Time-Memory tradeoffs for large-weight syndrome decoding in ternary codes

Pierre Karpman[1] Charlotte Lefevre[2]

PKC 2022

March 9

[1]Université Grenoble Alpes, Grenoble, France

[2]Radboud University, Nijmegen, The Netherlands

Definitions, motivation

The fixed-weight syndrome decoding problem (SDP)

- We consider $[n, k]$-ternary linear codes
- $n =$ length, $k =$ dimension

The fixed-weight syndrome decoding problem (SDP)

- We consider $[n, k]$-ternary linear codes
- $n =$ length, $k =$ dimension

SDP

Input: $\boldsymbol{H} \in \mathbb{F}_3^{(n-k)\times n}$ full-rank (parity-check matrix)
$\boldsymbol{s} \in \mathbb{F}_3^{n-k}$ (syndrome)
$w \in [\![0, n]\!]$ (weight)

Output: $\boldsymbol{e} \in \mathbb{F}_3^n$ s.t. $\boldsymbol{He} = \boldsymbol{s}$ and $\mathrm{wt}(\boldsymbol{e}) = w$

The fixed-weight syndrome decoding problem (SDP)

- We consider $[n, k]$-ternary linear codes
- n = length, k = dimension

SDP

Input: $\boldsymbol{H} \in \mathbb{F}_3^{(n-k)\times n}$ full-rank (parity-check matrix)
$\boldsymbol{s} \in \mathbb{F}_3^{n-k}$ (syndrome)
$w \in [\![0, n]\!]$ (weight)

Output: $\boldsymbol{e} \in \mathbb{F}_3^n$ s.t. $\boldsymbol{He} = \boldsymbol{s}$ and $\mathrm{wt}(\boldsymbol{e}) = w$

In our case:

- $\boldsymbol{H} \twoheadleftarrow \left\{ \boldsymbol{M} \in \mathbb{F}_3^{(n-k)\times n} \mid \boldsymbol{M} \text{ full-rank} \right\}$; $\boldsymbol{s} \twoheadleftarrow \mathbb{F}_3^{n-k}$
- $k/n, w/n$ fixed
- Number of solutions exponential in n

The Wave signature scheme

- Introducted by Debris-Alazard et al. in 2018

The Wave signature scheme

- Introducted by Debris-Alazard et al. in 2018
- High-level view:
  - ◇ Private key: structured parity-check matrix $\boldsymbol{H} \in \mathbb{F}_3^{(n-k)\times n}$
  - ◇ Public key: random obfuscation of the latter $\boldsymbol{H}_{\text{obfs}}$

The Wave signature scheme

- Introducted by Debris-Alazard et al. in 2018
- High-level view:
  - ◇ Private key: structured parity-check matrix $\boldsymbol{H} \in \mathbb{F}_3^{(n-k)\times n}$
  - ◇ Public key: random obfuscation of the latter $\boldsymbol{H}_{\text{obfs}}$
- $\texttt{Sign}(m) = \text{SDP}(\boldsymbol{H}_{\text{obfs}}, \texttt{hash}(m), w)$, where
  - ◇ $\texttt{hash}$: $\mathcal{M} \longrightarrow \mathbb{F}_3^{n-k}$ hash function
  - ◇ w fixed by the scheme ($w/n \approx 0.948$)

The Wave signature scheme

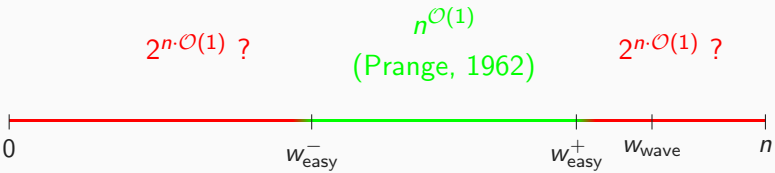
- Introducted by Debris-Alazard et al. in 2018
- High-level view:
  - ◇ Private key: structured parity-check matrix $\boldsymbol{H} \in \mathbb{F}_3^{(n-k)\times n}$
  - ◇ Public key: random obfuscation of the latter $\boldsymbol{H}_{\text{obfs}}$
- $\texttt{Sign}(m) = \text{SDP}(\boldsymbol{H}_{\text{obfs}}, \texttt{hash}(m), w)$, where
  - ◇ $\texttt{hash}$: $\mathcal{M} \longrightarrow \mathbb{F}_3^{n-k}$ hash function
  - ◇ w fixed by the scheme ($w/n \approx 0.948$)
- Decoding is easier with access to $\boldsymbol{H}$ $\implies$ trapdoor

