phase described in this section, is illustrated in Figure 2. We now give precise definitions of the algorithms forming the **Search** phase, and analyze them in full detail.

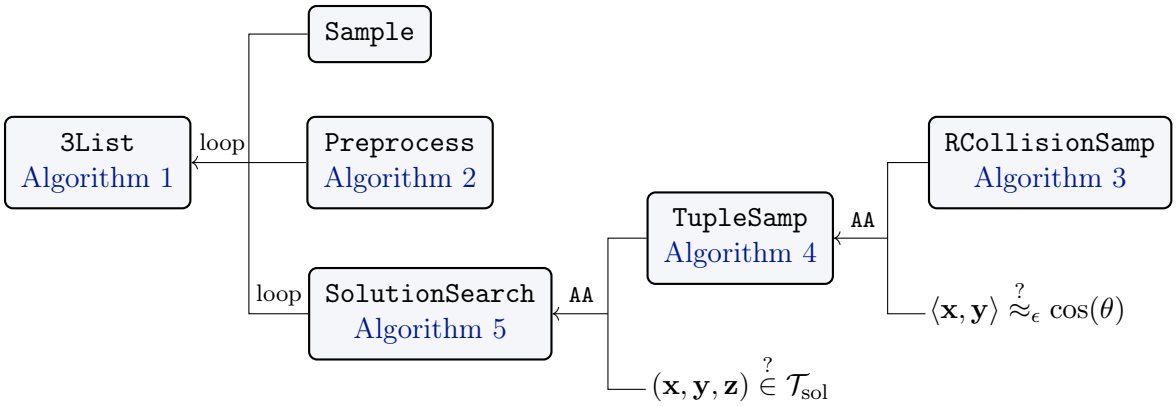

Figure 2: Structure of the algorithm 3List, which repeats the following. First, the **Sampling** phase produces a pair (R, R') of random relations that are stored in a data structure during the **Preprocessing** phase. The algorithm then repeatedly calls SolutionSearch, which is instructed to find an element of $\mathcal{T}_{\text{sol}}$ using a nested amplitude amplification (AA). Given a subroutine RCollisionSamp that creates a superposition over R-collisions $(\mathbf{x}, \mathbf{y}) \in L^2$, the first AA amplifies those that satisfy $\langle \mathbf{x}, \mathbf{y} \rangle \approx_\epsilon \cos(\theta)$. Next, TupleSamp extends them to triples $(\mathbf{x}, \mathbf{y}, \mathbf{z})$ such that $(\frac{\mathbf{x}-\mathbf{y}}{\|\mathbf{x}-\mathbf{y}\|}, \mathbf{z})$ forms an R'-collision (if such a $\mathbf{z}$ exists), and the final AA amplifies those triples that belong to $\mathcal{T}_{\text{sol}}$.

3.4.1 Sampling an R-collision

We start with the bottom layer, RCollisionSamp (Algorithm 3). For arbitrary sets L and C, this algorithm takes as input a QCRAM data structure $D(R)$ storing a relation $R \subseteq L \times C$ (see Section 2.3.5), and outputs a superposition over R-collisions, specifically over $(\mathbf{x}, \mathbf{c}, \mathbf{y}) \in L \times C \times L$ such that $\mathbf{c} \in R(\mathbf{x}) \cap R(\mathbf{y})$, as visualized in part (a) of Figure 3. While RCollisionSamp works for any relation, we will apply it to $L, C \subseteq \mathcal{S}^{d-1}$ (where L is an instance of Problem 1 and C an RPC), so we use vector notation such as $\mathbf{x}$ and $\mathbf{c}$ for elements of these sets.

RCollisionSamp starts by taking a uniform superposition over all $\mathbf{x} \in L$. Then, for all $\mathbf{x}$ such that $R(\mathbf{x}) \neq \emptyset$, it creates a superposition over all $\mathbf{c} \in R(\mathbf{x})$ in the second register, and subsequently over all $\mathbf{y} \in R^{\intercal}(\mathbf{c})$ in the third register. These steps are easy using the data structure $D(R)$. In case $R(\mathbf{x}) = \emptyset$, the second and third register are mapped to $|\perp\rangle\,|\perp\rangle$, but in our applications this typically accounts for a small fraction of the state, so it is easily suppressed using amplitude amplification when Algorithm 3 is called by TupleSamp (Algorithm 4).

Algorithm 3 `RCollisionSamp`$(D(R))$

Input: $|0,0,0\rangle$;
A QCRAM data structure $D(R)$ for $R \subseteq L \times C$, where L and C are finite sets

Output: A superposition $|\psi\rangle$ over $(\mathbf{x}, \mathbf{c}, \mathbf{y}) \in L \times (C \cup \perp) \times (L \cup \perp)$

1. Generate a uniform superposition over L in the first register: $|0,0,0\rangle \mapsto \frac{1}{\sqrt{|L|}} \sum_{\mathbf{x} \in L} |\mathbf{x}\rangle |0\rangle |0\rangle$.

2. Controlled on $\mathbf{x}$ in the first register, generate a uniform superposition over the set $R(\mathbf{x})$ in the second register, or map the second register to $|\perp\rangle$ if $R(\mathbf{x})$ is empty:

$$|\mathbf{x}\rangle |0\rangle |0\rangle \mapsto \begin{cases} |\mathbf{x}\rangle \left(\frac{1}{\sqrt{|R(\mathbf{x})|}} \sum_{\mathbf{c} \in R(\mathbf{x})} |\mathbf{c}\rangle \right) |0\rangle & \text{if } R(\mathbf{x}) \neq \emptyset \\ |\mathbf{x}\rangle |\perp\rangle |0\rangle & \text{if } R(\mathbf{x}) = \emptyset \end{cases}$$

3. Controlled on $\mathbf{c} \neq \perp$ in the second register, generate a superposition over the set $R^{\mathsf{T}}(\mathbf{c})$ in the third register, and map $|\perp\rangle |0\rangle$ to $|\perp\rangle |\perp\rangle$:

$$|\mathbf{x}\rangle |\mathbf{c}\rangle |0\rangle \mapsto \begin{cases} |\mathbf{x}\rangle |\mathbf{c}\rangle \left(\frac{1}{\sqrt{|R^{\mathsf{T}}(\mathbf{c})|}} \sum_{\mathbf{y} \in R^{\mathsf{T}}(\mathbf{c})} |\mathbf{y}\rangle \right) & \text{if } \mathbf{c} \neq \perp \\ |\mathbf{x}\rangle |\mathbf{c}\rangle |\perp\rangle & \text{if } \mathbf{c} = \perp \end{cases}$$

4. Return the final quantum state

Lemma 3.5 (Analysis of `RCollisionSamp`)**.** *For finite sets L and C, let $R \subseteq L \times C$. Let $D(R)$ be a QCRAM data structure for R.* `RCollisionSamp`$(D(R))$ *(Algorithm 3) outputs a state $|\psi\rangle$ such that, for all $(\mathbf{x}, \mathbf{c}, \mathbf{y}) \in L \times C \times L$ satisfying $\mathbf{c} \in R(\mathbf{x}) \cap R(\mathbf{y})$, we have*

$$\langle \mathbf{x}, \mathbf{c}, \mathbf{y} | \psi \rangle = \frac{1}{\sqrt{|L|\,|R(\mathbf{x})|\,|R^{\mathsf{T}}(\mathbf{c})|}}.$$

The algorithm uses $O(\log|L| + \log|C|)$ time and QCRAM queries, and $O(\log|L| + \log|C|)$ qubits.

Proof. The operations used by the subroutine (taking a uniform superposition over L, taking a uniform superposition over $R(\mathbf{x})$ for any $\mathbf{x} \in L$, and taking a uniform superposition over $R^{\mathsf{T}}(\mathbf{c})$ for any $\mathbf{c} \in C$) can all be done using $O(\log|L| + \log|C|)$ time and QCRAM queries, by Lemma 2.6. Since Algorithm 3 uses $O(\log|L| + \log|C|)$ qubits, the claim on the time and memory complexities follows. The output state of Algorithm 3 is

$$|\psi\rangle \coloneqq \frac{1}{\sqrt{|L|}} \sum_{\substack{\mathbf{x} \in L: \\ R(\mathbf{x}) \neq \emptyset}} \frac{1}{\sqrt{|R(\mathbf{x})|}} \sum_{\mathbf{c} \in R(\mathbf{x})} \frac{1}{\sqrt{|R^{\mathsf{T}}(\mathbf{c})|}} \sum_{\mathbf{y} \in R^{\mathsf{T}}(\mathbf{c})} |\mathbf{x}\rangle |\mathbf{c}\rangle |\mathbf{y}\rangle + \frac{1}{\sqrt{|L|}} \sum_{\substack{\mathbf{x} \in L: \\ R(\mathbf{x}) = \emptyset}} |\mathbf{x}\rangle |\perp\rangle |\perp\rangle .$$

In particular, $\langle \mathbf{x}, \mathbf{c}, \mathbf{y} | \psi \rangle = 1/\sqrt{|L|\,|R(\mathbf{x})|\,|R^{\mathsf{T}}(\mathbf{c})|}$ whenever $\mathbf{c} \in R(\mathbf{x})$ and $\mathbf{y} \in R^{\mathsf{T}}(\mathbf{c})$. □

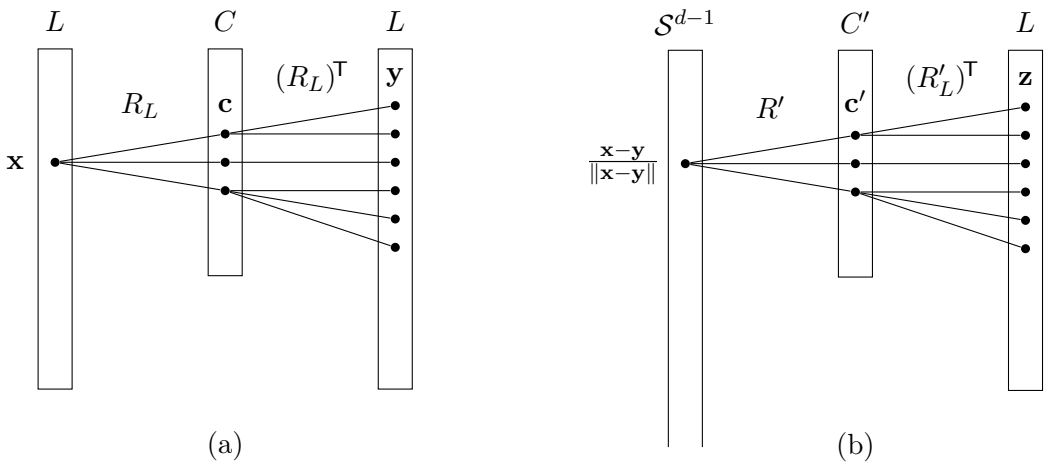

Figure 3: Summary of the relations between vectors encountered during the **Search** phase. Part (a) visualizes the relations during the subroutine `RCollisionSamp`, which first creates a superposition over all $\mathbf{x} \in L$, followed by taking, for each such $\mathbf{x}$, a superposition over all $\mathbf{c}$ such that $(\mathbf{x}, \mathbf{c}) \in R_L$, and then, for each such $\mathbf{c}$, over all $\mathbf{y}$ such that $(\mathbf{y}, \mathbf{c}) \in R_L$. This results in a superposition over all R_L-collisions (that is, all R-collisions in L^2). Part (b) visualizes what happens after the first step of `TupleSamp`, which amplifies those R_L-collisions $(\mathbf{x}, \mathbf{y})$ that satisfy $\langle \mathbf{x}, \mathbf{y} \rangle \approx_\epsilon \cos(\theta)$. Namely, the second step creates, for any such $(\mathbf{x}, \mathbf{y})$, a superposition over $\mathbf{c}' \in C'$ satisfying $(\frac{\mathbf{x}-\mathbf{y}}{\|\mathbf{x}-\mathbf{y}\|}, \mathbf{c}') \in R'$, and, for each such $\mathbf{c}'$, the third step creates a superposition over all $\mathbf{z}$ such that $(\mathbf{z}, \mathbf{c}') \in R'_L$, as visualized in the figure, and amplifies those $\mathbf{z}$ satisfying $\langle \frac{\mathbf{x}-\mathbf{y}}{\|\mathbf{x}-\mathbf{y}\|}, \mathbf{z} \rangle \approx_\epsilon \cos(\theta')$. As for most $(\mathbf{x}, \mathbf{y})$ no such $\mathbf{z}$ exists, `SolutionSearch` applies amplitude amplification on top of `TupleSamp` to amplify exactly those $(\mathbf{x}, \mathbf{y})$ where such a $\mathbf{z}$ does exist.

3.4.2 TupleSamp

The next layer is the subroutine `TupleSamp`, presented in Algorithm 4, which considers relations on $\mathcal{S}^{d-1}$. Its goal is to construct a quantum state $|\psi'\rangle$ that has sufficiently large overlap with $\mathcal{T}_{\text{sol}}(R, R')$ (Equation (3)) so that putting amplitude amplification on top (as will be done by `SolutionSearch`) allows to sample from $\mathcal{T}_{\text{sol}}(R, R')$ at not too high cost. In fact, for our applications, we want that *each* element in $\mathcal{T}_{\text{sol}}(R, R')$ has rather large overlap with $|\psi'\rangle$, ensuring that repeatedly calling `SolutionSearch` allows for finding *all* elements of $\mathcal{T}_{\text{sol}}(R, R')$, and not just the same element over and over again.

`TupleSamp` takes as input two relations $R \subseteq \mathcal{S}^{d-1} \times C$ and $R' \subseteq \mathcal{S}^{d-1} \times C'$, or rather their restrictions to L, given as data structures $D(R_L)$ and $D(R'_L)$. It also assumes the existence of an oracle that creates a uniform superposition over $R'(\frac{\mathbf{x}-\mathbf{y}}{\|\mathbf{x}-\mathbf{y}\|})$ for vectors $(\mathbf{x}, \mathbf{y}) \in L^2$, which we will implement using Lemma 3.4. `TupleSamp` first creates a superposition over R-collisions $(\mathbf{x}, \mathbf{y}) \in L^2$ that belong to $\mathcal{T}_1(R)$ (implying that $\mathbf{x}$ and $\mathbf{y}$ are relatively "close"), and then searches for a "close" $\mathbf{z} \in L$ among those that form an R'-collision with $\frac{\mathbf{x}-\mathbf{y}}{\|\mathbf{x}-\mathbf{y}\|}$. Such a $\mathbf{z}$ might not exist for all $(\mathbf{x}, \mathbf{y})$, so we use a flag qubit that is set to $|1\rangle$ whenever such a $\mathbf{z}$ was found.

In particular, step 1 of `TupleSamp` creates a superposition over $\mathcal{T}_1(R)$ by applying amplitude amplification to `RCollisionSamp` (Algorithm 3), amplifying those R-collisions $(\mathbf{x}, \mathbf{y}) \in L^2$ that satisfy $\langle \mathbf{x}, \mathbf{y} \rangle \approx_\epsilon \cos(\theta)$ and $|R(\mathbf{x})| \leq 2^{d/\log d}$.[8]

[8]Note that the amplification step also checks whether $\mathbf{y} \neq \perp$, because the superposition constructed by `RCollisionSamp` may have nonzero amplitude on pairs $(\mathbf{x}, \mathbf{y})$ with $\mathbf{y} = \perp$, namely if $R(\mathbf{x}) = \emptyset$.

