ASYMPTOTICS AND IMPROVEMENTS OF SIEVING FOR CODES

May 2024, Eurocrypt 2024

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

Motivation: Sieving is a well-known and widely used technique for attacking decoding problems in **lattice-based** cryptography.

Motivation: Sieving is a well-known and widely used technique for attacking decoding problems in **lattice-based** cryptography.

How well these techniques perform in the **code-based** setting?

Motivation: Sieving is a well-known and widely used technique for attacking decoding problems in **lattice-based** cryptography.

How well these techniques perform in the **code-based** setting?

Goal: Adapt sieving techniques to the **code-based setting** and make them competitive with state-of-the-art algorithms.

SIEVING FOR CODES

Problem definition

Decoding problem, DP(n, k, w)

Given an $[n, k]$ binary linear code $\mathcal{C}$ and a weight w, find a codeword of Hamming weight[1] w.

[1] **Hamming weight,** $|\cdot|$: The number of non-zero entries of a vector.

Information Set Decoding (ISD)

Information set decoding algorithms are the best known generic[2] attacks for the decoding problem.

[2] For certain parameter ranges, statistical decoding performs better. (Kevin Carrier et al. Statistical Decoding 2.0: Reducing Decoding to LPN. Cryptology ePrint Archive, Paper 2022/1000. 2022)

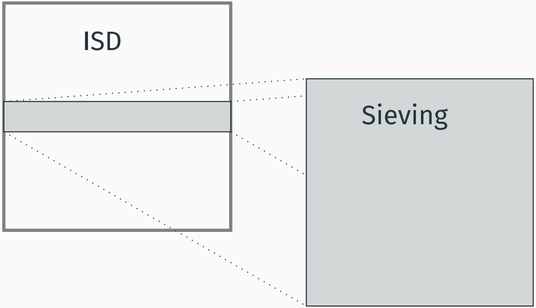
Information Set Decoding (ISD)

Information set decoding algorithms are the best known generic attacks for the decoding problem.

Recently, a new ISD algorithm based using **sieving** as a **subroutine** was proposed in [GJN23][2].

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

Inspiration Part I: [GJN23] approach[3]



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

Inspiration Part I: [GJN23] approach[5]

Provides slight improvements in asymptotic runtime over the baseline algorithm due to Prange[4].

[4] Eugene Prange. “The use of information sets in decoding cyclic codes”. In: IRE Trans. Inf. Theory 8.5 (1962), pp. 5–9.

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

Inspiration Part I: [GJN23] approach[5]

Provides slight improvements in asymptotic runtime over the baseline algorithm due to Prange[4].

Gives very good time-memory trade-offs.

[4]Eugene Prange. “The use of information sets in decoding cyclic codes”. In: IRE Trans. Inf. Theory 8.5 (1962), pp. 5–9.

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

Inspiration Part II: Near-Neighbor Search

In the lattice-based setting, **sieving** was successfully combined with **near-neighbor search**[6][7][8].

[6]Thijs Laarhoven. Sieving for shortest vectors in lattices using angular locality-sensitive hashing. Cryptology ePrint Archive, Paper 2014/744. 2014.

[7]Anja Becker, Nicolas Gama, and Antoine Joux. Speeding-up lattice sieving without increasing the memory, using sub-quadratic nearest neighbor search. Cryptology ePrint Archive, Paper 2015/522. 2015.

[8]Anja Becker, Léo Ducas, et al. New directions in nearest neighbor searching with applications to lattice sieving. 2015.

Inspiration Part II: Near-Neighbor Search

In the lattice-based setting, **sieving** was successfully combined with **near-neighbor search**.

[MO15][6], [BM18][7], etc. and Kévin Carrier's thesis[8] explored **near-neighbor search** in the coding setting.

