Quality Control for Pattern Counting



Cassandra Marcussen
Harvard



Ronitt Rubinfeld
MIT



Madhu Sudan
Harvard

Based on arxiv:2508.16531 (FOCS 2026) and arxiv:2608.03439 (SODA 2027)

Agenda

- Quality control:
  - Permutation counts
  - Broader discussion
- Permutation pattern counts
  - Query-efficient algorithm
  - Runtime-efficient algorithm
- Graphs
- Shared themes and open problems

Guiding question

Can we **efficiently check** whether a **fixed input** satisfies the **predictions** we rely on from a **random model**?

Random models give useful guarantees.

But in practice we have one fixed input.

Example: Does my data look i.i.d.?

i.i.d. = Independent and identically distributed

This is a popular distributional assumption that is impossible to verify on fixed data!

Example: Does my data look i.i.d.?

i.i.d. = Independent and identically distributed

Many examples of papers that **design theoretical algorithms assuming i.i.d. data**

Will the algorithm work **on my data**?



A Practical Algorithm for Topic Modeling with Provable Guarantees

Sanjeev Arora
Rong Ge
Yoni Halpern
David Mimno
Ankur Moitra
David Sontag
Yichen Wu
Michael Zhu

Abstract

Topic models provide a useful method for dimensionality reduction and exploratory data analysis in large text corpora. Most approaches to topic model learning have been based on a maximum likelihood objective. Efficient algorithms exist that attempt to approximate this objective, but they have no provable guarantees. Recently, algorithms have been introduced that provide provable bounds, but these algorithms are not practical because they are inefficient and not robust to violations of model assumptions. In this paper we present an algorithm for learning topic models that is both provable and practical. The algorithm produces results comparable to the best MCMC implementations while running orders of magnitude faster.

Online Stochastic Packing applied to Display Ad Allocation

Jon Feldman* Monika Henzinger † Nitish Korula ‡ Vahab S. Mirrokni*
Cliff Stein§

May 28, 2018

Abstract

Inspired by online ad allocation, we study online stochastic packing linear programs from theoretical and practical standpoints. We first present a near-optimal online algorithm for a general class of packing linear programs which model various online resource allocation problems including online variants of routing, ad allocations, generalized assignment, and combinatorial auctions. As our main theoretical result, we prove that a simple primal-dual training-based algorithm achieves a $(1 - o(1))$-approximation guarantee in the random order stochastic model. This is a significant improvement over logarithmic or constant-factor approximations for the adversarial variants of the same problems (e.g. factor $1 - \frac{1}{e}$ for online ad allocation, and $\log(m)$ for online routing). We then focus on the online display ad allocation problem and study the efficiency and fairness of various training-based and online allocation algorithms on data sets collected from real-life display ad allocation system. Our experimental evaluation confirms the effectiveness of training-based primal-dual algorithms on real data sets, and also indicate an intrinsic trade-off between fairness and efficiency.

Spectral redemption in clustering sparse networks

Florent Krzakala[a,b], Cristopher Moore[c], Elchanan Mossel[d,1], Joe Neeman[d], Allan Sly[d], Lenka Zdeborová[e], and Pan Zhang[a,c]

[a]Ecole Supérieure de Physique et de Chimie Industrielles, 75005 Paris, France; [b]Ecole Normale Supérieure, 75005 Paris, France; [c]Santa Fe Institute, Santa Fe, NM 87501; [d]University of California, Berkeley, CA 94720; and [e]Institut de Physique Théorique, Commissariat à l'Energie Atomique Saclay and Unité de Recherche Associée 2306, Centre National de la Recherche Scientifique, 91190 Gif-sur-Yvette, France

Edited* by Peter J. Bickel, University of California, Berkeley, CA, and approved October 23, 2013 (received for review July 2, 2013)

Robust Principal Component Analysis?

EMMANUEL J. CANDÈS and XIAODONG LI, Stanford University
YI MA, University of Illinois at Urbana-Champaign, Microsoft Research Asia
JOHN WRIGHT, Microsoft Research Asia

This article is about a curious phenomenon. Suppose we have a data matrix, which is the superposition of a low-rank component and a sparse component. Can we recover each component individually? We prove that under some suitable assumptions, it is possible to recover both the low-rank and the sparse components *exactly* by solving a very convenient convex program called *Principal Component Pursuit*; among all feasible decompositions, simply minimize a weighted combination of the nuclear norm and of the ℓ_1 norm. This suggests the possibility of a principled approach to robust principal component analysis since our methodology and results assert that one can recover the principal components of a data matrix even though a positive fraction of its entries are arbitrarily corrupted. This extends to the situation where a fraction of the entries are missing as well. We discuss an algorithm for solving this optimization problem, and present applications in the area of video surveillance, where our methodology allows for the detection of objects in a cluttered background, and in the area of face recognition, where it offers a principled way of removing shadows and specularities in images of faces.

11

Example: Does my data look i.i.d.?

i.i.d. = Independent and identically distributed

Many examples of papers that **design theoretical algorithms assuming i.i.d. data**

→

Test their algorithms on **a set of real-world data**

→

Will the algorithm work **on my data**?