Algorithm 4 $\texttt{TupleSamp}(D(R_L), D(R'_L))$

Input: $|0,0,0,0,0\rangle\,|0,0\rangle$, where the last two qubits form the flag register;
QCRAM data structures $D(R_L)$ and $D(R'_L)$ for the relations

$$R_L \coloneqq R|_{L\times C} \quad \text{and} \quad R'_L \coloneqq R'|_{L\times C'}$$

where $L, C, C' \subseteq \mathcal{S}^{d-1}$ are finite, $R \subseteq \mathcal{S}^{d-1} \times C$, and $R' \subseteq \mathcal{S}^{d-1} \times C'$

Output: A superposition $|\psi'\rangle$ over $(\mathbf{x}, \mathbf{c}, \mathbf{y}, \mathbf{c}', \mathbf{z}, b_F, b'_F)$ such that all basis states in its support with $b'_F = 1$ satisfy $(\mathbf{x}, \mathbf{y}, \mathbf{z}) \in \mathcal{T}_{\text{sol}}(R, R')$

1. Apply $\texttt{AA}_r(\texttt{RCollisionSamp}(D(R_L)), \texttt{RCollisionCheck})$ to the first three registers and first flag qubit, where:
   - $r = \max\{1, \sqrt{|\mathcal{T}_0(R)|}\}\, 2^{o(d)}$ (computed using $D(R_L)$)
   - $\texttt{RCollisionSamp}$ (Algorithm 3) generates a superposition $|\psi\rangle$ over $(\mathbf{x}, \mathbf{c}, \mathbf{y})$ such that either $\mathbf{c} \in R(\mathbf{x}) \cap R(\mathbf{y})$ or $\mathbf{y} = \perp$
   - $\texttt{RCollisionCheck}$ checks if $\mathbf{y} \neq \perp$, $\langle \mathbf{x}, \mathbf{y} \rangle \approx_\epsilon \cos(\theta)$, and $|R(\mathbf{x})| \leq 2^{d/\log d}$

   This step sets the first flag qubit to $|1\rangle$ *if and only if* $\mathcal{T}_1(R) \neq \emptyset$.

2. Controlled on the first three registers being in state $|\mathbf{x}, \mathbf{c}, \mathbf{y}\rangle$ and the first flag qubit in $|1\rangle$, apply the following unitary map to the fourth register:

$$|0\rangle \mapsto \begin{cases} \dfrac{1}{\sqrt{|R'(\frac{\mathbf{x}-\mathbf{y}}{\|\mathbf{x}-\mathbf{y}\|})|}} \displaystyle\sum_{\mathbf{c}' \in R'(\frac{\mathbf{x}-\mathbf{y}}{\|\mathbf{x}-\mathbf{y}\|})} |\mathbf{c}'\rangle & \text{if } |R'(\frac{\mathbf{x}-\mathbf{y}}{\|\mathbf{x}-\mathbf{y}\|})| \in [1, 2^{d/\log d}] \\ |\perp\rangle & \text{otherwise} \end{cases}$$

3. Controlled on the first four registers being in state $|\mathbf{x}, \mathbf{c}, \mathbf{y}, \mathbf{c}'\rangle$ and the first flag qubit in $|1\rangle$, apply $\texttt{AA}_{r'}(\texttt{zSamp}(\mathbf{c}'), \texttt{zCheck}(\mathbf{x}, \mathbf{y}))$ to the fifth register and second flag qubit, where:
   - $r' \coloneqq \sqrt{|(R'_L)^{\mathsf{T}}(\mathbf{c}')|}$ (computed using $D(R'_L)$)
   - $\texttt{zSamp}(\mathbf{c}')$ uses $D(R'_L)$ to apply the following map to the fifth register:

$$|0\rangle \mapsto \begin{cases} \dfrac{1}{\sqrt{|(R'_L)^{\mathsf{T}}(\mathbf{c}')|}} \displaystyle\sum_{\mathbf{z} \in (R'_L)^{\mathsf{T}}(\mathbf{c}')} |\mathbf{z}\rangle & \text{if } \mathbf{c}' \neq \perp \text{ and } (R'_L)^{\mathsf{T}}(\mathbf{c}') \neq \emptyset \\ |\perp\rangle & \text{otherwise} \end{cases}$$

   - $\texttt{zCheck}(\mathbf{x}, \mathbf{y})$ checks if $\mathbf{z} \neq \perp$ and $\langle \frac{\mathbf{x}-\mathbf{y}}{\|\mathbf{x}-\mathbf{y}\|}, \mathbf{z} \rangle \approx_\epsilon \cos(\theta')$

   This step ensures the second flag qubit has support on $|1\rangle$ *if and only if* $\mathcal{T}_{\text{sol}}(R, R') \neq \emptyset$.

4. Return the final quantum state

Lemma 3.6 (Analysis of `TupleSamp`). *For $L, C, C' \subseteq \mathcal{S}^{d-1}$ of size $2^{O(d)}$, let $R \subseteq \mathcal{S}^{d-1} \times C$ and $R' \subseteq \mathcal{S}^{d-1} \times C'$ be such that (R, R') is well-balanced on L (Definition 3.2). Let $D(R_L)$ and $D(R'_L)$ be QCRAM data structures for $R_L := R|_{L\times C}$ and $R'_L := R'|_{L\times C'}$. With probability at least $1 - 2^{-\omega(d)}$, $\texttt{TupleSamp}(D(R_L), D(R'_L))$ (Algorithm 4) generates a quantum state $|\psi'\rangle$ and satisfies:*

(1) Complexity: If $\mathcal{T}_1(R) \neq \emptyset$, then it uses

$$\left(\sqrt{\frac{|\mathcal{T}_0(R)|}{|\mathcal{T}_1(R)|}} + \sqrt{\frac{|L|}{|C'|}}\right) 2^{o(d)}$$

time and QCRAM queries, and $\max\{1, \sqrt{|\mathcal{T}_0(R)|}\}\, 2^{o(d)}$ otherwise. It uses $2^{o(d)}$ qubits.

(2) Near-uniform sampling: The probability that measuring $|\psi'\rangle$ yields outcome $(\mathbf{x}, \mathbf{y}, \mathbf{z}, b'_F)$ with $b'_F = 1$ satisfies

$$\Pr[(\mathbf{x}, \mathbf{y}, \mathbf{z}, 1)] =_d \begin{cases} \dfrac{1}{|\mathcal{T}_1(R)|} & \text{if } (\mathbf{x}, \mathbf{y}, \mathbf{z}) \in \mathcal{T}_{\mathrm{sol}}(R, R') \\ 0 & \text{otherwise.} \end{cases}$$

Proof. Suppose (R, R') is well-balanced on L. We will show that there exists an implementation of $\texttt{TupleSamp}(D(R_L), D(R'_L))$ (Algorithm 4) with the desired properties.

Let Π be the orthogonal projector onto the subspace marked by `RCollisionCheck`. Lemma 3.5 implies that $\texttt{RCollisionSamp}(D(R_L))$ has complexity $2^{o(d)}$ and generates a state $|\psi\rangle$ such that

$$\|\Pi\,|\psi\rangle\|^2 = \sum_{\substack{(\mathbf{x},\mathbf{y})\in\mathcal{T}_1(R),\\ \mathbf{c}\in R(\mathbf{x})\cap R(\mathbf{y})}} \frac{1}{|L|\,|R(\mathbf{x})|\,|(R_L)^{\mathsf{T}}(\mathbf{c})|} =_d \frac{|\mathcal{T}_1(R)|}{|\mathcal{T}_0(R)|}$$

if $\mathcal{T}_1(R) \neq \emptyset$, and $\|\Pi\,|\psi\rangle\| = 0$ otherwise. Here, we use that $|R(\mathbf{x})| \in [1, 2^{o(d)}]$ for all $(\mathbf{x}, \mathbf{y}) \in \mathcal{T}_1(R)$, and that condition (i) of Definition 3.2 implies $|L|\,|(R_L)^{\mathsf{T}}(\mathbf{c})| =_d |L|\,|R_L|/|C| =_d |\mathcal{T}_0(R)|$ for all $\mathbf{c} \in C$. Moreover, all $r =_d \max\{1, |L|/\sqrt{|C|}\}$ satisfy $r =_d \max\{1, \sqrt{|\mathcal{T}_0(R)|}\}$. Choosing $r = \max\{1, |L|/\sqrt{|C|}\}\, 2^{o(d)}$ sufficiently large (note that $|L|$ and $|C|$ are known) ensures $\|\Pi\,|\psi\rangle\| \geq 1/r$ whenever $\|\Pi\,|\psi\rangle\| \neq 0$ (the latter holds if and only if $\mathcal{T}_1(R) \neq \emptyset$). Next, note that `RCollisionCheck` can be implemented in time $2^{o(d)}$, using one query to $D(R_L)$ to compute $|R(\mathbf{x})|$. Therefore, Lemma 2.3 implies that, with probability $1 - 2^{-\omega(d)}$, step 1 of `TupleSamp` can be implemented using $t_1 \leq_d \sqrt{\max\{1, |\mathcal{T}_0(R)|\}/\max\{1, |\mathcal{T}_1(R)|\}}$ time and QCRAM queries and using one auxiliary qubit, and maps

$$|0,0,0\rangle\,|0\rangle \mapsto \frac{\Pi\,|\psi\rangle}{\|\Pi\,|\psi\rangle\|}\,|1\rangle$$

if $\mathcal{T}_1(R) \neq \emptyset$, and $|0,0,0\rangle\,|0\rangle \to |\psi\rangle\,|0\rangle$ otherwise. In the remainder of this proof, we assume step 1 was successful, so the first flag qubit is either fully supported on $|0\rangle$ or fully supported on $|1\rangle$. Since steps 2 and 3 are only applied in the latter case, in the remainder of the proof we omit writing this first flag qubit.

Step 2 can be implemented using the quantum algorithm $\texttt{Dec}(C')$ from Lemma 3.4 in time $t_2 = 2^{o(d)}$ using $2^{o(d)}$ auxiliary qubits.[9] For step 3, we use the data structure $D(R'_L)$ to implement

[9] We remark that $\texttt{Dec}(C')$ takes as input a description of C'. We assume without loss of generality that such a description (which is of size $2^{o(d)}$) is stored in $D(R'_L)$ during the **Preprocessing** phase.

$\texttt{zSamp}(\mathbf{c}')$ using $2^{o(d)}$ time and QCRAM queries, as established by Lemma 2.6. Note that $D(R'_L)$ stores the size $|(R'_L)^{\mathsf{T}}(\mathbf{c}')|$, so step 3 (controlled on $(\mathbf{x},\mathbf{c},\mathbf{y},\mathbf{c}')$) can be implemented using the algorithm $\texttt{AA}_{r'}$ from Lemma 2.3 with $r' \coloneqq \sqrt{|(R'_L)^{\mathsf{T}}(\mathbf{c}')|}$. In addition, $\texttt{zCheck}(\mathbf{x},\mathbf{y})$ is a straightforward check that takes time at most $2^{o(d)}$. By Lemma 2.3 (see also Remark 2), this implements step 3 with probability $1-2^{-\omega(d)}$ in time $t_3 = \sqrt{\max_{\mathbf{c}'\in C'} |(R'_L)^{\mathsf{T}}(\mathbf{c}')|}\, 2^{o(d)}$, and maps

$$|\mathbf{x},\mathbf{c},\mathbf{y},\mathbf{c}'\rangle\,|0\rangle\,|1,0\rangle \mapsto \begin{cases} |\mathbf{x},\mathbf{c},\mathbf{y},\mathbf{c}'\rangle \dfrac{1}{\sqrt{|L(\mathbf{x},\mathbf{y},\mathbf{c}')|}} \displaystyle\sum_{\mathbf{z}\in L(\mathbf{x},\mathbf{y},\mathbf{c}')} |\mathbf{z}\rangle\,|1,1\rangle & \text{if } \mathbf{c}'\neq\perp \text{ and } L(\mathbf{x},\mathbf{y},\mathbf{c}')\neq\emptyset \\ |\mathbf{x},\mathbf{c},\mathbf{y},\mathbf{c}'\rangle\,|\perp\rangle\,|1,0\rangle & \text{otherwise} \end{cases}$$

where $L(\mathbf{x},\mathbf{y},\mathbf{c}') \coloneqq \{\mathbf{z}\in L\colon \langle \frac{\mathbf{x}-\mathbf{y}}{\|\mathbf{x}-\mathbf{y}\|},\mathbf{z}\rangle \approx_\epsilon \cos(\theta'), \mathbf{c}'\in R'(\frac{\mathbf{x}-\mathbf{y}}{\|\mathbf{x}-\mathbf{y}\|})\cap R'(\mathbf{z})\}$. Since $t_3 \le_d |L|/|C'|$ by condition (ii) of being well-balanced, $\texttt{TupleSamp}$ uses

$$t_1+t_2+t_3 \le_d \sqrt{\frac{|\mathcal{T}_0(R)|}{|\mathcal{T}_1(R)|}} + \sqrt{\frac{|L|}{|C'|}}$$

time and QCRAM queries if $\mathcal{T}_1(R)\neq\emptyset$, and $t_1+t_2+t_3 \le_d \max\{1,\sqrt{|\mathcal{T}_0(R)|}\}$ otherwise. The claim on the memory complexity follows immediately from Lemma 3.4, Lemma 3.5, and the construction of the algorithm. This proves statement (1).

By construction of $\texttt{TupleSamp}$, its output state $|\psi'\rangle$ has nonzero overlap with $|\mathbf{x},\mathbf{y},\mathbf{z}\rangle\,|1\rangle$ (the latter register corresponding to the second flag qubit) if and only if $(\mathbf{x},\mathbf{y},\mathbf{z})\in\mathcal{T}_{\mathrm{sol}}(R,R')$, so fix an arbitrary $(\mathbf{x},\mathbf{y},\mathbf{z})\in\mathcal{T}_{\mathrm{sol}}(R,R')$. Note that this implies $\mathcal{T}_1(R)\neq\emptyset$, so the first flag qubit is fully supported on $|1\rangle$. By the above, $|\psi'\rangle$ satisfies

$$\langle \mathbf{x},\mathbf{c},\mathbf{y},\mathbf{c}',\mathbf{z},1,1|\psi'\rangle = \frac{1}{\sqrt{\|\Pi\,|\psi\rangle\|^2 |L|\,|R(\mathbf{x})|\,|(R_L)^{\mathsf{T}}(\mathbf{c})|\,|R'(\frac{\mathbf{x}-\mathbf{y}}{\|\mathbf{x}-\mathbf{y}\|})|\,|L(\mathbf{x},\mathbf{y},\mathbf{c}')|}}$$

if $\mathbf{c}\in R(\mathbf{x})\cap R(\mathbf{y})$ and $\mathbf{c}'\in R'(\frac{\mathbf{x}-\mathbf{y}}{\|\mathbf{x}-\mathbf{y}\|})\cap R'(\mathbf{z})$; and $\langle \mathbf{x},\mathbf{c},\mathbf{y},\mathbf{c}',\mathbf{z},1,1|\psi'\rangle = 0$ otherwise. Using condition (i) and (iii) of being well-balanced and the fact that $|R(\mathbf{x})| =_d |R'(\frac{\mathbf{x}-\mathbf{y}}{\|\mathbf{x}-\mathbf{y}\|})| =_d 1$ for $(\mathbf{x},\mathbf{y},\mathbf{z})\in\mathcal{T}_{\mathrm{sol}}(R,R')$, we conclude that a measurement of $|\psi'\rangle$ yields outcome $(\mathbf{x},\mathbf{y},\mathbf{z},1)$ with probability $1/|\mathcal{T}_1(R)|$, up to subexponential factors in d. □

Using the ingredients of step 1 of $\texttt{TupleSamp}$ (Algorithm 4), we can also estimate $|\mathcal{T}_1(R)|$ up to a subexponential factor in d, which will be used in Algorithm 5.

