A structural theorem for local algorithms with applications to coding, testing and privacy

Marcel Dall'Agnol
University of Warwick

Tom Gur
University of Warwick

Oded Lachish
Birkbeck, University of London

SODA 2021

An efficient transformation from local to sample-based algorithms.

An efficient transformation from local to sample-based algorithms.

Local: inspects a small part of the input string x
Sample-based: access by random (i, x_i) samples
[Goldreich and Ron, 2016]

An efficient transformation from local to sample-based algorithms.

Local: inspects a small part of the input string x
Sample-based: access by random (i, x_i) samples

But why?

Many (rich) algorithmic problems are inherently local:

- Testing
- Local codes
- PCPs and PCPPs
- LCAs

Many (rich) algorithmic problems are inherently local:

- Testing
- Local codes
- PCPs and PCPPs
- LCAs
- ...

Many (rich) algorithmic problems are inherently local:

- Testing
- Local codes
- PCPs and PCPPs
- LCAs
- ...

Sample-based access buys

- Privacy
- Efficient repetition: running $L_1, \ldots, L_t$ with query complexity q on the same input takes
  - $O(qt \log t)$ queries in general
  - $O(q \log t)$ queries by **reusing samples**

Many (rich) algorithmic problems are inherently local:

- Testing
- Local codes
- PCPs and PCPPs
- LCAs
- ...

Sample-based access buys

- Privacy
- Efficient repetition: running $L_1, \ldots, L_t$ with query complexity q on the same input takes
  - $O(qt \log t)$ queries in general
  - $O(q \log t)$ queries by **reusing samples**
- ...

Local + robust
($O(1)$ queries)

Local + sample-based
($o(n)$ queries)

$$L \implies S$$

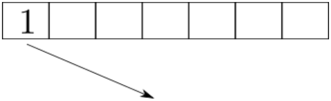
Local + robust
($O(1)$ queries)

Local + sample-based
($o(n)$ queries)

$$L \quad \Longrightarrow \quad S$$

Local + robust ($O(1)$ queries)

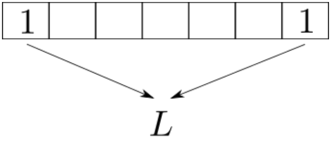
Local + sample-based ($o(n)$ queries)



$$L \implies S$$

Local + robust
($O(1)$ queries)

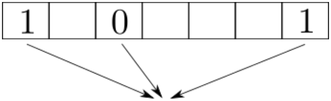
Local + sample-based
($o(n)$ queries)



$\Longrightarrow$ S

Local + robust
($O(1)$ queries)

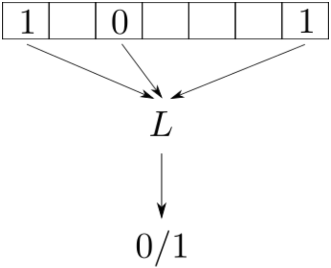
Local + sample-based
($o(n)$ queries)



$$L \quad \Longrightarrow \quad S$$

Local + robust
($O(1)$ queries)

Local + sample-based
($o(n)$ queries)



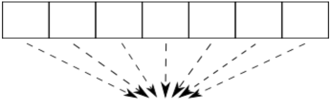
$$\Longrightarrow \qquad S$$

Local + robust ($O(1)$ queries) | Local + sample-based ($o(n)$ queries)

$$L \implies S$$

Local + robust
($O(1)$ queries)

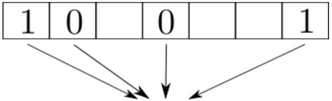
Local + sample-based
($o(n)$ queries)



$L \quad \Longrightarrow \quad S$

Local + robust
($O(1)$ queries)

Local + sample-based
($o(n)$ queries)

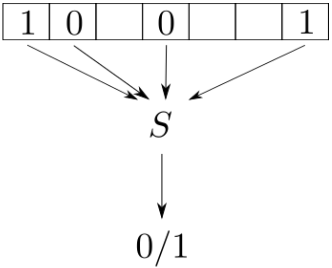


$$L \implies S$$

Local + robust
($O(1)$ queries)

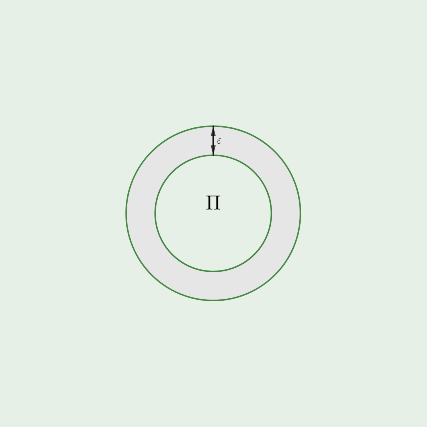
Local + sample-based
($o(n)$ queries)

$L \implies$



Property tester

Property $\Pi \subseteq \{0,1\}^n$, proximity parameter $\varepsilon > 0$

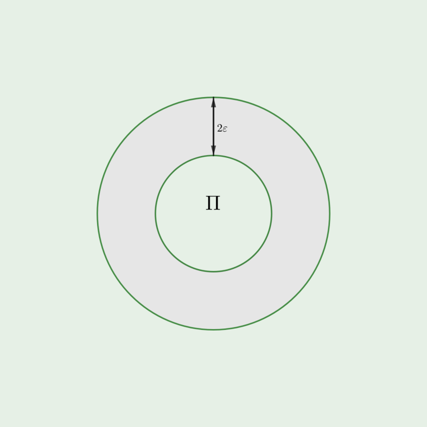


Tester T computes

$$f(x) = \begin{cases} 1, & \text{if } x \in \Pi \\ 0, & \text{if } x \text{ is } \varepsilon\text{-far from } \Pi. \end{cases}$$

