Quantum speedups for lattice sieves are tenuous at best

ePrint: 2019/1161

Martin R. Albrecht, Vlad Gheorghiu,
Eamonn W. Postlethwaite, John M. Schanck

October 18, 2019

Not this talk:

The security of Kyber768.

Not this talk:

The security of Kyber768.

Not this talk:

The security of Kyber768.

The **lattice security** of Kyber768.

Not this talk:

The security of Kyber768.

The **lattice security** of Kyber768.

The **cost of BKZ-k for k that determines** the lattice security of Kyber768.

Not this talk:

The security of Kyber768.

The **lattice security** of Kyber768.

The **cost of BKZ-k for k that determines** the lattice security of Kyber768.

The **core-SVP estimate for** the lattice security of Kyber768.

Not this talk:

The security of Kyber768.

The **lattice security** of Kyber768.

The **cost of BKZ-k for k that determines** the lattice security of Kyber768.

The **core-SVP estimate for** the lattice security of Kyber768.

The **cost of the sieving routine** inside the SVP solver used in the core-SVP estimate for the lattice security of Kyber768.

Not this talk:

The security of Kyber768.
The **lattice security** of Kyber768.
The **cost of BKZ-k for k that determines** the lattice security of Kyber768.
The **core-SVP estimate for** the lattice security of Kyber768.
The **cost of the sieving routine** inside the SVP solver used in the core-SVP estimate for the lattice security of Kyber768.
The **heuristic cost of the sieving routine** inside the SVP solver used in the core-SVP estimate for the lattice security of Kyber768.

Not this talk:

The security of Kyber768.

The **lattice security** of Kyber768.

The **cost of BKZ-k for k that determines** the lattice security of Kyber768.

The **core-SVP estimate for** the lattice security of Kyber768.

The **cost of the sieving routine** inside the SVP solver used in the core-SVP estimate for the lattice security of Kyber768.

The **heuristic cost of the sieving routine** inside the SVP solver used in the core-SVP estimate for the lattice security of Kyber768.

The **heuristic cost of one call to the sieving routine** inside the SVP solver used in the core-SVP estimate for the lattice security of Kyber768.

Not this talk:

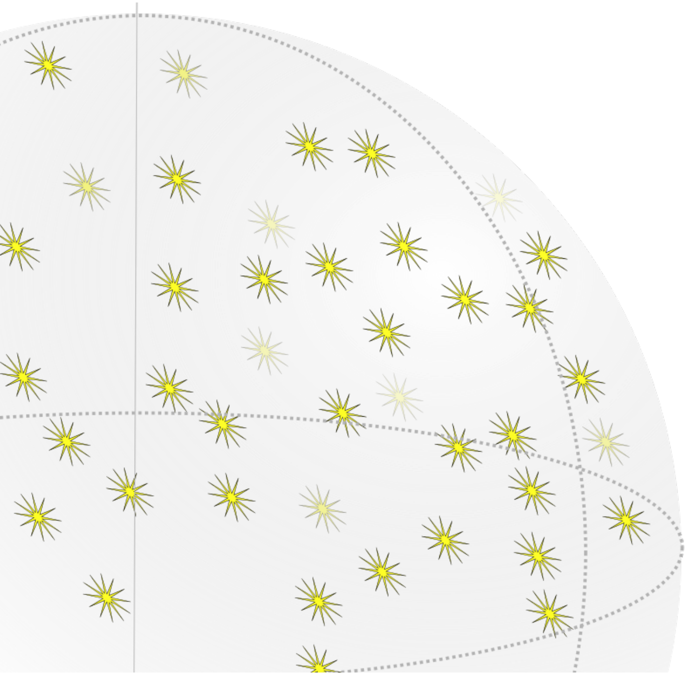
The security of Kyber768.
The **lattice security** of Kyber768.
The **cost of BKZ-k for k that determines** the lattice security of Kyber768.
The **core-SVP estimate for** the lattice security of Kyber768.
The **cost of the sieving routine** inside the SVP solver used in the core-SVP estimate for the lattice security of Kyber768.
The **heuristic cost of the sieving routine** inside the SVP solver used in the core-SVP estimate for the lattice security of Kyber768.
The **heuristic cost of one call to the sieving routine** inside the SVP solver used in the core-SVP estimate for the lattice security of Kyber768.

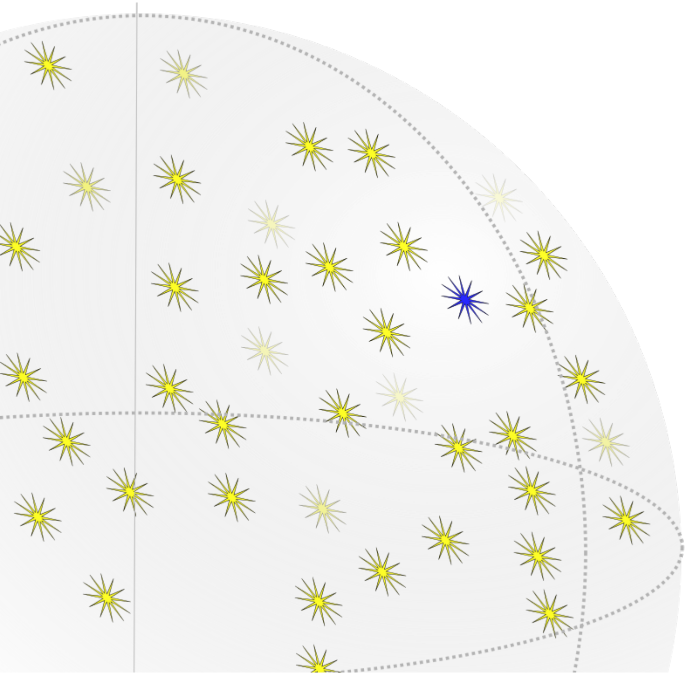
This talk:

