Estimating the Hidden Overheads in the BDGL Lattice Sieving Algorithm

Léo Ducas

Centrum Wiskunde & Informatica, Amsterdam
Leiden University, Mathematical Institute

PQCrypto, September 28–30, 2022

Lattice Sieving

Lattice Sieving [AKS01, NV08, MV10, ..., BDGL16]

A core algorithm for lattice cryptanalysis

- Lattice Sieving solves the Shortest Vector Problem (SVP)
- It is used inside BKZ for strong lattice reduction

Lattice Sieving

[AKS01, NV08, MV10, ..., BDGL16]

A core algorithm for lattice cryptanalysis

- Lattice Sieving solves the Shortest Vector Problem (SVP)
- It is used inside BKZ for strong lattice reduction

The Central task

- Given $N = 2^{.2075d+o(d)}$ random points on the sphere of $\mathbb{R}^d$
- Find all close-by pairs: $\vec{v}, \vec{w}$ s.t. $\langle \vec{v}, \vec{w} \rangle \leq \frac{1}{2}$

Lattice Sieving [AKS01, NV08, MV10, ..., BDGL16]

A core algorithm for lattice cryptanalysis

- Lattice Sieving solves the Shortest Vector Problem (SVP)
- It is used inside BKZ for strong lattice reduction

The Central task

- Given $N = 2^{.2075d+o(d)}$ random points on the sphere of $\mathbb{R}^d$
- Find all close-by pairs: $\vec{v}, \vec{w}$ s.t. $\langle \vec{v}, \vec{w} \rangle \leq \frac{1}{2}$

Complexity

- Naively: [AKS01, NV08, MV10]

$$O(N^2) = 2^{.415d+o(d)}$$

- Locality Sensitive Hashing (LSH): [Laarhoven, BDGL16]

$$2^{.292d+o(d)}$$

Diving in the small $o(d)$

Concrete security estimation requires more than asymptotics

e.g. for PQC NIST standardization

The concrete analysis of [AGPS20]

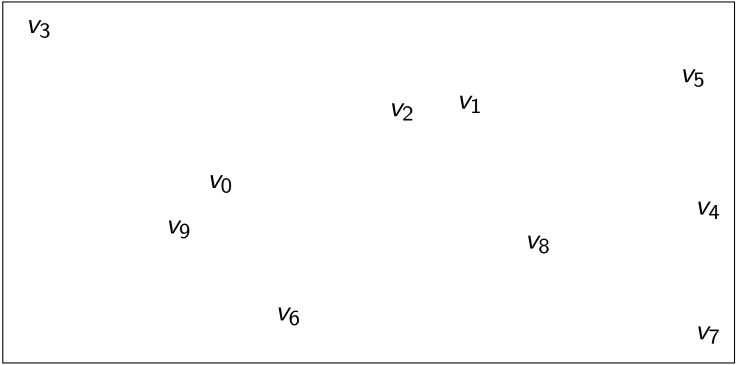
- Replaced asymptotic by exact formula
- Costed each elementary step in term of gates
- Used by NIST candidates to substanciate security claims

Remaining Innacuracies

- Overcosted some steps of BDGL, as shown by [Mat22]
- **Idealized Quality of List-Decodable Codes** (this talk) — Q2 of Kyber Spec, Sec 5.3
- And others... — Q1,3,4,5 of Kyber Spec, Sec 5.3

