SIEVING FOR CODES: FROM GJN TO HASH-BASED AND RPC-BASED APPROACHES

February 2024, ATTACC workshop, Germany

presenting: Simona Etinski (CWI)

based on joint work with: Léo Ducas (CWI, LEI), Andre Esser (TII), and Elena Kirshanova (TII, IKBFU)

Motivation: Sieving is a well-known and widely used technique in the lattice-based cryptography.

Motivation: Sieving is a well-known and widely used technique in the lattice-based cryptography.

Goal: Make the sieving ”work” for codes.

Motivation: Sieving is a well-known and widely used technique in the lattice-based cryptography.

Goal: Make the sieving ”work” for codes.

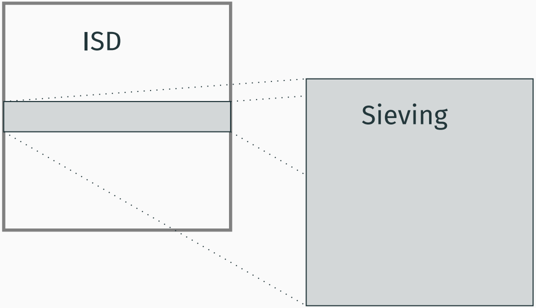
The idea of adapting the sieving to **information set decoding** framework was introduced in [GJN23][1].

[1] Qian Guo, Thomas Johansson, and Vu Nguyen. A New Sieving-Style Information-Set Decoding Algorithm. 2023.

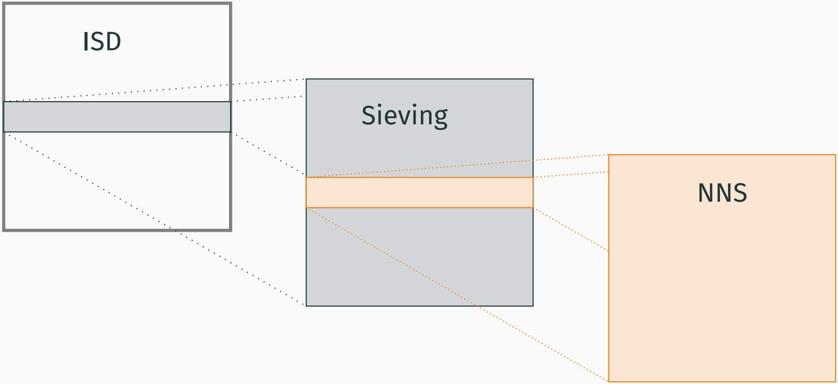
Sieving ISD framework

ISD

Sieving ISD framework



Sieving ISD framework



SIEVING ALGORITHM

(Informal) problem definition

Given a list of N vectors in $\mathcal{S}_w^n$, find N codewords in $\mathcal{C} \cap \mathcal{S}_w^n$.

(Informal) problem definition

Given a list of N vectors in $\mathcal{S}_w^n$, find N codewords in $\mathcal{C} \cap \mathcal{S}_w^n$.

Remarks on notation:

(Informal) problem definition

Given a list of N vectors in $\mathcal{S}_w^n$, find N codewords in $\mathcal{C} \cap \mathcal{S}_w^n$.

Remarks on notation:

- $\mathcal{S}_w^n$ - a sphere of radius w in $\mathbb{F}_2^n$;

(Informal) problem definition

Given a list of N vectors in $\mathcal{S}_w^n$, find N codewords in $\mathcal{C} \cap \mathcal{S}_w^n$.

Remarks on notation:

- $\mathcal{S}_w^n$ - a sphere of radius w in $\mathbb{F}_2^n$;
- $\mathcal{C} \subseteq \mathbb{F}_2^n$ - an $[n, k]$ binary linear code.

Algorithm Sieving

Input : $\mathcal{C}$ - $[n, k]$ binary linear code, w - weight, N - output size,

Algorithm Sieving

Input : $\mathcal{C}$ - [n, k] binary linear code, w - weight, N - output size,
$\mathcal{C}_f$ - bucket centers, α - bucketing parameter.