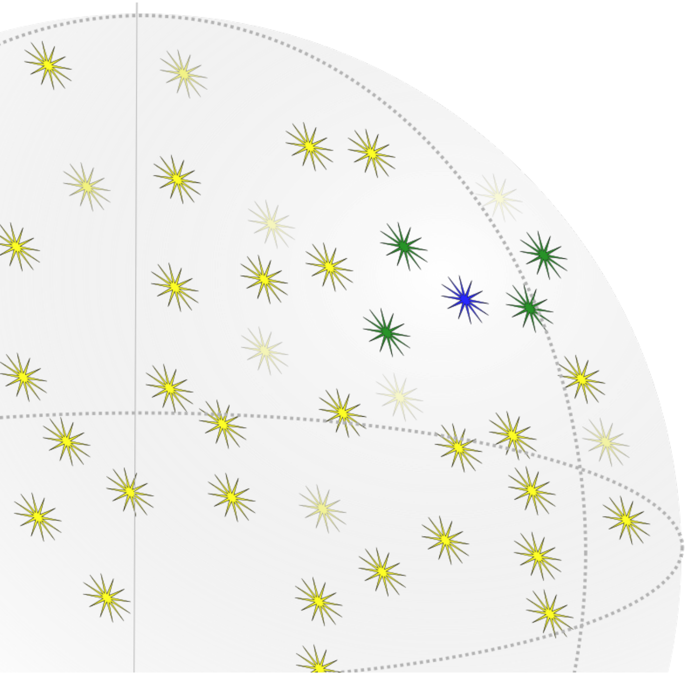
The heuristic cost—classical and quantum—of near neighbor search on spheres in dimension < 1000.

Cost estimates and numerically optimized parameters for the heuristic NNS algorithms underlying:

- Nguyen–Vidick sieve
- `bgj1`, i.e. Becker–Gama–Joux sieve w/o recursion
- The Becker–Ducas–Gama–Laarhoven sieve







Near neighbor search

A near neighbor search algorithm takes a list of N points, pre-processes it to make neighbor queries more efficient.

I want to find points that are close to u in angular distance.

- Angular distance: $\theta(u, v) = \arccos\langle u, v\rangle$.

I want to do this for many different u.

Near neighbor search

A near neighbor search algorithm takes a list of N points, pre-processes it to make neighbor queries more efficient.

I want to find points that are close to u in angular distance.

- Angular distance: $\theta(u, v) = \arccos\langle u, v\rangle$.

I want to do this for many different u.

List-size preserving parameterization

Special case:

- Input consists of N uniformly random points.
- N large enough to ensure that there are N neighboring pairs.

Write $C_d(\theta)$ for the spherical measure of

$$\mathrm{Cap}(u,\theta) = \{v : \theta(u,v) \le \theta\}.$$

Then

$$N \approx \binom{N}{2} C_d(\theta),$$

equiv.

$$N \approx 2/C_d(\theta)$$

List-size preserving parameterization

Special case:

- Input consists of N uniformly random points.
- N large enough to ensure that there are N neighboring pairs.

Write $C_d(\theta)$ for the spherical measure of

$$\mathrm{Cap}(u,\theta) = \{v : \theta(u,v) \leq \theta\}.$$

Then

$$N \approx \binom{N}{2} C_d(\theta),$$

equiv.

$$N \approx 2/C_d(\theta)$$

Algorithm: AllPairs / Nguyen–Vidick sieve

Input: list L of size N.

Search:

1. Number the points $v_1, v_2, v_3, \ldots, v_N$
2. Test $\theta(v_i, v_j) \leq \theta$ for $1 \leq i < j \leq N$

Cost of AllPairs / Nguyen–Vidick sieve

List-size preserving case

Classical search

Nguyen–Vidick (2008): $(1/C_d(\theta))^{2+o(1)}$

$$(1/C_d(\pi/3))^{2+o(1)} = 2^{c(d)} \text{ where } c(d) = (0.4150\ldots + o(1))d$$

Quantum search

Laarhoven–Mosca–van de Pol (2014): $(1/C_d(\theta))^{1.5+o(1)}$

$$(1/C_d(\pi/3))^{1.5+o(1)} = 2^{c(d)} \text{ where } c(d) = (0.3112\ldots + o(1))d$$

Cost of AllPairs / Nguyen–Vidick sieve

List-size preserving case

Classical search

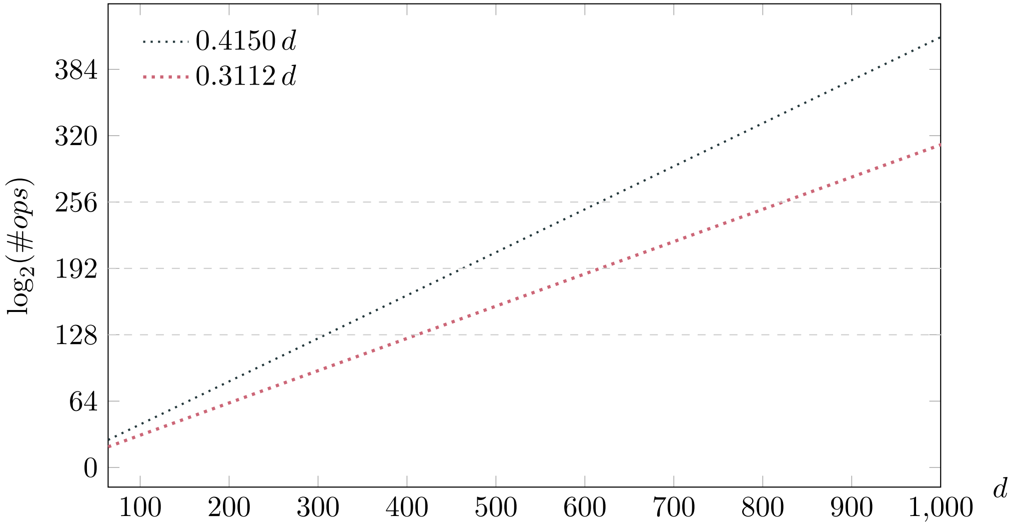
Nguyen–Vidick (2008): $(1/C_d(\theta))^{2+o(1)}$

$$(1/C_d(\pi/3))^{2+o(1)} = 2^{c(d)} \text{ where } c(d) = (0.4150\ldots + o(1))d$$

Quantum search

Laarhoven–Mosca–van de Pol (2014): $(1/C_d(\theta))^{1.5+o(1)}$

$$(1/C_d(\pi/3))^{1.5+o(1)} = 2^{c(d)} \text{ where } c(d) = (0.3112\ldots + o(1))d$$


$0.4150\,d$
$0.3112\,d$
$\log_2(\#ops)$
384
320
256
192
128
64
0
100
200
300
400
500
600
700
800
900
1,000
d

Why care about the polynomial terms?