Property tester

Property $\Pi \subseteq \{0,1\}^n$, proximity parameter $\varepsilon > 0$

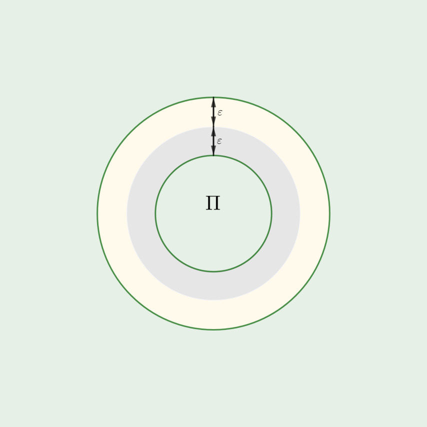


Tester T computes

$$f(x) = \begin{cases} 1, & \text{if } x \in \Pi \\ 0, & \text{if } x \text{ is } \varepsilon\text{-far from } \Pi. \end{cases}$$

Property tester

Property $\Pi \subseteq \{0,1\}^n$, proximity parameter $\varepsilon > 0$

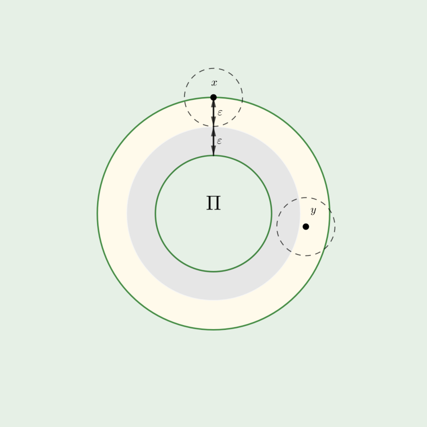


Tester T **robustly** computes

$$g(x) = \begin{cases} 1, & \text{if } x \in \Pi \\ 0, & \text{if } x \text{ is } 2\varepsilon\text{-far from } \Pi. \end{cases}$$

Property tester

Property $\Pi \subseteq \{0,1\}^n$, proximity parameter $\varepsilon > 0$

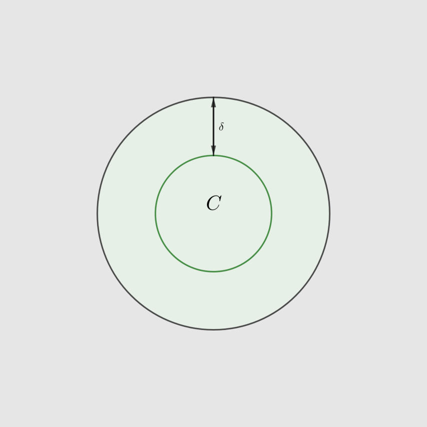


Tester T **robustly** computes

$$g(x) = \begin{cases} 1, & \text{if } x \in \Pi \\ 0, & \text{if } x \text{ is } 2\varepsilon\text{-far from } \Pi. \end{cases}$$

Local decoder

Code $C : \{0,1\}^k \to \{0,1\}^n$, decoding radius $\delta > 0$

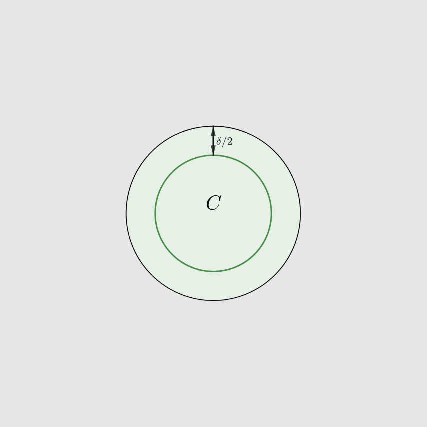


Decoder D computes

$$f(w, i) = x_i, \text{when } w \text{ is } \delta\text{-close to } C(x).$$

Local decoder

Code $C : \{0,1\}^k \to \{0,1\}^n$, decoding radius $\delta > 0$

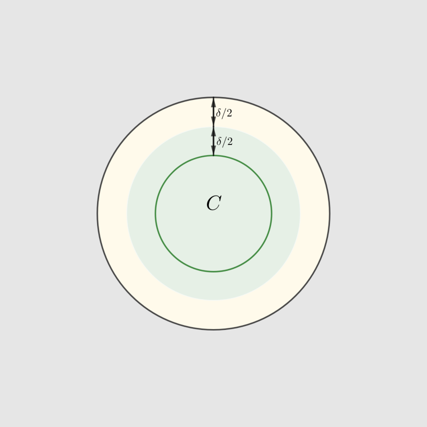


Decoder D computes

$$f(w,i) = x_i, \text{when } w \text{ is } \delta\text{-close to } C(x).$$

Local decoder

Code $C : \{0,1\}^k \to \{0,1\}^n$, decoding radius $\delta > 0$

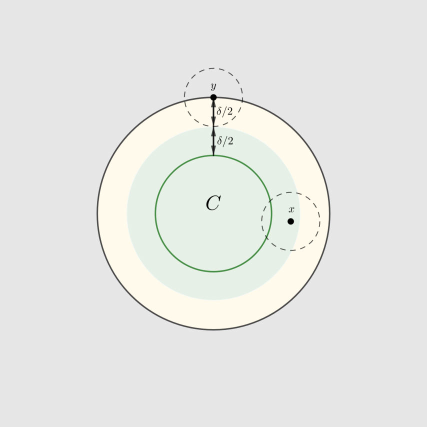


Decoder D **robustly** computes

$$g(w, i) = x_i, \text{when } w \text{ is } \delta/2\text{-close to } C(x).$$

Local decoder

Code $C : \{0,1\}^k \to \{0,1\}^n$, decoding radius $\delta > 0$



Decoder D **robustly** computes

$g(w, i) = x_i,$ when w is $\delta/2$-close to $C(x)$.