Algorithm Sieving

Input : $\mathcal{C}$ - $[n, k]$ binary linear code, w - weight, N - output size,
$\mathcal{C}_f$ - bucket centers, α - bucketing parameter.

Output: $\mathcal{L}$ - list of codewords $\mathbf{c} \in \mathcal{C} \cap \mathcal{S}_w^n$ of size $|\mathcal{L}| = N$.

Algorithm Sieving

Input : $\mathcal{C}$ - $[n, k]$ binary linear code, w - weight, N - output size,
$\mathcal{C}_f$ - bucket centers, α - bucketing parameter.

Output: $\mathcal{L}$ - list of codewords $\mathbf{c} \in \mathcal{C} \cap \mathcal{S}_w^n$ of size $|\mathcal{L}| = N$.

Sample a tower of codes $\{\mathbb{F}_2^n = \mathcal{C}_0, \ldots, \mathcal{C}_{n-k} = \mathcal{C}\}$.

Algorithm Sieving

Input : $\mathcal{C}$ - $[n, k]$ binary linear code, w - weight, N - output size,
$\mathcal{C}_f$ - bucket centers, α - bucketing parameter.

Output: $\mathcal{L}$ - list of codewords $\mathbf{c} \in \mathcal{C} \cap \mathcal{S}_w^n$ of size $|\mathcal{L}| = N$.

Sample a tower of codes $\{\mathbb{F}_2^n = \mathcal{C}_0, \ldots, \mathcal{C}_{n-k} = \mathcal{C}\}$.

Sample N distinct vectors from $\mathcal{S}_w^n$ and set as the initial $\mathcal{L}'$.

Algorithm Sieving

Input : $\mathcal{C}$ - $[n, k]$ binary linear code, w - weight, N - output size,
$\mathcal{C}_f$ - bucket centers, α - bucketing parameter.

Output: $\mathcal{L}$ - list of codewords $\mathbf{c} \in \mathcal{C} \cap \mathcal{S}_w^n$ of size $|\mathcal{L}| = N$.

Sample a tower of codes $\{\mathbb{F}_2^n = \mathcal{C}_0, \ldots, \mathcal{C}_{n-k} = \mathcal{C}\}$.

Sample N distinct vectors from $\mathcal{S}_w^n$ and set as the initial $\mathcal{L}'$.

for $i = 1$ to $n - k$ **do**

Algorithm Sieving

Input : $\mathcal{C}$ - $[n, k]$ binary linear code, w - weight, N - output size,
$\mathcal{C}_f$ - bucket centers, α - bucketing parameter.

Output: $\mathcal{L}$ - list of codewords $\mathbf{c} \in \mathcal{C} \cap \mathcal{S}_w^n$ of size $|\mathcal{L}| = N$.

Sample a tower of codes $\{\mathbb{F}_2^n = \mathcal{C}_0, \ldots, \mathcal{C}_{n-k} = \mathcal{C}\}$.

Sample N distinct vectors from $\mathcal{S}_w^n$ and set as the initial $\mathcal{L}'$.

for $i = 1$ to $n - k$ **do**

    Invoke near neighbour search oracle NNS$(\mathcal{L}, \mathcal{C}_f, \alpha)$ to obtain $\mathcal{P}$.

Algorithm Sieving

Input : $\mathcal{C}$ - $[n, k]$ binary linear code, w - weight, N - output size,
$\mathcal{C}_f$ - bucket centers, α - bucketing parameter.

Output: $\mathcal{L}$ - list of codewords $\mathbf{c} \in \mathcal{C} \cap \mathcal{S}_w^n$ of size $|\mathcal{L}| = N$.

Sample a tower of codes $\{\mathbb{F}_2^n = \mathcal{C}_0, \ldots, \mathcal{C}_{n-k} = \mathcal{C}\}$.

Sample N distinct vectors from $\mathcal{S}_w^n$ and set as the initial $\mathcal{L}'$.

for $i = 1$ to $n - k$ **do**

  Invoke near neighbour search oracle NNS$(\mathcal{L}, \mathcal{C}_f, \alpha)$ to obtain $\mathcal{P}$.

  for $(x, y) \in \mathcal{P}$ **do**

Algorithm Sieving

Input : $\mathcal{C}$ - $[n, k]$ binary linear code, w - weight, N - output size,
$\mathcal{C}_f$ - bucket centers, α - bucketing parameter.