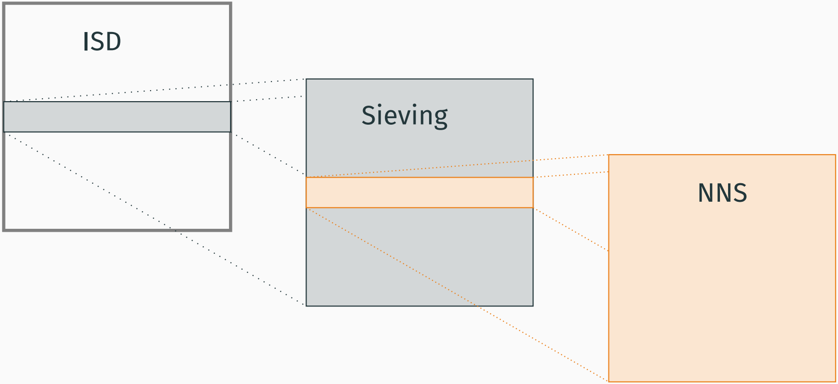
---

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

[7] 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.

[8] 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.

Our generalization



NEAR-NEIGHBOR SEARCH

DEFINITIONS AND NOTATION

Sphere of radius p: $\mathcal{S}_p^m := \{x \in \mathbb{F}_2^m : |x| = p\}.$

Definitions and notation

Sphere of radius p: $\mathcal{S}_p^m := \{x \in \mathbb{F}_2^m : |x| = p\}.$

Near neighbors: $\big\{(x, y) \in \mathcal{S}_p^m \times \mathcal{S}_p^m : |x + y| = p\big\},$

where $+$ denotes bitwise XOR.

Definitions and notation

Sphere of radius p: $$\mathcal{S}_p^m := \{\mathbf{x} \in \mathbb{F}_2^m : |\mathbf{x}| = p\}.$$

Near neighbors: $$\big\{(\mathbf{x}, \mathbf{y}) \in \mathcal{S}_p^m \times \mathcal{S}_p^m : |\mathbf{x} \wedge \mathbf{y}| = p/2\big\},$$

where $\wedge$ denotes bitwise AND.

Near-Neighbor Search Problem

Near-Neighbor Search (NNS), NNS($\mathcal{L}$, p)

Given a target weight p and an input list $\mathcal{L} \subseteq \mathcal{S}_p^m$, find all pairs $(\mathbf{x}, \mathbf{y}) \in \mathcal{L} \times \mathcal{L}$ satisfying $|\mathbf{x} + \mathbf{y}| = p$.

Near-Neighbor Search Problem

Near-Neighbor Search (NNS), NNS($\mathcal{L}$, p)

Given an input list $\mathcal{L} \subseteq \mathcal{S}_p^m$ and a target weight p, find all pairs $(\mathbf{x}, \mathbf{y}) \in \mathcal{L} \times \mathcal{L}$ satisfying $|\mathbf{x} \wedge \mathbf{y}| = p/2$.

Near-Neighbor Search Problem

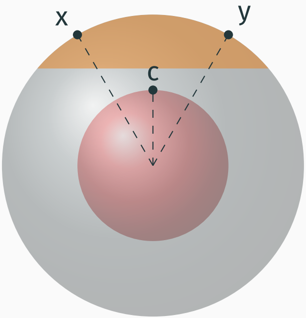
Near-Neighbor Search (NNS), NNS($\mathcal{L}$, p)

Given an input list $\mathcal{L} \subseteq \mathcal{S}_{\mathrm{p}}^{\mathrm{m}}$ and a target weight p, find all pairs $(\mathbf{x}, \mathbf{y}) \in \mathcal{L} \times \mathcal{L}$ satisfying $|\mathbf{x} \wedge \mathbf{y}| = \mathrm{p}/2$.

$\rightarrow$ Brute-force search runtime: $\tilde{\mathcal{O}}(|\mathcal{L}|^2)$.

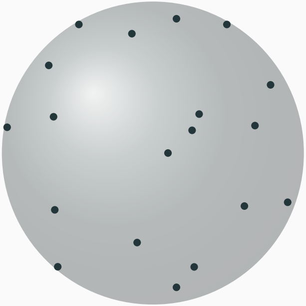
Localized search

For a suitable choice of α, if $|\mathbf{x} \wedge \mathbf{c}| = |\mathbf{y} \wedge \mathbf{c}| = \alpha$, **x** and **y** are likely **near neighbors**.