Lemma 3.7 (Estimating $|\mathcal{T}_1(R)|$)**.** *For $L,C\subseteq\mathcal{S}^{d-1}$ of size $2^{O(d)}$, let $R\subseteq\mathcal{S}^{d-1}\times C$ satisfy condition (i) in Definition 3.2, and let $D(R_L)$ be a QCRAM data structure for $R_L \coloneqq R|_{L\times C}$. There exists a quantum algorithm that, with probability $1-2^{-\omega(d)}$, returns an integer $\tilde\tau$ satisfying $\tilde\tau=0$ if $\mathcal{T}_1(R)=\emptyset$ and $\tilde\tau =_d |\mathcal{T}_1(R)|$ otherwise, using $\sqrt{\max\{1,|\mathcal{T}_0(R)|\}/\max\{1,|\mathcal{T}_1(R)|\}}\,2^{o(d)}$ time and QCRAM queries, and using $2^{o(d)}$ qubits.*

Proof. Define $\texttt{RCollisionSamp}$ and $\texttt{RCollisionCheck}$ as in step 1 of Algorithm 4. Let $|\psi\rangle$ be the state generated by $\texttt{RCollisionSamp}$, and Π the orthogonal projector onto the subspace marked by $\texttt{RCollisionCheck}$. Note that $\mathcal{T}_1(R)=\emptyset$ if and only if $\|\Pi\,|\psi\rangle\|=0$. Moreover, by the proof

of Lemma 3.6 (using the assumption on R), if $\mathcal{T}_1(R) \neq \emptyset$, then $\|\Pi|\psi\rangle\|^2 =_d |\mathcal{T}_1(R)|/|\mathcal{T}_0(R)|$, where $|\mathcal{T}_0(R)|$ is known up to a subexponential factor in d. To prove the lemma, it thus suffices to show we can decide whether $\|\Pi|\psi\rangle\| = 0$, and compute an estimate $\tilde{a} =_d \|\Pi|\psi\rangle\|$, in the desired complexity and with success probability $1 - 2^{-\omega(d)}$.

To decide whether $\|\Pi|\psi\rangle\| = 0$, we can apply step 1 of `TupleSamp` (Algorithm 4) to the state $|0,0,0\rangle|0\rangle$, and measure the last (flag) qubit. By the proof of Lemma 3.6, with probability $1-2^{-\omega(d)}$, the measurement outcome is 0 if and only if $\|\Pi|\psi\rangle\| = 0$, and the complexity is within the desired bounds.

Moreover, $\|\Pi|\psi\rangle\|$ can be estimated using a variant of amplitude estimation. For instance, by [AR20, Thm. 3], there is a quantum algorithm that, with probability $1 - 2^{-\omega(d)}$, returns an integer $\tilde{a} = \Theta(\|\Pi|\psi\rangle\|)$ if $0 < \|\Pi|\psi\rangle\| < 1$, using at most $\|\Pi|\psi\rangle\|^{-1}2^{o(d)} \leq \sqrt{|\mathcal{T}_0(R)|/|\mathcal{T}_1(R)|}2^{o(d)}$ applications of `RCollisionSamp` and `RCollisionCheck`, and using $2^{o(d)}$ qubits. The possibility that $\|\Pi|\psi\rangle\| = 1$ can be dealt with in affordable time using standard methods. □

3.4.3 SolutionSearch

Finally, the outermost layer of the **Search** phase is Algorithm 5, which uses amplitude amplification to project the output state $|\psi'\rangle$ of `TupleSamp` (Algorithm 4) onto $\mathcal{T}_{\text{sol}}(R, R')$. If $\mathcal{T}_{\text{sol}}(R, R')$ is nonempty, then this state $|\psi'\rangle$ is solely supported on $(\mathbf{x}, \mathbf{y}, \mathbf{z}, b'_F)$ with $(\mathbf{x}, \mathbf{y}) \in \mathcal{T}_1(R)$, and with the flag bit set to $b'_F = 1$ only if $(\mathbf{x}, \mathbf{y}, \mathbf{z}) \in \mathcal{T}_{\text{sol}}(R, R')$. Thus, as long as we perform sufficiently many rounds of amplitude amplification, Algorithm 5 successfully outputs an element of $\mathcal{T}_{\text{sol}}(R, R')$.

Algorithm 5 `SolutionSearch`$(D(R_L), D(R'_L))$

Input: QCRAM data structures $D(R_L)$ and $D(R'_L)$ for the relations

$$R_L \coloneqq R|_{L\times C} \quad \text{and} \quad R'_L \coloneqq R'|_{L\times C'}$$

where $L, C, C' \subseteq \mathcal{S}^{d-1}$ are finite, $R \subseteq \mathcal{S}^{d-1} \times C$, and $R' \subseteq \mathcal{S}^{d-1} \times C'$

Output: An element $(\mathbf{x}, \mathbf{y}, \mathbf{z}) \in \mathcal{T}_{\text{sol}}(R, R')$ if $\mathcal{T}_{\text{sol}}(R, R') \neq \emptyset$, and "no solution" otherwise

1. Initialize $|0,0,0,0,0\rangle|0,0\rangle$
2. If $\mathcal{T}_1(R) \neq \emptyset$ (decided using Lemma 3.7), then apply $\texttt{AA}_r(\texttt{TupleSamp}(D(R_L), D(R'_L)), \texttt{TupleCheck})$, where:
   - $r = \sqrt{|\mathcal{T}_1(R)|}\, 2^{o(d)}$ (computed using Lemma 3.7)
   - `TupleSamp` (Algorithm 4) generates a superposition over $|\mathbf{x}, \mathbf{c}, \mathbf{y}, \mathbf{c}', \mathbf{z}\rangle|b_F, b'_F\rangle$ such that either $b'_F = 0$ or $(\mathbf{x}, \mathbf{y}, \mathbf{z}) \in \mathcal{T}_{\text{sol}}(R, R')$
   - `TupleCheck` checks if $b'_F = 1$
3. Measure. From outcome $(\mathbf{x}, \mathbf{y}, \mathbf{z}, b'_F)$, output $(\mathbf{x}, \mathbf{y}, \mathbf{z})$ if $b'_F = 1$, and "no solution" otherwise.

The relations between the vectors encountered during `SolutionSearch` are visualized in Figure 3.

Lemma 3.8 (Analysis of `SolutionSearch`). *For $L, C, C' \subseteq \mathcal{S}^{d-1}$ of size $2^{O(d)}$, let $R \subseteq \mathcal{S}^{d-1} \times C$ and $R' \subseteq \mathcal{S}^{d-1} \times C'$ be such that (R, R') is well-balanced on L (Definition 3.2). Let $D(R_L)$ and $D(R'_L)$ be QCRAM data structures for $R_L \coloneqq R|_{L\times C}$ and $R'_L \coloneqq R'|_{L\times C'}$. With probability $1-2^{-\omega(d)}$, $\texttt{SolutionSearch}(D(R_L), D(R'_L))$ (Algorithm 5) satisfies the following:*

(1) Complexity: If $\mathcal{T}_{\mathrm{sol}}(R, R') \neq \emptyset$, then it outputs $(\mathbf{x}, \mathbf{y}, \mathbf{z}) \in \mathcal{T}_{\mathrm{sol}}(R, R')$ using

$$\sqrt{\frac{|\mathcal{T}_1(R)|}{|\mathcal{T}_{\mathrm{sol}}(R, R')|}} \left(\sqrt{\frac{|\mathcal{T}_0(R)|}{|\mathcal{T}_1(R)|}} + \sqrt{\frac{|L|}{|C'|}} \right) 2^{o(d)}$$

time and QCRAM queries; otherwise, it outputs "no solution" using

$$\left(\max\left\{1, \sqrt{|\mathcal{T}_0(R)|}\right\} + \sqrt{|\mathcal{T}_1(R)| \frac{|L|}{|C'|}} \right) 2^{o(d)}$$

time and QCRAM queries. It uses $2^{o(d)}$ qubits.

(2) Near-uniform sampling: For all $(\mathbf{x}, \mathbf{y}, \mathbf{z}) \in \mathcal{T}_{\mathrm{sol}}(R, R')$,

$$\Pr\left[\texttt{SolutionSearch}(D(R_L), D(R'_L)) \text{ outputs } (\mathbf{x}, \mathbf{y}, \mathbf{z})\right] =_d \frac{1}{|\mathcal{T}_{\mathrm{sol}}(R, R')|}$$

where the probability is over the internal randomness of `SolutionSearch`.

Proof. For step 2 of `SolutionSearch`, we invoke the algorithm from Lemma 3.7 to decide whether $\mathcal{T}_1(R) \neq \emptyset$ and to estimate $|\mathcal{T}_1(R)|$ up to a subexponential factor in d. This uses $2^{o(d)}$ auxiliary qubits. If $\mathcal{T}_1(R) = \emptyset$, then the lemma statement follows immediately, so suppose $\mathcal{T}_1(R) \neq \emptyset$. By Lemma 3.6, with probability at least $1 - 2^{-\omega(d)}$, the output state $|\psi'\rangle$ of `TupleSamp` (Algorithm 4) satisfies

$$\|\Pi' |\psi'\rangle\|^2 =_d \begin{cases} \dfrac{|\mathcal{T}_{\mathrm{sol}}(R, R')|}{|\mathcal{T}_1(R)|} & \text{if } \mathcal{T}_{\mathrm{sol}}(R, R') \neq \emptyset \\ 0 & \text{otherwise} \end{cases}$$

where Π' denotes the orthogonal projector onto the subspace where $b'_F = 1$. If $\mathcal{T}_{\mathrm{sol}}(R, R') \neq \emptyset$, then $\|\Pi' |\psi'\rangle\| \geq_d 1/\sqrt{|\mathcal{T}_1(R)|}$. Choosing $r = \sqrt{|\mathcal{T}_1(R)|}\, 2^{o(d)}$ sufficiently large (using the computed estimate for $|\mathcal{T}_1(R)|$) thus ensures $\|\Pi' |\psi'\rangle\| \geq 1/r$, allowing us to invoke algorithm $\texttt{AA}_r$ from Lemma 2.3 to implement step 2. Since `TupleCheck` can be implemented in time $2^{o(d)}$, statements (1) and (2) now follow from Lemma 3.6. □

3.4.4 Overall analysis of the Search phase

We now complete our analysis of the **Search** phase of `3List` (Algorithm 1). By applying a coupon-collector argument (see Lemma 2.4), it follows that for a sufficiently large search threshold $t = 2^{o(d)}$, the **Search** phase finds, with overwhelming probability, all elements of $\mathcal{T}_{\mathrm{sol}}(R, R')$ using at most $|\mathcal{T}_{\mathrm{sol}}(R, R')|2^{o(d)}$ calls to `SolutionSearch`, even if $|\mathcal{T}_{\mathrm{sol}}(R, R')|$ is not known to the algorithm a priori. This establishes the overall complexity of the **Search** phase.

Lemma 3.9 (Analysis of the **Search** phase of `3List`)**.** *Let $t = 2^{o(d)}$ be sufficiently large.*[10] *Consider a single repetition of the **Search** phase of* `3List` *(Algorithm 1) for $L, C, C' \subseteq \mathcal{S}^{d-1}$ of size $2^{O(d)}$, given the QCRAM data structures $D(R_L)$ and $D(R'_L)$ constructed in the **Preprocessing** phase. Suppose (R, R') is well-balanced on L (Definition 3.2). With probability at least $1 - 2^{-\omega(d)}$, the **Search** phase constructs a list consisting of all elements of $\mathcal{T}_{\mathrm{sol}}(R, R')$ and uses*

$$\sqrt{\max\{1, |\mathcal{T}_{\mathrm{sol}}(R,R')|\}} \left(\max\{1, \sqrt{|\mathcal{T}_0(R)|}\} + \sqrt{|\mathcal{T}_1(R)| \frac{|L|}{|C'|}} \right) 2^{o(d)}$$

time and QCRAM queries, $2^{o(d)}$ qubits, and $\max\{1, |\mathcal{T}_{\mathrm{sol}}(R,R')|\}\, 2^{o(d)}$ additional classical memory.

Proof. The **Search** phase of `3List` is exactly the algorithm $\mathcal{A}(\texttt{Samp}, t)$ in the proof of Lemma 2.4 with $X = \mathcal{T}_{\mathrm{sol}}(R, R')$ and `Samp` = `SolutionSearch`. If $\mathcal{T}_{\mathrm{sol}}(R, R') = \emptyset$, then the **Search** phase will finish after at most t repetitions of step *ii*, correctly yielding an empty set S. Otherwise, by Lemma 3.8, with probability $1 - 2^{-\omega(d)}$, there exists $\varepsilon \geq 2^{-o(d)}$ such that

$$\Pr[\texttt{SolutionSearch} \text{ outputs } (\mathbf{x}, \mathbf{y}, \mathbf{z})] \geq \varepsilon / |\mathcal{T}_{\mathrm{sol}}(R, R')|$$

for all $(\mathbf{x}, \mathbf{y}, \mathbf{z}) \in \mathcal{T}_{\mathrm{sol}}(R, R')$. Therefore, for all $t \geq \frac{1}{\varepsilon} \ln(|\mathcal{T}_{\mathrm{sol}}(R, R')|) + \omega(d)$, Lemma 2.4 implies that the **Search** phase outputs all elements of $\mathcal{T}_{\mathrm{sol}}(R, R')$ in time $t\,\mathsf{S}\,|\mathcal{T}_{\mathrm{sol}}(R, R')|^{1+o(d)}$, where S denotes the time complexity of `SolutionSearch`. Since $|\mathcal{T}_{\mathrm{sol}}(R, R')| \leq |L|^3 \leq 2^{O(d)}$, it suffices to take a sufficiently large $t = 2^{o(d)}$. The complexity of the **Search** phase then follows from Lemma 3.8. □

3.5 Final ingredients

This section presents a few lemmas that will be used to prove our main theorem, Theorem 3.1.