Output: $\mathcal{L}$ - list of codewords $\mathbf{c} \in \mathcal{C} \cap \mathcal{S}_w^n$ of size $|\mathcal{L}| = N$.

Sample a tower of codes $\{\mathbb{F}_2^n = \mathcal{C}_0, \ldots, \mathcal{C}_{n-k} = \mathcal{C}\}$.

Sample N distinct vectors from $\mathcal{S}_w^n$ and set as the initial $\mathcal{L}'$.

for $i = 1$ to $n - k$ **do**

    Invoke near neighbour search oracle $\text{NNS}(\mathcal{L}, \mathcal{C}_f, \alpha)$ to obtain $\mathcal{P}$.

    for $(\mathbf{x}, \mathbf{y}) \in \mathcal{P}$ **do**

        if $(\mathbf{x} + \mathbf{y}) \in \mathcal{C}_i$ **then**

            Add $(\mathbf{x} + \mathbf{y})$ to $\mathcal{L}$.

        end

Algorithm Sieving

Input : $\mathcal{C}$ - $[n, k]$ binary linear code, w - weight, N - output size,
$\mathcal{C}_f$ - bucket centers, α - bucketing parameter.

Output: $\mathcal{L}$ - list of codewords $\mathbf{c} \in \mathcal{C} \cap \mathcal{S}_w^n$ of size $|\mathcal{L}| = N$.

Sample a tower of codes $\{\mathbb{F}_2^n = \mathcal{C}_0, \ldots, \mathcal{C}_{n-k} = \mathcal{C}\}$.

Sample N distinct vectors from $\mathcal{S}_w^n$ and set as the initial $\mathcal{L}'$.

for $i = 1$ to $n - k$ **do**

  Invoke near neighbour search oracle $\mathrm{NNS}(\mathcal{L}, \mathcal{C}_f, \alpha)$ to obtain $\mathcal{P}$.

  for $(\mathbf{x}, \mathbf{y}) \in \mathcal{P}$ **do**

    if $(\mathbf{x} + \mathbf{y}) \in \mathcal{C}_i$ **then**

      Add $(\mathbf{x} + \mathbf{y})$ to $\mathcal{L}$.

    end

    Discard some elements if $|\mathcal{L}'| > N$ and set $\mathcal{L}' \leftarrow \mathcal{L}$.

  end

end

return $\mathcal{L}$

Sieving

The **running time** and the **memory**:

$$T_{\text{SIEVING}} = \tilde{\mathcal{O}}(T_{\text{NNS}}), \quad M_{\text{SIEVING}} = \tilde{\mathcal{O}}(M_{\text{NNS}}).$$

Remarks

Heuristics: The input list elements at any step of the sieving algorithm behave like **uniformly random** and **independent** vectors from the sphere $\mathcal{S}_w^n$.

Remarks

Choice of N: We choose N such that there exist N distinct codewords of weight w in $\mathcal{C}$ and that we maintain the list size in each iteration, namely

$$\frac{c\binom{n}{w}}{\binom{w}{w/2}\binom{n-w}{w/2}} \leq N \leq \binom{n}{w} \cdot 2^{k-n}.$$

NEAR NEIGHBOR SEARCH ALGORITHMS

Previous work

[MO15][2], [BM18][3], etc. and Kévin Carrier's thesis[4] explored **near neighbor search** in the coding setting.

[2] Alexander May and Ilya Ozerov. "On Computing Nearest Neighbors with Applications to Decoding of Binary Linear Codes". In: 2015.

[3] Leif Both and Alexander May. "Decoding Linear Codes with High Error Rate and Its Impact for LPN Security". In: ed. by Tanja Lange and Rainer Steinwandt. 2018.