Locality-Sensitive Filtering (LSF)[9]

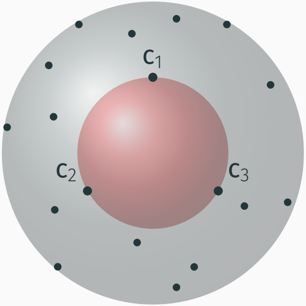
Given $\mathcal{L} \subseteq \mathcal{S}_p^m$



[9] Anja Becker, Léo Ducas, et al. New directions in nearest neighbor searching with applications to lattice sieving. 2015.

Locality-Sensitive Filtering (LSF)[9]

Given $\mathcal{L} \subseteq \mathcal{S}_p^m$, set of centers $\mathcal{C}_f$ and parameter α

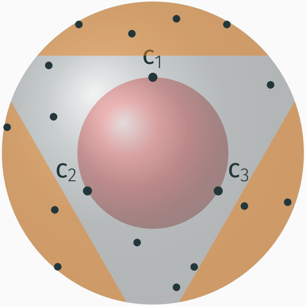


[9] Anja Becker, Léo Ducas, et al. New directions in nearest neighbor searching with applications to lattice sieving. 2015.

Locality-Sensitive Filtering (LSF)[9]

Given $\mathcal{L} \subseteq \mathcal{S}_p^m$, set of centers $\mathcal{C}_f$ and parameter α, perform

- **bucketing phase**: for each element $\mathbf{x} \in \mathcal{L}$, if $|\mathbf{x} \wedge \mathbf{c}| = \alpha$, assign $\mathbf{x}$ to a bucket $\mathcal{B}_\alpha(\mathbf{c})$,

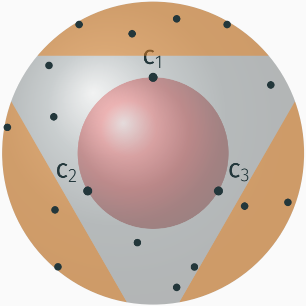


[9] Anja Becker, Léo Ducas, et al. New directions in nearest neighbor searching with applications to lattice sieving. 2015.

Locality-Sensitive Filtering (LSF)[9]

Given $\mathcal{L} \subseteq \mathcal{S}_p^m$, set of centers $\mathcal{C}_f$ and parameter α, perform

- **bucketing phase**: for each element $\mathbf{x} \in \mathcal{L}$, if $|\mathbf{x} \wedge \mathbf{c}| = \alpha$, assign $\mathbf{x}$ to a bucket $\mathcal{B}_\alpha(\mathbf{c})$,
- **checking phase**: for each $\mathbf{c} \in \mathcal{C}_f$, check which $(\mathbf{x}, \mathbf{y}) \in \mathcal{B}_\alpha(\mathbf{c}) \times \mathcal{B}_\alpha(\mathbf{c})$ are near neighbors and add them to the output list.



[9] Anja Becker, Léo Ducas, et al. New directions in nearest neighbor searching with applications to lattice sieving. 2015.

Algorithm NNS using locality-sensitive filtering

Input : Weight p, input list $\mathcal{L} \subseteq \mathcal{S}_{\mathrm{p}}^{\mathrm{m}}$, set of centers $\mathcal{C}_{\mathrm{f}}$, and a bucketing parameter α.

Algorithm NNS using locality-sensitive filtering

Input : Weight p, input list $\mathcal{L} \subseteq \mathcal{S}_p^m$, set of centers $\mathcal{C}_f$, and a bucketing parameter α.

Output: Output list $\mathcal{L}'$ containing pairs $(\mathbf{x}, \mathbf{y}) \in \mathcal{L} \times \mathcal{L}$ with $|\mathbf{x} \wedge \mathbf{y}| = p/2$.

Algorithm NNS using locality-sensitive filtering

Input : Weight p, input list $\mathcal{L} \subseteq \mathcal{S}_p^m$, set of centers $\mathcal{C}_f$, and a bucketing parameter α.

Output: Output list $\mathcal{L}'$ containing pairs $(x, y) \in \mathcal{L} \times \mathcal{L}$ with $|x \wedge y| = p/2$.