Definition

Let $X \subset \{0,1\}^n$. An algorithm L computes $f : X \times Z \to \{0,1\}$ with error rate σ if

$$\mathbb{P}[L^x(z) = f(x,z)] \geq 1 - \sigma. \qquad (\forall x \in X)$$

If L makes $q = o(n)$ queries to the bit string, it is **local**.

Definition

Let $X \subset \{0,1\}^n$. An algorithm L computes $f : X \times Z \to \{0,1\}$ with error rate σ if

$$\mathbb{P}[L^x(z) = f(x,z)] \geq 1 - \sigma. \qquad (\forall x \in X)$$

If L makes $q = o(n)$ queries to the bit string, it is **local**. If

$$\mathbb{P}[L^y(z) = 0] \geq 1 - \sigma, \qquad \forall y \ \rho\text{-close to a 0-input } x,$$

it is ρ-**robust**.

Definition

Let $X \subset \{0,1\}^n$. An algorithm L computes $f : X \times Z \to \{0,1\}$ with error rate σ if

$$\mathbb{P}[L^x(z) = f(x,z)] \geq 1 - \sigma. \qquad (\forall x \in X)$$

If L makes $q = o(n)$ queries to the bit string, it is **local**. If

$$\mathbb{P}[L^y(z) = 0] \geq 1 - \sigma, \qquad \forall y \ \rho\text{-close to a 0-input } x,$$

it is ρ-**robust**.

Remarks:

- Robust 0-inputs without loss of generality.
- f is either partial or constant.
- Captures **LTCs, LCCs, MAPs, PCPPs**...

Theorem

Any function computed by an $\Omega(1)$*-robust local algorithm with query complexity* q *admits a sample-based algorithm with sample complexity*

$$n^{1-\Omega(1/(q^2 \log^2 q))}.$$

$$(q = \Omega(\sqrt{\log n}) \implies \text{sample complexity } \Omega(n))$$

Theorem

Any function computed by an $\Omega(1)$-robust local algorithm with query complexity q admits a sample-based algorithm with sample complexity

$$n^{1-\Omega(1/(q^2 \log^2 q))}.$$

$$(q = \Omega(\sqrt{\log n}) \implies \text{sample complexity } \Omega(n))$$

Theorem

This transformation cannot achieve sample complexity

$$n^{1-\omega(1/q)}.$$

Applications ($q = O(1)$)

Improved lower bound on the blocklength of a *relaxed* LDC.
State-of-the-art was $n = k^{1+\tilde{\Omega}\left(1/2^{2q}\right)}$ [Gur and Lachish, 2020].

Corollary

Any code $C : \{0,1\}^k \to \{0,1\}^n$ that is relaxed locally decodable with q queries satisfies

$$n = k^{1+\tilde{\Omega}(1/q^2)}.$$

Applications ($q = O(1)$)

Improved lower bound on the blocklength of a *relaxed* LDC.
State-of-the-art was $n = k^{1+\tilde{\Omega}\left(1/2^{2q}\right)}$ [Gur and Lachish, 2020].

Corollary

Any code $C : \{0,1\}^k \to \{0,1\}^n$ that is relaxed locally decodable with q queries satisfies

$$n = k^{1+\tilde{\Omega}(1/q^2)}.$$

([Asadi and Shinkar, 2020] achieves $n = k^{1+O(1/q)}$, improving on the construction of [Ben-Sasson et al., 2004])

Applications ($q = O(1)$)

Known "P vs. NP separation for testing" is essentially optimal.
$\exists$ property testable with q queries and $O(\log n)$-long proof, while $n^{1-O(1/q)}$ are needed without a proof [Gur and Rothblum, 2018].

Corollary

If a property is ε-testable with a short proof and q queries, then it is 2ε-testable with $n^{1-\tilde{\Omega}(1/q^2)}$ queries and no proof. In particular, a $O(1)$ vs. $\Omega(n)$ separation is impossible.

Applications ($q = O(1)$)

Known "P vs. NP separation for testing" is essentially optimal.
$\exists$ property testable with q queries and $O(\log n)$-long proof, while $n^{1-O(1/q)}$ are needed without a proof [Gur and Rothblum, 2018].

Corollary

If a property is ε-testable with a short proof and q queries, then it is 2ε-testable with $n^{1-\tilde{\Omega}(1/q^2)}$ queries and no proof. In particular, a $O(1)$ vs. $\Omega(n)$ separation is impossible.

(short: sublinear in the sample complexity)

Applications ($q = O(1)$)

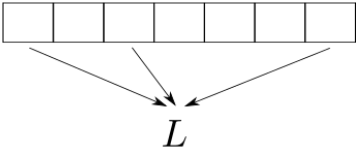
Extension of [Fischer et al., 2015] to adaptive testers.

Corollary

Any property ε-testable with q queries admits a sample-based 2ε-tester with sample complexity $n^{1-\tilde{\Omega}(1/q^2)}$.