Lemma 3.10. *Let m be a positive integer and let $\alpha \in (0, \pi/2)$ satisfy $\alpha = \Omega(1)$ and $mp_\alpha = \omega(d)$. Let $C \subseteq \mathcal{S}^{d-1}$ be of size $2^{O(d)}$. With probability $1 - 2^{-\omega(d)}$ over $L \sim \mathcal{U}(\mathcal{S}^{d-1}, m)$, $|(R_L)^{\mathsf{T}}(\mathbf{c})| =_d mp_\alpha$ for all $\mathbf{c} \in C$, and thus $|R_L| =_d |C| mp_\alpha$, where $R_L \coloneqq \{(\mathbf{x}, \mathbf{c}) \in L \times C \colon \langle \mathbf{x}, \mathbf{c} \rangle \approx_\epsilon \cos(\alpha)\}$.*

Proof. Fix $\mathbf{c} \in C$. View $L \sim \mathcal{U}(\mathcal{S}^{d-1}, m)$ as an ordered sequence $(\mathbf{x}_1, \ldots, \mathbf{x}_m)$ of i.i.d. uniformly random unit vectors, and consider the sum $X = \sum_{i \in [m]} X_i$ of i.i.d. random variables $X_i \in \{0, 1\}$ defined by $X_i = 1$ if and only if $\langle \mathbf{x}_i, \mathbf{c} \rangle \approx_\epsilon \cos(\alpha)$. Note that $X = |(R_L)^{\mathsf{T}}(\mathbf{c})|$. By Lemma 2.9 and by assumption, $\mathbb{E}_L[X] =_d mp_\alpha = \omega(d)$, so Corollary 2.2 implies $|(R_L)^{\mathsf{T}}(\mathbf{c})| =_d mp_\alpha$ except with probability at most $\exp(-\omega(d))$. Hence, applying a union bound over all $2^{O(d)}$ vectors $\mathbf{c} \in C$ yields $|(R_L)^{\mathsf{T}}(\mathbf{c})| =_d mp_\alpha$ for all $\mathbf{c} \in C$, except with probability $|C| \exp(-\omega(d)) = \exp(-\omega(d))$. □

Lemma 3.11. *Let $m = 2^{O(d)}$ and let $\theta' \in (0, \pi/2)$ satisfy $\theta' = \Omega(1)$. With probability $1 - 2^{-\omega(d)}$ over $L \sim \mathcal{U}(\mathcal{S}^{d-1}, m)$, $|\{\mathbf{z} \in L \colon \langle \frac{\mathbf{x}-\mathbf{y}}{\|\mathbf{x}-\mathbf{y}\|}, \mathbf{z} \rangle \approx_\epsilon \cos(\theta')\}| \leq_d \max\{1, mp_{\theta'}\}$ for all $(\mathbf{x}, \mathbf{y}) \in L^2$.*

Proof. View L as an ordered sequence $(\mathbf{x}_1, \ldots, \mathbf{x}_m)$ and let $(i, j) \in [m]^2$. For all $k \in [m-2]$, define the random variable $X_k \in \{0, 1\}$ by $X_k = 1$ if and only if $\langle (\mathbf{x}_i - \mathbf{x}_j)/\|\mathbf{x}_i - \mathbf{x}_j\|, \mathbf{x}_k \rangle \approx_\epsilon \cos(\theta')$. If

[10] By "sufficiently large", we mean that there exists some $f(d) \leq 2^{o(d)}$, determined by the proof of Lemma 3.9, such that the lemma holds for all $t = 2^{o(d)}$ with $t \geq f(d)$.

$L \sim \mathcal{U}(\mathcal{S}^{d-1}, m)$, then the X_k are i.i.d. and $\mathbb{E}_L[\sum_{k\in[m-2]} X_k] =_d mp_{\theta'}$ by Lemma 2.9. Therefore, Corollary 2.2 implies $\sum_{k\in[m-2]} X_k \leq_d \max\{1, mp_{\theta'}\}$ except with probability at most $\exp(-\omega(d))$. The lemma statement now follows by applying a union bound over all $(i,j) \in [m]^2$, using that $m = 2^{O(d)}$ and that including $\mathbf{x}_i$ or $\mathbf{x}_j$ in the count does not affect the asymptotic upper bound. □

Lemma 3.12 (Size of $\mathcal{T}_1(R)$)**.** *Let $m = 2^{O(d)}$ and let $\theta, \alpha \in (0, \pi/2)$ satisfy $\min\{\theta, 2\alpha - \theta\} = \Omega(1)$ and $m\mathcal{W}(\theta, \alpha \mid \alpha) = \omega(d)$. Let $C \subseteq \mathcal{S}^{d-1}$ be of size $2^{O(d)}$, $R \coloneqq \{(\mathbf{x}, \mathbf{c}) \in \mathcal{S}^{d-1} \times C \colon \langle \mathbf{x}, \mathbf{c}\rangle \approx_\epsilon \cos(\alpha)\}$, and $R_L \coloneqq R|_{L\times C}$. With probability $1 - 2^{-\omega(d)}$ over $L \sim \mathcal{U}(\mathcal{S}^{d-1}, m)$, $|\mathcal{T}_1(R)| \leq_d |R_L| m\mathcal{W}(\theta, \alpha \mid \alpha)$, where $\mathcal{T}_1(R)$ is defined in Equation (4).*

Proof. Viewing L as an ordered sequence $(\mathbf{x}_1, \dots, \mathbf{x}_m)$ of unit vectors, let $i \in [m]$ and $\mathbf{c} \in R(\mathbf{x}_i)$. Note that $L \sim \mathcal{U}(\mathcal{S}^{d-1}, m)$ implies $L_{-i} \sim \mathcal{U}(\mathcal{S}^{d-1}, m-1)$, where $L_{-i} \coloneqq (\mathbf{x}_1, \dots, \mathbf{x}_{i-1}, \mathbf{x}_{i+1}, \dots, \mathbf{x}_m)$. Therefore, by Lemma 2.10, $\Pr_{\mathbf{y}\sim\mathcal{U}(L_{-i})}[\langle \mathbf{x}_i, \mathbf{y}\rangle \approx_\epsilon \cos(\theta), \langle \mathbf{y}, \mathbf{c}\rangle \approx_\epsilon \cos(\alpha)] =_d \mathcal{W}(\theta, \alpha \mid \alpha)$, giving

$$\mathbb{E}_{L_{-i}}[|\{\mathbf{y} \in L_{-i} \colon \langle \mathbf{x}_i, \mathbf{y}\rangle \approx_\epsilon \cos(\theta), \langle \mathbf{y}, \mathbf{c}\rangle \approx_\epsilon \cos(\alpha)\}|] =_d (m-1)\mathcal{W}(\theta, \alpha \mid \alpha) =_d m\mathcal{W}(\theta, \alpha \mid \alpha)$$

which is $\omega(d)$. Thus, $|\{\mathbf{y} \in L_{-i} \colon \langle \mathbf{x}_i, \mathbf{y}\rangle \approx_\epsilon \cos(\theta), \langle \mathbf{y}, \mathbf{c}\rangle \approx_\epsilon \cos(\alpha)\}| =_d m\mathcal{W}(\theta, \alpha \mid \alpha)$ except with probability $\exp(-\omega(d))$ by Corollary 2.2. Using a union bound over all $i \in [m]$ and $\mathbf{c} \in R(\mathbf{x}_i)$ yields $\sum_{i\in[m]} \sum_{\mathbf{c}\in R(\mathbf{x}_i)} |\{\mathbf{y} \in L_{-i} \colon \langle \mathbf{x}_i, \mathbf{y}\rangle \approx_\epsilon \cos(\theta), \langle \mathbf{y}, \mathbf{c}\rangle \approx_\epsilon \cos(\alpha)\}| \leq_d |R_L| m\mathcal{W}(\theta, \alpha \mid \alpha)$, except with probability $m|C| \exp(-\omega(d)) = \exp(-\omega(d))$, so the statement follows by definition of $\mathcal{T}_1(R)$. □

Lemma 3.13 (Size of $\mathcal{T}_{\mathrm{sol}}(L, \theta, \theta')$)**.** *Let $m = 2^{\Omega(d)}$ and let $\theta, \theta' \in (0, \pi/2)$ satisfy $\min\{\theta, \theta'\} = \Omega(1)$ and $\min\{m^2 p_\theta, m^2 p_{\theta'}, m^3 p_\theta p_{\theta'}\} = 2^{\Omega(d)}$. With probability $1 - 2^{-\Omega(d)}$ over $L \sim \mathcal{U}(\mathcal{S}^{d-1}, m)$, we have $|\{(\mathbf{x}, \mathbf{y}, \mathbf{z}) \in \mathcal{T}_{\mathrm{sol}}(L, \theta, \theta') \colon \mathbf{x}, \mathbf{y}, \mathbf{z} \text{ are distinct}\}| =_d m^3 p_\theta p_{\theta'}$, where $\mathcal{T}_{\mathrm{sol}}(L, \theta, \theta')$ is defined in Equation (2).*

Proof. We think of L as an ordered sequence $(\mathbf{x}_1, \dots, \mathbf{x}_m)$ of i.i.d. uniformly random unit vectors. Let I denote the set of all $m(m-1)(m-2)$ triples (i,j,k) of distinct indices from $[m]$. For each $(i,j,k) \in I$, let Z_{ijk} be the indicator random variable defined by $Z_{ijk} = 1$ if and only if both $\langle \mathbf{x}_i, \mathbf{x}_j\rangle \approx_\epsilon \cos(\theta)$ and $\langle \frac{\mathbf{x}_i - \mathbf{x}_j}{\|\mathbf{x}_i - \mathbf{x}_j\|}, \mathbf{x}_k\rangle \approx_\epsilon \cos(\theta')$. Then $Z \coloneqq \sum_{(i,j,k)\in I} Z_{ijk}$ counts the number of elements in $\mathcal{T}_{\mathrm{sol}}(L, \theta, \theta')$ that consist of distinct vectors. By linearity of expectation, $\mu \coloneqq \mathbb{E}[Z]$ satisfies $\mu = m(m-1)(m-2)pp'$, where

$$p \coloneqq \Pr[\langle \mathbf{x}, \mathbf{y}\rangle \approx_\epsilon \cos(\theta)] \quad \text{and} \quad p' \coloneqq \Pr[\langle \tfrac{\mathbf{x}-\mathbf{y}}{\|\mathbf{x}-\mathbf{y}\|}, \mathbf{z}\rangle \approx_\epsilon \cos(\theta')]$$

for independent samples $\mathbf{x}, \mathbf{y}, \mathbf{z} \sim \mathcal{U}(\mathcal{S}^{d-1})$. By Lemma 2.9, $p =_d p_\theta$ and $p' =_d p_{\theta'}$, so it suffices to show $Z =_d \mu$. We will now argue that the variance of Z is $\sigma^2 \leq \mu^2 2^{-\Omega(d)}$. Chebyshev's inequality (which gives $\Pr[|Z - \mu| \geq k\sigma] \leq 1/k^2$ for any real $k > 0$) then implies tight concentration: the probability that $|Z - \mu| \geq \mu/2$ is at most $4\sigma^2/\mu^2$, so $Z =_d \mu$ except with probability $2^{-\Omega(d)}$.

To upper bound $\sigma^2 = \mathbb{E}[Z^2] - \mu^2$, we note $\mathbb{E}[Z^2] = \sum_{(i,j,k)\in I} \sum_{(i',j',k')\in I} \mathbb{E}[Z_{ijk} Z_{i'j'k'}]$. Fix $(i,j,k) \in I$ and partition I into four disjoint sets I_0, I_1, I_2, I_3, where $I_\ell \coloneqq \{(i',j',k') \in I \colon |\{i,j,k\} \cap \{i',j',k'\}| = \ell\}$. By independence and by definition of the sets I_ℓ, we obtain

$$\sum_{(i',j',k')\in I_\ell} \Pr[Z_{i'j'k'} = 1 \mid Z_{ijk} = 1] \leq \begin{cases} (m-3)(m-4)(m-5)pp' & \text{if } \ell = 0 \\ 3(m-3)(m-4)pp' & \text{if } \ell = 1 \\ 3(m-3)\max\{p, p'\} & \text{if } \ell = 2 \\ 1 & \text{if } \ell = 3. \end{cases}$$

Since $\mu = m(m-1)(m-2)pp'$, we obtain $\mathbb{E}[Z^2] \le \mu^2 + O(\mu^2/\min\{m, m^2p, m^2p', \mu\})$. By assumption, $\min\{m, m^2p, m^2p', \mu\} =_d \min\{m, m^2p_\theta, m^2p_{\theta'}, m^3p_\theta p_{\theta'}\} = 2^{\Omega(d)}$, which implies $\sigma^2 \le \mu^2 2^{-\Omega(d)}$. As explained before, the lemma statement now follows from Chebyshev's inequality. □

Lemma 3.14. *Let $\theta, \alpha \in (0, \pi/2)$ satisfy $\min\{\theta, 2\alpha - \theta\} = \Omega(1)$. For all $\mathbf{x}, \mathbf{y} \in \mathcal{S}^{d-1}$ that satisfy $\langle \mathbf{x}, \mathbf{y} \rangle \approx_\epsilon \cos(\theta)$,*

$$\Pr_{C \sim \mathrm{RPC}(d, \log d, 1/p_\alpha)} \big[R(\mathbf{x}) \cap R(\mathbf{y}) \neq \emptyset \text{ and } |R(\mathbf{x})| \le 2^{d/\log d}\big] =_d \frac{\mathcal{W}(\alpha, \alpha \mid \theta)}{p_\alpha}$$

where $R \coloneqq \{(\mathbf{x}, \mathbf{c}) \in \mathcal{S}^{d-1} \times C \colon \langle \mathbf{x}, \mathbf{c} \rangle \approx_\epsilon \cos(\alpha)\}$.

Proof. Define $b \coloneqq \log d$. The upper bound follows from Lemma 2.10, because $\langle \mathbf{x}, \mathbf{y} \rangle \approx_\epsilon \cos(\theta)$. Since $\Pr_C\big[|R(\mathbf{x})| \in [1, 2^{d/\log d}]\big] \ge_d 1$ by Lemma 2.13 (applied with $\theta = 0$) and a careful application of Markov's inequality, we obtain

$$\begin{aligned}\Pr_C\big[R(\mathbf{x}) \cap R(\mathbf{y}) \neq \emptyset \text{ and } |R(\mathbf{x})| \le 2^{d/\log d}\big] &= \Pr_C\big[R(\mathbf{x}) \cap R(\mathbf{y}) \neq \emptyset \text{ and } |R(\mathbf{x})| \in [1, 2^{d/\log d}]\big] \\ &\ge_d \Pr_C\big[R(\mathbf{x}) \cap R(\mathbf{y}) \neq \emptyset \,\big|\, |R(\mathbf{x})| \in [1, 2^{d/\log d}]\big].\end{aligned}$$

Therefore, it remains to lower bound the latter, i.e., the probability of the event E that there is a center point $\mathbf{c} \in C$ that lands in the region

$$\widetilde{\mathcal{W}}_{\mathbf{x},\alpha,\mathbf{y},\alpha} = \{\mathbf{s} \in \mathcal{S}^{d-1} \colon \langle \mathbf{x}, \mathbf{s} \rangle \approx_\epsilon \cos(\alpha), \langle \mathbf{y}, \mathbf{s} \rangle \approx_\epsilon \cos(\alpha)\},$$

conditioned on the event E' that the number of center points $\mathbf{c} \in C$ that lie in the "approximate" spherical cap $\widetilde{\mathcal{H}}_{\mathbf{x},\alpha} = \{\mathbf{s} \in \mathcal{S}^{d-1} \colon \langle \mathbf{x}, \mathbf{s} \rangle \approx_\epsilon \cos(\alpha)\} \supseteq \widetilde{\mathcal{W}}_{\mathbf{x},\alpha,\mathbf{y},\alpha}$ is between 1 and $2^{d/\log d}$. Thus, suppose event E' holds. Then event E holds if one of those center points $\mathbf{c}^*$ that lie in $\widetilde{\mathcal{H}}_{\mathbf{x},\alpha}$ (let's say $\mathbf{c}^*$ is the first such center point, assuming without loss of generality some ordering on C) also satisfies $\langle \mathbf{y}, \mathbf{c}^* \rangle \approx_\epsilon \cos(\alpha)$, i.e., $\mathbf{c}^* \in \widetilde{\mathcal{W}}_{\mathbf{x},\alpha,\mathbf{y},\alpha}$.