The Wave signature scheme

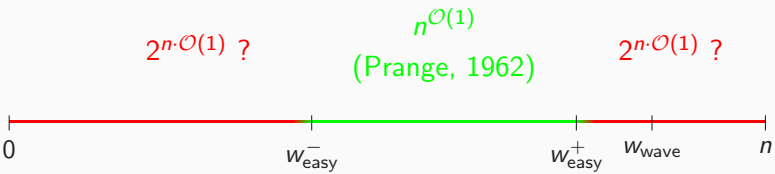
- Introducted by Debris-Alazard et al. in 2018
- High-level view:
  - ◇ Private key: structured parity-check matrix $\boldsymbol{H} \in \mathbb{F}_3^{(n-k)\times n}$
  - ◇ Public key: random obfuscation of the latter $\boldsymbol{H}_{\text{obfs}}$
- $\texttt{Sign}(m) = \text{SDP}(\boldsymbol{H}_{\text{obfs}}, \texttt{hash}(m), w)$, where
  - ◇ $\texttt{hash}$: $\mathcal{M} \longrightarrow \mathbb{F}_3^{n-k}$ hash function
  - ◇ w fixed by the scheme ($w/n \approx 0.948$)
- Decoding is easier with access to $\boldsymbol{H}$ $\implies$ trapdoor
- One security requirement: SDP with a random parity-check matrix and syndrome must be hard

The SDP in the Wave regime

- With random parity-check matrix and syndrome, k/n fixed:

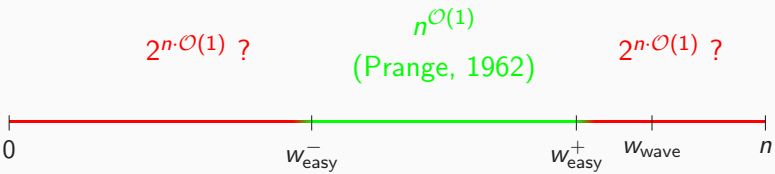


- With random parity-check matrix and syndrome, k/n fixed:



- Best known attack (Bricout et al., 2019): based on the PGE+SS framework with k-tree+representations: $\mathcal{O}(2^{0.0176n})$

- With random parity-check matrix and syndrome, k/n fixed:



- Best known attack (Bricout et al., 2019): based on the PGE+SS framework with k-tree+representations: $\mathcal{O}(2^{0.0176n})$ in time **and** memory
- In this work, we investigate tradeoffs for this problem

The PGE+SS framework

$SDP(\boldsymbol{H} \in \mathbb{F}_3^{(n-k)\times n}, \boldsymbol{s} \in \mathbb{F}_3^{n-k}, w \in [\![0, n]\!])$:

$\boldsymbol{e} \in \mathbb{F}_3^n$ st., $\boldsymbol{He} = \boldsymbol{s}$, $\mathrm{wt}(e) = w$

- Formalized by Bricout et al.

The PGE+SS framework

$SDP(\boldsymbol{H} \in \mathbb{F}_3^{(n-k)\times n}, \boldsymbol{s} \in \mathbb{F}_3^{n-k}, w \in [\![0, n]\!])$:

$\boldsymbol{e} \in \mathbb{F}_3^n$ st., $\boldsymbol{He} = \boldsymbol{s}$, $\mathrm{wt}(e) = w$

- Formalized by Bricout et al.
- Inputs: $\boldsymbol{H} \in \mathbb{F}_3^{(n-k)\times n}, \boldsymbol{s} \in \mathbb{F}_3^{n-k}, \ell \in [\![0, w-k]\!]$

$$\boldsymbol{SHP} = \left(\begin{array}{c|c} \overbrace{\boldsymbol{I}_{n-k-\ell}}^{n-k-\ell} & \overbrace{\boldsymbol{H}_1}^{k+\ell} \\ \hline 0 & \boldsymbol{H}_2 \end{array}\right) \qquad \boldsymbol{Ss} = \left(\begin{array}{c} \boldsymbol{s}_1 \\ \hline \boldsymbol{s}_2 \end{array}\right) \begin{array}{l} \updownarrow n-k-\ell \\ \updownarrow \ell \end{array} \qquad \boldsymbol{P}^{-1}\boldsymbol{e} = \left(\begin{array}{c} \boldsymbol{e}_1 \\ \hline \boldsymbol{e}_2 \end{array}\right) \begin{array}{l} \updownarrow n-k-\ell \\ \updownarrow k+\ell \end{array}$$

The PGE+SS framework

$SDP(\boldsymbol{H} \in \mathbb{F}_3^{(n-k)\times n}, \boldsymbol{s} \in \mathbb{F}_3^{n-k}, w \in [\![0, n]\!])$:

$\boldsymbol{e} \in \mathbb{F}_3^n$ st., $\boldsymbol{He} = \boldsymbol{s}$, $\mathrm{wt}(e) = w$

- Formalized by Bricout et al.
- Inputs: $\boldsymbol{H} \in \mathbb{F}_3^{(n-k)\times n}, \boldsymbol{s} \in \mathbb{F}_3^{n-k}, \ell \in [\![0, w-k]\!]$

$$\boldsymbol{SHP} = \left(\begin{array}{c|c} \overbrace{\boldsymbol{I}_{n-k-\ell}}^{n-k-\ell} & \overbrace{\boldsymbol{H}_1}^{k+\ell} \\ \hline 0 & \boldsymbol{H}_2 \end{array}\right) \qquad \boldsymbol{Ss} = \left(\begin{array}{c} \boldsymbol{s}_1 \\ \hline \boldsymbol{s}_2 \end{array}\right) \begin{array}{l} \updownarrow n-k-\ell \\ \updownarrow \ell \end{array} \qquad \boldsymbol{P}^{-1}\boldsymbol{e} = \left(\begin{array}{c} \boldsymbol{e}_1 \\ \hline \boldsymbol{e}_2 \end{array}\right) \begin{array}{l} \updownarrow n-k-\ell \\ \updownarrow k+\ell \end{array}$$