- Quantum and classical variants have *different* polynomial factors.
- Quantum advantage is small. Even smaller in more advanced algorithms.
- Polynomial factors are significant in low dimension.

Why care about the polynomial terms?

- Quantum and classical variants have *different* polynomial factors.
- Quantum advantage is small. Even smaller in more advanced algorithms.
- Polynomial factors are significant in low dimension.

Why care about the polynomial terms?

- Quantum and classical variants have *different* polynomial factors.
- Quantum advantage is small. Even smaller in more advanced algorithms.
- Polynomial factors are significant in low dimension.

Why care about the polynomial terms?

- Quantum and classical variants have *different* polynomial factors.
- Quantum advantage is small. Even smaller in more advanced algorithms.
- Polynomial factors are significant in low dimension.

What are the polynomial factors?

- Volume estimates.
- Cost of testing $\theta(u, v)$.

What are the polynomial factors?

- Volume estimates.
- Cost of testing $\theta(u, v)$.

Search predicates

- Search predicate on $\mathcal{X}$:

$$f : \mathcal{X} \to \{0, 1\}$$

- Kernel of f:

$$\mathrm{Ker}(f) = \{x : f(x) = 0\}$$

$$|f| = |\,\mathrm{Ker}(f)|$$

- Predicate $f \cap g$ defined by:

$$\mathrm{Ker}(f \cap g) = \mathrm{Ker}(f) \cap \mathrm{Ker}(g)$$

Exhaustive search

$$g(1) \qquad g(2) \qquad g(3) \qquad g(4) \qquad g(5) \qquad \dots \qquad \dots$$

Exhaustive search

$1 \qquad g(2) \qquad g(3) \qquad g(4) \qquad g(5) \qquad \ldots \qquad \ldots$

Exhaustive search

$$1 \qquad 1 \qquad g(3) \qquad g(4) \qquad g(5) \qquad \ldots \qquad \ldots$$

Exhaustive search

$$1 \qquad 1 \qquad 1 \qquad \mathrm{g}(4) \qquad \mathrm{g}(5) \qquad \ldots \qquad \ldots$$

Exhaustive search

$$1 \qquad 1 \qquad 1 \qquad 1 \qquad \mathrm{g}(5) \qquad \ldots \qquad \ldots$$

Exhaustive search

$$1 \qquad 1 \qquad 1 \qquad 1 \qquad 1 \qquad \ldots \qquad \ldots$$

Exhaustive search

$$1 \qquad 1 \qquad 1 \qquad 1 \qquad 1 \qquad \ldots \qquad \mathrm{g}(57)$$

Exhaustive search

$$1 \qquad 1 \qquad 1 \qquad 1 \qquad 1 \qquad \dots \qquad 0$$

Filtered search

f(1)	f(2)	f(3)	f(4)	f(5)	...	f(57)
g(1)	g(2)	g(3)	g(4)	g(5)	...	g(57)

Filtered search

1	1	f(3)	f(4)	f(5)	...	f(57)
g(1)	g(2)	g(3)	g(4)	g(5)	...	g(57)

Filtered search

1	1	1	1	f(5)	...	f(57)
g(1)	g(2)	g(3)	g(4)	g(5)	...	g(57)

Filtered search

1	1	1	1	1	$\dots$	f(57)
g(1)	g(2)	g(3)	g(4)	g(5)	$\dots$	g(57)

Filtered search

1	1	1	1	1	...	0
g(1)	g(2)	g(3)	g(4)	g(5)	...	g(57)

Filtered search

1	1	1	1	1	$\dots$	0
g(1)	g(2)	g(3)	g(4)	g(5)	$\dots$	0

Quantum search

For any predicate g and unitary $\mathbf{A}$, define the amplification operator:

$$\mathbf{G}(\mathbf{A}, g) := \mathbf{A}\mathbf{R}_0\mathbf{A}^\dagger\mathbf{R}_g$$

where

$$\mathbf{R}_0 |x\rangle = \begin{cases} -|x\rangle & \text{if } x = 0 \\ |x\rangle & \text{otherwise} \end{cases}$$

$$\mathbf{R}_g |x\rangle = (-1)^{g(x)} |x\rangle .$$

Quantum search

Suppose that measuring $\mathbf{A}\,|0\rangle$ yields an element of $\mathrm{Ker}(g)$ with probability p.

Grover–Brassard–Høyer–Mosca–Tapp:

- Measuring
$$\mathbf{G}(\mathbf{A},g)^k\mathbf{A}\,|0\rangle$$
with $k \approx \sqrt{1/p}$ yields a root of g w.p. $\approx 1 \ldots$

Boyer–Brassard–Høyer–Tapp:

- …even if p is not known.

Quantum search

Suppose that measuring $\mathbf{A}|0\rangle$ yields an element of $\mathrm{Ker}(g)$ with probability p.

Grover–Brassard–Høyer–Mosca–Tapp:

- Measuring
$$\mathbf{G}(\mathbf{A}, g)^k \mathbf{A}|0\rangle$$
with $k \approx \sqrt{1/p}$ yields a root of g w.p. $\approx 1\ldots$

Boyer–Brassard–Høyer–Tapp:

- $\ldots$ even if p is not known.

Quantum search

Suppose that measuring $\mathbf{A}\left|0\right\rangle$ yields an element of $\mathrm{Ker}(g)$ with probability p.

Grover–Brassard–Høyer–Mosca–Tapp:

- Measuring
$$\mathbf{G}(\mathbf{A}, g)^k \mathbf{A}\left|0\right\rangle$$
with $k \approx \sqrt{1/p}$ yields a root of g w.p. $\approx 1\ldots$

Boyer–Brassard–Høyer–Tapp:

- ... even if p is not known.

Filtered quantum search

Parameters m_1 and m_2.

1. Sample j uniformly from $\{0, \ldots, m_1 - 1\}$
2. Sample k uniformly from $\{0, \ldots, m_2 - 1\}$
3. Define

$$\mathbf{A}_j = \mathbf{G}(\mathbf{D}, f)^j \mathbf{D}$$
$$\mathbf{B}_k = \mathbf{G}(\mathbf{A}_j, f \cap g)^k$$

4. Prepare and measure the state:

$$\mathbf{B}_k \mathbf{A}_j \left|0\right\rangle$$

Cost of filtered quantum search

Suppose that we know $P/\gamma \leq |g| \leq \gamma P$.