We recall that the distribution $C \sim \mathrm{RPC}(d, b, 1/p_\alpha)$ is defined as taking $C = \mathbf{Q}(C^{(1)} \times \cdots \times C^{(b)})$, where $\mathbf{Q} \sim \mathcal{U}(\mathrm{SO}(d))$ and $C^{(i)} \sim \mathcal{U}(\frac{1}{\sqrt{b}}\mathcal{S}^{d/b-1}, (1/p_\alpha)^{1/b})$ for $i \in \{1, \ldots, b\}$. Therefore, sampling $C \sim \mathrm{RPC}(d, b, 1/p_\alpha)$ conditional on $|R(\mathbf{x})| \in [1, 2^{d/\log d}]$ is equivalent to sampling C as follows:

1. Sample $\mathbf{c}^* \coloneqq \mathbf{Q}(\mathbf{v}_1^*, \ldots, \mathbf{v}_b^*)$, by first sampling $\mathbf{Q} \sim \mathcal{U}(\mathrm{SO}(d))$ and then sampling the tuple $(\mathbf{v}_1^*, \ldots, \mathbf{v}_b^*) \sim \mathcal{U}(\frac{1}{\sqrt{b}}\mathcal{S}^{d/b-1}) \times \cdots \times \mathcal{U}(\frac{1}{\sqrt{b}}\mathcal{S}^{d/b-1})$ conditional on $\langle \mathbf{x}, \mathbf{c}^* \rangle \approx_\epsilon \cos(\alpha)$.
2. Sample $\widetilde{C}^{(i)} \sim \mathcal{U}(\frac{1}{\sqrt{b}}\mathcal{S}^{d/b-1}, (1/p_\alpha)^{1/b} - 1)$ for all $i \in \{1, \ldots, b\}$, conditional on $|C \cap \widetilde{\mathcal{H}}_{\mathbf{x},\alpha}| \in [1, 2^{d/\log d}]$ where $C \coloneqq \mathbf{Q}(C^{(1)} \times \cdots \times C^{(b)})$ for $C^{(i)} \coloneqq \{\mathbf{v}_i^*\} \cup \widetilde{C}^{(i)}$.

The second step does not influence the probability that the vector $\mathbf{c}^*$ sampled in the first step lands in $\widetilde{\mathcal{W}}_{\mathbf{x},\alpha,\mathbf{y},\alpha}$. Moreover, the distribution of the vector $\mathbf{c}^*$ sampled in the first step is the same as sampling $\mathbf{c}^* \sim \mathcal{U}(\mathcal{S}^{d-1})$ conditional on $\langle \mathbf{x}, \mathbf{c}^* \rangle \approx_\epsilon \cos(\alpha)$. In other words, we obtain

$$\begin{aligned}\Pr_C\big[R(\mathbf{x}) \cap R(\mathbf{y}) \neq \emptyset \mid |R(\mathbf{x})| \in [1, 2^{d/\log d}]\big] &\ge \Pr_C\big[\langle \mathbf{y}, \mathbf{c}^* \rangle \approx_\epsilon \cos(\alpha) \,\big|\, |R(\mathbf{x})| \in [1, 2^{d/\log d}]\big] \\ &= \Pr_{\mathbf{c}^* \sim \mathcal{U}(\mathcal{S}^{d-1})}\big[\langle \mathbf{y}, \mathbf{c}^* \rangle \approx_\epsilon \cos(\alpha) \,\big|\, \langle \mathbf{x}, \mathbf{c}^* \rangle \approx_\epsilon \cos(\alpha)\big] \\ &=_d \frac{\mathcal{W}(\alpha, \alpha \mid \theta)}{p_\alpha}.\end{aligned}$$

by Lemma 2.9 and Lemma 2.10 (again, using $\langle \mathbf{x}, \mathbf{y} \rangle \approx_\epsilon \cos(\theta)$). □

3.6 Final analysis of our quantum algorithm

We now show that `3List` (i.e., Algorithm 1) solves Problem 1 for a random list $L \subseteq \mathcal{S}^{d-1}$ of size $m = (27/16)^{d/4+o(d)}$ in complexity $2^{0.284551d+o(d)}$, thereby proving Theorem 3.1. First, we formalize the claim we made in the introduction of Section 3 regarding $\mathcal{T}_{\text{sol}}(L,\theta,\theta')$.

Lemma 3.15. *Let $L \subseteq \mathcal{S}^{d-1}$ and let $\theta,\theta' \in (0,\pi/2)$ satisfy $\cos(\theta) = \frac{1}{3}$ and $\cos(\theta') = \epsilon + \sqrt{\frac{1}{3}+\frac{\epsilon}{2}}$. Then $\|\mathbf{x}-\mathbf{y}-\mathbf{z}\| \leq 1$ for all $(\mathbf{x},\mathbf{y},\mathbf{z}) \in \mathcal{T}_{\text{sol}}(L,\theta,\theta')$, where $\mathcal{T}_{\text{sol}}(L,\theta,\theta')$ is defined in Equation (2). Moreover, there exists $m' = (27/16)^{d/4+o(d)}$ such that for all $m \geq m'$:*

(i) $\mathbb{E}_{L\sim\mathcal{U}(\mathcal{S}^{d-1},m)}[|\mathcal{T}_{\text{sol}}(L,\theta,\theta')|] \geq m$.

(ii) With probability $1-2^{-\Omega(d)}$ over $L \sim \mathcal{U}(\mathcal{S}^{d-1},m)$, $m \leq |\mathcal{T}_{\text{sol}}(L,\theta,\theta')| \leq_d m^3 p_\theta p_{\theta'}$ and every $(\mathbf{x},\mathbf{y},\mathbf{z}) \in \mathcal{T}_{\text{sol}}(L,\theta,\theta')$ satisfies $\mathbf{x}\neq\mathbf{y}$, $\mathbf{y}\neq\mathbf{z}$, and $\mathbf{z}\neq\mathbf{x}$.

Proof. For all $\mathbf{x},\mathbf{y},\mathbf{z} \in \mathcal{S}^{d-1}$, we have $\|\mathbf{x}-\mathbf{y}-\mathbf{z}\|^2 \leq 1$ if and only if $\langle \frac{\mathbf{x}-\mathbf{y}}{\|\mathbf{x}-\mathbf{y}\|}, \mathbf{z}\rangle \geq \sqrt{(1-\langle\mathbf{x},\mathbf{y}\rangle)/2}$. Consider arbitrary $\theta \in (0,\pi/2)$, and define θ' by $\cos(\theta') = \epsilon + \sqrt{(1-\cos(\theta)+\epsilon)/2}$. Then all $(\mathbf{x},\mathbf{y},\mathbf{z}) \in \mathcal{T}_{\text{sol}}(L,\theta,\theta')$ satisfy

$$\sqrt{\frac{1-\langle\mathbf{x},\mathbf{y}\rangle}{2}} \leq \sqrt{\frac{1-\cos(\theta)+\epsilon}{2}} = \cos(\theta') - \epsilon \leq \langle \frac{\mathbf{x}-\mathbf{y}}{\|\mathbf{x}-\mathbf{y}\|}, \mathbf{z}\rangle$$

and thus $\|\mathbf{x}-\mathbf{y}-\mathbf{z}\| \leq 1$. We will now show that the lemma statement holds for $\cos(\theta) = 1/3$ (and thus $\cos(\theta') = \epsilon + \sqrt{1/3+\epsilon/2}$).

Note that $p_{\theta'} = (1-\cos^2(\theta'))^{d/2} =_d (1-(\frac{1-\cos(\theta)}{2}))^{d/2} = (\frac{1+\cos(\theta)}{2})^{d/2} = \cos(\frac{\theta}{2})^d$ by the fixed choice of $\epsilon = 1/(\log d)^2$. Therefore, if $\theta = \Omega(1)$ and $\theta' = \Omega(1)$, we obtain (for $L \sim \mathcal{U}(\mathcal{S}^{d-1},m)$)

$$\begin{aligned}\mathbb{E}_L\big[|\mathcal{T}_{\text{sol}}(L,\theta,\theta')|\big] &= \sum_{(\mathbf{x},\mathbf{y},\mathbf{z})\in L^3} \Pr_L\big[\langle\mathbf{x},\mathbf{y}\rangle \approx_\epsilon \cos(\theta)\big] \cdot \Pr_L\big[\langle\tfrac{\mathbf{x}-\mathbf{y}}{\|\mathbf{x}-\mathbf{y}\|},\mathbf{z}\rangle \approx_\epsilon \cos(\theta') \,\big|\, \langle\mathbf{x},\mathbf{y}\rangle \approx_\epsilon \cos(\theta)\big] \\ &=_d m^3 p_\theta p_{\theta'} \\ &=_d m^3 (\sin(\theta)\cos(\tfrac{\theta}{2}))^d\end{aligned}$$

by Lemma 2.9. In particular, there exists $m' = (\sin(\theta)\cos(\frac{\theta}{2}))^{-d/2}2^{o(d)}$ such that, whenever the list size $|L| = m$ is $\geq m'$, the expected size of $\mathcal{T}_{\text{sol}}(L,\theta,\theta')$ is at least m. As $(\sin(\theta)\cos(\frac{\theta}{2}))^{-d/2}$ is minimized for $\cos(\theta) = 1/3$, we choose $\cos(\theta) = 1/3$ and choose θ' accordingly. The corresponding m' then satisfies $m' = (27/16)^{d/4+o(d)}$.

It remains to prove part (ii). For all sufficiently large d, with probability 1 over $L \sim \mathcal{U}(\mathcal{S}^{d-1},m)$, every $(\mathbf{x},\mathbf{y},\mathbf{z}) \in \mathcal{T}_{\text{sol}}(L,\theta,\theta')$ satisfies $\mathbf{x}\neq\mathbf{y}$, $\mathbf{y}\neq\mathbf{z}$, and $\mathbf{z}\neq\mathbf{x}$ (recall Remark 3). Indeed, consider $(\mathbf{x},\mathbf{y},\mathbf{z}) \in \mathcal{T}_{\text{sol}}(L,\theta,\theta')$, which implies $\|\mathbf{x}-\mathbf{y}-\mathbf{z}\| \leq 1$. By definition, $\langle\mathbf{x},\mathbf{y}\rangle \approx_\epsilon \cos(\theta)$, so $\mathbf{x}\neq\mathbf{y}$ for sufficiently large d. Next, $\mathbf{y}\neq\mathbf{z}$, because otherwise $\|\mathbf{x}-2\mathbf{y}\| > 1$, a contradiction. Finally, if $\mathbf{x}\neq\mathbf{y}$ while $\mathbf{x}=\mathbf{z}$, then $\langle\mathbf{x},\mathbf{y}\rangle = \cos(\theta)-\epsilon$ (by definition of $\mathcal{T}_{\text{sol}}(L,\theta,\theta')$ and θ'), which occurs with measure 0 for independent $\mathbf{x},\mathbf{y} \sim \mathcal{U}(\mathcal{S}^{d-1})$. Therefore, by choosing $m' = (27/16)^{d/4+o(d)}$ sufficiently large, part (ii) follows from Lemma 3.13 (note that θ and θ' satisfy the constraints). □

The following lemma implies that if $|\mathcal{T}_{\text{sol}}(L,\theta,\theta')| \geq m$ and the parameters (α,α',ℓ,t) of `3List` are chosen appropriately, then, with high probability, `3List` outputs m distinct elements of $\mathcal{T}_{\text{sol}}(L,\theta,\theta')$.

To prove Theorem 3.1, it then remains to show that, for $\theta, \theta' \in (0, \pi/2)$ as given in Lemma 3.15, there exists a suitable choice of (α, α') such that the complexity is as desired.

Lemma 3.16. *Let $m = 2^{\Theta(d)}$ and let $\theta, \theta', \alpha, \alpha' \in (0, \pi/2)$ satisfy $\min\{\theta, \theta', 2\alpha - \theta, 2\alpha' - \theta'\} = \Omega(1)$, $mp_{\theta'} \leq 2^{o(d)}$, and $\min\{mp_\alpha, mp_{\alpha'}, m\,\mathcal{W}(\theta, \alpha \mid \alpha), m^2\,\mathcal{W}(\theta', \alpha' \mid \alpha')\} = \omega(d)$. Let $\ell = \frac{p_\alpha}{\mathcal{W}(\alpha,\alpha|\theta)} \frac{p_{\alpha'}}{\mathcal{W}(\alpha',\alpha'|\theta')} 2^{o(d)}$ and $t = 2^{o(d)}$ be sufficiently large.*[11] *With probability $1 - 2^{-\Omega(d)}$ over $L \sim \mathcal{U}(\mathcal{S}^{d-1}, m)$ and the internal randomness of the algorithm, $\texttt{3List}(L, \theta, \theta')$ (Algorithm 1) with parameters $(\alpha, \alpha', \ell, t)$ outputs a list containing m elements of $\mathcal{T}_{\mathrm{sol}}(L, \theta, \theta')$, if m elements exist, in time*

$$\ell \left(m + \sqrt{\frac{|\mathcal{T}_{\mathrm{sol}}(L, \theta, \theta')|}{\ell}} \left(\sqrt{m^2 p_\alpha} + \sqrt{m^3 \mathcal{W}(\theta, \alpha \mid \alpha) p_{\alpha'}} \right) \right) 2^{o(d)} \tag{5}$$

using $\max\{m, |\mathcal{T}_{\mathrm{sol}}(L, \theta, \theta')|\}\, 2^{o(d)}$ classical memory and QCRAM bits, and $2^{o(d)}$ qubits.