- Solve the sub-SDP problem $\boldsymbol{H}_2\boldsymbol{e}_2 = \boldsymbol{s}_2$ with $\boldsymbol{e}_2 \in \mathbb{F}_3^{k+\ell}$ **full-weight** where **many solutions are required**

The PGE+SS framework

$SDP(\boldsymbol{H} \in \mathbb{F}_3^{(n-k)\times n}, \boldsymbol{s} \in \mathbb{F}_3^{n-k}, w \in [\![0, n]\!])$:

$\boldsymbol{e} \in \mathbb{F}_3^n$ st., $\boldsymbol{He} = \boldsymbol{s}$, $\mathrm{wt}(e) = w$

- Formalized by Bricout et al.
- Inputs: $\boldsymbol{H} \in \mathbb{F}_3^{(n-k)\times n}, \boldsymbol{s} \in \mathbb{F}_3^{n-k}, \ell \in [\![0, w-k]\!]$

$$\boldsymbol{SHP} = \left(\begin{array}{c|c} \overbrace{\boldsymbol{I}_{n-k-\ell}}^{n-k-\ell} & \overbrace{\boldsymbol{H}_1}^{k+\ell} \\ \hline 0 & \boldsymbol{H}_2 \end{array}\right) \qquad \boldsymbol{Ss} = \left(\begin{array}{c} \boldsymbol{s}_1 \\ \hline \boldsymbol{s}_2 \end{array}\right) \begin{array}{l} \updownarrow n-k-\ell \\ \updownarrow \ell \end{array} \qquad \boldsymbol{P}^{-1}\boldsymbol{e} = \left(\begin{array}{c} \boldsymbol{e}_1 \\ \hline \boldsymbol{e}_2 \end{array}\right) \begin{array}{l} \updownarrow n-k-\ell \\ \updownarrow k+\ell \end{array}$$

- Solve the sub-SDP problem $\boldsymbol{H}_2\boldsymbol{e}_2 = \boldsymbol{s}_2$ with $\boldsymbol{e}_2 \in \mathbb{F}_3^{k+\ell}$ **full-weight** where **many solutions are required**
- Every $\boldsymbol{e}_2$ completes to a unique $\boldsymbol{e}_1$ which gives $\boldsymbol{He} = \boldsymbol{s}$
- Solution of SDP iff $\mathrm{wt}(\boldsymbol{e}_1) = w - k - \ell$

The PGE+SS framework

$SDP(\boldsymbol{H} \in \mathbb{F}_3^{(n-k)\times n}, \boldsymbol{s} \in \mathbb{F}_3^{n-k}, w \in [\![0, n]\!])$:

$\boldsymbol{e} \in \mathbb{F}_3^n$ st., $\boldsymbol{He} = \boldsymbol{s}$, $\mathrm{wt}(e) = w$

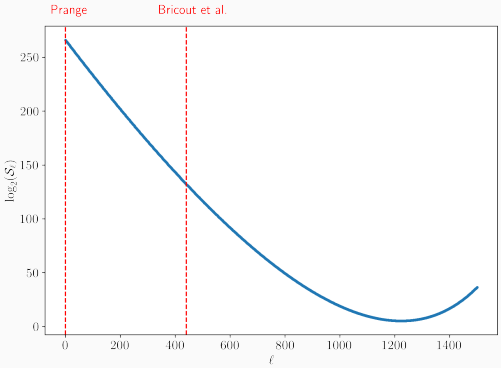
- Formalized by Bricout et al.
- Inputs: $\boldsymbol{H} \in \mathbb{F}_3^{(n-k)\times n}, \boldsymbol{s} \in \mathbb{F}_3^{n-k}, \ell \in [\![0, w-k]\!]$

$$\boldsymbol{SHP} = \left(\begin{array}{c|c} \overbrace{\boldsymbol{I}_{n-k-\ell}}^{n-k-\ell} & \overbrace{\boldsymbol{H}_1}^{k+\ell} \\ \hline 0 & \boldsymbol{H}_2 \end{array}\right) \qquad \boldsymbol{Ss} = \left(\begin{array}{c} \boldsymbol{s}_1 \\ \hline \boldsymbol{s}_2 \end{array}\right) \begin{array}{l} \updownarrow n-k-\ell \\ \updownarrow \ell \end{array} \qquad \boldsymbol{P}^{-1}\boldsymbol{e} = \left(\begin{array}{c} \boldsymbol{e}_1 \\ \hline \boldsymbol{e}_2 \end{array}\right) \begin{array}{l} \updownarrow n-k-\ell \\ \updownarrow k+\ell \end{array}$$

- Solve the sub-SDP problem $\boldsymbol{H}_2\boldsymbol{e}_2 = \boldsymbol{s}_2$ with $\boldsymbol{e}_2 \in \mathbb{F}_3^{k+\ell}$ **full-weight** where **many solutions are required**
- Every $\boldsymbol{e}_2$ completes to a unique $\boldsymbol{e}_1$ which gives $\boldsymbol{He} = \boldsymbol{s}$
- Solution of SDP iff $\mathrm{wt}(\boldsymbol{e}_1) = w - k - \ell$