BUCKETING PHASE:

Algorithm NNS using locality-sensitive filtering

Input : Weight p, input list $\mathcal{L} \subseteq \mathcal{S}_p^m$, set of centers $\mathcal{C}_f$, and a bucketing parameter α.

Output: Output list $\mathcal{L}'$ containing pairs $(x, y) \in \mathcal{L} \times \mathcal{L}$ with $|x \wedge y| = p/2$.

BUCKETING PHASE:

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

Algorithm NNS using locality-sensitive filtering

Input : Weight p, input list $\mathcal{L} \subseteq \mathcal{S}_p^m$, set of centers $\mathcal{C}_f$, and a bucketing parameter α.

Output: Output list $\mathcal{L}'$ containing pairs $(\mathbf{x}, \mathbf{y}) \in \mathcal{L} \times \mathcal{L}$ with $|\mathbf{x} \wedge \mathbf{y}| = p/2$.

BUCKETING PHASE:

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

  for $\mathbf{c} \in \mathrm{FindValidCenters}(\mathcal{C}_f, \mathbf{x}, \alpha)$ **do**

Algorithm NNS using locality-sensitive filtering

Input : Weight p, input list $\mathcal{L} \subseteq \mathcal{S}_p^m$, set of centers $\mathcal{C}_f$, and a bucketing parameter α.

Output: Output list $\mathcal{L}'$ containing pairs $(\mathsf{x}, \mathsf{y}) \in \mathcal{L} \times \mathcal{L}$ with $|\mathsf{x} \wedge \mathsf{y}| = p/2$.

BUCKETING PHASE:

for $x \in \mathcal{L}$ **do**
- **for** $c \in$ FindValidCenters$(\mathcal{C}_f, \mathsf{x}, \alpha)$ **do**
  - Add x to $\mathcal{B}_\alpha(\mathsf{c})$.

Algorithm NNS using locality-sensitive filtering

Input : Weight p, input list $\mathcal{L} \subseteq \mathcal{S}_p^m$, set of centers $\mathcal{C}_f$, and a bucketing parameter α.

Output: Output list $\mathcal{L}'$ containing pairs $(x, y) \in \mathcal{L} \times \mathcal{L}$ with $|x \wedge y| = p/2$.

BUCKETING PHASE:

for $x \in \mathcal{L}$ **do**
  for $c \in \text{FindValidCenters}(\mathcal{C}_f, x, \alpha)$ **do**
    Add x to $\mathcal{B}_\alpha(c)$.

CHECKING PHASE:

Algorithm NNS using locality-sensitive filtering

Input : Weight p, input list $\mathcal{L} \subseteq \mathcal{S}_p^m$, set of centers $\mathcal{C}_f$, and a bucketing parameter α.

Output: Output list $\mathcal{L}'$ containing pairs $(\mathbf{x}, \mathbf{y}) \in \mathcal{L} \times \mathcal{L}$ with $|\mathbf{x} \wedge \mathbf{y}| = p/2$.

BUCKETING PHASE:

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

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

    Add $\mathbf{x}$ to $\mathcal{B}_\alpha(\mathbf{c})$.

CHECKING PHASE:

for $\mathbf{c} \in \mathcal{C}_f$ **do**

Algorithm NNS using locality-sensitive filtering

Input : Weight p, input list $\mathcal{L} \subseteq \mathcal{S}_p^m$, set of centers $\mathcal{C}_f$, and a bucketing parameter α.

Output: Output list $\mathcal{L}'$ containing pairs $(x, y) \in \mathcal{L} \times \mathcal{L}$ with $|x \wedge y| = p/2$.

BUCKETING PHASE:

for $x \in \mathcal{L}$ **do**
- **for** $c \in$ FindValidCenters$(\mathcal{C}_f, x, \alpha)$ **do**
  - Add x to $\mathcal{B}_\alpha(c)$.

CHECKING PHASE:

for $c \in \mathcal{C}_f$ **do**
- **for** $(x, y) \in \mathcal{B}_\alpha(c) \times \mathcal{B}_\alpha(c)$ **do**

Algorithm NNS using locality-sensitive filtering

Input : Weight p, input list $\mathcal{L} \subseteq \mathcal{S}_p^m$, set of centers $\mathcal{C}_f$, and a bucketing parameter α.

