Finding monotone patterns

Shoham Letzter

University College London

joint with Omri Ben-Eliezer and Erik Waingarten

ICALP

July 2022

Property testing

Aim: design fast (randomised) algorithms that determine, with probability at least 0.99, if a given (large) object

- has property $\mathcal{P}$,
- or is far from having property $\mathcal{P}$.

Property testing

Aim: design fast (randomised) algorithms that determine, with probability at least 0.99, if a given (large) object

- has property $\mathcal{P}$,
- or is far from having property $\mathcal{P}$.

We consider testing with **one-sided error**:

Property testing

Aim: design fast (randomised) algorithms that determine, with probability at least 0.99, if a given (large) object

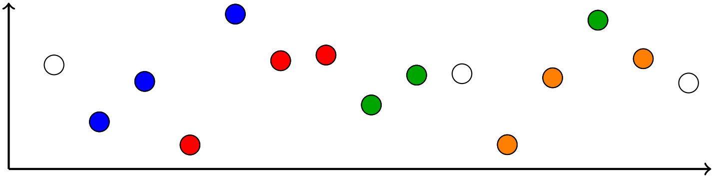
- has property $\mathcal{P}$,
- or is far from having property $\mathcal{P}$.

We consider testing with **one-sided error**: given an object which is far from having $\mathcal{P}$, provide evidence of not being in $\mathcal{P}$, with high probability.

Testing for $(1 \dots k)$-freeness

Fix $k \geq 2$.

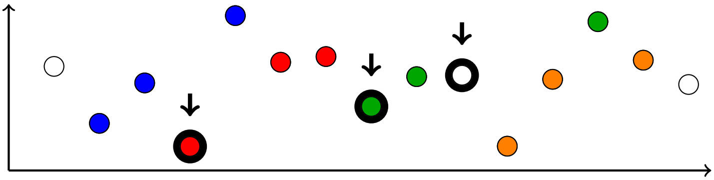
- **Input.** $f : [n] \to \mathbb{R}$ with $\Omega(n)$ disjoint increasing k-tuples.



Testing for $(1 \ldots k)$-freeness

Fix $k \geq 2$.

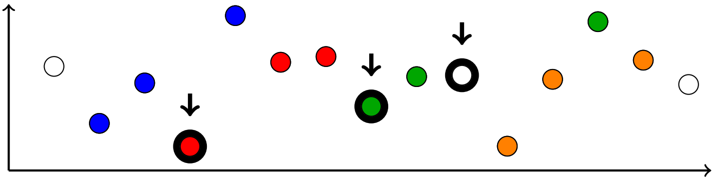
- **Input.** $f : [n] \to \mathbb{R}$ with $\Omega(n)$ disjoint increasing k-tuples.
- **Aim.** Find, with high probability, an increasing k-tuple.



Testing for $(1 \ldots k)$-freeness

Fix $k \geq 2$.

- **Input.** $f : [n] \to \mathbb{R}$ with $\Omega(n)$ disjoint increasing k-tuples.
- **Aim.** Find, with high probability, an increasing k-tuple.



We sometimes refer to an increasing k-tuple as a **$(1 \ldots k)$-copy**.

History

History

$\boldsymbol{k = 2}$**:** monotonicity testing (with one-sided error).

History

k $= 2$**:** monotonicity testing (with one-sided error).

Ergün–Kannan–Kumar–Rubinfeld–Viswanathan ’98. Optimal non-adaptive monotonicity testers make $\Theta(\log n)$ queries.

(non-adaptivity: queries do not depend on previous outcomes.)

History

$k = 2$: monotonicity testing (with one-sided error).

Ergün–Kannan–Kumar–Rubinfeld–Viswanathan '98. Optimal non-adaptive monotonicity testers make $\Theta(\log n)$ queries.

(non-adaptivity: queries do not depend on previous outcomes.)

Fischer '09. Adaptivity does not help monotonicity testing!

History

$\boldsymbol{k = 2}$**:** monotonicity testing (with one-sided error).

Ergün–Kannan–Kumar–Rubinfeld–Viswanathan '98. Optimal non-adaptive monotonicity testers make $\Theta(\log n)$ queries.

(non-adaptivity: queries do not depend on previous outcomes.)

Fischer '09. Adaptivity does not help monotonicity testing!

Newman–Rabinovich–Rajendraprasad–Sohler '17. For $k \geq 2$, there is a (non-adaptive) tester which makes $(\log n)^{O(k^2)}$ queries.

Our results

Theorem (Ben-Eliezer–Canonne–L.–Waingarten)

An optimal non-adaptive algorithm for testing $(1\dots k)$-freeness makes $\Theta_k\left((\log n)^{\lfloor \log_2 k \rfloor}\right)$ queries.

Our results

Theorem (Ben-Eliezer–Canonne–L.–Waingarten)

An optimal non-adaptive algorithm for testing $(1\ldots k)$-freeness makes $\Theta_k\left((\log n)^{\lfloor \log_2 k \rfloor}\right)$ queries.