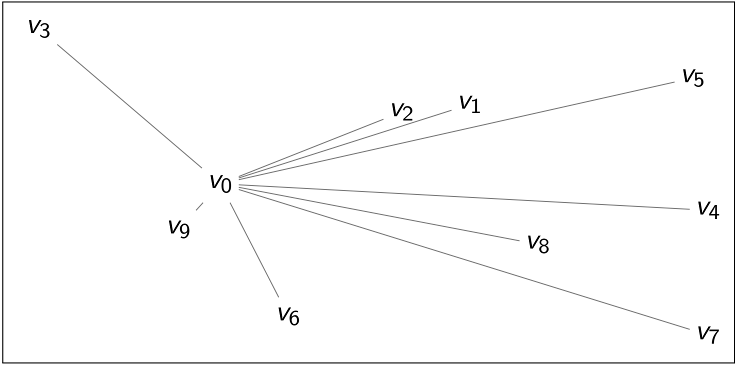
Naive Close Pair Search

Compute all distances and check



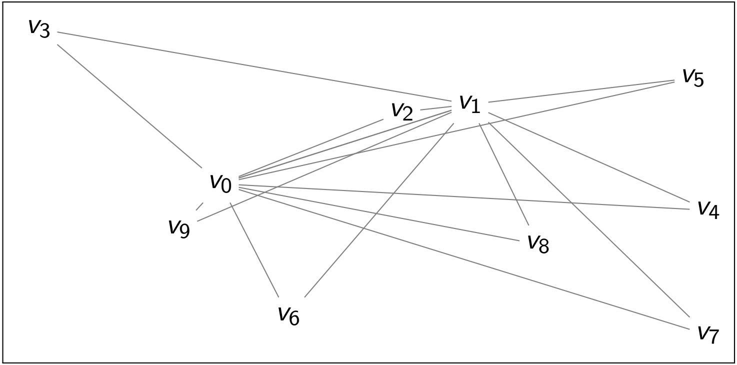
Naive Close Pair Search

Compute all distances and check



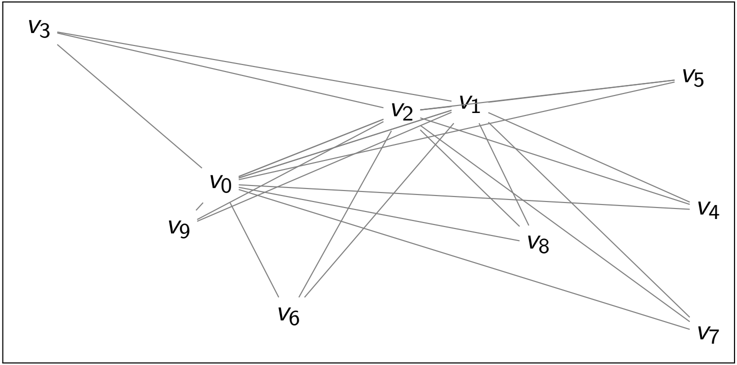
Naive Close Pair Search

Compute all distances and check



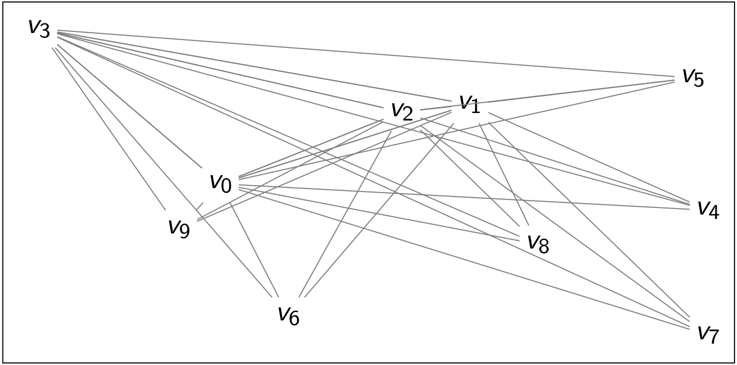
Naive Close Pair Search

Compute all distances and check



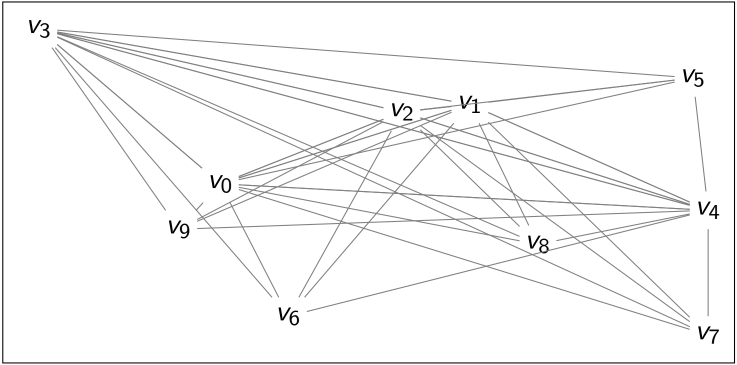
Naive Close Pair Search

Compute all distances and check



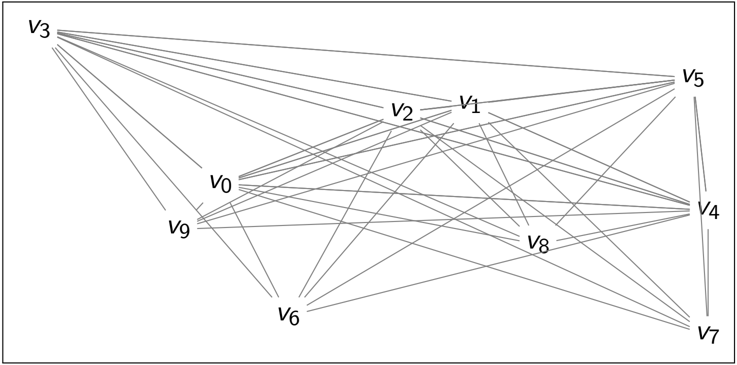
Naive Close Pair Search

Compute all distances and check



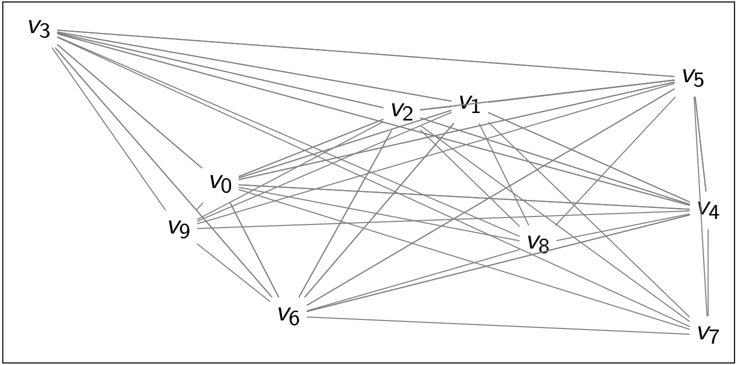