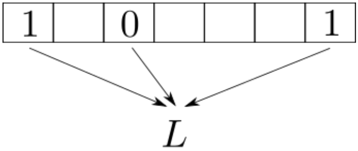
Warm-up: a special case

L **non-adaptive**, one-sided and its query sets form a sunflower



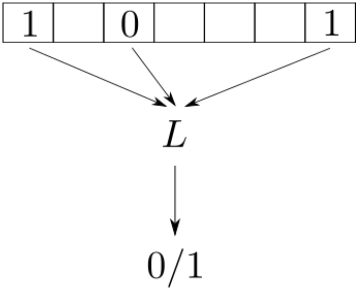
Warm-up: a special case

L **non-adaptive**, one-sided and its query sets form a sunflower



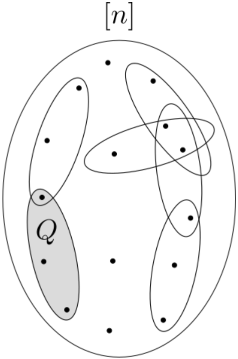
Warm-up: a special case

L **non-adaptive**, one-sided and its query sets form a sunflower



Warm-up: a special case

L **non-adaptive**, one-sided and its query sets form a sunflower



$Q \in \mathcal{Q}$ sampled with probability $\mu(Q)$

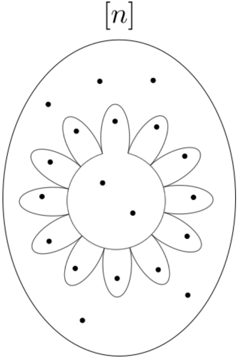
Warm-up: a special case

L non-adaptive, **one-sided** and its query sets form a sunflower

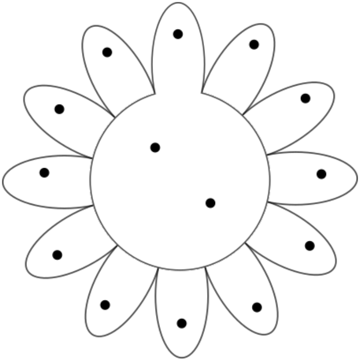
If $f(x) = 1$, then L^x outputs 1 *with certainty*.

Warm-up: a special case

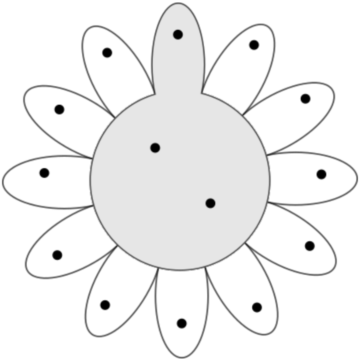
L non-adaptive, one-sided and its **query sets form a sunflower**



Goal: sample many query sets of L and aggregate its "votes"
$(p \approx 1/n^{\alpha})$

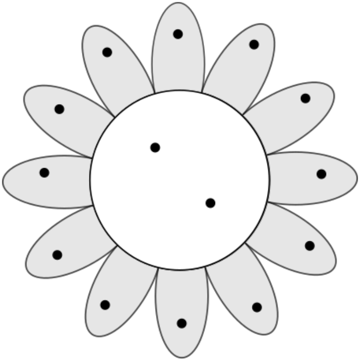


Goal: sample many query sets of L and aggregate its "votes"
$(p \approx 1/n^{\alpha})$



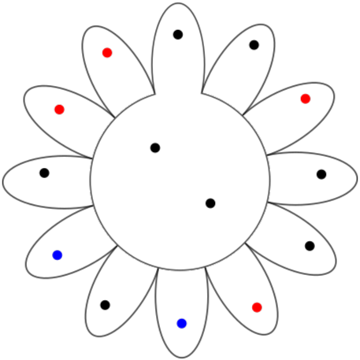
Query at least one set with probability $\leq p$ ☹

Goal: sample many query sets of L and aggregate its "votes"
$(p \approx 1/n^{\alpha})$

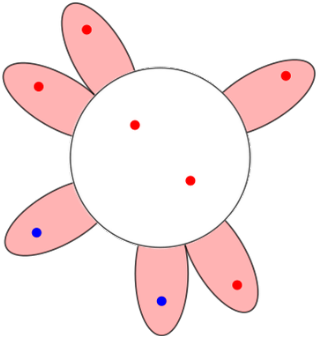


Query $\approx p|\mathcal{Q}|$ *partial* sets with high probability ☺
Fill in the kernel arbitrarily – robustness!

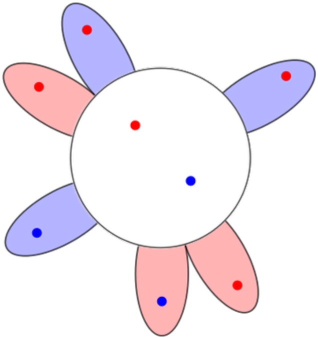
Output 1 $\iff$ some kernel assignment yields *all votes* for 1



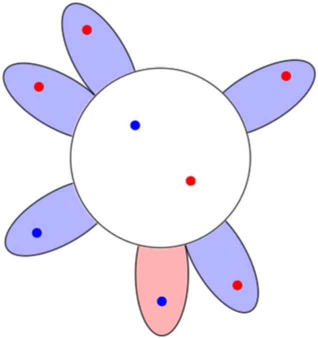
($\bullet = 0$, $\bullet = 1$; $\circ$ = vote for 0, $\circ$ = vote for 1)



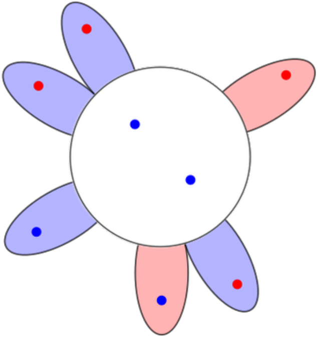
(● = 0, ● = 1; ⬭ = vote for 0, ⬭ = vote for 1)



($\bullet$ = 0, $\bullet$ = 1; $\circ$ = vote for 0, $\circ$ = vote for 1)



(● = 0, ● = 1; ⬭ = vote for 0, ⬭ = vote for 1)

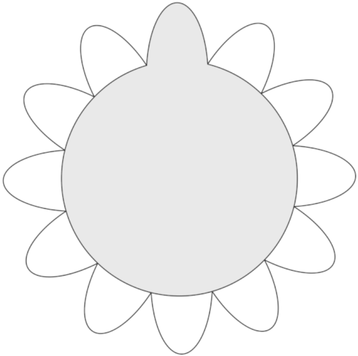


Output 0!