Proposition

We can choose m_1 and m_2 such that $\mathrm{FilteredQuantumSearch}$ *finds a root of $f \cap g$ with probability at least $1/8$ and has a cost that is dominated by (approximately)*

- $\gamma \frac{1}{2}\sqrt{N}$ *times the cost of $\mathbf{G}(g)$, or*
- $\frac{4}{3}\sqrt{\gamma P}$ *times the cost of $\mathbf{R}_{f \cap g}$.*

Cost of filtered quantum search

Suppose that we know $P/\gamma \leq |g| \leq \gamma P$.

Proposition

We can choose m_1 and m_2 such that $\mathrm{FilteredQuantumSearch}$ finds a root of $f \cap g$ with probability at least $1/8$ and has a cost that is dominated by (approximately)

- $\gamma \frac{1}{2}\sqrt{N}$ *times the cost of $\mathbf{G}(g)$, or*
- $\frac{4}{3}\sqrt{\gamma P}$ *times the cost of $\mathbf{R}_{f \cap g}$.*

Cost of filtered quantum search

Suppose that we know $P/\gamma \leq |g| \leq \gamma P$.

Proposition

We can choose m_1 *and* m_2 *such that* $\mathrm{FilteredQuantumSearch}$ *finds a root of* $f \cap g$ **with probability at least $1/8$** *and has a cost that is dominated by* **(approximately)**

- $\gamma \frac{1}{2}\sqrt{N}$ *times the cost of* $\mathbf{G}(g)$*, or*
- $\frac{4}{3}\sqrt{\gamma P}$ *times the cost of* $\mathbf{R}_{f \cap g}$*.*

Cost of filtered quantum search

Suppose that we know $P/\gamma \leq |g| \leq \gamma P$.

Idealized Proposition

We can choose m_1 *and* m_2 *such that* $\mathrm{FilteredQuantumSearch}$ *finds a root of* $f \cap g$ *and has a cost that is dominated by*

- $\frac{1}{2}\sqrt{N}$ *times the cost of* $\mathbf{G}(g)$*, or*
- $\frac{4}{3}\sqrt{P}$ *times the cost of* $\mathbf{R}_{f\cap g}$*.*

Algorithm: AllPairs / Nguyen–Vidick sieve

Input: list L of size N

1. Number the points $v_1, v_2, v_3, \ldots, v_N$
2. For $i = 1, \ldots, N$
3. For $j = i + 1, \ldots, N$
4. Test $g_i(v_j)$ where $g_i(v_j) = [\theta(v_i, v_j) > \pi/3]$.

Algorithm: AllPairs / Nguyen–Vidick sieve

Input: list L of size N

1. Number the points $v_1, v_2, v_3, \ldots, v_N$
2. For $i = 1, \ldots, N$
3.   For $j = i + 1, \ldots, N$
4.   **If $\boldsymbol{f_i(v_j)}$ then** test $g_i(v_j)$ where $g_i(v_j) = [\theta(v_i, v_j) > \pi/3]$.

Algorithm: AllPairs / Nguyen–Vidick sieve

Input: list L of size N

1. Number the points $v_1, v_2, v_3, \ldots, v_N$
2. For $i = 1, \ldots, N$
3. For $j = i + 1, \ldots, N$
4. **If $\boldsymbol{f_i(v_j)}$ then** test $g_i(v_j)$ where $g_i(v_j) = [\theta(v_i, v_j) > \pi/3]$.

What to use for f_i in a filtered search?

XOR + population count

Define a hash function family:

$$\mathcal{H} = \{u \mapsto \operatorname{sgn}(\langle r, u \rangle) \ : \ r \in \mathcal{S}\}$$

XOR + population count

Fact: $$\Pr_{h \leftarrow \mathcal{H}}[h(u) \neq h(v)] = \frac{\theta(u,v)}{\pi}.$$

Let $H_n(x) = (h_1(x), \ldots h_n(x))$ for random $h_i \in \mathcal{H}$.

For large n, we have

$$\frac{\mathrm{HammingWeight}(H_n(u) \oplus H_n(v))}{n} \approx \frac{\theta(u,v)}{\pi}$$

XOR + population count

Fact: $$\Pr_{h \leftarrow \mathcal{H}}[h(u) \neq h(v)] = \frac{\theta(u,v)}{\pi}.$$

Let $H_n(x) = (h_1(x), \ldots h_n(x))$ for random $h_i \in \mathcal{H}$.

For large n, we have

$$\frac{\mathrm{HammingWeight}(H_n(u) \oplus H_n(v))}{n} \approx \frac{\theta(u,v)}{\pi}$$

XOR + population count

Fact: $$\Pr_{h \leftarrow \mathcal{H}}[h(u) \neq h(v)] = \frac{\theta(u,v)}{\pi}.$$

Let $H_n(x) = (h_1(x), \ldots h_n(x))$ for random $h_i \in \mathcal{H}$.

For large n, we have

$$\frac{\mathrm{HammingWeight}(H_n(u) \oplus H_n(v))}{n} \approx \frac{\theta(u,v)}{\pi}$$

XOR + population count

Used as a filter in implementations of sieving algorithms:

- 2014 Fitzpatrick–Bischof–Buchmann–Dagdelen–Göpfert–Mariano–Yang
- 2018 Ducas
- 2019 Albrecht–Ducas–Herold–Kirshanova–Postlethwaite–Stevens

Earlier algorithmic use

- 1995 Goemans–Williamson
- 2002 Charikar

Algorithm: AllPairs / Nguyen–Vidick sieve

Input: list L of size N

Setup:

1. Fix H_n
2. Construct a table $(i, H_n(v_i))$

Search:

For all i:

1. Load $H_n(v_i)$
2. For $j = i + 1, \ldots, N$
3. Load $H_n(v_j)$
4. If $\mathrm{HammingWt}(H_n(v_i) \oplus H_n(v_j)) \leq k$
5. Test $\theta(v_i, v_j) \leq \theta$.