Proof. On input (L, θ, θ') and with parameters $(\alpha, \alpha', \ell, t)$, `3List` (Algorithm 1) samples ℓ RPC pairs, independently. For each sampled RPC pair (C, C'), it obtains corresponding data structures (D, D') from $\texttt{Preprocess}(L, C, C')$ (Algorithm 2), and repeatedly applies $\texttt{SolutionSearch}(D, D')$ (Algorithm 5). In the remainder of this proof, we write $\mathcal{T}_{\mathrm{sol}}$ as shorthand for $\mathcal{T}_{\mathrm{sol}}(L, \theta, \theta')$ and, for each $j \in [\ell]$, we write (R_j, R'_j) for the relations corresponding to the j-th sampled RPC pair.

We claim that, for all sufficiently large $\ell = \frac{p_\alpha}{\mathcal{W}(\alpha,\alpha|\theta)} \frac{p_{\alpha'}}{\mathcal{W}(\alpha',\alpha'|\theta')} 2^{o(d)}$, the following statements simultaneously hold, except with probability $2^{-\omega(d)}$ over L and the ℓ sampled RPC pairs:

(I) For all $j \in [\ell]$, (R_j, R'_j) is well-balanced on L (Definition 3.2).

(II) For all $j \in [\ell]$, $|\mathcal{T}_1(R_j)| \leq_d m^2 \mathcal{W}(\theta, \alpha \mid \alpha)$.

(III) For all $(\mathbf{x}, \mathbf{y}, \mathbf{z}) \in \mathcal{T}_{\mathrm{sol}}$, $|\{j \in [\ell] \colon (\mathbf{x}, \mathbf{y}, \mathbf{z}) \in \mathcal{T}_{\mathrm{sol}}(R_j, R'_j)\}| \in [1, 2^{o(d)}]$.

By Lemma 3.9, (I) implies that for sufficiently large $t = 2^{o(d)}$, for all $j \in [\ell]$, the j-th repetition of the outer loop of `3List` finds all elements of the corresponding set $\mathcal{T}_{\mathrm{sol}}(R_j, R'_j)$. Since (III) implies $\bigcup_{j=1}^{\ell} \mathcal{T}_{\mathrm{sol}}(R_j, R'_j) = \mathcal{T}_{\mathrm{sol}}$, the output of `3List` contains all elements of $\mathcal{T}_{\mathrm{sol}}$.

Next, an upper bound on the time complexity of `3List` is given by

$$T_{\texttt{3List}} = \sum_{j=1}^{\ell} (T_{\texttt{Sample}}(j) + T_{\texttt{Preprocess}}(j) + T_{\texttt{Search}}(j))$$

where $T_{\texttt{Sample}}(j), T_{\texttt{Preprocess}}(j), T_{\texttt{Search}}(j)$ are upper bounds on the time complexity of the respective phases in the j-th repetition of the outer loop of `3List`. It is immediate that $T_{\texttt{Sample}}(j) = 2^{o(d)}$ for all $j \in [\ell]$, and the **Sampling** phases together use at most $2^{o(d)}$ classical memory (as the memory can be reinitialized in each repetition). By Lemma 3.3 and (I), $T_{\texttt{Preprocess}}(j) = m2^{o(d)}$ for all $j \in [\ell]$, and the **Preprocessing** phases together use at most $m2^{o(d)}$ classical memory and QCRAM bits (as

[11] Similarly to Lemma 3.9, "sufficiently large" means that there exists some $f(d) \leq 2^{o(d)}$ such that Lemma 3.16 holds for all $\ell = \frac{p_\alpha}{\mathcal{W}(\alpha,\alpha|\theta)} \frac{p_{\alpha'}}{\mathcal{W}(\alpha',\alpha'|\theta')} 2^{o(d)}$ and $t = 2^{o(d)}$ with $\ell \geq \frac{p_\alpha}{\mathcal{W}(\alpha,\alpha|\theta)} \frac{p_{\alpha'}}{\mathcal{W}(\alpha',\alpha'|\theta')} f(d)$ and $t \geq f(d)$.

the memory can be reinitialized in each repetition). By Lemma 3.9, (I) and (II) imply that, with probability at least $1 - 2^{-\omega(d)}$,

$$T_{\texttt{Search}}(j) = \sqrt{\max\{1, |\mathcal{T}_{\mathrm{sol}}(R_j, R'_j)|\}} \left(\sqrt{m^2 p_\alpha} + \sqrt{m^3 \mathcal{W}(\theta, \alpha \mid \alpha) p_{\alpha'}}\right) 2^{o(d)}$$

time and QCRAM queries, $2^{o(d)}$ qubits, and $|\mathcal{T}_{\mathrm{sol}}(R_j, R'_j)| 2^{o(d)}$ additional classical memory. Here, we use that (I) implies $|\mathcal{T}_0(R_j)| =_d |L|^2/|C| =_d m^2 p_\alpha$, which is $\omega(d)$ by assumption. By the Cauchy-Schwarz inequality and (III), we obtain

$$\sum_{j=1}^{\ell} \sqrt{\max\{1, |\mathcal{T}_{\mathrm{sol}}(R_j, R'_j)|\}} \le \sqrt{\ell \sum_{j=1}^{\ell} \max\{1, |\mathcal{T}_{\mathrm{sol}}(R_j, R'_j)|\}} \le_d \sqrt{\ell \max\{\ell, |\mathcal{T}_{\mathrm{sol}}|\}}.$$

By assumption, $\min\{m\mathcal{W}(\theta, \alpha \mid \alpha), m^2 \mathcal{W}(\theta', \alpha' \mid \alpha')\} = \omega(d)$, so $m^3 \mathcal{W}(\theta, \alpha \mid \alpha)\mathcal{W}(\theta', \alpha' \mid \alpha') = \omega(d)$. Thus, $\ell \le_d m^3 p_\theta p_{\theta'}$ and $\min\{m^2 p_\theta, m^2 p_{\theta'}, m^3 p_\theta p_{\theta'}\} = 2^{\Omega(d)}$, because $p_\theta = \mathcal{W}(\theta, \alpha \mid \alpha)\, 2^{\Omega(d)}$ and $p_{\theta'} = \mathcal{W}(\theta', \alpha' \mid \alpha')\, 2^{\Omega(d)}$. Therefore, with probability at least $1 - 2^{-\Omega(d)}$, $|\mathcal{T}_{\mathrm{sol}}| \ge_d m^3 p_\theta p_{\theta'}$ by Lemma 3.13, which implies $\sqrt{\ell \max\{\ell, |\mathcal{T}_{\mathrm{sol}}|\}} =_d \sqrt{\ell |\mathcal{T}_{\mathrm{sol}}|}$. In particular, we obtain

$$T_{\texttt{3List}} \le_d \ell m + \sqrt{\ell |\mathcal{T}_{\mathrm{sol}}|} \left(\sqrt{m^2 p_\alpha} + \sqrt{m^3 \mathcal{W}(\theta, \alpha \mid \alpha) p_{\alpha'}}\right)$$

from which the lemma follows (as the total amount of classical memory used is $\max\{m, |\mathcal{T}_{\mathrm{sol}}|\}\, 2^{o(d)}$).

It remains to prove the conditions (I), (II), and (III) claimed above. By Lemma 3.10, Lemma 3.11, and a union bound over all $j \in [\ell]$, (I) holds except with probability $2^{-\omega(d)}$ over L. Here, we use that the RPCs are of size $1/p_\alpha$ and $1/p_{\alpha'}$ (resp.), that $m \min\{p_\alpha, p_{\alpha'}\} = \omega(d)$, and that $m p_{\theta'} \le_d 1$. Moreover, by applying Lemma 3.12 (using $m\mathcal{W}(\theta, \alpha \mid \alpha) = \omega(d)$) together with a union bound over the $\ell = 2^{O(d)}$ sampled RPC pairs, we obtain that (II) follows from (I), except with probability $2^{-\omega(d)}$. Finally, to prove (III), let $(\mathbf{x}, \mathbf{y}, \mathbf{z}) \in \mathcal{T}_{\mathrm{sol}}$ and define $p_{(\mathbf{x},\mathbf{y},\mathbf{z})} \coloneqq \Pr_{(C,C')}[(\mathbf{x}, \mathbf{y}, \mathbf{z}) \in \mathcal{T}_{\mathrm{sol}}(R, R')]$, where the distribution is over $C \sim \mathrm{RPC}(d, \log d, 1/p_\alpha)$ and $C' \sim \mathrm{RPC}(d, \log d, 1/p_{\alpha'})$. By Lemma 3.14,

$$\begin{aligned} p_{(\mathbf{x},\mathbf{y},\mathbf{z})} &= \Pr_{C}\left[|R(\mathbf{x})| \le 2^{d/\log d}, R(\mathbf{x}) \cap R(\mathbf{y}) \ne \emptyset\right] \cdot \Pr_{C'}\left[|R'(\tfrac{\mathbf{x}-\mathbf{y}}{\|\mathbf{x}-\mathbf{y}\|})| \le 2^{d/\log d}, R'(\tfrac{\mathbf{x}-\mathbf{y}}{\|\mathbf{x}-\mathbf{y}\|}) \cap R'(\mathbf{z}) \ne \emptyset\right] \\ &=_d \frac{\mathcal{W}(\alpha, \alpha \mid \theta)}{p_\alpha} \frac{\mathcal{W}(\alpha', \alpha' \mid \theta')}{p_{\alpha'}} \end{aligned}$$

where the first equality uses that C and C' are independent. Consequently, for sufficiently large $\ell = \frac{p_\alpha}{\mathcal{W}(\alpha,\alpha|\theta)} \frac{p_{\alpha'}}{\mathcal{W}(\alpha',\alpha'|\theta')} 2^{o(d)}$, we obtain $\mu \coloneqq \mathbb{E}[|\{j \in [\ell] \colon (\mathbf{x}, \mathbf{y}, \mathbf{z}) \in \mathcal{T}_{\mathrm{sol}}(R_j, R'_j)\}|] = \omega(d)$ and $\mu \le 2^{o(d)}$, where the expectation is taken over the ℓ RPC pairs that are (independently) sampled during `3List`. Corollary 2.2 then implies $|\{j \in [\ell] \colon (\mathbf{x}, \mathbf{y}, \mathbf{z}) \in \mathcal{T}_{\mathrm{sol}}(R_j, R'_j)\}| \in [1, 2^{o(d)}]$, except with probability $2^{-\omega(d)}$. By applying a union bound over all (at most $2^{O(d)}$) $(\mathbf{x}, \mathbf{y}, \mathbf{z}) \in \mathcal{T}_{\mathrm{sol}}$, we conclude that (III) holds with probability $1 - 2^{-\omega(d)}$. □

We complete our analysis by proving Theorem 3.1.

Proof of Theorem 3.1. Consider an instance $L \sim \mathcal{U}(\mathcal{S}^{d-1}, m)$ of Problem 1 for sufficiently large $m = (27/16)^{d/4+o(d)}$, and let θ, θ' be according to Lemma 3.15. By Lemma 3.15, with probability $1 - 2^{-\Omega(d)}$, $\mathcal{T}_{\mathrm{sol}}(L, \theta, \theta')$ consists only of 3-tuple solutions and satisfies $m \le |\mathcal{T}_{\mathrm{sol}}(L, \theta, \theta')| \le_d m^3 p_\theta p_{\theta'} =_d m$. Therefore, it suffices to show there exists a quantum algorithm that solves Problem 1 with the claimed time and memory complexity if $m \le |\mathcal{T}_{\mathrm{sol}}(L, \theta, \theta')| \le_d m$, except with probability $2^{-\Omega(d)}$.

We remark that $\cos(\theta')$ is so close to $1/\sqrt{3}$ (when $d \to \infty$) that we may without loss of generality assume they are equal (as it only affects $p_{\theta'}$ and $\mathcal{W}(\alpha', \alpha' \mid \theta')$ by subexponential factors, for similar reasons as in the proofs of Lemma 2.9 and Lemma 2.10). Suppose $\alpha, \alpha' \in (0, \pi/2)$ are constants that satisfy:

(I) $\min\{\theta, \theta', 2\alpha - \theta, 2\alpha' - \theta'\} = \Omega(1)$;

(II) $\min\{mp_\alpha, mp_{\alpha'}, m\mathcal{W}(\theta, \alpha \mid \alpha), m^2\mathcal{W}(\theta', \alpha' \mid \alpha')\} = \omega(d)$.

Then, by Lemma 3.16 (using that $mp_{\theta'} \leq 2^{o(d)}$, $m^3 p_\theta p_{\theta'} =_d m$, and that $\theta \geq \theta' = \Omega(1)$), there is a quantum algorithm that, with probability $1 - 2^{-\Omega(d)}$ over the randomness of L and the internal randomness of the algorithm, finds m elements of $\mathcal{T}_{\text{sol}}(L, \theta, \theta')$ (if they exist) with time complexity given by Equation (5) for some $\ell = \frac{p_\alpha}{\mathcal{W}(\alpha, \alpha|\theta)} \frac{p_{\alpha'}}{\mathcal{W}(\alpha', \alpha'|\theta')} 2^{o(d)}$, using $\max\{m, |\mathcal{T}_{\text{sol}}(L, \theta, \theta')|\}\, 2^{o(d)}$ classical memory and QCRAM bits, and $2^{o(d)}$ qubits.

To conclude the proof (with $\cos(\theta) = 1/3$ and $\cos(\theta') = 1/\sqrt{3}$), we show that setting $\alpha, \alpha' \in (0, \pi/2)$ such that $\cos(\alpha) = 0.347606$ and $\cos(\alpha') = 0.427124$ implies that (I) and (II) are satisfied, and results in the desired time complexity. It is easy to verify that condition (I) follows from the choice of $\theta, \theta', \alpha, \alpha'$. Moreover, this particular choice of (α, α') implies that condition (II) is satisfied and that Equation (5) is at most $2^{0.284551d + o(d)}$, as can be verified using the script available at `https://github.com/lynnengelberts/3List`. □

4 Application to lattice problems

In this section, we detail how our quantum algorithm for Problem 1 aids in finding short vectors in a lattice, and can be turned into a quantum algorithm for solving approximate SVP with small approximation factor.

4.1 Lattices and the shortest vector problem

A matrix $\mathbf{B} \in \mathbb{R}^{d \times d}$ with linearly independent columns generates a lattice Λ defined as the set of all integer linear combinations of the columns of $\mathbf{B}$. Such a matrix $\mathbf{B}$ is called a *basis* of Λ, and we remark that (for $d > 1$) such a basis is not unique: by combining elements we can form infinitely many bases for the same lattice. Every lattice has at least one shortest nonzero element (with respect to the Euclidean norm), and we denote its length by $\lambda_1 > 0$. This gives rise to the Shortest Vector Problem (SVP) that we mentioned in the introduction.

Problem 3 (Shortest Vector Problem)**.** Given a basis of a lattice $\Lambda \subseteq \mathbb{R}^d$, find a lattice vector of Euclidean norm λ_1.

SVP is known to be NP-hard under randomized reductions [Emd81, Ajt96]. For cryptanalytic purposes, it would suffice to find a nonzero lattice vector of norm $\leq \gamma\lambda_1$ for some reasonably small approximation factor $\gamma \geq 1$ (for fixed γ, this variant of approximate SVP is often called γ-SVP), since the security of lattice-based cryptosystems is based on the hardness of (a decision variant of) this problem. In particular, many algorithms for attacking lattice-based cryptoschemes — such as the BKZ algorithm [Sch87, SE94] — require a subroutine for solving approximate SVP.