Theorem (Ben-Eliezer–L.–Waingarten)

An optimal adaptive algorithm for testing $(1\ldots k)$-freeness makes $\Theta_k(\log n)$ queries.

Structure theorem (BCLW '19) – first outcome

Two outcomes if $f : [n] \to \mathbb{R}$ has $\Omega(n)$ disjoint increasing k-tuples:

Structure theorem (BCLW '19) – first outcome

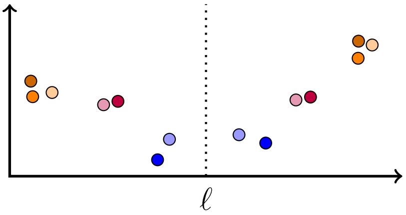
Two outcomes if $f : [n] \to \mathbb{R}$ has $\Omega(n)$ disjoint increasing k-tuples:

1. f **is chaotic**:

Structure theorem (BCLW '19) – first outcome

Two outcomes if $f : [n] \to \mathbb{R}$ has $\Omega(n)$ disjoint increasing k-tuples:

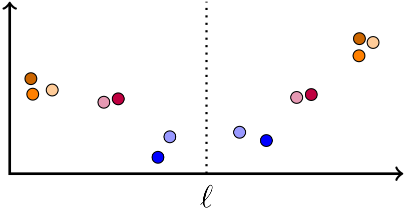
1. f **is chaotic**: there are $\Omega(n)$ values of ℓ like this:



Structure theorem (BCLW '19) – first outcome

Two outcomes if $f : [n] \to \mathbb{R}$ has $\Omega(n)$ disjoint increasing k-tuples:

1. f **is chaotic**: there are $\Omega(n)$ values of ℓ like this:

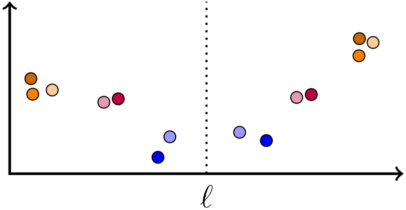


Can find (w.h.p.) an increasing k-tuple with $\Theta(\log n)$ queries:

Structure theorem (BCLW '19) – first outcome

Two outcomes if $f : [n] \to \mathbb{R}$ has $\Omega(n)$ disjoint increasing k-tuples:

1. f **is chaotic**: there are $\Omega(n)$ values of ℓ like this:



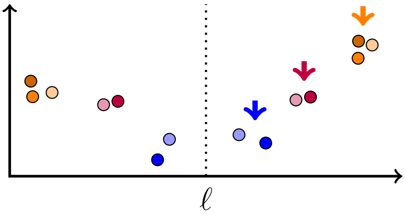
Can find (w.h.p.) an increasing k-tuple with $\Theta(\log n)$ queries:

- $\Theta(1)$ queries to find good ℓ,

Structure theorem (BCLW '19) – first outcome

Two outcomes if $f : [n] \to \mathbb{R}$ has $\Omega(n)$ disjoint increasing k-tuples:

1. f **is chaotic**: there are $\Omega(n)$ values of ℓ like this:



Can find (w.h.p.) an increasing k-tuple with $\Theta(\log n)$ queries:

- $\Theta(1)$ queries to find good ℓ,
- $\Theta(1)$ queries in $[\ell, \ell + 2^i]$ for $i \in [\log n]$ and sampled ℓ.

Structure thereom (BCLW '19) – second outcome

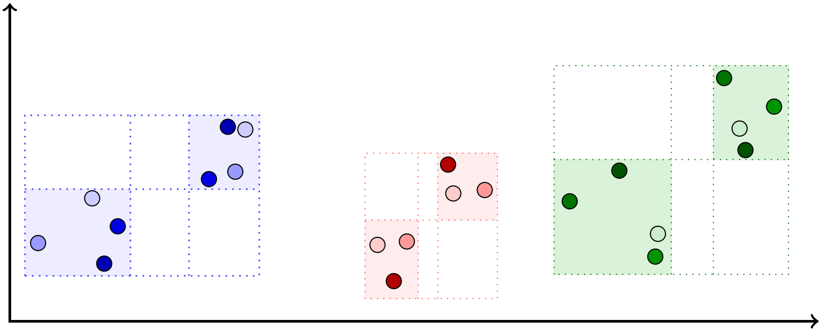
Two outcomes if $f : [n] \to \mathbb{R}$ has $\Omega(n)$ disjoint increasing k-tuples:

2. f **is structured**:

Structure thereom (BCLW '19) – second outcome

Two outcomes if $f : [n] \to \mathbb{R}$ has $\Omega(n)$ disjoint increasing k-tuples:

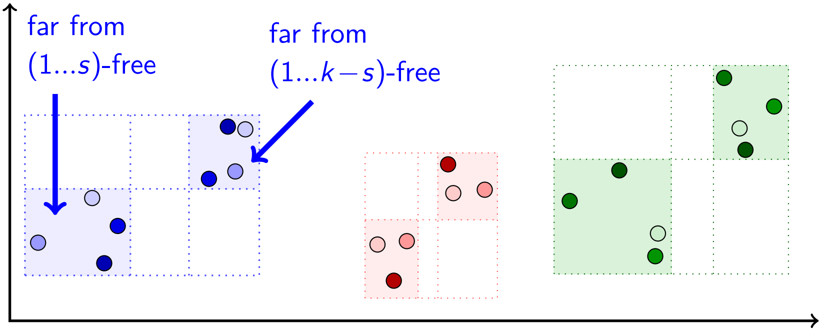
2. f **is structured**: can cover $\Omega(n)$ entries with disjoint 'splittable intervals'.



Structure thereom (BCLW '19) – second outcome

Two outcomes if $f : [n] \to \mathbb{R}$ has $\Omega(n)$ disjoint increasing k-tuples:

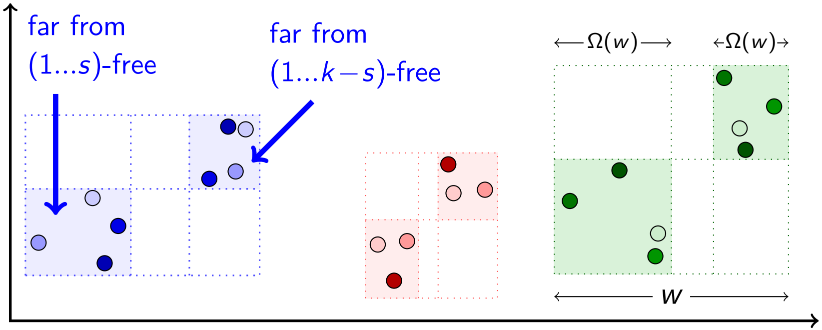
2. f **is structured**: can cover $\Omega(n)$ entries with disjoint 'splittable intervals'.



Structure thereom (BCLW '19) – second outcome

Two outcomes if $f : [n] \to \mathbb{R}$ has $\Omega(n)$ disjoint increasing k-tuples:

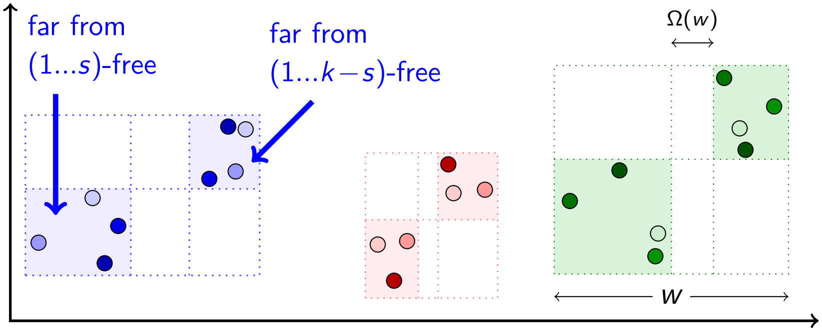
2. f **is structured**: can cover $\Omega(n)$ entries with disjoint 'splittable intervals'.



Structure thereom (BCLW '19) – second outcome

Two outcomes if $f : [n] \to \mathbb{R}$ has $\Omega(n)$ disjoint increasing k-tuples:

2. f **is structured**: can cover $\Omega(n)$ entries with disjoint 'splittable intervals'.



A combinatorial lemma

Lemma

Let $\mathcal{I}$ be a family of disjoint intervals in $[n]$ s.t. $\sum_{I \in \mathcal{I}} |I| \geq \alpha n$.

A combinatorial lemma

Lemma

Let $\mathcal{I}$ be a family of disjoint intervals in $[n]$ s.t. $\sum_{I \in \mathcal{I}} |I| \geq \alpha n$. Then there is $\mathcal{J} \subseteq \mathcal{I}$ s.t.

- $\sum_{J \in \mathcal{J}} |J| \geq \frac{\alpha}{4} n$,

A combinatorial lemma

Lemma

Let $\mathcal{I}$ be a family of disjoint intervals in $[n]$ s.t. $\sum_{I \in \mathcal{I}} |I| \geq \alpha n$. Then there is $\mathcal{J} \subseteq \mathcal{I}$ s.t.

- $\sum_{J \in \mathcal{J}} |J| \geq \frac{\alpha}{4} n$,
- *if an interval K contains $J \in \mathcal{J}$, then $\sum_{I \in \mathcal{I}:\, I \subseteq K} |I| \geq \frac{\alpha}{4} |K|$.*

A combinatorial lemma

Lemma

Let $\mathcal{I}$ be a family of disjoint intervals in $[n]$ s.t. $\sum_{I \in \mathcal{I}} |I| \geq \alpha n$. Then there is $\mathcal{J} \subseteq \mathcal{I}$ s.t.