This work: New python/mpmath package

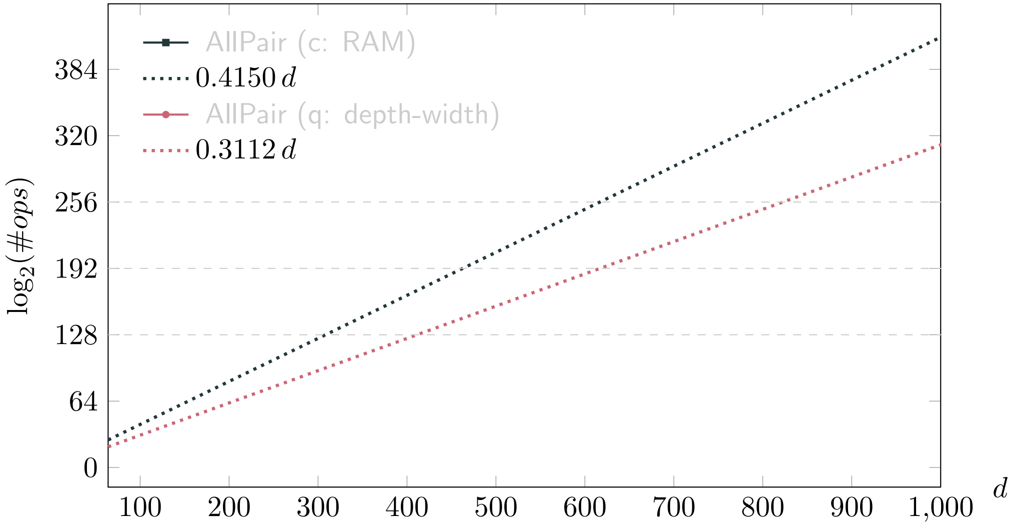
Calculates the circuit depth, width, gate count (etc.) for popcount and filtered quantum search subroutines.

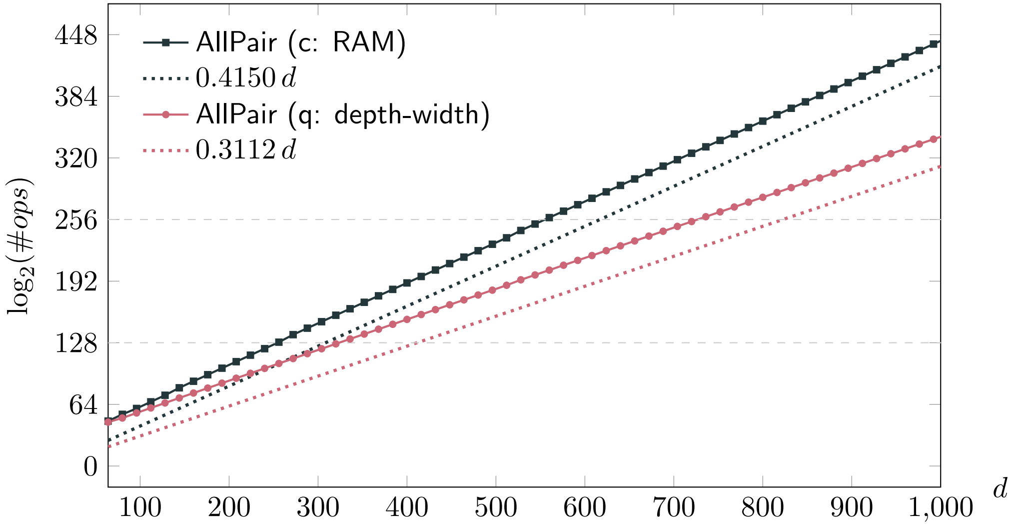
Calculates the accuracy of random popcount filters given

- points uniformly distributed on sphere;
- points uniformly distributed in a cap of angle β.

Calculates the (normalized) spherical measure of

- caps, using ${}_2F_1$ representation of $C_d(\theta)$
- intersections of caps, using an integral representation.


AllPair (c: RAM)
$0.4150\,d$
AllPair (q: depth-width)
$0.3112\,d$
$\log_2(\#ops)$
384
320
256
192
128
64
0
100
200
300
400
500
600
700
800
900
1,000
d



AllPair (c: RAM)
$0.4150\,d$
AllPair (q: depth-width)
$0.3112\,d$
$\log_2(\#ops)$
448
384
320
256
192
128
64
0
100
200
300
400
500
600
700
800
900
1,000
d

Error correction

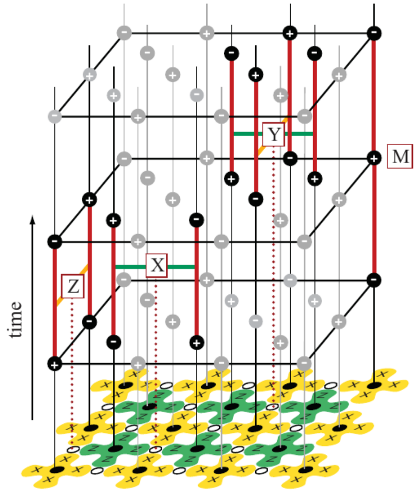
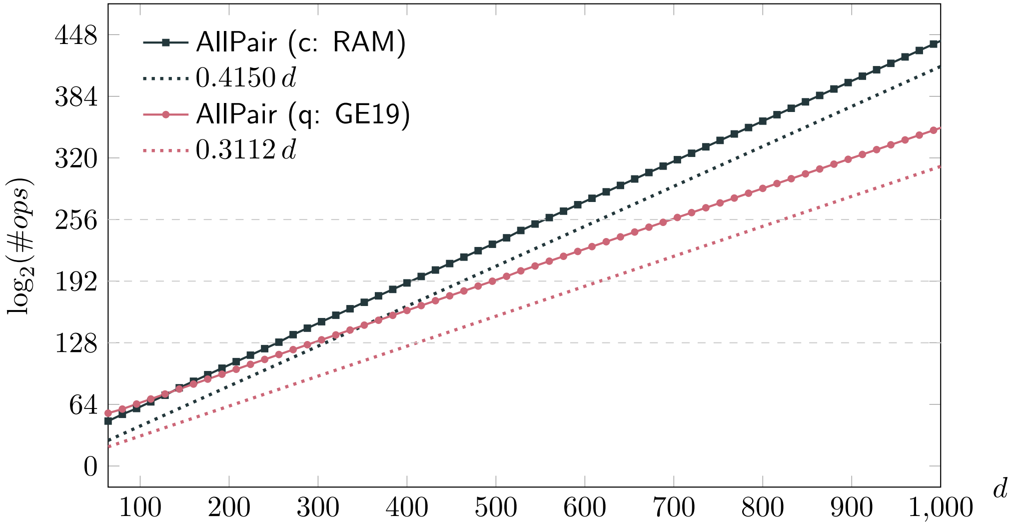


Image: Fowler, Mariantoni, Martinis, Cleland. (2012)

Error correction

We consider the added cost of *reading* syndromes, but not processing them.