[4] Kévin Carrier. "Recherche de Presque-Collisions pour le Décodage et la Reconnaissance de Codes Correcteurs. (Near-collisions finding problem for decoding and recognition of error correcting codes)". PhD thesis. 2020.

Previous work

[MO15], [BM18], etc. and Kévin Carrier's thesis explored **near neighbor search** in the coding setting.

In the lattice-based setting, **sieving** was successfully combined with **locality-sensitive hashing (filtering)** introduced in [BDGL15][2].

[2]Anja Becker et al. New directions in nearest neighbor searching with applications to lattice sieving. 2015.

(Informal) problem definition

Given a list $\mathcal{L}$ of N vectors of weight w, return a list of pairs of vectors $(\mathbf{x}, \mathbf{y})$ from $\mathcal{L} \times \mathcal{L}$ that satisfy $|\mathbf{x} + \mathbf{y}| = w$.

(Informal) problem definition

Given a list $\mathcal{L}$ of N vectors of weight w, return a list of pairs of vectors $(\mathbf{x}, \mathbf{y})$ from $\mathcal{L} \times \mathcal{L}$ that satisfy $|\mathbf{x} + \mathbf{y}| = w$.

High-level idea

If two vectors overlap in α positions, they are more likely to be close in space (aka these are ”near neighbors”).

Algorithm Near Neighbour Search

Input : $\mathcal{L}$ - list of of weight w vectors, $\mathcal{C}_{\mathrm{f}}$ - bucket centers,
α - bucketing parameter.

Algorithm Near Neighbour Search

Input : $\mathcal{L}$ - list of of weight w vectors, $\mathcal{C}_\mathrm{f}$ - bucket centers,
α - bucketing parameter.

Output: $\mathcal{P}$ - list of pairs $(\mathbf{x}, \mathbf{y})$ satisfying $|\mathbf{x} + \mathbf{y}| = \mathrm{w}$.

Algorithm Near Neighbour Search

Input : $\mathcal{L}$ - list of of weight w vectors, $\mathcal{C}_f$ - bucket centers,
α - bucketing parameter.

Output: $\mathcal{P}$ - list of pairs $(\mathbf{x}, \mathbf{y})$ satisfying $|\mathbf{x} + \mathbf{y}| = w$.

for $\mathbf{x} \in \mathcal{L}$ **do**

Algorithm Near Neighbour Search

Input : $\mathcal{L}$ - list of of weight w vectors, $\mathcal{C}_f$ - bucket centers, α - bucketing parameter.

Output: $\mathcal{P}$ - list of pairs $(\mathbf{x}, \mathbf{y})$ satisfying $|\mathbf{x} + \mathbf{y}| = w$.

for $\mathbf{x} \in \mathcal{L}$ **do**

  for ValidFilters($\mathcal{C}_f, \alpha, \mathbf{x}$) **do**

Algorithm Near Neighbour Search

Input : $\mathcal{L}$ - list of of weight w vectors, $\mathcal{C}_f$ - bucket centers,
α - bucketing parameter.

Output: $\mathcal{P}$ - list of pairs $(\mathbf{x}, \mathbf{y})$ satisfying $|\mathbf{x} + \mathbf{y}| = w$.

```
for x ∈ L do
    for ValidFilters(C_f, α, x) do
        add x to B_c
    end
end
```

Algorithm Near Neighbour Search

Input : $\mathcal{L}$ - list of of weight w vectors, $\mathcal{C}_f$ - bucket centers, α - bucketing parameter.

Output: $\mathcal{P}$ - list of pairs $(\mathbf{x}, \mathbf{y})$ satisfying $|\mathbf{x} + \mathbf{y}| = w$.

```
for x ∈ L do
    for ValidFilters(C_f, α, x) do
        add x to B_c
    end
end
for x ∈ L do
```

Algorithm Near Neighbour Search

Input : $\mathcal{L}$ - list of of weight w vectors, $\mathcal{C}_\mathsf{f}$ - bucket centers, α - bucketing parameter.