There isn't always a sunflower...

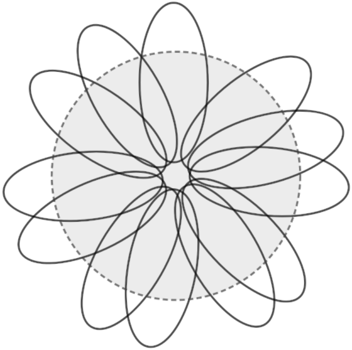
But there is always a daisy!

There isn't always a sunflower...



$|\mathcal{S}| = o(|\mathcal{Q}|)$

But there is always a daisy!



$|\mathcal{D}| = \Omega(|\mathcal{Q}|)$

The bottlenecks

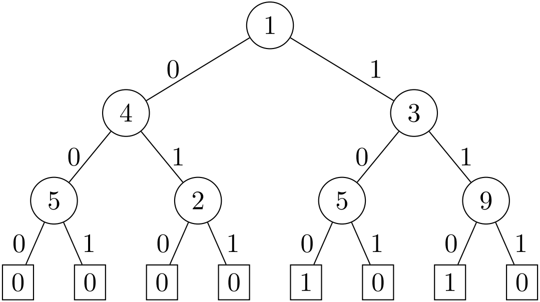
With error $\sigma \approx 1/q$, some daisy approximates L and we throw away the rest. [Gur and Lachish, 2020]

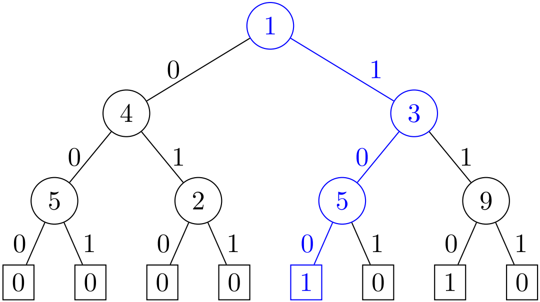
The bottlenecks

With error $\sigma \approx 1/q$, some daisy approximates L and we throw away the rest. [Gur and Lachish, 2020]

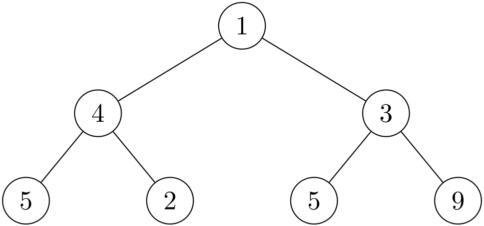
Adaptivity: decision trees vs. query sets. Some daisy works, but we **don't know which**!

Two-sided error: no hope for a consensus.

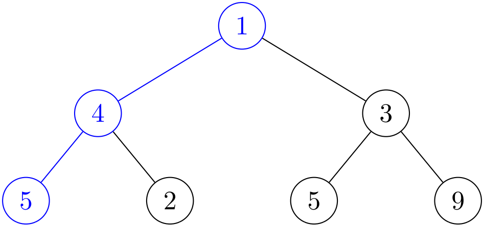

1
0
1
4
3
0
1
0
1
5
2
5
9
0
1
0
1
0
1
0
1
0
0
0
0
1
0
1
0



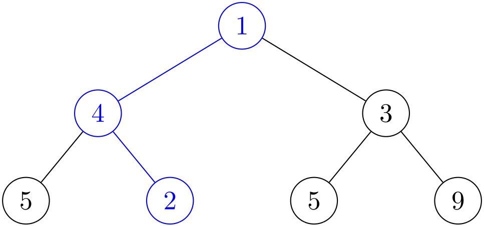
$$x_1 = 1$$
$$x_3 = 0$$
$$x_5 = 0$$



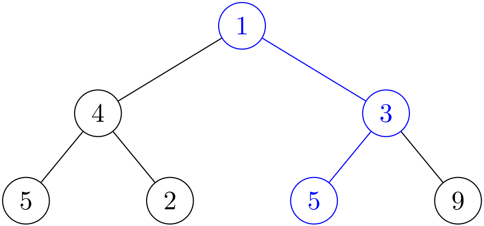
$$\mathcal{Q}_T = \left\{ \quad \left\{ \begin{array}{c} 1 \\ 4 \\ 5 \end{array} \right\}, \quad \left\{ \begin{array}{c} 1 \\ 4 \\ 2 \end{array} \right\}, \quad \left\{ \begin{array}{c} 1 \\ 3 \\ 5 \end{array} \right\}, \quad \left\{ \begin{array}{c} 1 \\ 3 \\ 9 \end{array} \right\} \quad \right\}$$



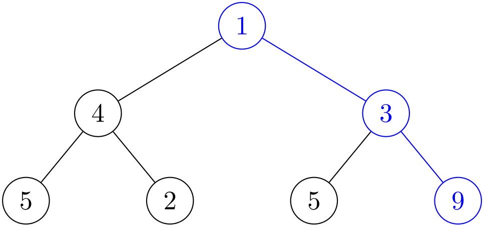
$$\mathcal{Q}_T = \left\{ \quad \left\{ \begin{array}{c} 1 \\ 4 \\ 5 \end{array} \right\}, \quad \left\{ \begin{array}{c} 1 \\ 4 \\ 2 \end{array} \right\}, \quad \left\{ \begin{array}{c} 1 \\ 3 \\ 5 \end{array} \right\}, \quad \left\{ \begin{array}{c} 1 \\ 3 \\ 9 \end{array} \right\} \quad \right\}$$