(Under the same physical assumptions as Gidney–Ekera (2019))


AllPair (c: RAM)
$0.4150\,d$
AllPair (q: GE19)
$0.3112\,d$
$\log_2(\#ops)$
448
384
320
256
192
128
64
0
100
200
300
400
500
600
700
800
900
1,000
d

Algorithm: RandomBucketSearch / `bgj1`

Parameters: t, θ_1

Input: list L of size N

Search:

1. Repeat t times:
2. Pick a random point f.
3. Run AllPairs on $L_f = L \cap Cap(f, \theta_1)$.

Note: Optimal choice of t and θ_1 is based on volume of the intersection of caps of angle θ_1 with centers at distance $\pi/3$.

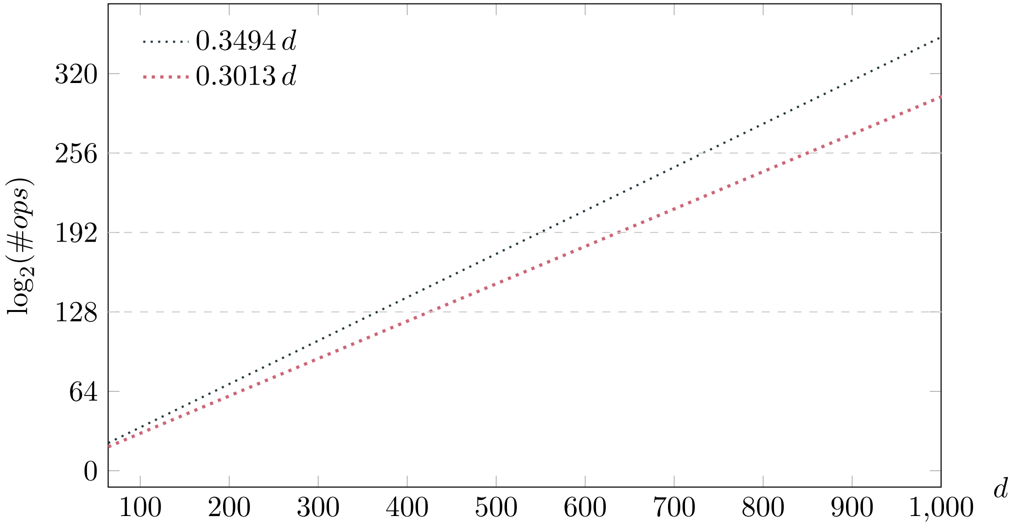
Cost of RandomBucketSearch
List-size preserving case

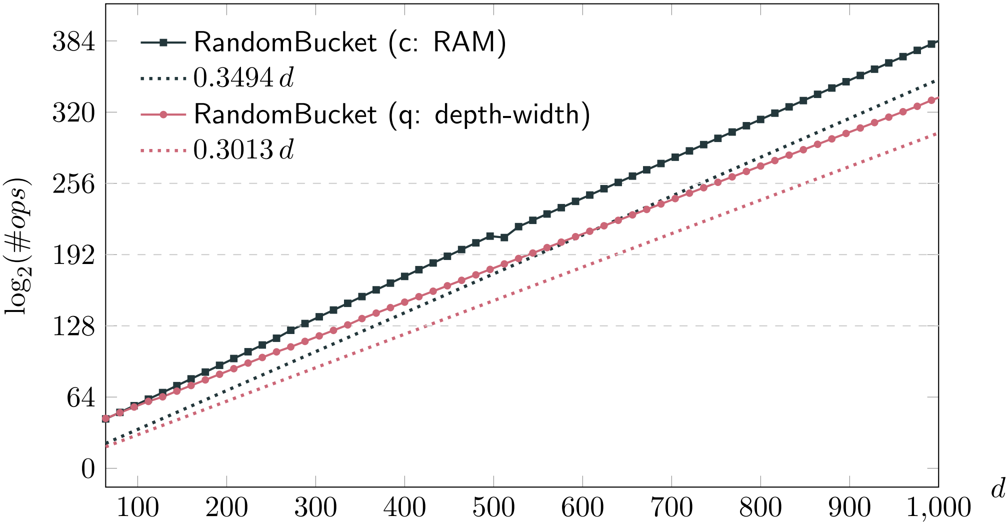
Classical search
Albrecht–Ducas–Herold–Kirshanova–Postlethwaite–Stevens

$$2^{c(d)} \text{ where } c(d) = (0.3494\ldots + o(1))d$$

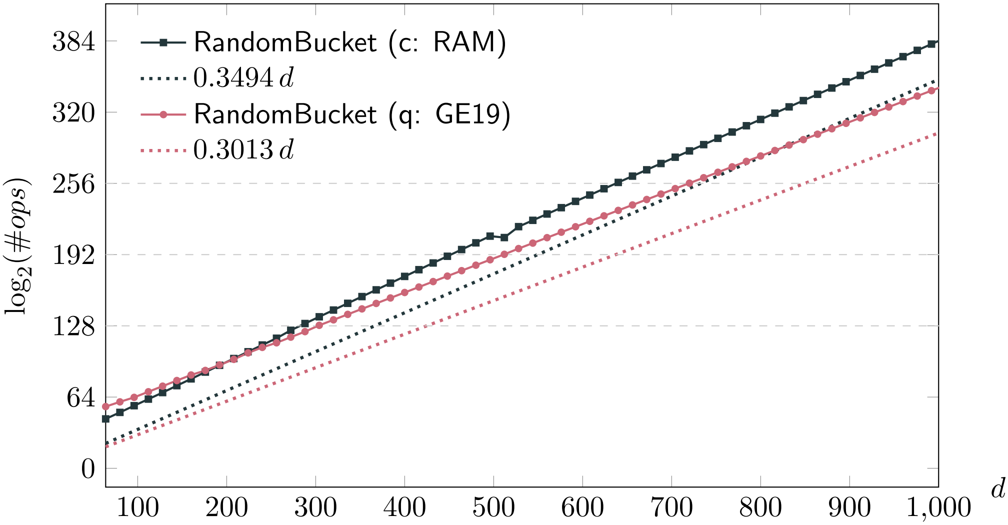
Quantum search

$$2^{c(d)} \text{ where } c(d) = (0.3013\ldots + o(1))d$$


0.3494 d
0.3013 d
320
256
192
128
64
0
log2(#ops)
100
200
300
400
500
600
700
800
900
1,000
d