Output: $\mathcal{P}$ - list of pairs $(\mathbf{x}, \mathbf{y})$ satisfying $|\mathbf{x}+\mathbf{y}| = \mathsf{w}$.

```
for x ∈ L do
    for ValidFilters(C_f, α, x) do
        add x to B_c
    end
end

for x ∈ L do
    for c ∈ ValidFilters(C_f, α, x),    y ∈ B_c do
```

Algorithm Near Neighbour Search

Input : $\mathcal{L}$ - list of of weight w vectors, $\mathcal{C}_f$ - bucket centers, α - bucketing parameter.

Output: $\mathcal{P}$ - list of pairs $(\mathbf{x}, \mathbf{y})$ satisfying $|\mathbf{x} + \mathbf{y}| = w$.

for $x \in \mathcal{L}$ **do**
  for ValidFilters$(\mathcal{C}_f, \alpha, \mathbf{x})$ **do**
    add $\mathbf{x}$ to $\mathcal{B}_c$
  end
end

for $x \in \mathcal{L}$ **do**
  for $\mathbf{c} \in$ ValidFilters$(\mathcal{C}_f, \alpha, \mathbf{x})$, $\mathbf{y} \in \mathcal{B}_c$ **do**
    if $|\mathbf{x} + \mathbf{y}| = w$ **then**

Algorithm Near Neighbour Search

Input : $\mathcal{L}$ - list of of weight w vectors, $\mathcal{C}_f$ - bucket centers, α - bucketing parameter.

Output: $\mathcal{P}$ - list of pairs $(\mathbf{x}, \mathbf{y})$ satisfying $|\mathbf{x} + \mathbf{y}| = w$.

```
for x ∈ L do
    for ValidFilters(C_f, α, x) do
        add x to B_c
    end
end
for x ∈ L do
    for c ∈ ValidFilters(C_f, α, x),    y ∈ B_c do
        if |x + y| = w then
            add (x, y) to P
        end
    end
end
return P
```

Near Neighbor Search

The **running time**:

$$T_{\text{NNS}} = \tilde{\mathcal{O}}(N \cdot T_{\text{VALIDFILTERS}}) + \tilde{\mathcal{O}}(N \cdot \mathbb{E}(\text{VALIDFILTERS}) \cdot \mathbb{E}(\mathcal{B})).$$

The **memory**:

$$M_{\text{NNS}} = \tilde{\mathcal{O}}(N \cdot \mathbb{E}(\text{VALIDFILTERS})).$$

GJN, HASH-BASED AND RPC-BASED

Guo, Johansson and Nguyen [GJN] approach[3]

Main idea

For any $\mathbf{x}, \mathbf{y} \in \mathcal{S}_w^n$ satisfying $|\mathbf{x} + \mathbf{y}| = w$, there exists $\mathbf{c} \in \mathcal{S}_{w/2}^n$ such that $|\mathbf{x} \wedge \mathbf{c}| = |\mathbf{y} \wedge \mathbf{c}| = w/2$.

[3] Qian Guo, Thomas Johansson, and Vu Nguyen. A New Sieving-Style Information-Set Decoding Algorithm. 2023.

Guo, Johansson and Nguyen [GJN] approach

Main idea

For any $\mathbf{x}, \mathbf{y} \in \mathcal{S}_w^n$ satisfying $|\mathbf{x} + \mathbf{y}| = w$, there exists $\mathbf{c} \in \mathcal{S}_{w/2}^n$ such that $|\mathbf{x} \wedge \mathbf{c}| = |\mathbf{y} \wedge \mathbf{c}| = w/2$.

*Initially, the approach was not presented in the locality-sensitive filtering fashion, yet it aligns with the framework.

Guo, Johansson and Nguyen [GJN] approach

Parameters:

$$\mathcal{C}_{\mathsf{f}} = \mathcal{S}^{\mathsf{n}}_{\mathsf{w}/2}, \quad \alpha = \mathsf{w}/2.$$

Guo, Johansson and Nguyen [GJN] approach

Parameters:

$$\mathcal{C}_f = \mathcal{S}^n_{w/2}, \quad \alpha = w/2.$$

Valid Filters Subroutine

For each $\mathbf{x} \in \mathcal{L}$, returns all $\mathbf{c} \in \mathcal{S}^n_{w/2}$ such that $|\mathbf{x} \wedge \mathbf{c}| = w/2$.