Naive Close Pair Search

Compute all distances and check



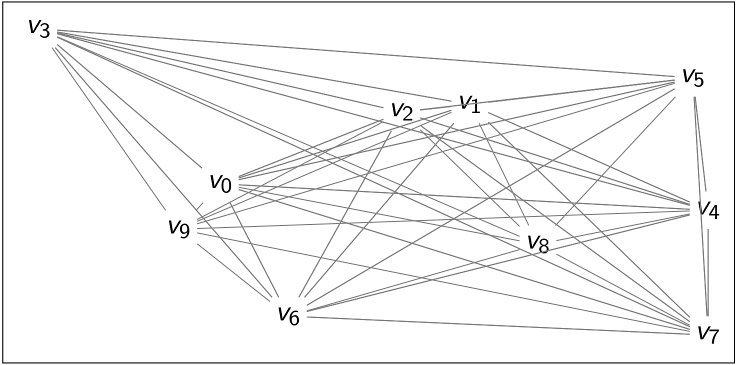
Naive Close Pair Search

Compute all distances and check



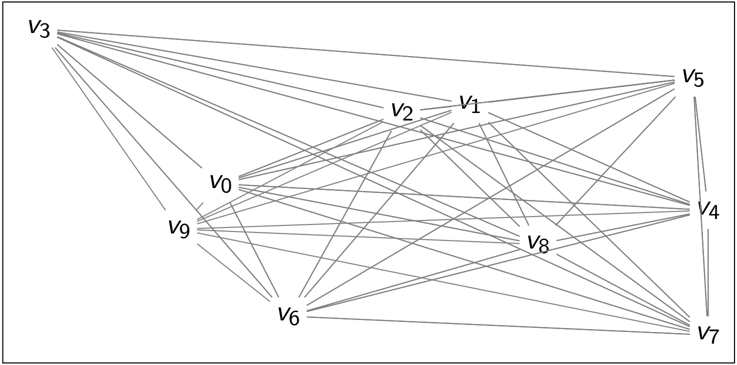
Naive Close Pair Search

Compute all distances and check



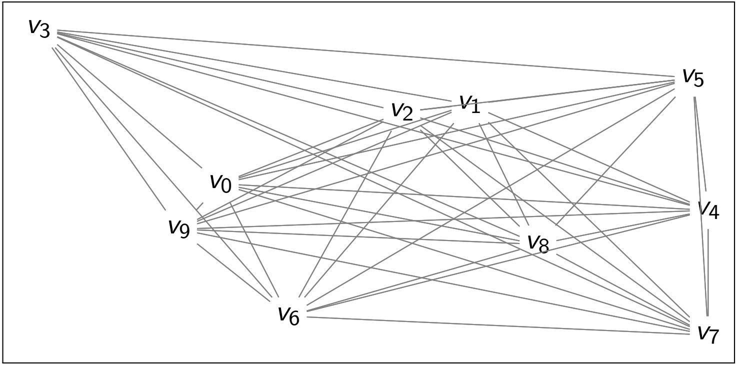
Naive Close Pair Search

Compute all distances and check



Naive Close Pair Search

Compute all distances and check



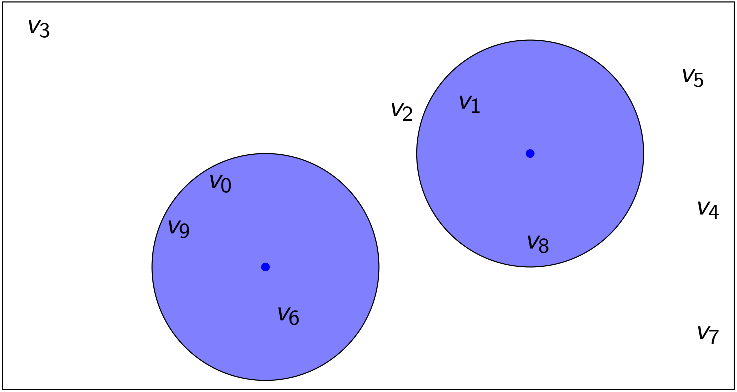
for a cost of $O(N^2)$.

BDGL Lattice Sieving

(Becker-D.-Gama-Laarhoven, SODA 2016)

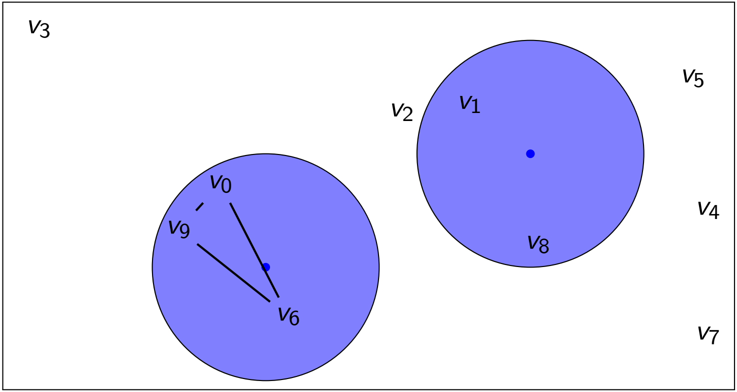
Locality Sensitive Filtering