- $\sum_{J \in \mathcal{J}} |J| \geq \frac{\alpha}{4} n$,
- *if an interval K contains $J \in \mathcal{J}$, then* $\sum_{I \in \mathcal{I}:\, I \subseteq K} |I| \geq \frac{\alpha}{4} |K|$.

A combinatorial lemma

Lemma

Let $\mathcal{I}$ be a family of disjoint intervals in $[n]$ s.t. $\sum_{I \in \mathcal{I}} |I| \geq \alpha n$. Then there is $\mathcal{J} \subseteq \mathcal{I}$ s.t.

- $\sum_{J \in \mathcal{J}} |J| \geq \frac{\alpha}{4} n$,
- *if an interval K contains $J \in \mathcal{J}$, then* $\sum_{I \in \mathcal{I}:\, I \subseteq K} |I| \geq \frac{\alpha}{4} |K|$.

A combinatorial lemma

Lemma

Let $\mathcal{I}$ be a family of disjoint intervals in $[n]$ s.t. $\sum_{I\in\mathcal{I}} |I| \geq \alpha n$. Then there is $\mathcal{J} \subseteq \mathcal{I}$ s.t.

- $\sum_{J\in\mathcal{J}} |J| \geq \frac{\alpha}{4} n$,
- *if an interval K contains $J \in \mathcal{J}$, then* $\sum_{I\in\mathcal{I}:\, I\subseteq K} |I| \geq \frac{\alpha}{4}|K|$.

A combinatorial lemma

Lemma

Let $\mathcal{I}$ be a family of disjoint intervals in $[n]$ s.t. $\sum_{I\in\mathcal{I}} |I| \geq \alpha n$. Then there is $\mathcal{J} \subseteq \mathcal{I}$ s.t.

- $\sum_{J\in\mathcal{J}} |J| \geq \frac{\alpha}{4} n$,
- *if an interval K contains $J \in \mathcal{J}$, then $\sum_{I\in\mathcal{I}:\, I\subseteq K} |I| \geq \frac{\alpha}{4}|K|$.*

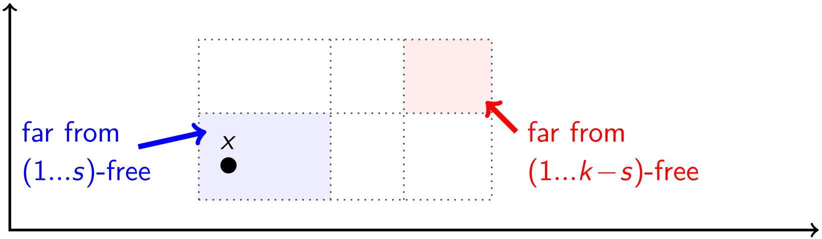
Thus, if f is structured, there is a **robust collection** of disjoint 'splittable intervals' $\mathcal{J}$ s.t.

- $\sum_{J\in\mathcal{J}} |J| = \Omega(n)$,
- if K contains $J \in \mathcal{J}$ then K has $\Omega(|K|)$ disjoint increasing k-tuples.

Handling the structured case

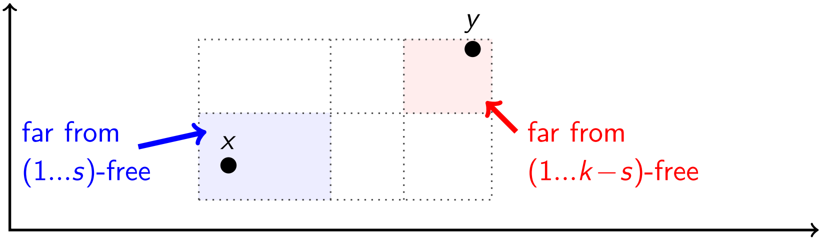
Handling the structured case

- $\Theta(1)$ queries to find x in bottom-left of robust splittable interval I.



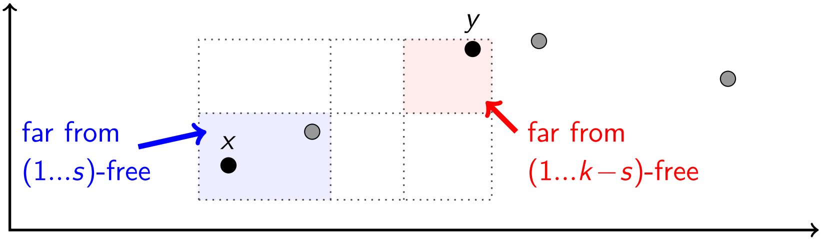
Handling the structured case

- $\Theta(1)$ queries to find x in bottom-left of robust splittable interval I.
- $\Theta(\log n)$ queries to find y in top-right of I.