$\Longrightarrow$ Need to solve efficiently $\boldsymbol{H}_2\boldsymbol{e}_2 = \boldsymbol{s}_2$

- Let $\mathcal{S}_\ell$ be the number of required $\boldsymbol{e}_2$ full weight solutions to $\boldsymbol{H}_2\boldsymbol{e}_2 = \boldsymbol{s}_2$
- Large $\ell \implies$ hard sub-problem, but smaller $\mathcal{S}_\ell$
- One ℓ parameter can provide one tradeoff



r-list sum problem

Transform the SDP into a r-list problem

- Want to solve $\boldsymbol{He} = \boldsymbol{s}, \boldsymbol{H} \in \mathbb{F}_3^{\ell \times (k+\ell)}$, $\boldsymbol{e}$ full-weight
- We use algorithms solving the r-list problem

Transform the SDP into a r-list problem

- Want to solve $\boldsymbol{He} = \boldsymbol{s}, \boldsymbol{H} \in \mathbb{F}_3^{\ell \times (k+\ell)}$, $\boldsymbol{e}$ full-weight
- We use algorithms solving the r-list problem

The r-list sum problem

Input: r lists $L_1, L_2, \ldots, L_r$
Output: $x_1 \in L_1, \ldots, x_r \in L_r$ st $x_1 + \cdots + x_r = 0$

Transform the SDP into a r-list problem

- Want to solve $\boldsymbol{He} = \boldsymbol{s}, \boldsymbol{H} \in \mathbb{F}_3^{\ell \times (k+\ell)}$, $\boldsymbol{e}$ full-weight
- We use algorithms solving the r-list problem

The r-list sum problem

Input: r lists $L_1, L_2, \ldots, L_r$
Output: $x_1 \in L_1, \ldots, x_r \in L_r$ st $x_1 + \cdots + x_r = 0$

Decompose $\boldsymbol{H} = \left(\begin{array}{c|c|c|c} \boldsymbol{H}_1 & \boldsymbol{H}_2 & \cdots & \boldsymbol{H}_r \end{array} \right)$ $\boldsymbol{e} = \left(\begin{array}{c} \boldsymbol{e}_1 \\ \hline \boldsymbol{e}_2 \\ \hline \cdots \\ \hline \boldsymbol{e}_r \end{array} \right)$

- $\boldsymbol{He} = \boldsymbol{s} \iff \underbrace{\boldsymbol{H}_1\boldsymbol{e}_1}_{L_1} + \underbrace{\boldsymbol{H}_2\boldsymbol{e}_2}_{L_2} \cdots + \underbrace{\boldsymbol{H}_r\boldsymbol{e}_r - \boldsymbol{s}}_{L_r} = \boldsymbol{0}$

Transform the SDP into a r-list problem

- Want to solve $\boldsymbol{He} = \boldsymbol{s}, \boldsymbol{H} \in \mathbb{F}_3^{\ell \times (k+\ell)}$, $\boldsymbol{e}$ full-weight
- We use algorithms solving the r-list problem

The r-list sum problem

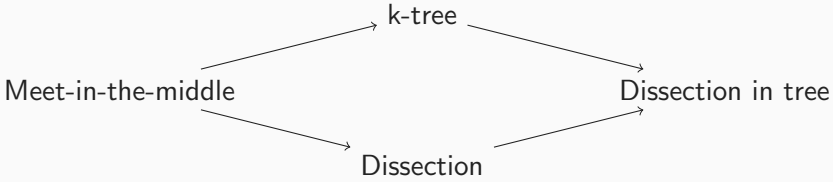
Input: r lists $L_1, L_2, \ldots, L_r$
Output: $x_1 \in L_1, \ldots, x_r \in L_r$ st $x_1 + \cdots + x_r = 0$

Decompose $\boldsymbol{H} = \left(\begin{array}{c|c|c|c} \boldsymbol{H}_1 & \boldsymbol{H}_2 & \cdots & \boldsymbol{H}_r \end{array} \right)$ $\boldsymbol{e} = \left(\begin{array}{c} \boldsymbol{e}_1 \\ \hline \boldsymbol{e}_2 \\ \hline \cdots \\ \hline \boldsymbol{e}_r \end{array} \right)$

- $\boldsymbol{He} = \boldsymbol{s} \iff \underbrace{\boldsymbol{H}_1\boldsymbol{e}_1}_{L_1} + \underbrace{\boldsymbol{H}_2\boldsymbol{e}_2}_{L_2} \cdots + \underbrace{\boldsymbol{H}_r\boldsymbol{e}_r - \boldsymbol{s}}_{L_r} = \boldsymbol{0}$
- Do not need to add **all** possible e_i's in L_i
- ⚠ Constraint: $\forall i, \#L_i \leq 2^{\frac{k+\ell}{r}}$

The algorithms used

- Freedom on r, ℓ, $\#L_i$ parameter
- Generalizations of the meet-in-the-middle algorithm:



The granularity

- The algorithms presented can provide solutions in constant amortized time

The granularity

- The algorithms presented can provide solutions in constant amortized time
- Problem: sometimes these algorithms return too much solutions

The granularity

- The algorithms presented can provide solutions in constant amortized time
- Problem: sometimes these algorithms return too much solutions

Definition

The **granularity** of an algorithm is the minimal number of solutions that can be returned by one iteration without changing its amortized cost

The granularity

- The algorithms presented can provide solutions in constant amortized time
- Problem: sometimes these algorithms return too much solutions

Definition

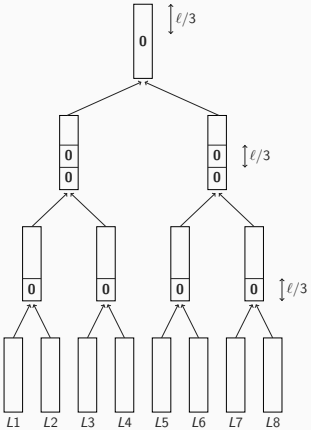
The **granularity** of an algorithm is the minimal number of solutions that can be returned by one iteration without changing its amortized cost

- Some algorithms are very memory friendly, but have a granularity too coarse

The k-tree algorithm

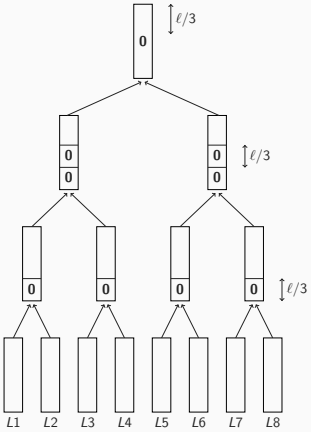
The k-tree algorithm (Wagner, 2002)

- Need $r = 2^a$
- Key idea: apply the meet-in-the-middle pairwise on small constraints



The k-tree algorithm (Wagner, 2002)

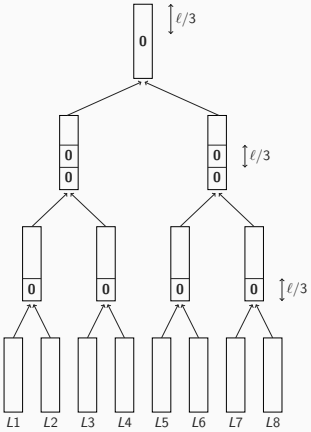
- Need $r = 2^a$
- Key idea: apply the meet-in-the-middle pairwise on small constraints



- Decimates solutions, but efficient

The k-tree algorithm (Wagner, 2002)

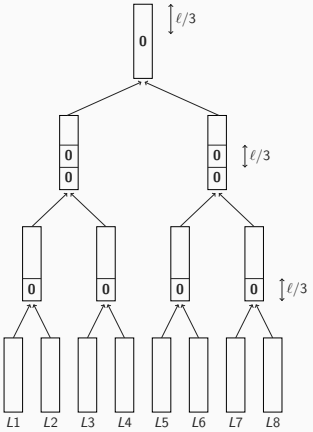
- Need $r = 2^a$
- Key idea: apply the meet-in-the-middle pairwise on small constraints



- Decimates solutions, but efficient
- Minimal list cardinal and granularity: $3^{\ell/a}$

The k-tree algorithm (Wagner, 2002)

- Need $r = 2^a$
- Key idea: apply the meet-in-the-middle pairwise on small constraints

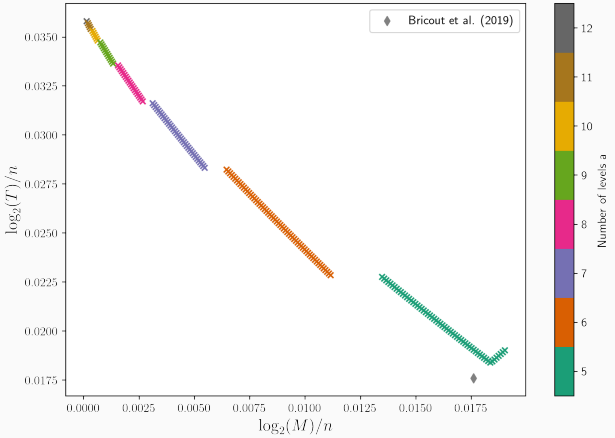


- Decimates solutions, but efficient
- Minimal list cardinal and granularity: $3^{\ell/a}$

  $\Longrightarrow$ Want a as large as possible
- ⚠ Constraint:

$$3^{\ell/a} < 2^{\frac{k+\ell}{2^a}}$$

Time-Memory tradeoffs using the k-tree algorithm

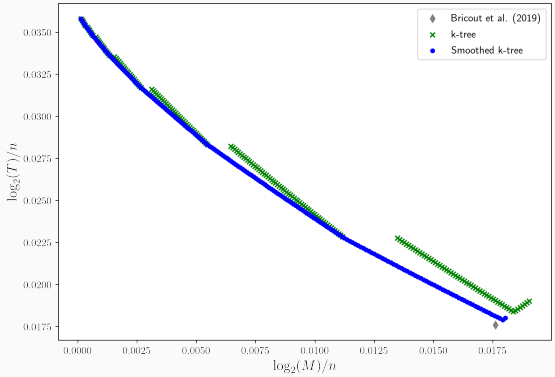


The smoothing technique

- The extended k-tree algorithm (Minder and Sinclair, 2011), adapted in ternary high-weight SDP as smoothing technique (Bricout et al., 2019)
- Relax the constraint: enables to add one level