Make local buckets, and search within them



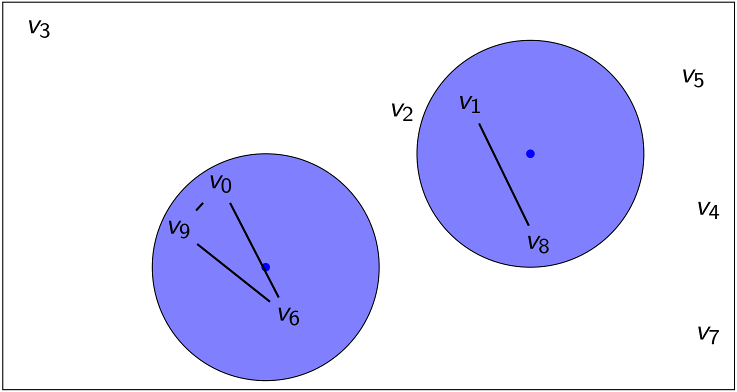
Locality Sensitive Filtering

Make local buckets, and search within them



Locality Sensitive Filtering

Make local buckets, and search within them



All kinds of Shapes, and Time-Memory-Success trade-offs.
[Laarhoven, BGJ15, BDGL16]

BDGL Idealized

1. Filters are spherical caps
2. Filter centers are **efficiently list-decodable**
3. Filter centers are uniform and **independent**

Complexity:

$$T = \sqrt{3/2}^{\,n+o(n)} \qquad M = \sqrt{4/3}^{\,n+o(n)}$$

Requirements 2 and 3 in tensions

To be efficiently list-decodable, the centers need some structure...