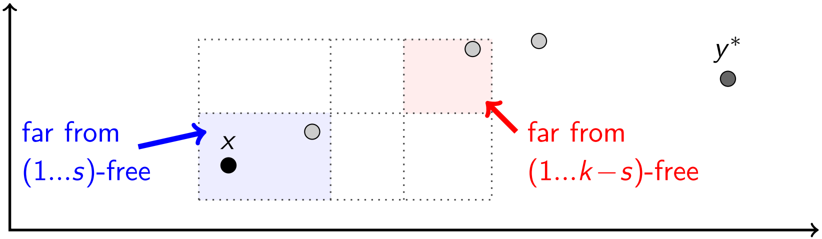
Handling the structured case

- $\Theta(1)$ queries to find x in bottom-left of robust splittable interval I.
- $\Theta(\log n)$ queries to find y in top-right of I.



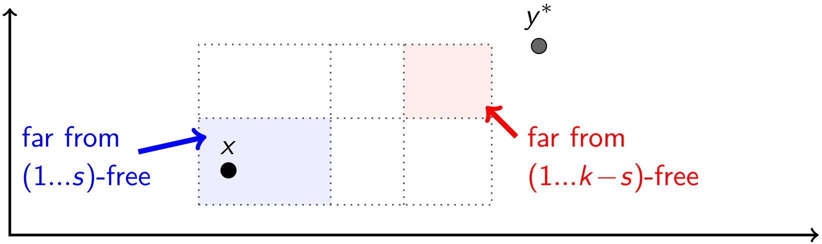
Handling the structured case

- $\Theta(1)$ queries to find x in bottom-left of robust splittable interval I.
- $\Theta(\log n)$ queries to find y in top-right of I.
- $y^* =$ maximal sampled element with $f(y^*) > f(x)$.



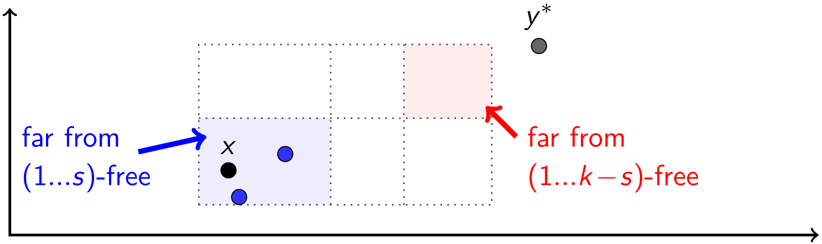
Handling the structured case

- $\Theta(1)$ queries to find x in bottom-left of robust splittable interval I.
- $\Theta(\log n)$ queries to find y in top-right of I.
- $y^* =$ maximal sampled element with $f(y^*) > f(x)$.
- If $y^* \approx y$:



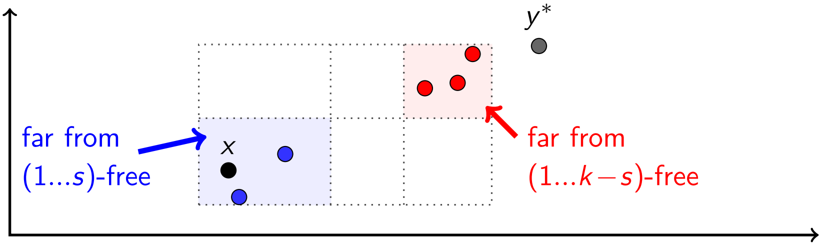
Handling the structured case

- $\Theta(1)$ queries to find x in bottom-left of robust splittable interval I.
- $\Theta(\log n)$ queries to find y in top-right of I.
- $y^* =$ maximal sampled element with $f(y^*) > f(x)$.
- If $y^* \approx y$: $\Theta(\log n)$ queries to find increasing s-tuple π_1 near x



Handling the structured case

- $\Theta(1)$ queries to find x in bottom-left of robust splittable interval I.
- $\Theta(\log n)$ queries to find y in top-right of I.
- $y^* =$ maximal sampled element with $f(y^*) > f(x)$.
- If $y^* \approx y$: $\Theta(\log n)$ queries to find increasing s-tuple π_1 near x and $(k-s)$-tuple π_2 near y^* and above π_1.

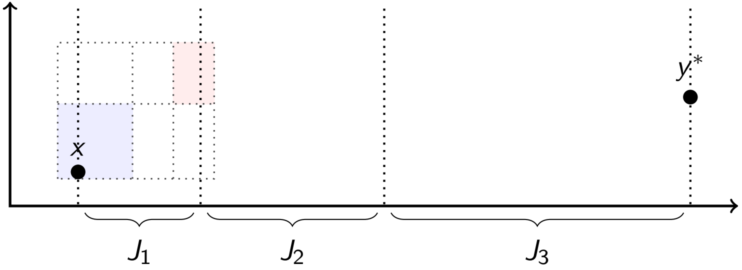


Handling overshooting in the structured case – I

- x in bottom-left of robust splittable interval I,
- y in top-right of I,
- $y^* \gg y$ and $f(y^*) > f(x)$.