Coded hashing approach (HASH)[3]

Parameters

$$\mathcal{C}_f = \mathcal{S}^n_\alpha \cap \mathcal{C}_\mathcal{H}, \quad \alpha \leq w/2,$$

where $\mathcal{C}_\mathcal{H}$ is $[n, n-r]$ binary linear code.

[3] Léo Ducas et al. Asymptotics and Improvements of Sieving for Codes. 2023.

Coded hashing approach (HASH)[3]

Parameters

$$\mathcal{C}_f = \mathcal{S}^n_\alpha \cap \mathcal{C}_\mathcal{H}, \quad \alpha \leq w/2,$$

where $\mathcal{C}_\mathcal{H}$ is $[n, n-r]$ binary linear code.

Valid Filters Subroutine

For each $\mathbf{x} \in \mathcal{L}$, returns all $\mathbf{c} \in \mathcal{S}^n_\alpha \cap \mathcal{C}_\mathcal{H}$ such that $|\mathbf{x} \wedge \mathbf{c}| = \alpha$.

[3] Léo Ducas et al. Asymptotics and Improvements of Sieving for Codes. 2023.

Random product codes approach (RPC)[4]

Parameters:

$$\mathcal{C}_{\mathcal{H}}^{(i)} \subseteq \mathcal{S}_{v/t}^{n/t}, \quad \mathcal{C}_{\mathcal{H}} = \mathcal{C}_{\mathcal{H}}^{(1)} \times \cdots \times \mathcal{C}_{\mathcal{H}}^{(t)}, \quad \alpha, v \leq w/2 \text{ - to be optimized.}$$

Valid Filters Subroutine

For each $\mathbf{x} = (\mathbf{x}^{(1)}, \ldots \mathbf{x}^{(t)}) \in \mathcal{L}$, returns all $\mathbf{c} = (\mathbf{c}^{(1)}, \ldots \mathbf{c}^{(t)}) \in \mathcal{S}_v^n \cap \mathcal{C}_{\mathcal{H}}$ such that

$$|\mathbf{x}^{(i)} \wedge \mathbf{c}^{(i)}| = \alpha/t \text{ for all } i \in \{1, \ldots, t\}.$$

[4] Léo Ducas et al. Asymptotics and Improvements of Sieving for Codes. 2023.

Memory optimal versions (HASH and RPC memo-opt)[5]

High-level idea

We interleave the bucketing and the checking phase.

[5] Léo Ducas et al. Asymptotics and Improvements of Sieving for Codes. 2023.

Memory optimal versions (HASH and RPC memo-opt)[5]

High-level idea

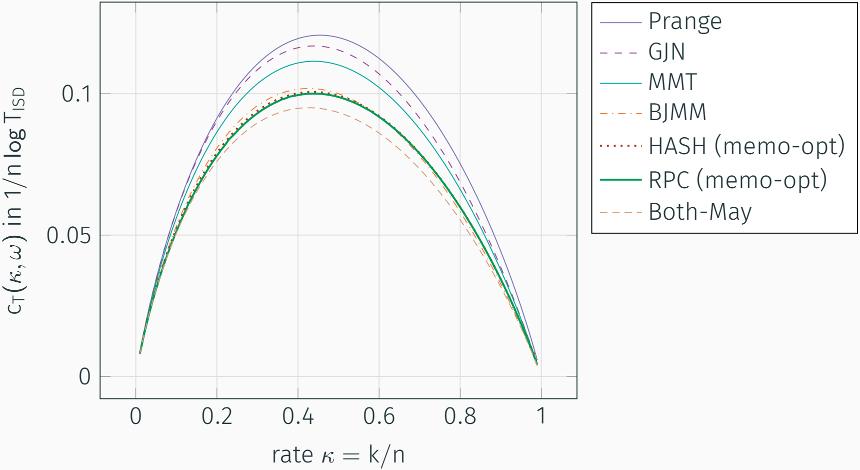
We interleave the bucketing and the checking phase.