BDGL, instantiated with Random Product Codes

The set of Filters centers F is a direct product:

$$F = F_1 \times F_2 \times \cdots \times F_k$$

where each F_i has uniform independent elements in dimension n/k.

Cost Overhead for List-Decoding

$$|F|^{1/k} + \text{number of passing filters}$$

Probability Overhead Lack of Independance [BDGL16, Thm. 5.1]

For $k = O(\log n)$ the success probability loss compared to idealized is at most $2^{\tilde{O}(\sqrt{n})}$

This Work

The Problem Considered in This Work

Concrete Estimation of the Probability Overhead

- A concrete analytic approach ? No clue how . . .
- Run the algorithm and see ? Infeasible in large dim . . .

The Problem Considered in This Work

Concrete Estimation of the Probability Overhead

- A concrete analytic approach ? — No clue how ...
- Run the algorithm and see ? — Infeasible in large dim ...

Focused Experiments on Random Product Codes

- Accelerate Intersected List-Decoding
- Conditional Sampling

The Problem Considered in This Work

Concrete Estimation of the Probability Overhead

- A concrete analytic approach ? No clue how . . .
- Run the algorithm and see ? Infeasible in large dim . . .

Focused Experiments on Random Product Codes

- Accelerate Intersected List-Decoding
- Conditional Sampling

Concrete Results

- Time overhead of about 2^6 in dimension $n = 380$
- Costly and limited trade-off with Memory Overhead

What We Want to Measure ?

- For a random close pair v, w, and a random product code F
- What is the probability that $\exists f \in F$ close to both v and w ?

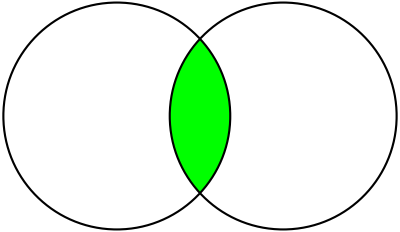
Naive Solution

- Sample a close pair v, w and an RPC F
- List-decode F around v and w
- Check if intersection is non-empty

Techniques

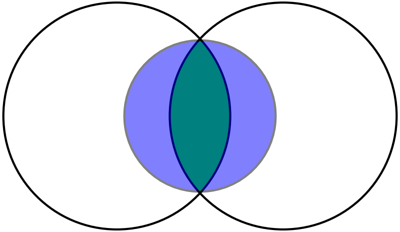
Trick 1: Intersected List-Decoding

Is there a point $f \in F$ in this intersection ?



Trick 1: Intersected List-Decoding

Is there a point $f \in F$ in this intersection ?



Faster Solution

- Sample a close pair v, w and an RPC F
- List decode around $m = (v + w)/2$, **with a smaller radius**
- In that list, check if a point is in the intersection

Toward Trick 2: A Needle in a Haystack

For each $f \in F$ consider the event that f leads to finding v, w

$$E_f : w \in f \wedge v \in f$$

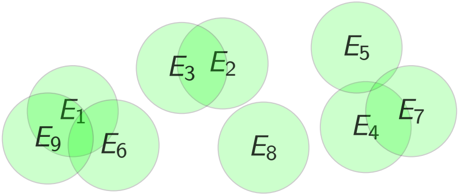
We want to estimate $S = \mathbb{P}[\bigcup_{f \in F} E_f]$.

Note: All event E_f have the same and known probability, but are not independent.