RandomBucket (c: RAM)
$0.3494\,d$
RandomBucket (q: depth-width)
$0.3013\,d$
$\log_2(\#ops)$
0
64
128
192
256
320
384
d
100
200
300
400
500
600
700
800
900
1,000



RandomBucket (c: RAM)
$0.3494\,d$
RandomBucket (q: GE19)
$0.3013\,d$
$\log_2(\#ops)$
384
320
256
192
128
64
0
100
200
300
400
500
600
700
800
900
1,000
d

Algorithm: ListDecodingSearch / BDGL

Parameters: t, θ_1, θ_2

Input: list L of size N

Setup:
Pick a set of t random points F
Initialize t buckets $\{L_f : f \in F\}$

Fill:

1. For each v in L
2. insert v into L_f if $f \in Cap(v, \theta_2)$

Query:

1. For each v in L
2. $F_i = F \cap Cap(v, \theta_1)$
3. Run AllPairs on $L_F = \coprod\{L_f : f \in F_i\}$.

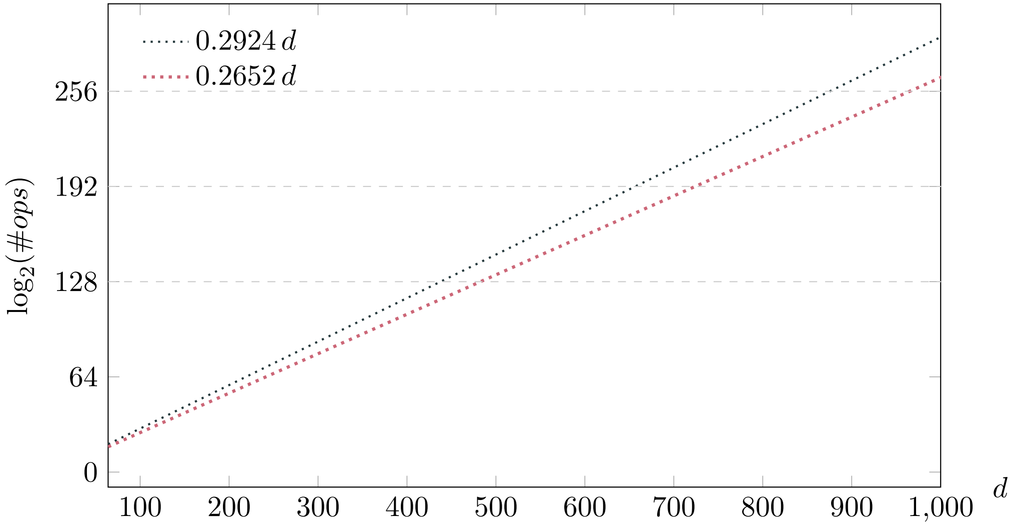
Cost of ListDecodingSearch / BDGL

Classical search
Becker–Ducas–Gama–Laarhoven:

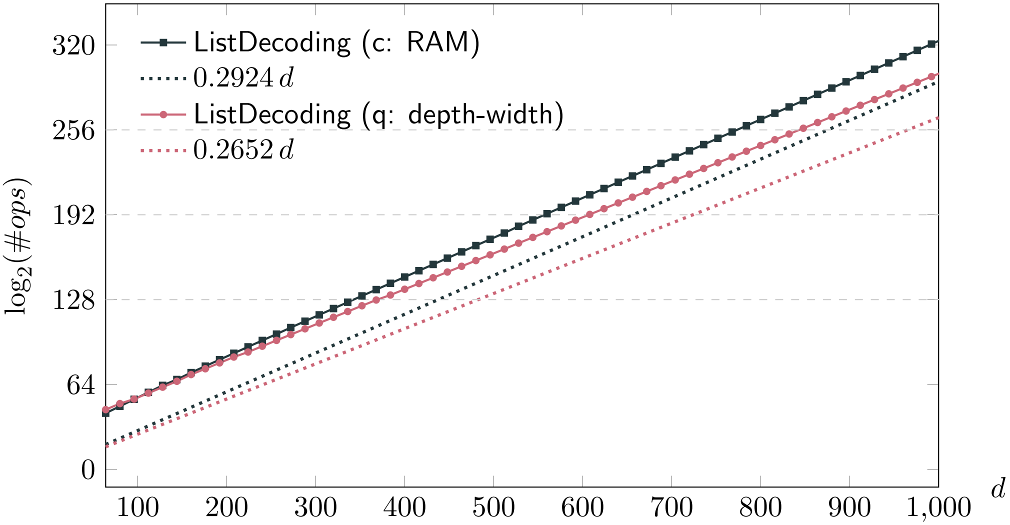
$$2^{c(d)} \text{ where } c(d) = (0.2924\ldots + o(1))d$$

Quantum search
Laarhoven:

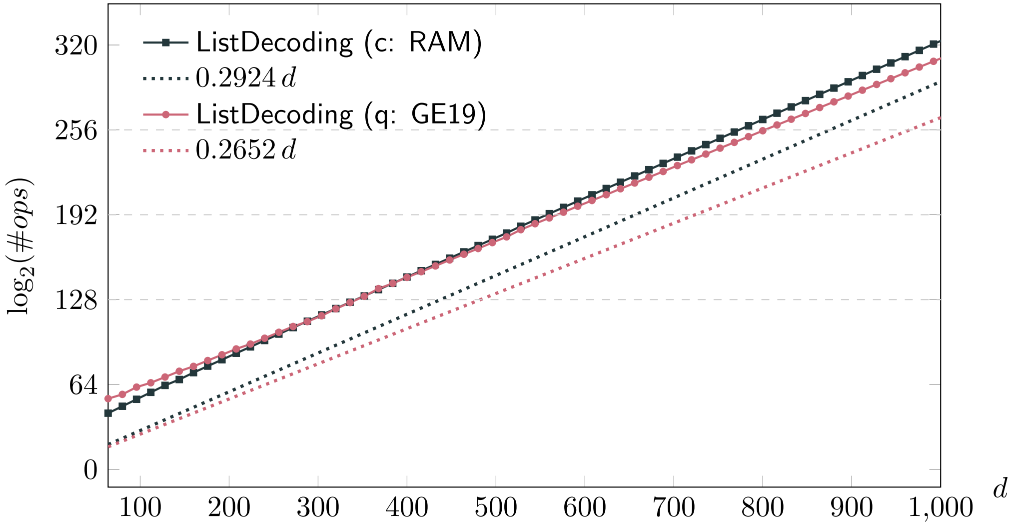
$$2^{c(d)} \text{ where } c(d) = (0.2652\ldots + o(1))d$$



$0.2924\,d$
$0.2652\,d$
256
192
128
64
0
$\log_2(\#ops)$
100
200
300
400
500
600
700
800
900
1,000
d



ListDecoding (c: RAM)
$0.2924\,d$
ListDecoding (q: depth-width)
$0.2652\,d$
$\log_2(\#ops)$
320
256
192
128
64
0
100
200
300
400
500
600
700
800
900
1,000
d



ListDecoding (c: RAM)
$0.2924\,d$
ListDecoding (q: GE19)
$0.2652\,d$
$\log_2(\#ops)$
320
256
192
128
64
0
100
200
300
400
500
600
700
800
900
1,000
d

qRAM

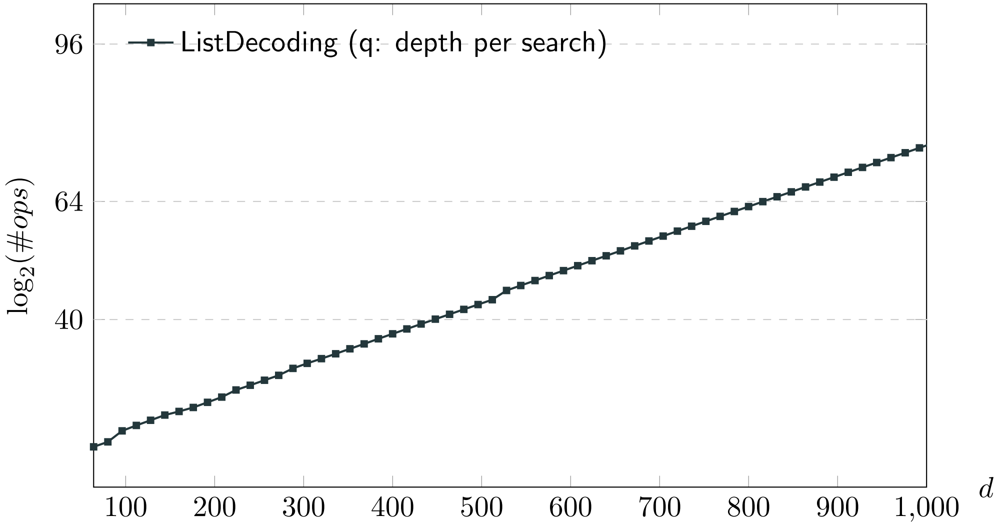
- Known constructions have some cost that grows like $N^{O(1)}$.
- qRAM computations are not necessarily “localizable”.

Error correction overhead

- Cost of processing syndromes
- Cost of state distillation
- Locality constraints introduced by code
- Probability of failure from logical errors

Barriers to a quantum speedup

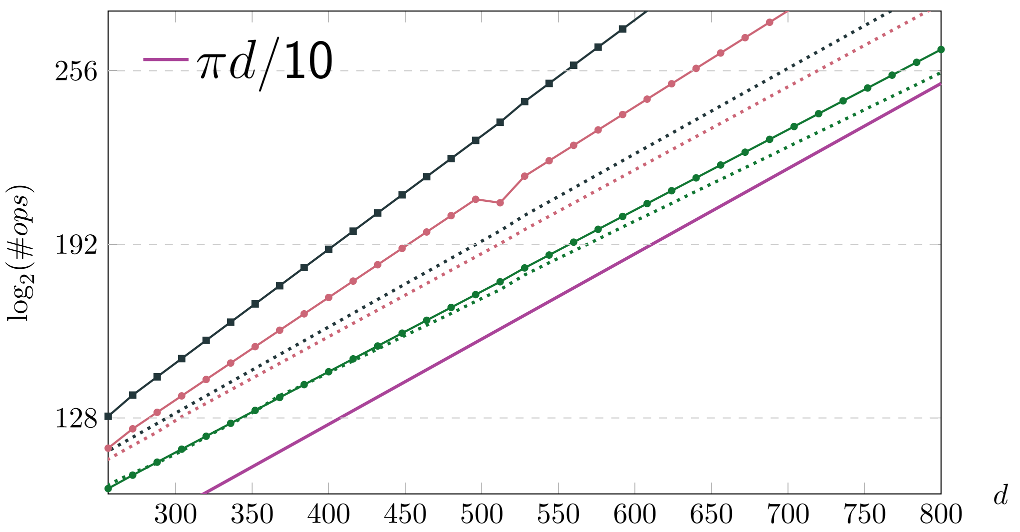
Poor parallelization


ListDecoding (q: depth per search)
96
64
40
log2(#ops)
100
200
300
400
500
600
700
800
900
1,000
d

Barriers to a quantum speedup

Cost underestimates

- "Idealized proposition":
  $P/\gamma \leq |g| \leq \gamma P; \quad \Pr[\text{success}] \geq 1/8$.
- Use of $\mathbf{G}(\mathbf{H}, f)$.
  "Run AllPairs on $L_F = \coprod\{B_f : f \in F_i\}$."


$\pi d/10$
$\log_2(\#ops)$
256
192
128
300
350
400
450
500
550
600
650
700
750
800
d