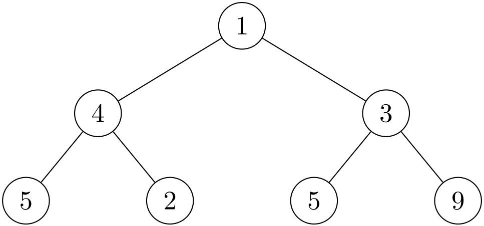
$$\mathcal{Q}_T = \left\{ \quad \left\{ \begin{matrix} 1 \\ 4 \\ 5 \end{matrix} \right\}, \quad \left\{ \begin{matrix} 1 \\ 4 \\ 2 \end{matrix} \right\}, \quad \left\{ \begin{matrix} 1 \\ 3 \\ 5 \end{matrix} \right\}, \quad \left\{ \begin{matrix} 1 \\ 3 \\ 9 \end{matrix} \right\} \quad \right\}$$



$$\mathcal{Q}_T = \left\{ \quad \left\{ \begin{array}{c} 1 \\ 4 \\ 5 \end{array} \right\}, \quad \left\{ \begin{array}{c} 1 \\ 4 \\ 2 \end{array} \right\}, \quad \left\{ \begin{array}{c} 1 \\ 3 \\ 5 \end{array} \right\}, \quad \left\{ \begin{array}{c} 1 \\ 3 \\ 9 \end{array} \right\} \quad \right\}$$



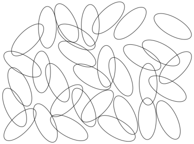
$$\mathcal{Q}_T = \left\{ \quad \left\{\begin{matrix}1\\4\\5\end{matrix}\right\}, \quad \left\{\begin{matrix}1\\4\\2\end{matrix}\right\}, \quad \left\{\begin{matrix}1\\3\\5\end{matrix}\right\}, \quad \left\{\begin{matrix}1\\3\\9\end{matrix}\right\} \quad \right\}$$



$$\mathcal{Q}_T = \left\{ \left\{ \begin{array}{c} 1 \\ 4 \\ 5 \end{array} \right\}, \left\{ \begin{array}{c} 1 \\ 4 \\ 2 \end{array} \right\}, \left\{ \begin{array}{c} 1 \\ 3 \\ 5 \end{array} \right\}, \left\{ \begin{array}{c} 1 \\ 3 \\ 9 \end{array} \right\} \right\}$$

$$\mathcal{Q} = \bigcup_T \mathcal{Q}_T$$

Daisy partition theorem

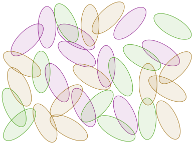


$\mathcal{Q}$

An *arbitrary* $\mathcal{Q}$ ($|\mathcal{Q}| \approx n$) can be partitioned into $\mathcal{D}_0, \ldots, \mathcal{D}_q$.
$\mathcal{D}_i$ has:

- petals of size i
- small kernel $|K_i|$
- small intersection

Daisy partition theorem

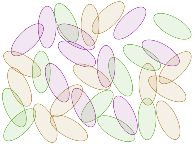


$\mathcal{Q}$

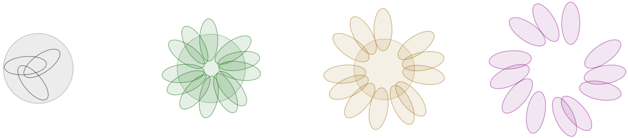
An *arbitrary* $\mathcal{Q}$ ($|\mathcal{Q}| \approx n$) can be partitioned into $\mathcal{D}_0, \ldots, \mathcal{D}_q$.
$\mathcal{D}_i$ has:

- petals of size i
- small kernel $|K_i|$
- small intersection

Daisy partition theorem



$\downarrow$

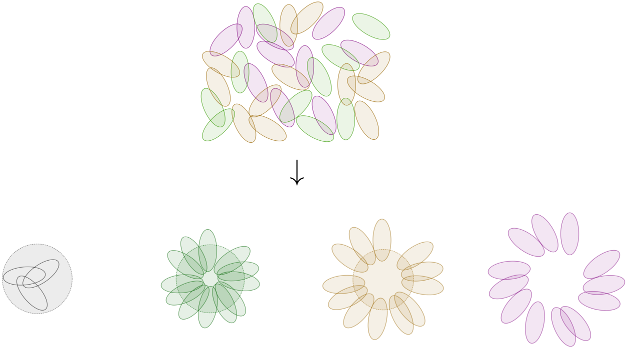


$\mathcal{D}_0$ $\mathcal{D}_1$ $\mathcal{D}_2$ $\mathcal{D}_3$

An *arbitrary* $\mathcal{Q}$ ($|\mathcal{Q}| \approx n$) can be partitioned into $\mathcal{D}_0, \ldots, \mathcal{D}_q$.
$\mathcal{D}_i$ has:

- petals of size i
- small kernel $|K_i|$
- small intersection

Daisy partition theorem



$\mathcal{D}_0$ $\mathcal{D}_1$ $\mathcal{D}_2$ $\mathcal{D}_3$

An *arbitrary* $\mathcal{Q}$ ($|\mathcal{Q}| \approx n$) can be partitioned into $\mathcal{D}_0, \ldots, \mathcal{D}_q$.
$\mathcal{D}_i$ has:

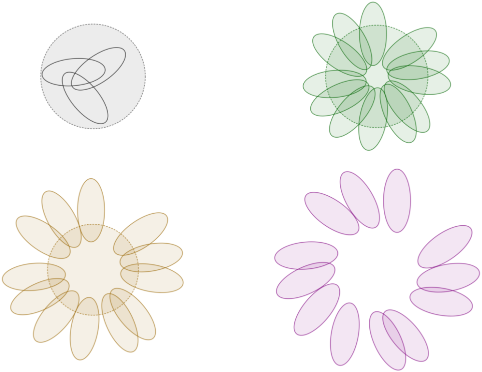
- petals of size i
- small kernel $|K_i|$ ($\implies$ few kernel assignments)
- small intersection (+ Hajnal-Szemerédi $\implies$ many sampled petals)

The algorithm

1. Sample each element of $[n]$ with probability p.
2. For every $i \in [q]$ and assignment κ_i to K_i:
   If L votes 1 on $\geq \tau_i$ sets with sampled petals, output 1.
3. Otherwise, output 0.

The algorithm

1. Sample each element of $[n]$ with probability p.
2. For every $i \in [q]$ and assignment κ_i to K_i:
   If L votes 1 on $\geq \tau_i$ sets with sampled petals, output 1.
3. Otherwise, output 0.

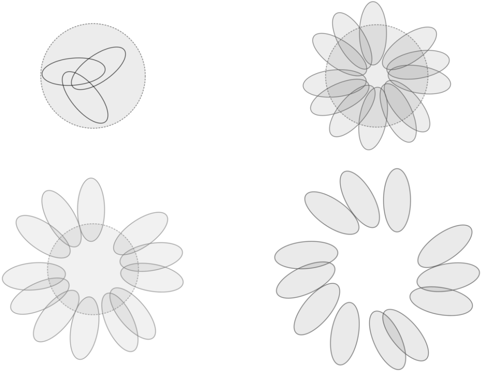


The algorithm

1. **Sample each element of $[n]$ with probability p.**

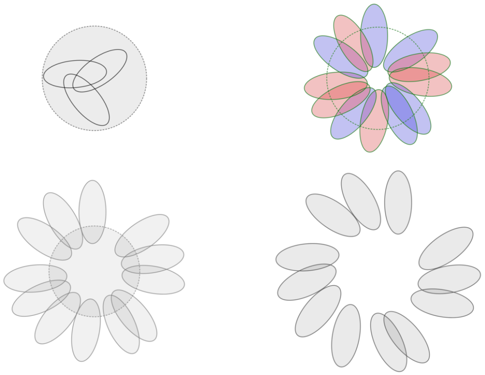
2. For every $i \in [q]$ and assignment κ_i to K_i:
   If L votes 1 on $\geq \tau_i$ sets with sampled petals, output 1.

3. Otherwise, output 0.



The algorithm

1. Sample each element of $[n]$ with probability p.
2. **For every $i \in [q]$ and assignment κ_i to K_i:**
   If L votes 1 on $\geq \tau_i$ sets with sampled petals, output 1.
3. Otherwise, output 0.

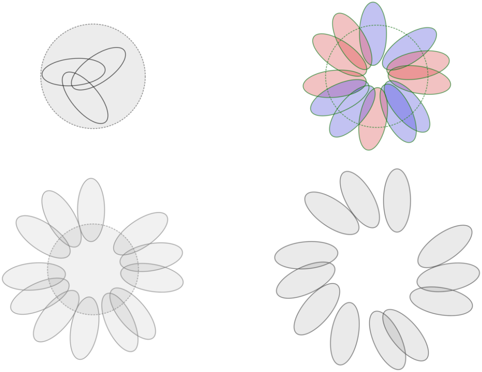


The algorithm

1. Sample each element of $[n]$ with probability p.

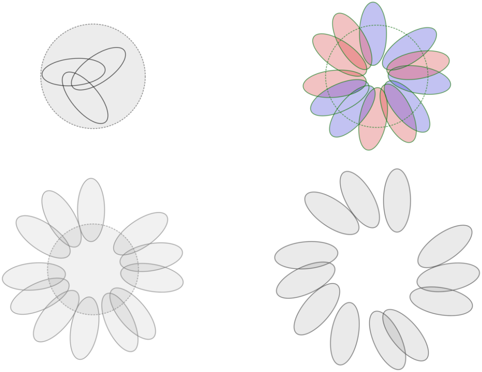
2. **For every $i \in [q]$ and assignment κ_i to K_i:**
   If L votes 1 on $\geq \tau_i$ sets with sampled petals, output 1.

3. Otherwise, output 0.



The algorithm

1. Sample each element of $[n]$ with probability p.
2. **For every $i \in [q]$ and assignment κ_i to K_i:**
   If L votes 1 on $\geq \tau_i$ sets with sampled petals, output 1.
3. Otherwise, output 0.

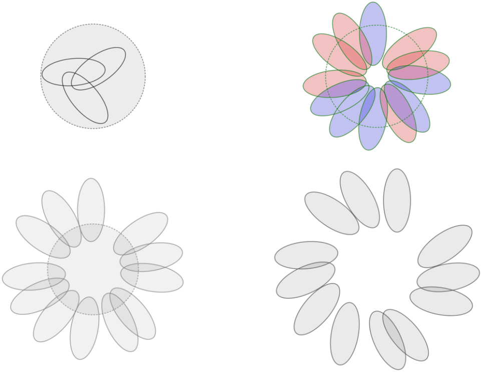


The algorithm

1. Sample each element of $[n]$ with probability p.

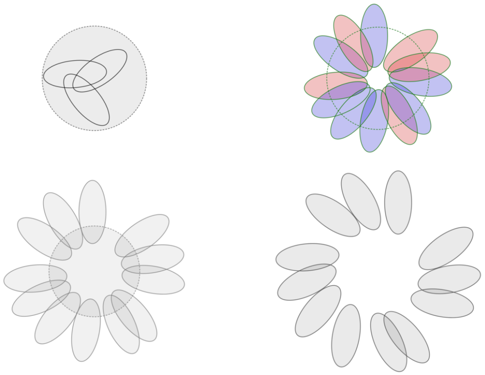
2. **For every** $i \in [q]$ **and assignment** κ_i **to** K_i**:**
   If L votes 1 on $\geq \tau_i$ sets with sampled petals, output 1.