Output: Output list $\mathcal{L}'$ containing pairs $(x, y) \in \mathcal{L} \times \mathcal{L}$ with $|x \wedge y| = p/2$.

BUCKETING PHASE:

```
for x ∈ L do
    for c ∈ FindValidCenters(C_f, x, α) do
        Add x to B_α(c).
```

CHECKING PHASE:

```
for c ∈ C_f do
    for (x, y) ∈ B_α(c) × B_α(c) do
        if |x ∧ y| = p/2 then
            Add (x, y) to L'.
```

Algorithm NNS using locality-sensitive filtering

Input : Weight p, input list $\mathcal{L} \subseteq \mathcal{S}_p^m$, set of centers $\mathcal{C}_f$, and a bucketing parameter α.

Output: Output list $\mathcal{L}'$ containing pairs $(\mathbf{x}, \mathbf{y}) \in \mathcal{L} \times \mathcal{L}$ with $|\mathbf{x} \wedge \mathbf{y}| = p/2$.

Bucketing Phase:

```
for x ∈ L do
    for c ∈ FindValidCenters(C_f, x, α) do
        Add x to B_α(c).
```

Checking Phase:

```
for c ∈ C_f do
    for (x, y) ∈ B_α(c) × B_α(c) do
        if |x ∧ y| = p/2 then
            Add (x, y) to L'.
return L'
```

Guo, Johansson and Nguyen [GJN] approach[10]

Basic idea: For any $(\mathbf{x}, \mathbf{y}) \in \mathcal{S}_p^m \times \mathcal{S}_p^m$ satisfying $|\mathbf{x} \wedge \mathbf{y}| = p/2$, there exists a unique $\mathbf{c} \in \mathcal{S}_{p/2}^m$ such that $|\mathbf{x} \wedge \mathbf{c}| = |\mathbf{y} \wedge \mathbf{c}| = p/2$.

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

Guo, Johansson and Nguyen [GJN] approach[10]

Basic idea: For any $(\mathbf{x}, \mathbf{y}) \in \mathcal{S}_p^m \times \mathcal{S}_p^m$ satisfying $|\mathbf{x} \wedge \mathbf{y}| = p/2$, there exists a unique $\mathbf{c} \in \mathcal{S}_{p/2}^m$ such that $|\mathbf{x} \wedge \mathbf{c}| = |\mathbf{y} \wedge \mathbf{c}| = p/2$.

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

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

Guo, Johansson and Nguyen [GJN] approach[10]

Basic idea: For any $(\mathbf{x}, \mathbf{y}) \in \mathcal{S}_p^m \times \mathcal{S}_p^m$ satisfying $|\mathbf{x} \wedge \mathbf{y}| = p/2$, there exists a unique $\mathbf{c} \in \mathcal{S}_{p/2}^m$ such that $|\mathbf{x} \wedge \mathbf{c}| = |\mathbf{y} \wedge \mathbf{c}| = p/2$.

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

Parameters:

$$\mathcal{C}_f = \mathcal{S}_{p/2}^m, \quad \alpha = p/2.$$

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

Coded hashing approach (HASH)

Basic idea: Increase the size of buckets but reduce the number of buckets efficiently.

Coded hashing approach (HASH)

Basic idea: Increase the size of buckets but reduce the number of buckets efficiently.

Parameters

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

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

CODED HASHING APPROACH (HASH)

Basic idea: Increase the size of buckets but reduce the number of buckets efficiently.

Parameters

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

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

$\to$ FINDVALIDCENTERS subroutine needs to perform efficient decoding.

Random Product Codes approach (RPC)

Basic idea: Improve efficiency of FindValidCenters subroutine using **random product codes**.

Random Product Codes approach (RPC)

Basic idea: Improve efficiency of FindValidCenters subroutine using **random product codes**.

Parameters:

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

where t is chosen to guarantee random behavior of the $\mathcal{C}_{\mathcal{H}}$ and an efficient decodability.

Memory optimal versions (HASH and RPC memo-opt)

High-level idea

We interleave the bucketing and the checking phase.