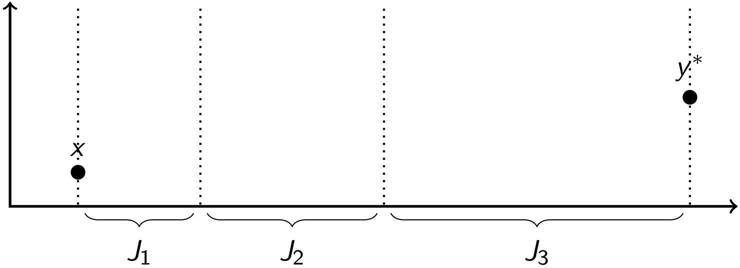
Handling overshooting in the structured case – I

- x in bottom-left of robust splittable interval I,
- y in top-right of I,
- $y^* \gg y$ and $f(y^*) > f(x)$.
- Take consecutive intervals $J_1, \ldots, J_{k-2} \subseteq [x, y^*]$ s.t. $|J_{i+1}| \gg |J_i|$.



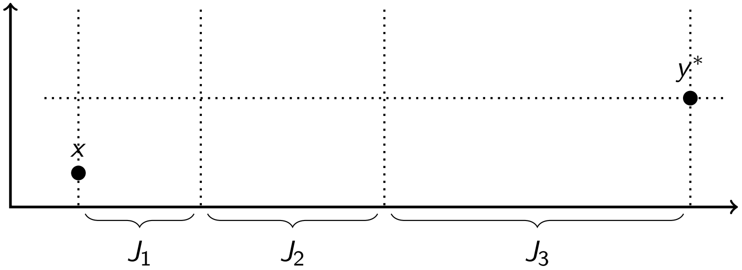
Handling overshooting in the structured case – I

- x in bottom-left of robust splittable interval I,
- y in top-right of I,
- $y^* \gg y$ and $f(y^*) > f(x)$.
- Take consecutive intervals $J_1, \ldots, J_{k-2} \subseteq [x, y^*]$ s.t. $|J_{i+1}| \gg |J_i|$.
  By robustness: J_i has $\Omega(|J_i|)$ increasing k-tuples.



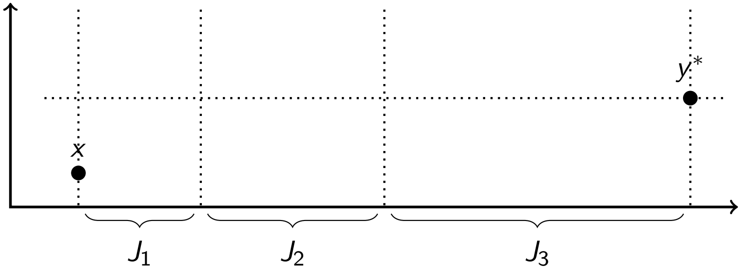
Handling overshooting in the structured case – II

- $y^* \gg y$ and $f(y^*) > f(x)$.
- $J_1, \ldots, J_{k-2} \subseteq [x, y^*]$ consecutive intervals, J_i has $\Omega(|J_i|)$ increasing k-tuples.



Handling overshooting in the structured case – II

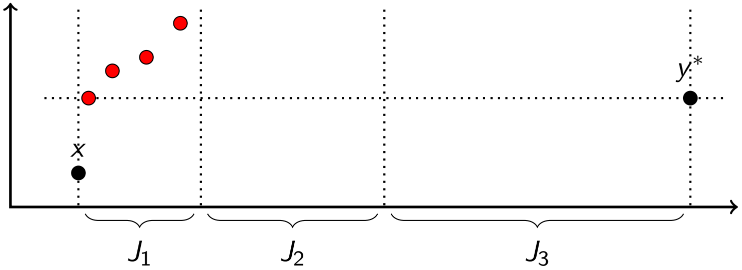
- $y^* \gg y$ and $f(y^*) > f(x)$.
- $J_1, \ldots, J_{k-2} \subseteq [x, y^*]$ consecutive intervals, J_i has $\Omega(|J_i|)$ increasing k-tuples. So,
  1. J_i has $\Omega(|J_i|)$ increasing $(i+1)$-tuples strictly below $f(y^*)$ (case Ai), or
  2. J_i has $\Omega(|J_i|)$ increasing $(k-i)$-tuples above $f(y^*)$ (case Bi).