S is very small in the regime of interest

- Number of samples for a fixed relative error is $\Theta(1/S)$.
- In the low memory regime, $S = 2^{-0.084n+o(n)}$

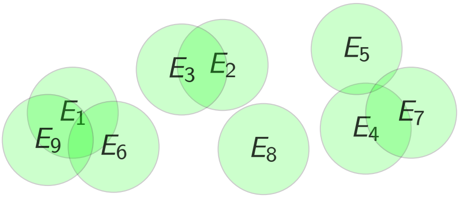
Trick 2: Focus on Successes



A conditioned Experiment

- Sample F
- Choose a uniform $f \in F$
- Sample close pair v, w conditioned on E_f
- Return $i \geq 1$, the number of successful events E_f

Trick 2: Focus on Successes



A conditioned Experiment

- Sample F
- Choose a uniform $f \in F$
- Sample close pair v, w conditioned on E_f
- Return $i \geq 1$, the number of successful events E_f

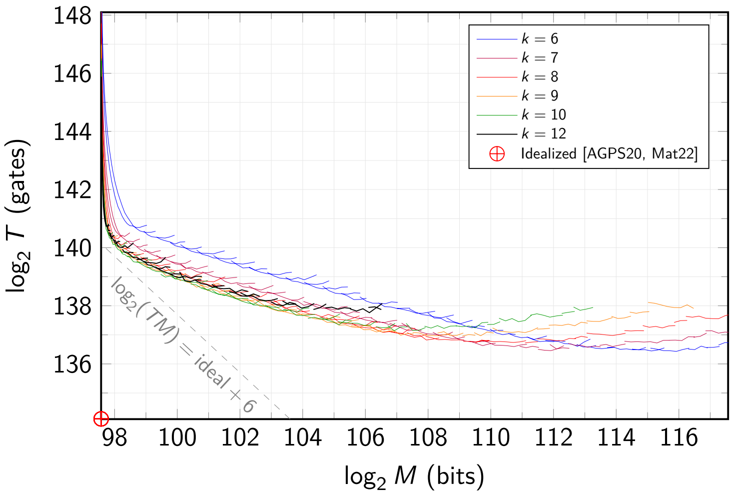
Lemma (Conditioned Estimation)

$$\frac{\mathbb{P}[\bigcup E_f]}{\sum \mathbb{P}[E_f]} = \mathbb{E}\left[\frac{1}{i}\right]$$

Convergence is much faster !

Results

Results in dimension 384



- Sieve Time-Memory cost 2^6 more than previously estimated

Mitigating the Overheads

This is an overhead on a subroutine

- In full attacks: many BDGL calls, with various dimension
- One can play with more memory in smaller dimensions as

$$T_{\text{total}} = \sum T_i \quad M_{\text{total}} = \max M_i$$

E.g. inside progressive-BKZ, one may save back a factor $2^{0.9}$.

Closing Remarks

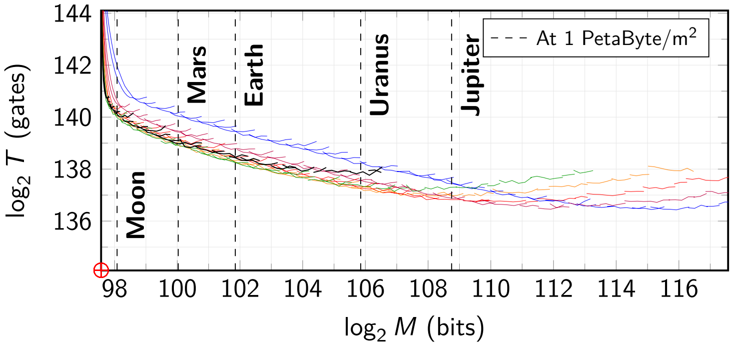
+5 bits for lattice schemes targetting NIST level 1 ?

- Depends on memory cost model *That's no moon!*
- Maybe a bit less for Guo-Johannson / Matzof attack

Closing Remarks

+5 bits for lattice schemes targetting NIST level 1 ?

- Depends on memory cost model *That's no moon!*
- Maybe a bit less for Guo-Johannson / Matzof attack



Closing Remarks

+5 bits for lattice schemes targetting NIST level 1 ?

- Depends on memory cost model *That's no moon!*
- Maybe a bit less for Guo-Johannson / Matzof attack

Hard to systematize

- Getting those curve is costly 40 core-days for dim 384
- Not feasible in an estimator

? An analytic estimate would be much preferable !

Alternative instantiation of BDGL16

? Can we do better than Random Product Codes ?