Memory optimal versions (HASH and RPC memo-opt)

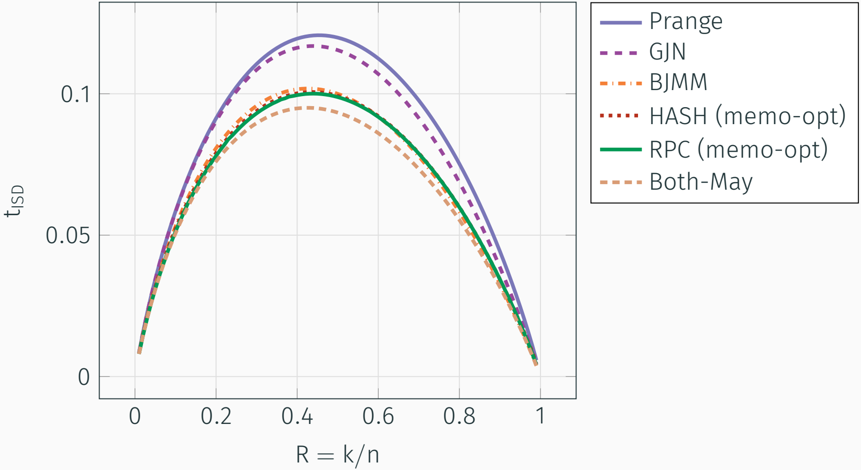
High-level idea

We interleave the bucketing and the checking phase.

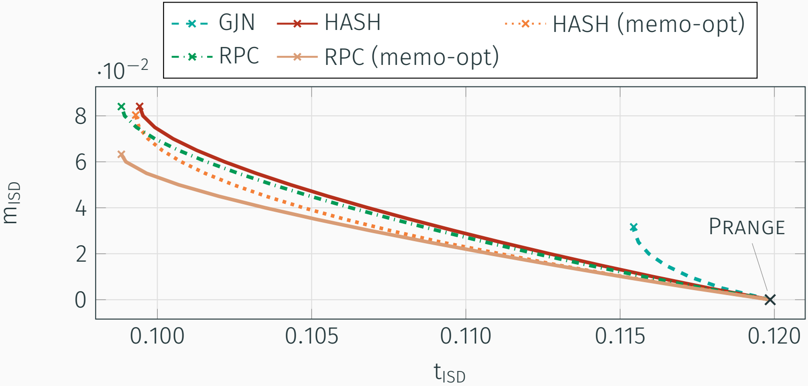
Memory optimal approach

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

COMPARISONS AND CONCLUSIONS



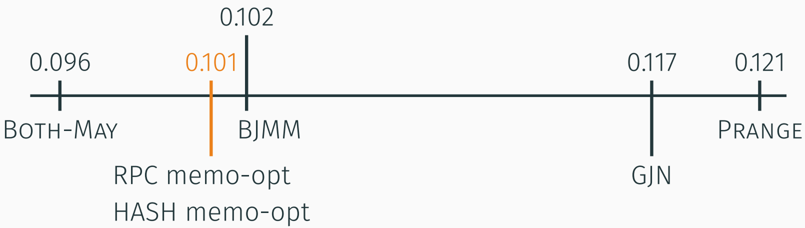
Asymptotic runtime exponent for different ISD variants for the unique-decoding regime.



Time-memory trade-off curves of different SievingISD instantiations.

Summa summarum

We introduce **sieving-based ISD** algorithms whose asymptotic runtime and memory are close to those of the state-of-the-art.



Summa summarum

We introduce **sieving-based ISD** algorithms whose asymptotic runtime and memory are close to those of the state-of-the-art.

A new alignment of the lattice-based and code-based framework.

Summa summarum

We introduce sieving-based ISD algorithms whose asymptotic runtime and memory are close to those of the state-of-the-art.

A new alignment of the lattice-based and code-based framework.

How practical is code-sieving?

Summa summarum

We introduce **sieving-based ISD** algorithms whose asymptotic runtime and memory are close to those of the state-of-the-art.

A new alignment of the lattice-based and code-based framework.

How practical is code-sieving?

eprint GitHub repo

Thank you for your attention!