Memory optimal approach

The initial set of filters contains $|\mathcal{C}_f|/2^d$ centers but we repeat the algorithm 2^d times.

[5] Léo Ducas et al. Asymptotics and Improvements of Sieving for Codes. 2023.

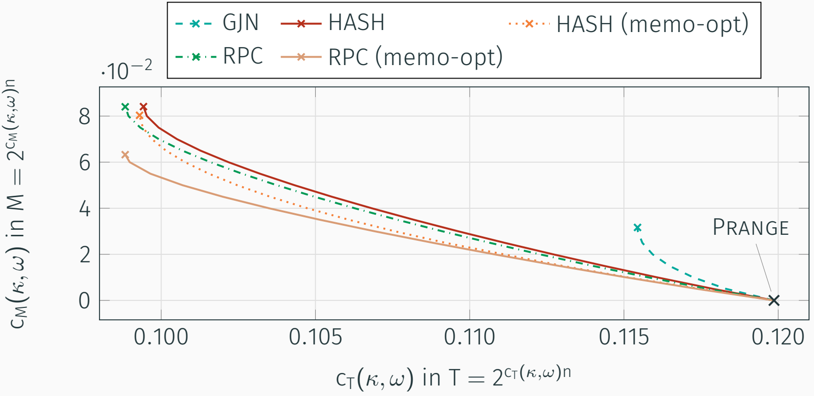
NUMERICAL RESULTS



Runtime exponent for different ISD and SievingISD variants.

Type	Algorithm	κ	$c_T(\kappa,\omega)$	$c_M(\kappa,\omega)$
SievingISD	GJN	0.44	0.1169	0.0279
	HASH	0.44	0.1007	0.0849
	HASH memo-opt	0.44	0.1007	0.0830
	RPC	0.44	0.1001	0.0852
	RPC memo-opt	0.44	0.1001	0.0636
Conventional ISD	Prange	0.45	0.1207	0.0000
	MMT	0.45	0.1116	0.0541
	BJMM	0.43	0.1020	0.0728
	Both-May	0.42	0.0951	0.0754

Table: Worst case running time $2^{c_T(\kappa,\omega)n}$ and corresponding memory usage $2^{c_M(\kappa,\omega)n}$ for different ISD algorithms.



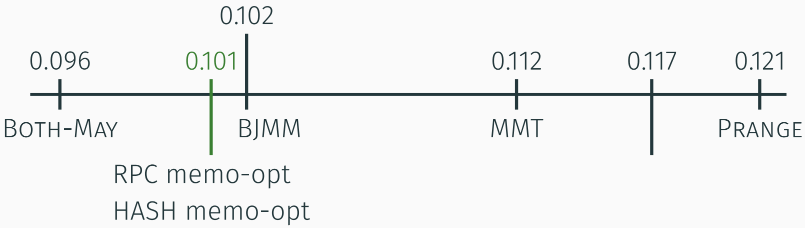
Time-memory trade-off curves of different SievingISD instantiations, for $\kappa = 0.5$ and $\omega = H^{-1}(0.5)$.

CONCLUDING REMARKS

An efficient **sieving-based** algorithm for **codes**.

An efficient **sieving-based** algorithm for **codes**.

→ For the worst case, the efficiency is comparable with BJMM.



In comparison to lattice sieving

A new alignment of the lattice-based and code-based framework (equivalent to [BDGL15]).

In comparison to lattice sieving

A new alignment of the lattice-based and code-based framework (equivalent to [BDGL15]).

Instead of sieving on a full instance, we **sieve** on the **ISD sub-instance**.

In comparison to lattice sieving

A new alignment of the lattice-based and code-based framework (equivalent to [BDGL15]).

Instead of sieving on a full instance, we **sieve** on the **ISD sub-instance**.

Instead of shortening vectors, we iteratively reduce the coset size till we find vectors in the code but we keep their length unchanged.

Open questions

How applicable it is?

Open questions

How applicable it is?

Is it inherently different from the other ISD algorithms?

Thank you for your attention!

eprint

GitHub repo

MCCL fork