The smoothing technique

- The extended k-tree algorithm (Minder and Sinclair, 2011), adapted in ternary high-weight SDP as smoothing technique (Bricout et al., 2019)
- Relax the constraint: enables to add one level



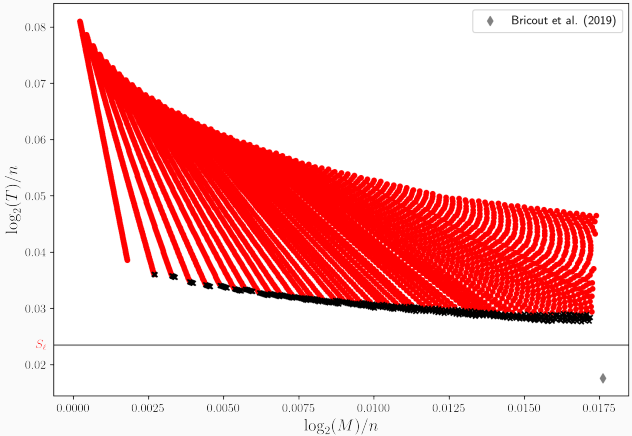
The dissection framework

Dissection features

- The dissection framework (Dinur et al., 2012) is a generalization of the meet-in-the-middle
- Memory-friendly family of exhaustive algorithms
- Proposes Time-Memory tradeoffs with initial list size 3^m
- Splits the r-list problem in an asymmetric way

r-dissection TM tradeoffs, $r < 400$

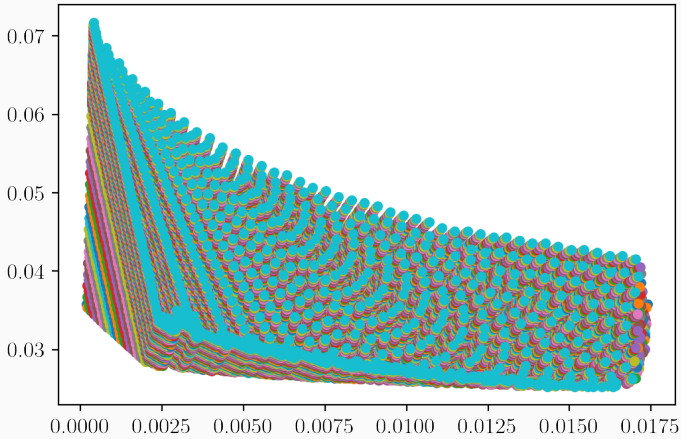
Fixed ℓ parameter ($\ell = 0.04n$)