Handling overshooting in the structured case – II

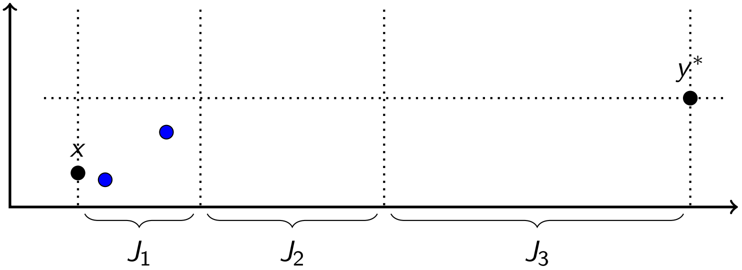
- $y^* \gg y$ and $f(y^*) > f(x)$.
- $J_1, \ldots, J_{k-2} \subseteq [x, y^*]$ consecutive intervals, J_i has $\Omega(|J_i|)$ increasing k-tuples. So,
  1. J_i has $\Omega(|J_i|)$ increasing $(i+1)$-tuples strictly below $f(y^*)$ (case Ai), or
  2. J_i has $\Omega(|J_i|)$ increasing $(k-i)$-tuples above $f(y^*)$ (case Bi).

- B1



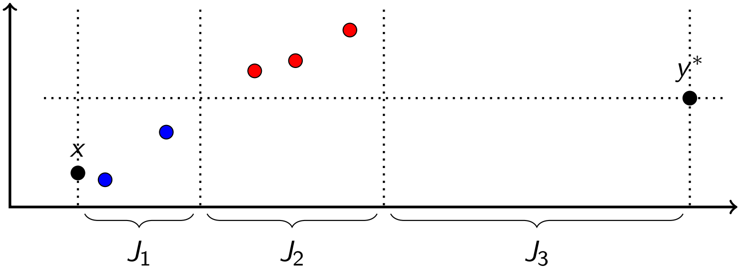
Handling overshooting in the structured case – II

- $y^* \gg y$ and $f(y^*) > f(x)$.
- $J_1, \ldots, J_{k-2} \subseteq [x, y^*]$ consecutive intervals, J_i has $\Omega(|J_i|)$ increasing k-tuples. So,
  1. J_i has $\Omega(|J_i|)$ increasing $(i+1)$-tuples strictly below $f(y^*)$ (case Ai), or
  2. J_i has $\Omega(|J_i|)$ increasing $(k-i)$-tuples above $f(y^*)$ (case Bi).
- A1,



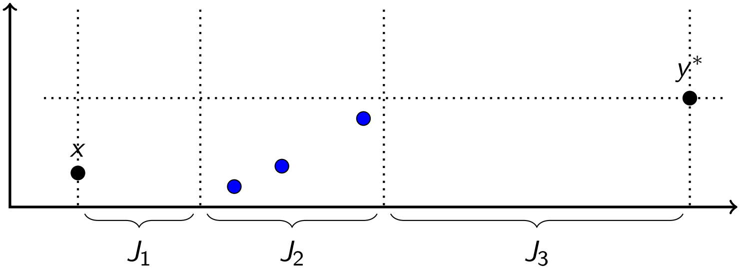
Handling overshooting in the structured case – II

- $y^* \gg y$ and $f(y^*) > f(x)$.
- $J_1, \ldots, J_{k-2} \subseteq [x, y^*]$ consecutive intervals, J_i has $\Omega(|J_i|)$ increasing k-tuples. So,
  1. J_i has $\Omega(|J_i|)$ increasing $(i+1)$-tuples strictly below $f(y^*)$ (case Ai), or
  2. J_i has $\Omega(|J_i|)$ increasing $(k-i)$-tuples above $f(y^*)$ (case Bi).
- A1, B2



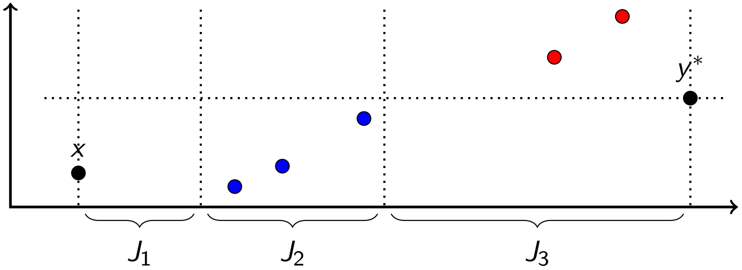
Handling overshooting in the structured case – II

- $y^* \gg y$ and $f(y^*) > f(x)$.
- $J_1, \ldots, J_{k-2} \subseteq [x, y^*]$ consecutive intervals, J_i has $\Omega(|J_i|)$ increasing k-tuples. So,
  1. J_i has $\Omega(|J_i|)$ increasing $(i+1)$-tuples strictly below $f(y^*)$ (case Ai), or
  2. J_i has $\Omega(|J_i|)$ increasing $(k-i)$-tuples above $f(y^*)$ (case Bi).
- A2,



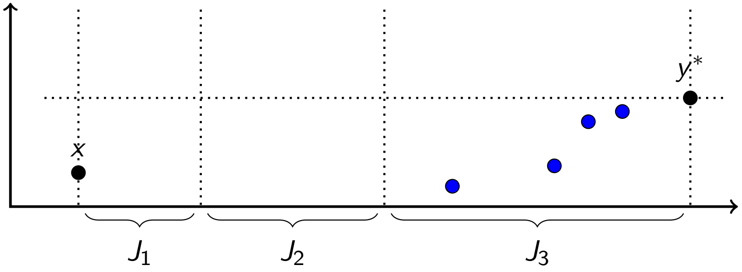
Handling overshooting in the structured case – II