4.2 Sieving algorithms for SVP and the uniform heuristic

This paper is about sieving algorithms, which are an important class of algorithms, including the fastest known classical algorithm [ADRS15] for (provably) solving SVP. First developed by Ajtai, Kumar, and Sivakumar [AKS01] and then improved by a series of subsequent works [Reg04, NV08, PS09, MV10, ADRS15, AS18], the strategy is to begin by generating numerous lattice vectors, and then iteratively combine and "sieve" them to create shorter and shorter vectors.

To prove the correctness of the runtime of sieving algorithms, existing methods add random perturbations to the lattice vectors (to make them continuous) or carefully control the distribution of the lattice points that are formed in each iteration [AKS01, MV10, HPS11, ADRS15, AS18]. However, these approaches incur extra time and are often the major contributors to the runtime of these — provably correct — algorithms. Nguyen and Vidick [NV08, Section 4] therefore suggested a heuristic assumption to simplify the analysis for a subclass of those algorithms, which we will refer to as *heuristic sieving algorithms*.[12] Roughly speaking, this heuristic assumes that after each iteration of the sieving algorithm, the obtained lattice points are uniformly and independently distributed within some thin annulus.

Heuristic 1 (Uniform heuristic)**.** *Let $L \subseteq \mathbb{R}^d$ be a list of lattice vectors obtained after some iteration of a heuristic sieving algorithm. After appropriate rescaling, every element of L behaves as if it is an i.i.d. uniform sample from $\mathsf{ann}_\rho = \{\mathbf{x} \in \mathbb{R}^d : \rho \leq \|\mathbf{x}\| \leq 1\}$ for some fixed $\rho < 1$.*

This heuristic is typically considered for $\rho \to 1$ (that is, the list vectors are assumed to be essentially i.i.d. uniform on $\mathcal{S}^{d-1}$). Note that this heuristic is technically incorrect, since the output list could include vectors of the form $\mathbf{x} + \mathbf{y}$ and $\mathbf{x} + \mathbf{z}$, which are correlated to each other (and hence not independent). However, it has been verified by several experimental studies that it does appear to capture the empirical behavior of sieving, and under the uniform heuristic, one can end up with more efficient algorithms for solving SVP (both classical and quantum ones) than are known in the provable setting. (E.g., see [NV08, MV10, BLS16, ADH+19].)

Assuming the uniform heuristic, (heuristic) 2-*tuple sieving* algorithms have the following high-level form (as already described in the introduction). The algorithm first generates a large number of lattice vectors of norm roughly R (for large R), and then tries to find pairs of vectors whose sum or difference yields a vector of smaller length. This process is repeated until the algorithm has found a sufficiently short vector to solve (approximate) SVP. If we instruct the algorithm in a way that each iteration, given lattice vectors of norm ≤ 1 (recall that this norm bound is without loss of generality, by scaling the lattice), constructs lattice vectors of norm $\leq 1 - 1/\tau$, then there is a choice of $\tau = \mathrm{poly}(d)$ such that τ^2 (which is polynomial in d) sieving iterations suffice to end up with a bunch of sufficiently short lattice vectors.[13] If each iteration runs in time T, then the uniform heuristic (Heuristic 1) would ensure we solve SVP in time $\mathrm{poly}(d)\mathsf{T}$. Hence, we typically care about analyzing the time T and the amount of memory used in each iteration. Note that if we start with

[12] The use of heuristics is common in cryptology, since cryptographers are primarily concerned with the practical solvability of problem and with the concrete runtimes of algorithms. Indeed, the practical performance of algorithms directly relates to the security of the cryptosystem, and helps determine appropriate security parameters. When a heuristic assumption holds in the sense that the observed runtime of an algorithm closely matches the asymptotic prediction obtained by assuming the heuristic, then this motivates analyzing the best possible runtime under this heuristic.

[13] Although the heuristic may fail when vectors become very short, it is typically assumed in this case that the (approximate) SVP instance has already been solved.

too few vectors, then we may end up with not enough vectors for the next iteration. However, the uniform heuristic allows us to calculate how many vectors are needed in the initial list: for 2-tuple sieving, $(4/3)^{d/2+o(d)} \approx 2^{0.2075d}$ vectors suffice (see [NV08] for a detailed calculation).

Bai, Laarhoven, and Stehlé [BLS16] further generalized the idea of this (2-tuple) sieving algorithm to k-tuple sieving: instead of just considering pairs of vectors, they introduced the idea of combining 3-tuples (or even k-tuples) of vectors; that is, trying to find 3-tuples $(\mathbf{x}, \mathbf{y}, \mathbf{z})$ in the current list such that $\mathbf{x} \pm \mathbf{y} \pm \mathbf{z}$ is of reduced length. They observed that the memory complexity of k-tuple sieving decreases as k increases (at the cost of increased time complexity). Later on, Herold and Kirshanova [HK17] extended the uniform heuristic to the case of k-tuple sieving (i.e., the output of each iteration of k-tuple sieving is distributed uniformly over the sphere) and introduced the k-list problem. Given a list L of m i.i.d. uniform samples from $\mathcal{S}^{d-1}$, this problem asks to find m k-tuples of distinct elements $\mathbf{x}_1, \ldots, \mathbf{x}_k \in L$ such that $\|\mathbf{x}_1 - \mathbf{x}_2 - \cdots - \mathbf{x}_k\| \leq 1$. Indeed, for $k = 3$, this is exactly Problem 1. Herold and Kirshanova observed that under the uniform heuristic (Heuristic 1), solving the k-list problem suffices to solve SVP, at least for a small constant approximation factor. Moreover, the uniform heuristic allows us to calculate the minimal required list size m so that m k-tuples solutions to the k-list problem (and hence k-tuple sieving) exist, at least with high probability over the list L. When k is constant, this minimal list size is $((k^{\frac{k}{k-1}})/(k+1))^{\frac{d}{2}}$ up to subexponential factors in d [HK17].[14] When $k = 3$, the minimal list size satisfies $|L| = (27/16)^{d/4+o(d)}$, which is exactly the regime we considered for Theorem 3.1.

4.3 An improved quantum algorithm for SVP using 3-tuple sieving

Combining Heuristic 1 and Theorem 3.1 yields a quantum algorithm that heuristically solves SVP in time $2^{0.284551d+o(d)}$ using $(27/16)^{d/4+o(d)} = 2^{0.188722d+o(d)}$ classical bits and QCRAM bits, and subexponentially many qubits. This is the fastest known (heuristic) quantum algorithm for SVP when the total memory is limited to $2^{0.188722d+o(d)}$.

Heuristic Claim 1. *Let $\gamma = O(1)$. Under Heuristic 1, there exists a quantum algorithm that solves γ-SVP in dimension d in time $2^{0.284551d+o(d)}$ with probability $1 - 2^{-\Omega(d)}$. This algorithm uses $(27/16)^{d/4+o(d)}$ classical memory and QCRAM bits, and $2^{o(d)}$ qubits.*

As mentioned in Section 1.2 and shown in Table 1, under the same heuristic and the same memory complexity (optimal for 3-tuple sieving), our quantum algorithm improves upon all prior algorithms. Yet, compared to the fastest known heuristic quantum algorithm for SVP [BCSS23, Proposition 4], which uses 2-tuple sieving, the gain in memory complexity that we obtain by using 3-tuple sieving still does not fully compensate for the increase in time complexity (despite our quantum speedup): time-memory product $2^{(0.1887+0.2846)d} \approx 2^{0.4733d}$ versus $2^{(0.2075+0.2563)d} \approx 2^{0.4638d}$.[15] However, our quantum algorithm uses only $2^{o(d)}$ qubits, whereas [BCSS23] uses an exponential number of qubits and QQRAM (i.e., quantum-readable quantum-writable *quantum* memory, which is technologically more demanding than the QCRAM required in our approach) for implementing quantum walks. Considering the class of heuristic quantum algorithms for SVP that use at most $2^{o(d)}$ qubits and

[14] This can be shown by an argument similar to the proof of [HK17, Corollary 1], using the fact that [HK17, Theorem 1] also gives a lower bound on the probability that a k-tuple is a solution.

[15] Note that a time-memory product may not be the best measure for comparing these algorithms, as time and memory are not directly interchangeable.

no QQRAM, the best known time complexity is $2^{0.2571d}$ [Hei21], achieved using $2^{0.2075d}$ classical memory (and also QCRAM of exponential size), which therefore still yields a better time-memory product than our result: $2^{0.4646d}$ versus $2^{0.4733d}$.

Acknowledgements. The authors thank Léo Ducas, Elena Kirshanova, Johanna Loyer, Eamonn Postlethwaite, and Yixin Shen for helpful discussions and references on sieving, and the anonymous reviewers for their valuable feedback.

References

[ACKS25] Divesh Aggarwal, Yanlin Chen, Rajendra Kumar, and Yixin Shen. Improved classical and quantum algorithms for the shortest vector problem via bounded distance decoding. *SIAM Journal on Computing*, 54(2):233–278, 2025. 1

[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. In *Advances in Cryptology – EUROCRYPT 2019*, volume 11477 of *Lecture Notes in Computer Science*, pages 717–746. Springer, 2019. 1, 37

[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 ACM Symposium on Theory of Computing (STOC 2015)*, pages 733–742. ACM, 2015. 1, 37

[Ajt96] Miklós Ajtai. Generating hard instances of lattice problems (extended abstract). In *Proceedings of the ACM Symposium on Theory of Computing (STOC 1996)*, pages 99–108, 1996. 36

[AKS01] Miklós Ajtai, Ravi Kumar, and D. Sivakumar. A sieve algorithm for the shortest lattice vector problem. In *Proceedings of the ACM Symposium on Theory of Computing (STOC 2001)*, pages 601–610. ACM, 2001. 1, 37

[Amb04] Andris Ambainis. Quantum walk algorithm for element distinctness. In *Proceedings of the IEEE Symposium on Foundations of Computer Science (FOCS 2004)*, pages 22–31, 2004. 5

[Amb10] Andris Ambainis. Quantum search with variable times. *Theory of Computing Systems*, 47(3):786–807, 2010. 9, 18

[AR20] Scott Aaronson and Patrick Rall. Quantum approximate counting, simplified. In *Symposium on Simplicity in Algorithms (SOSA 2020)*, 2020. 28

[AS18] Divesh Aggarwal and Noah Stephens-Davidowitz. Just take the average! an embarrassingly simple 2^n-time algorithm for SVP (and CVP). In *Symposium on Simplicity in Algorithms (SOSA 2018)*, volume 61 of *Open Access Series in Informatics*, pages 12:1–12:19, 2018. 37

[BCSS23] Xavier Bonnetain, André Chailloux, André Schrottenloher, and Yixin Shen. Finding many collisions via reusable quantum walks. In *Advances in Cryptology – EUROCRYPT 2023*, volume 14008, pages 221–251, 2023. 2, 3, 5, 38

[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 ACM-SIAM Symposium on Discrete Algorithms (SODA 2016)*, pages 10–24, 2016. 2, 4, 13, 14, 15, 17

[BHMT02] Gilles Brassard, Peter Høyer, Michele Mosca, and Alain Tapp. Quantum amplitude amplification and estimation. In *Quantum Computation and Information*, volume 305 of *Contemporary Mathematics*, pages 53–74. American Mathematical Society, 2002. 7, 8

[BHT98] Gilles Brassard, Peter Høyer, and Alain Tapp. Quantum algorithm for the collision problem. In *Latin American Symposium on Theoretical Informatics (LATIN 1998)*, Lecture Notes in Computer Science, pages 163–169, 1998. 5

[BLS16] Shi Bai, Thijs Laarhoven, and Damien Stehlé. Tuple lattice sieving. *LMS Journal of Computational Mathematics*, 19(A):146–162, 2016. 1, 2, 37, 38

[Che52] Herman Chernoff. A measure of asymptotic efficiency for tests of a hypothesis based on the sum of observations. *The Annals of Mathematical Statistics*, 23(4):493 – 507, 1952. 6

[CL21] André Chailloux and Johanna Loyer. Lattice sieving via quantum random walks. In *Advances in Cryptology – ASIACRYPT 2021*, volume 13093 of *Lecture Notes in Computer Science*, pages 63–91, 2021. 2, 3, 5

[CL23] André Chailloux and Johanna Loyer. Classical and quantum 3 and 4-sieves to solve SVP with low memory. In *International Conference on Post-Quantum Cryptography*, pages 225–255. Springer, 2023. 2, 4

[Emd81] Peter van Emde Boas. *Another NP-complete partition problem and the complexity of computing short vectors in a lattice*. Report. Department of Mathematics. University of Amsterdam. 1981. 36

[Gro96] Lov K. Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the ACM Symposium on Theory of Computing (STOC 1996)*, pages 212–219. ACM, 1996. 8

[GSLW19] András Gilyén, Yuan Su, Guang Hao Low, and Nathan Wiebe. Quantum singular value transformation and beyond: Exponential improvements for quantum matrix arithmetics. In *Proceedings of the ACM Symposium on Theory of Computing (STOC 2019)*, pages 193–204, 2019. 7

[Hei21] Max Heiser. Improved quantum hypercone locality sensitive filtering in lattice sieving. Cryptology ePrint Archive, Paper 2021/1295, 2021. 39

[HK17] Gottfried Herold and Elena Kirshanova. Improved algorithms for the approximate k-list problem in Euclidean norm. In *Public-Key Cryptography – PKC 2017*, volume 10174 of *Lecture Notes in Computer Science*, pages 16–40. Springer, 2017. 3, 4, 15, 16, 38

[HKL18] Gottfried Herold, Elena Kirshanova, and Thijs Laarhoven. Speed-ups and time-memory trade-offs for tuple lattice sieving. In *Public-Key Cryptography – PKC 2018*, volume 10769 of *Lecture Notes in Computer Science*, pages 407–436. Springer, 2018. 2

[HPS11] Guillaume Hanrot, Xavier Pujol, and Damien Stehlé. Algorithms for the shortest and closest lattice vector problems. In *Coding and Cryptology (IWCC 2011)*, volume 6639 of *Lecture Notes in Computer Science*, pages 159–190, 2011. 37

[Iir20] Iiridayn. Answer to "When to stop enumerating a fixed set of unknown cardinality via random sampling?". Cross Validated (Stack Exchange), 2020. `https://stats.stackexchange.com/a/486813`, accessed October 2025. 10

[KLL15] Shelby Kimmel, Cedric Yen-Yu Lin, and Han-Hsuan Lin. Oracles with costs. In *Theory of Quantum Computation, Communication and Cryptography (TQC 2015)*, volume 44 of *LIPIcs*, pages 1–26, 2015. 4, 5, 9, 18

[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 *Advances in Cryptology – ASIACRYPT 2019*, volume 11921 of *Lecture Notes in Computer Science*, pages 521–551. Springer, 2019. 2

[Laa15] Thijs Laarhoven. *Search Problems in Cryptography: From Fingerprinting to Lattice Sieving*. PhD thesis, Eindhoven University of Technology, 2015. 2