Black dot = too many returned solutions $\implies$ granularity problems

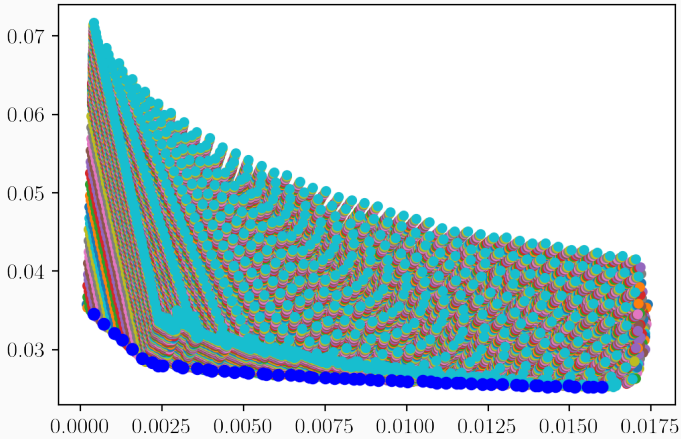
Dissection: obtain the best tradeoffs

The best tradeoffs are provided by a range of ℓ parameters

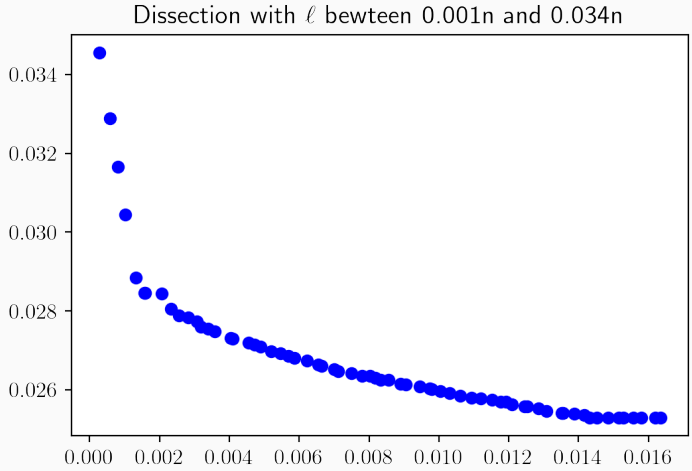


Dissection: obtain the best tradeoffs

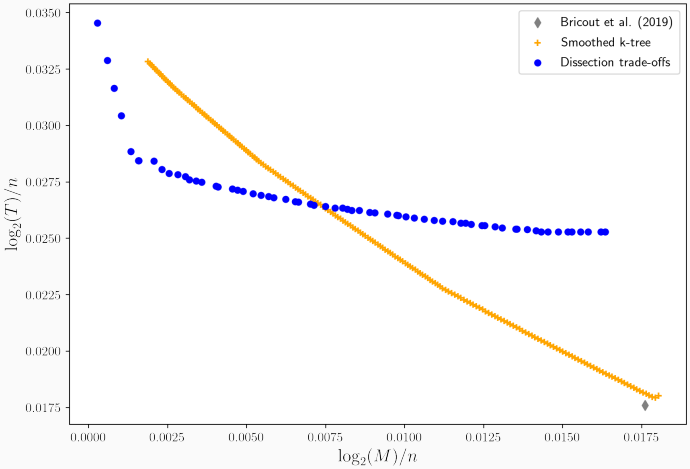
The best tradeoffs are provided by a range of ℓ parameters



Dissection: obtain the best tradeoffs



The dissection is better than the k-tree for small memory

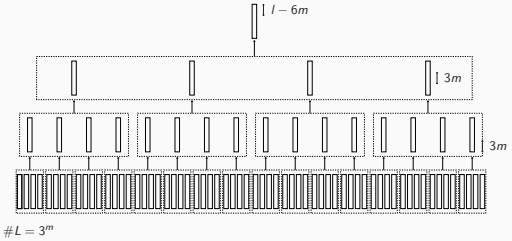


Dissection in tree

Principle

Combine dissection in a tree structure (Dinur, 2019)

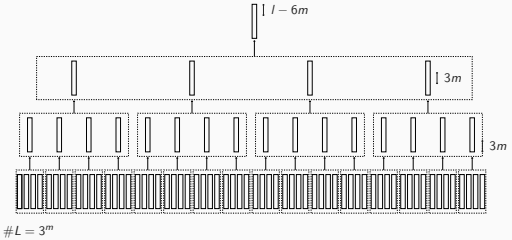
Example: 4-dissection with 3 levels ($X = 4, h = 3$)



Principle

Combine dissection in a tree structure (Dinur, 2019)

Example: 4-dissection with 3 levels ($X = 4, h = 3$)

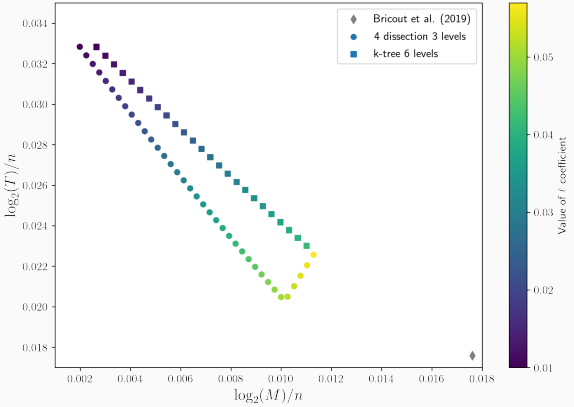


- Each dissection done is exhaustive
- Solutions are returned in constant amortized time
- ⚠ Again same constraint:

$$3^m < 2^{\frac{k+\ell}{x^h}}$$

4-dissection with three levels versus k-tree

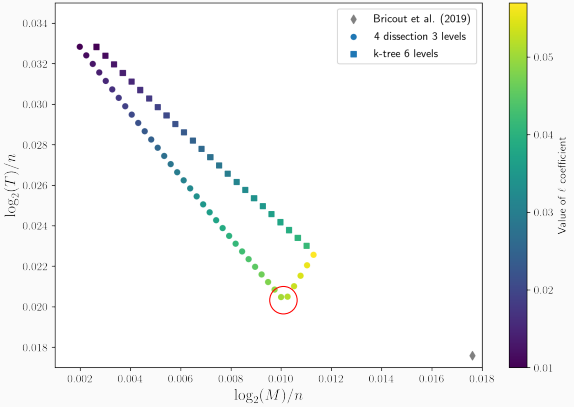
Both have the same number of lists, but different merging strategies



- Dissection in tree needs less memory $\implies$ it is applicable for more ℓ parameters

4-dissection with three levels versus k-tree

Both have the same number of lists, but different merging strategies



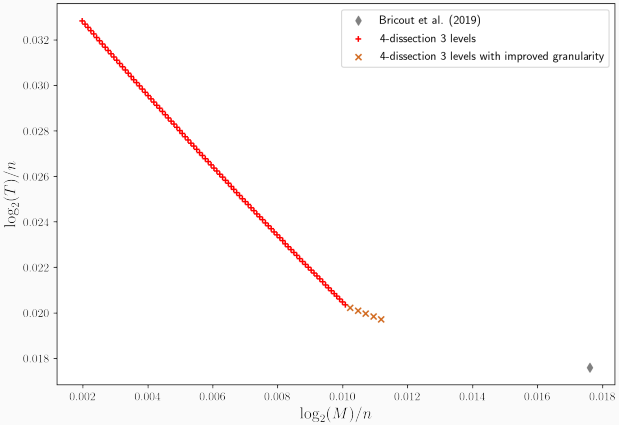
- Dissection in tree needs less memory $\implies$ it is applicable for more ℓ parameters
- The curve rebounds $\Rightarrow$ granularity problems