- $y^* \gg y$ and $f(y^*) > f(x)$.
- $J_1, \ldots, J_{k-2} \subseteq [x, y^*]$ consecutive intervals, J_i has $\Omega(|J_i|)$ increasing k-tuples. So,
  1. J_i has $\Omega(|J_i|)$ increasing $(i+1)$-tuples strictly below $f(y^*)$ (case Ai), or
  2. J_i has $\Omega(|J_i|)$ increasing $(k-i)$-tuples above $f(y^*)$ (case Bi).
- A2, B3



Handling overshooting in the structured case – II

- $y^* \gg y$ and $f(y^*) > f(x)$.
- $J_1, \ldots, J_{k-2} \subseteq [x, y^*]$ consecutive intervals, J_i has $\Omega(|J_i|)$ increasing k-tuples. So,
  1. J_i has $\Omega(|J_i|)$ increasing $(i+1)$-tuples strictly below $f(y^*)$ (case Ai), or
  2. J_i has $\Omega(|J_i|)$ increasing $(k-i)$-tuples above $f(y^*)$ (case Bi).
- A3



Open problems

Open problems

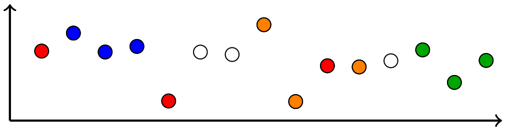
- k not fixed?

Open problems

- k not fixed?
- **Testing for other permutations**.

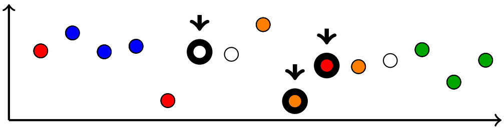
Open problems

- k not fixed?
- **Testing for other permutations**. E.g. $\pi = (312)$.



Open problems

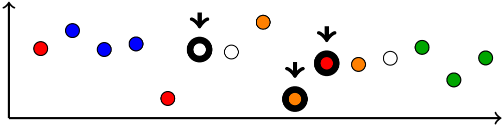
- k not fixed?
- **Testing for other permutations**. E.g. $\pi = (312)$.



Open problems

- k not fixed?

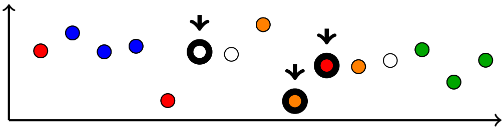
- **Testing for other permutations**. E.g. $\pi = (312)$.



Newman–Varma '21: adaptive π-freeness tester with $n^{o(1)}$ queries.

Open problems

- k not fixed?
- **Testing for other permutations**. E.g. $\pi = (312)$.

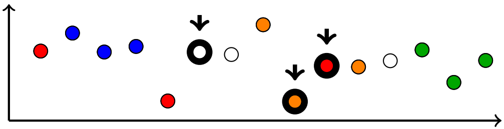


Newman–Varma '21: adaptive π-freeness tester with $n^{o(1)}$ queries.

Is there such an algorithm using polylog n queries?

Open problems

- k not fixed?
- **Testing for other permutations**. E.g. $\pi = (312)$.



Newman–Varma '21: adaptive π-freeness tester with $n^{o(1)}$ queries.

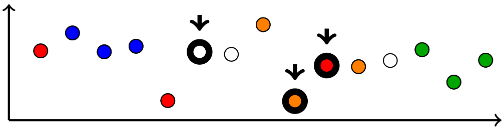
Is there such an algorithm using polylog n queries?

- Finding a π-copy (length k) in a permutation of length n:

Fox, '13. $2^{O(k^2)}n$.

Open problems

- k not fixed?
- **Testing for other permutations**. E.g. $\pi = (312)$.



Newman–Varma '21: adaptive π-freeness tester with $n^{o(1)}$ queries.

Is there such an algorithm using polylog n queries?

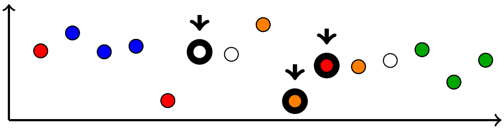
- Finding a π-copy (length k) in a permutation of length n:

Fox, '13. $2^{O(k^2)}n$. Better algorithms?

Open problems

- k not fixed?

- **Testing for other permutations**. E.g. $\pi = (312)$.



Newman–Varma '21: adaptive π-freeness tester with $n^{o(1)}$ queries.

Is there such an algorithm using polylog n queries?

- Finding a π-copy (length k) in a permutation of length n:

Fox, '13. $2^{O(k^2)}n$. Better algorithms?

Thank you!!!