3. Otherwise, output 0.



The algorithm

1. Sample each element of $[n]$ with probability p.
2. **For every $i \in [q]$ and assignment κ_i to K_i:**
   If L votes 1 on $\geq \tau_i$ sets with sampled petals, output 1.
3. Otherwise, output 0.

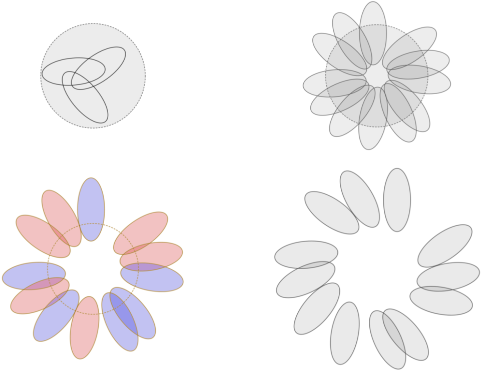


The algorithm

1. Sample each element of $[n]$ with probability p.

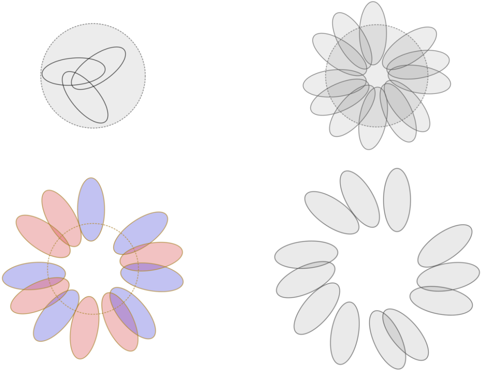
2. **For every $i \in [q]$ and assignment κ_i to K_i:**
   If L votes 1 on $\geq \tau_i$ sets with sampled petals, output 1.

3. Otherwise, output 0.



The algorithm

1. Sample each element of $[n]$ with probability p.
2. **For every $i \in [q]$ and assignment κ_i to K_i:**
   If L votes 1 on $\geq \tau_i$ sets with sampled petals, output 1.
3. Otherwise, output 0.

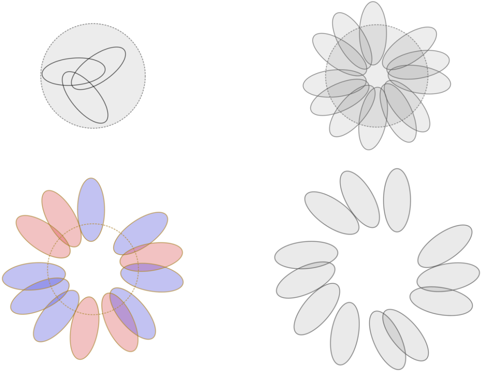


The algorithm

1. Sample each element of $[n]$ with probability p.

2. **For every $i \in [q]$ and assignment κ_i to K_i:**
   If L votes 1 on $\geq \tau_i$ sets with sampled petals, output 1.

3. Otherwise, output 0.

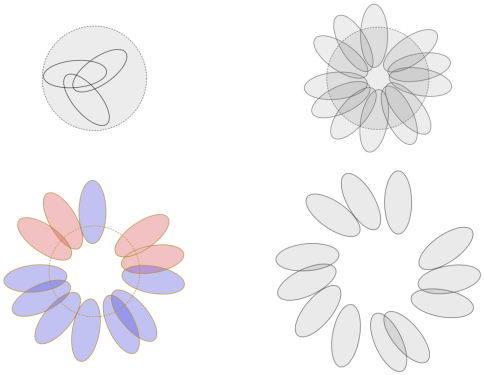


The algorithm

1. Sample each element of $[n]$ with probability p.

2. For every $i \in [q]$ and assignment κ_i to K_i:
   If L votes 1 on $\geq \tau_i$ sets with sampled petals, output 1.

3. Otherwise, output 0.



Open problems

- Closing the quadratic gap between RLDC blocklength lower and upper bounds ($k^{1+\tilde{\Omega}(1/q^2)}$ vs. $k^{1+O(1/q)}$).
- PCPP lower bounds by similar techniques?
- Capturing, e.g., PAC learning or local computation algorithms?

References

Asadi, V. R. and Shinkar, I. (2020).
Relaxed locally correctable codes with improved parameters.
arXiv preprint arXiv:2009.07311.

Ben-Sasson, E., Goldreich, O., Harsha, P., Sudan, M., and Vadhan, S. P. (2004).
Robust PCPs of proximity, shorter PCPs and applications to coding.

Fischer, E., Lachish, O., and Vasudev, Y. (2015).
Trading query complexity for sample-based testing and multi-testing scalability.

Goldreich, O. and Ron, D. (2016).
On sample-based testers.
ACM Transactions on Computation Theory (TOCT), 8(2):1–54.

Gur, T. and Lachish, O. (2020).
On the power of relaxed local decoding algorithms.

Gur, T. and Rothblum, R. D. (2018).
Non-interactive proofs of proximity.