Granularity improvement

Idea: be not exhaustive in the dissection

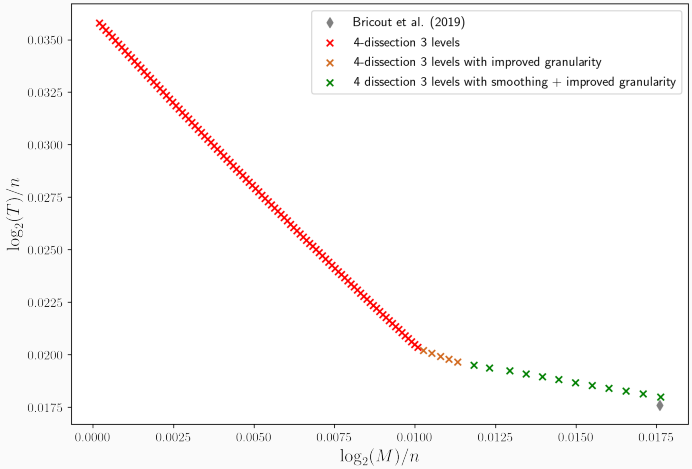
$\Longrightarrow$ solutions are more quickly decimated

$\Longrightarrow$ memory efficiency decreases



Combination of smoothing and granularity improvement

When the previous improvement can not be applied because of the constraint

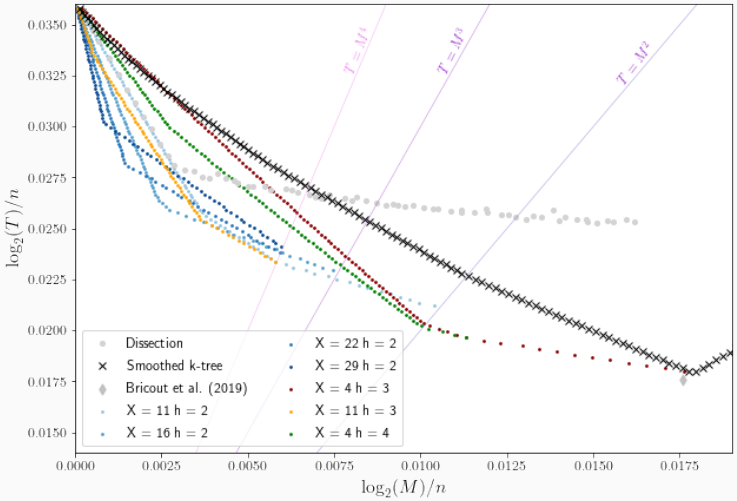


Results

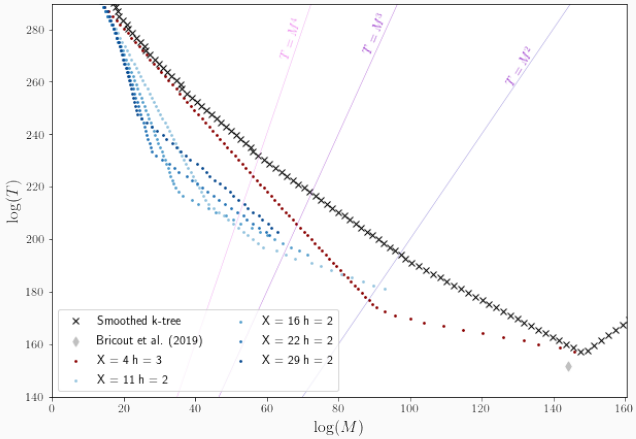
Numerical results

Time	Memory	Time $\times$ Memory	Tradeoff	Algorithm
$2^{0.0176n}$	$2^{0.0176n}$	$2^{0.0352n}$	$T = M$	k-tree + representations (Bricout et al.)
$2^{0.02014n}$	$2^{0.01007n}$	$2^{0.03021n}$	$T = M^2$	4,4-dissection
$2^{0.02256n}$	$2^{0.007521n}$	$2^{0.03008n}$	$T = M^3$	2,11-dissection
$2^{0.02335n}$	$2^{0.005838n}$	$2^{0.02919n}$	$T = M^4$	3,11-dissection

Time/Memory plots



Time/Memory plot with Wave parameters



$n = 7236, k = 4892, w = 6862$

Conclusion

- We investigated Time-Memory tradeoffs for the ternary syndrome decoding problem in the Wave regime
- The studied algorithms are k-tree, dissection and dissection in tree, embedded within the PGE+SS framework
- Some tweaks were necessary to decrease the *granularity* of the building blocks

Conclusion

- We investigated Time-Memory tradeoffs for the ternary syndrome decoding problem in the Wave regime
- The studied algorithms are k-tree, dissection and dissection in tree, embedded within the PGE+SS framework
- Some tweaks were necessary to decrease the *granularity* of the building blocks

Thank you for your attention!