[LK14] Yongjae Lee and Woo Chang Kim. Concise formulas for the surface area of the intersection of two hyperspherical caps. Technical report, KAIST, 2014. 13

[LMP15] Thijs Laarhoven, Michele Mosca, and Joop van de Pol. Finding shortest lattice vectors faster using quantum search. *Designs, Codes and Cryptography*, 77, 12 2015. 2

[MR08] Daniele Micciancio and Oded Regev. Lattice-based cryptography, 2008. `https://cims.nyu.edu/~regev/papers/pqc.pdf`. 1

[MU05] Michael Mitzenmacher and Eli Upfal. *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*. Cambridge University Press, 2005. 6

[MV10] Daniele Micciancio and Panagiotis Voulgaris. Faster exponential time algorithms for the shortest vector problem. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms (SODA 2010)*, pages 1468–1480. SIAM, 2010. 37

[NV08] Phong Q. Nguyen and Thomas Vidick. Sieve algorithms for the shortest vector problem are practical. *Journal of Mathematical Cryptology*, 2(2):181–207, 2008. 1, 2, 37, 38

[PS09] Xavier Pujol and Damien Stehlé. Solving the shortest lattice vector problem in time $2^{2.465n}$. Cryptology ePrint Archive, Paper 2009/605, 2009. 37

[Reg04] Oded Regev. Lattices in computer science, lecture 8, 2004. https://cims.nyu.edu/~regev/teaching/lattices_fall_2004/ln/svpalg.pdf. 37

[Sch87] Claus-Peter Schnorr. A hierarchy of polynomial time lattice basis reduction algorithms. *Theoretical Computer Science*, 53:201–224, 1987. 36

[SE94] Claus-Peter Schnorr and M. Euchner. Lattice basis reduction: Improved practical algorithms and solving subset sum problems. *Mathematical Programming*, 66:181–199, 1994. 36

[Sho97] Peter Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Journal on Computing*, 26(5):1484–1509, 1997. Earlier version in FOCS 1994. 1

[YLC14] Theodore J. Yoder, Guang Hao Low, and Isaac L. Chuang. Fixed-point quantum search with an optimal number of queries. *Physical Review Letters*, 113(21):210501, 2014. 7

A Appendix

A.1 Proofs of Lemma 2.9 and Lemma 2.10

Proof of Lemma 2.9. Let $\mathbf{x} \in \mathcal{S}^{d-1}$, and define $p := \Pr_{\mathbf{c}\sim\mathcal{U}(\mathcal{S}^{d-1})}[\langle \mathbf{x}, \mathbf{c}\rangle \approx_\epsilon \cos(\alpha)]$. By Lemma 2.7,

$$\begin{aligned}
p &\leq \Pr_{\mathbf{c}\sim\mathcal{U}(\mathcal{S}^{d-1})}[\langle \mathbf{x}, \mathbf{c}\rangle \geq \cos(\alpha) - \epsilon] \\
&=_d (1 - (\cos(\alpha) - \epsilon)^2)^{d/2} \\
&= (1-\cos^2(\alpha))^{d/2}\left(1 + \frac{2\epsilon\cos(\alpha) - \epsilon^2}{1-\cos^2(\alpha)}\right)^{d/2} \\
&\leq (1-\cos^2(\alpha))^{d/2}\left(1 + \frac{2\epsilon}{1-\cos^2(\alpha)}\right)^{d/2} \\
&\leq (1-\cos^2(\alpha))^{d/2}\exp(O(\epsilon d))
\end{aligned}$$

since $\alpha = \Omega(1)$ implies $1-\cos^2(\alpha) = \Omega(1)$, and using the fact that $1+x \leq \exp(x)$ for all $x \in \mathbb{R}$. As $\epsilon = 1/(\log d)^2$, we obtain $p \leq_d p_\alpha$.

Moreover, by the union bound and Lemma 2.7, we obtain that for some constant $k > 0$,

$$\begin{aligned}
p &\geq \Pr_{\mathbf{c}\sim\mathcal{U}(\mathcal{S}^{d-1})}[\langle \mathbf{x}, \mathbf{c}\rangle \geq \cos(\alpha)] - \Pr_{\mathbf{c}\sim\mathcal{U}(\mathcal{S}^{d-1})}[\langle \mathbf{x}, \mathbf{c}\rangle \geq \cos(\alpha) + \epsilon] \\
&\geq \frac{1}{d^k}(1-\cos^2(\alpha))^{d/2} - d^k(1-(\cos(\alpha)+\epsilon)^2)^{d/2} \\
&= (1-\cos^2(\alpha))^{d/2}\left(\frac{1}{d^k} - d^k\left(1 - \frac{2\epsilon\cos(\alpha)+\epsilon^2}{1-\cos^2(\alpha)}\right)^{d/2}\right).
\end{aligned}$$

Since $(1 - \frac{2\epsilon\cos(\alpha)+\epsilon^2}{1-\cos^2(\alpha)})^{d/2} \leq (1-\epsilon^2)^{d/2} \leq \exp(-\epsilon^2 d/2)$, the choice $\epsilon = 1/(\log d)^2$ yields $p \geq_d p_\alpha$. □

Proof of Lemma 2.10. Let $\mathbf{x}, \mathbf{y} \in \mathcal{S}^{d-1}$ be such that $\langle \mathbf{x}, \mathbf{y} \rangle \approx_\epsilon \cos(\theta)$, and define

$$p \coloneqq \Pr_{\mathbf{c} \sim \mathcal{U}(\mathcal{S}^{d-1})} [\langle \mathbf{x}, \mathbf{c} \rangle \approx_\epsilon \cos(\alpha), \langle \mathbf{y}, \mathbf{c} \rangle \approx_\epsilon \cos(\beta)].$$

By Lemma 2.8, it suffices to show that $p =_d (1-\gamma^2)^{d/2}$, where

$$\gamma^2 \coloneqq \frac{\cos^2(\alpha) + \cos^2(\beta) - 2\cos(\alpha)\cos(\beta)\cos(\theta)}{\sin^2(\theta)}.$$

We proceed by showing that $(1-\widetilde{\gamma}^2)^{d/2} \leq_d p \leq_d (1-\overline{\gamma}^2)^{d/2}$ for some $\widetilde{\gamma}^2 \leq \gamma^2 + O(\epsilon)$ and $\overline{\gamma}^2 \geq \gamma^2 - O(\epsilon)$ that are defined in terms of $(\alpha, \beta, \theta, \epsilon)$, allowing us to prove $p =_d (1-\gamma^2)^{d/2}$.

More precisely, define $\cos(\phi) \coloneqq \langle \mathbf{x}, \mathbf{y} \rangle$. Then $\cos(\theta) - \epsilon \leq \cos(\phi) \leq \cos(\theta) + \epsilon$, which implies $-3\epsilon \leq \sin^2(\phi) - \sin^2(\theta) \leq 2\epsilon$. In addition, the assumption $|\alpha - \beta| + \Omega(1) \leq \theta \leq \alpha + \beta - \Omega(1)$ implies $\theta = \Omega(1)$. Using $\varepsilon = o(1)$, it follows that there exists a constant $\kappa > 0$ such that (for sufficiently large d) $\sin^2(\phi) \geq \kappa$. Using standard trigonometric identities, we also obtain

$$\begin{aligned}
\sin^2(\theta)(1-\gamma^2) &= \sin^2(\theta) - \cos^2(\alpha) - \cos^2(\beta) + 2\cos(\alpha)\cos(\beta)\cos(\theta) \\
&= 4\sin\left(\frac{(\beta+\theta)+\alpha}{2}\right)\sin\left(\frac{(\beta+\theta)-\alpha}{2}\right)\sin\left(\frac{\alpha+(\beta-\theta)}{2}\right)\sin\left(\frac{\alpha-(\beta-\theta)}{2}\right)
\end{aligned}$$

and thus the assumption on α, β, θ implies $1-\gamma^2 \geq \sin^2(\theta)(1-\gamma^2) = \Omega(1)$. In other words, there also exists a constant $\kappa' > 0$ such that $1-\gamma^2 \geq \kappa'$ (for sufficiently large d).
Now, for the upper bound, note that Lemma 2.8 implies

$$p \leq \Pr_{\mathbf{c} \sim \mathcal{U}(\mathcal{S}^{d-1})} [\langle \mathbf{x}, \mathbf{c} \rangle \geq \cos(\alpha) - \epsilon, \langle \mathbf{y}, \mathbf{c} \rangle \geq \cos(\beta) - \epsilon] =_d (1-\overline{\gamma}^2)^{d/2}$$

where

$$\begin{aligned}
\overline{\gamma}^2 &\coloneqq \frac{(\cos(\alpha)-\epsilon)^2 + (\cos(\beta)-\epsilon)^2 - 2(\cos(\alpha)-\epsilon)(\cos(\beta)-\epsilon)\cos(\phi)}{\sin^2(\phi)} \\
&\geq \frac{\cos^2(\alpha) + \cos^2(\beta) - 2\cos(\alpha)\cos(\beta)\cos(\theta) - 2\epsilon(\cos(\alpha) + \cos(\beta) + \cos(\alpha)\cos(\beta))}{\sin^2(\phi)} \\
&\geq \frac{\cos^2(\alpha) + \cos^2(\beta) - 2\cos(\alpha)\cos(\beta)\cos(\theta) - 6\epsilon}{\sin^2(\phi)} \\
&= \gamma^2 \frac{\sin^2(\theta)}{\sin^2(\phi)} - \frac{6\epsilon}{\sin^2(\phi)} \\
&= \gamma^2 \left(1 - \frac{\sin^2(\phi) - \sin^2(\theta)}{\sin^2(\phi)}\right) - \frac{6\epsilon}{\sin^2(\phi)}.
\end{aligned}$$

Since $\gamma^2 \leq 1$, $\sin^2(\phi) - \sin^2(\theta) \leq 2\epsilon$, and $\sin^2(\phi) \geq \kappa$, we obtain $\overline{\gamma}^2 \geq \gamma^2 - \frac{8\epsilon}{\kappa}$. The desired upper bound now follows, because (using $1-\gamma^2 \geq \kappa'$ and $\epsilon = 1/(\log d)^2$)

$$(1-\overline{\gamma}^2)^{d/2} = (1-\gamma^2)^{d/2}\left(1 + \frac{\gamma^2 - \overline{\gamma}^2}{1-\gamma^2}\right)^{d/2} \leq (1-\gamma^2)^{d/2}\left(1 + \frac{8\epsilon}{\kappa\kappa'}\right)^{d/2} \leq_d (1-\gamma^2)^{d/2}.$$

For the lower bound, note that the union bound implies

$$\begin{aligned}
p \geq & \Pr_{\mathbf{c}\sim\mathcal{U}(\mathcal{S}^{d-1})}[\langle \mathbf{x},\mathbf{c}\rangle \geq \cos(\alpha)-\epsilon, \langle \mathbf{y},\mathbf{c}\rangle \geq \cos(\beta)-\epsilon] \\
& - \Pr_{\mathbf{c}\sim\mathcal{U}(\mathcal{S}^{d-1})}[\langle \mathbf{x},\mathbf{c}\rangle > \cos(\alpha)+\epsilon, \langle \mathbf{y},\mathbf{c}\rangle \geq \cos(\beta)-\epsilon] \\
& - \Pr_{\mathbf{c}\sim\mathcal{U}(\mathcal{S}^{d-1})}[\langle \mathbf{x},\mathbf{c}\rangle \geq \cos(\alpha)-\epsilon, \langle \mathbf{y},\mathbf{c}\rangle > \cos(\beta)+\epsilon] \\
\geq_d & \Pr_{\mathbf{c}\sim\mathcal{U}(\mathcal{S}^{d-1})}[\langle \mathbf{x},\mathbf{c}\rangle \geq \cos(\alpha)-\epsilon, \langle \mathbf{y},\mathbf{c}\rangle \geq \cos(\beta)-\epsilon] \\
\geq & \Pr_{\mathbf{c}\sim\mathcal{U}(\mathcal{S}^{d-1})}[\langle \mathbf{x},\mathbf{c}\rangle \geq \cos(\alpha), \langle \mathbf{y},\mathbf{c}\rangle \geq \cos(\beta)]
\end{aligned}$$

where the second inequality can be shown using similar methods as how we have shown the upper bound, using that $\epsilon = 1/(\log d)^2$. By Lemma 2.8, we thus obtain

$$p \geq_d (1-\widetilde{\gamma}^2)^{d/2} = (1-\gamma^2)^{d/2}\left(1-\frac{\widetilde{\gamma}^2-\gamma^2}{1-\gamma^2}\right)^{d/2}$$

where

$$\begin{aligned}
\widetilde{\gamma}^2 &:= \frac{\cos^2(\alpha)+\cos^2(\beta)-2\cos(\alpha)\cos(\beta)\cos(\phi)}{\sin^2(\phi)} \\
&= \frac{\cos^2(\alpha)+\cos^2(\beta)-2\cos(\alpha)\cos(\beta)(\cos(\theta)+\cos(\phi)-\cos(\theta))}{\sin^2(\theta)}\frac{\sin^2(\theta)}{\sin^2(\phi)} \\
&= \left(\gamma^2 - \frac{2\cos(\alpha)\cos(\beta)(\cos(\phi)-\cos(\theta))}{\sin^2(\theta)}\right)\frac{\sin^2(\theta)}{\sin^2(\phi)}.
\end{aligned}$$

If $\sin^2(\theta) \geq \sin^2(\phi)$ (meaning $\theta \geq \phi$ and $\cos(\theta) \leq \cos(\phi)$), then $\widetilde{\gamma}^2 \leq \gamma^2\frac{\sin^2(\theta)}{\sin^2(\phi)}$. Otherwise,

$$\widetilde{\gamma}^2 = \left(\gamma^2 + \frac{2\cos(\alpha)\cos(\beta)(\cos(\theta)-\cos(\phi))}{\sin^2(\theta)}\right)\frac{\sin^2(\theta)}{\sin^2(\phi)} \leq \gamma^2\frac{\sin^2(\theta)}{\sin^2(\phi)} + \frac{2\epsilon}{\sin^2(\phi)}.$$

In either case, we have $\sin^2(\theta)-\sin^2(\phi) \leq 3\epsilon$, so $\gamma^2 \leq 1$ and $\sin^2(\phi) \geq \kappa$ imply

$$\gamma^2\frac{\sin^2(\theta)}{\sin^2(\phi)} = \gamma^2 + \gamma^2\frac{\sin^2(\theta)-\sin^2(\phi)}{\sin^2(\phi)} \leq \gamma^2 + \frac{3\epsilon}{\kappa}.$$

In other words, we have shown $\widetilde{\gamma}^2 - \gamma^2 \leq \frac{5\epsilon}{\kappa}$. Hence,

$$p \geq_d (1-\gamma^2)^{d/2}\left(1-\frac{\widetilde{\gamma}^2-\gamma^2}{1-\gamma^2}\right)^{d/2} \geq (1-\gamma^2)^{d/2}\left(1-\frac{5\epsilon}{\kappa\kappa'}\right)^{d/2} \geq_d (1-\gamma^2)^{d/2}$$

as desired. □