A Practical Algorithm for Topic Modeling with Provable Guarantees

Sanjeev Arora
Rong Ge
Yoni Halpern
David Mimno
Ankur Moitra
David Sontag
Yichen Wu
Michael Zhu

Abstract

Topic models provide a useful method for dimensionality reduction and exploratory data analysis in large text corpora. Most approaches to topic model learning have been based on a maximum likelihood objective. Efficient algorithms exist that attempt to approximate this objective, but they have no provable guarantees. Recently, algorithms have been introduced that provide provable bounds, but these algorithms are not practical because they are inefficient and not robust to violations of model assumptions. In this paper we present an algorithm for learning topic models that is both provable and practical. The algorithm produces results comparable to the best MCMC implementations while running orders of magnitude faster.

Online Stochastic Packing applied to Display Ad Allocation

Jon Feldman* Monika Henzinger † Nitish Korula ‡ Vahab S. Mirrokni*
Cliff Stein§

May 28, 2018

Abstract

Inspired by online ad allocation, we study online stochastic packing linear programs from theoretical and practical standpoints. We first present a near-optimal online algorithm for a general class of packing linear programs which model various online resource allocation problems including online variants of routing, ad allocations, generalized assignment, and combinatorial auctions. As our main theoretical result, we prove that a simple primal-dual training-based algorithm achieves a $(1 - o(1))$-approximation guarantee in the random order stochastic model. This is a significant improvement over logarithmic or constant-factor approximations for the adversarial variants of the same problems (e.g. factor $1 - \frac{1}{e}$ for online ad allocation, and $\log(m)$ for online routing). We then focus on the online display ad allocation problem and study the efficiency and fairness of various training-based and online allocation algorithms on data sets collected from real-life display ad allocation system. Our experimental evaluation confirms the effectiveness of training-based primal-dual algorithms on real data sets, and also indicate an intrinsic trade-off between fairness and efficiency.

Spectral redemption in clustering sparse networks

Florent Krzakala, Cristopher Moore, Elchanan Mossel, Joe Neeman, Allan Sly, Lenka Zdeborová, and Pan Zhang

Robust Principal Component Analysis?

EMMANUEL J. CANDÈS and XIAODONG LI, Stanford University
YI MA, University of Illinois at Urbana-Champaign, Microsoft Research Asia
JOHN WRIGHT, Microsoft Research Asia

This article is about a curious phenomenon. Suppose we have a data matrix, which is the superposition of a low-rank component and a sparse component. Can we recover each component individually? We prove that under some suitable assumptions, it is possible to recover both the low-rank and the sparse components *exactly* by solving a very convenient convex program called *Principal Component Pursuit*; among all feasible decompositions, simply minimize a weighted combination of the nuclear norm and of the ℓ_1 norm. This suggests the possibility of a principled approach to robust principal component analysis since our methodology and results assert that one can recover the principal components of a data matrix even though a positive fraction of its entries are arbitrarily corrupted. This extends to the situation where a fraction of the entries are missing as well. We discuss an algorithm for solving this optimization problem, and present applications in the area of video surveillance, where our methodology allows for the detection of objects in a cluttered background, and in the area of face recognition, where it offers a principled way of removing shadows and specularities in images of faces.

11

Example: Does my data look i.i.d.?

i.i.d. = Independent and identically distributed

Many examples of papers that **design theoretical algorithms assuming i.i.d. data**

→

Test their algorithms on **a set of real-world data**

→

Will the algorithm work **on my data**?

I need to make sure that my data is suitable!

Hope: My real-world instance is i.i.d.

In actuality: Does my data have the predictions used by the algorithms & possessed by i.i.d. instances?

Example: Does my data look i.i.d.?

A basic prediction of the model: every relative ordering of k selected entries occurs with probability $\frac{1}{k!}$



$x_3 < x_2 < x_4 < x_1$

Example: Does my data look i.i.d.?

A basic prediction of the model: every **relative ordering** of k selected entries occurs with probability $\frac{1}{k!}$

We call this a "permutation pattern"



$x_3 < x_2 < x_4 < x_1$

Does this prediction hold on the sequence we have?

Let $C_k(x)$ be the count of a particular length-k permutation pattern in sequence x of length n.

$$C_k(x) \approx \binom{n}{k} \frac{1}{k!}?$$



$x_3 < x_2 < x_4 < x_1$

Does this prediction hold on the sequence we have?

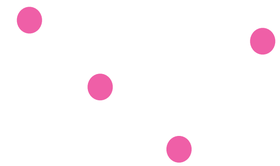
Let $C_k(x)$ be the count of a particular length-k permutation pattern in sequence x of length n.

$$C_k(x) \approx \binom{n}{k} \frac{1}{k!}?$$

This is a concrete certificate of one aspect of “i.i.d.-like” behavior.

Natural properties:

1) distribution-free for continuous data
2) connections to quasirandomness



$x_3 < x_2 < x_4 < x_1$

Our goals

Interested in: Is our data i.i.d.?

Benchmark: The prediction of

$$C_k(x) \approx \binom{n}{k} \frac{1}{k!}?$$

Our goals

Interested in: Is our data i.i.d.?

Benchmark: The prediction of

$$C_k(x) \approx \binom{n}{k} \frac{1}{k!}?$$

Efficiently?

Naively $\exp(\Omega(k))$ queries.

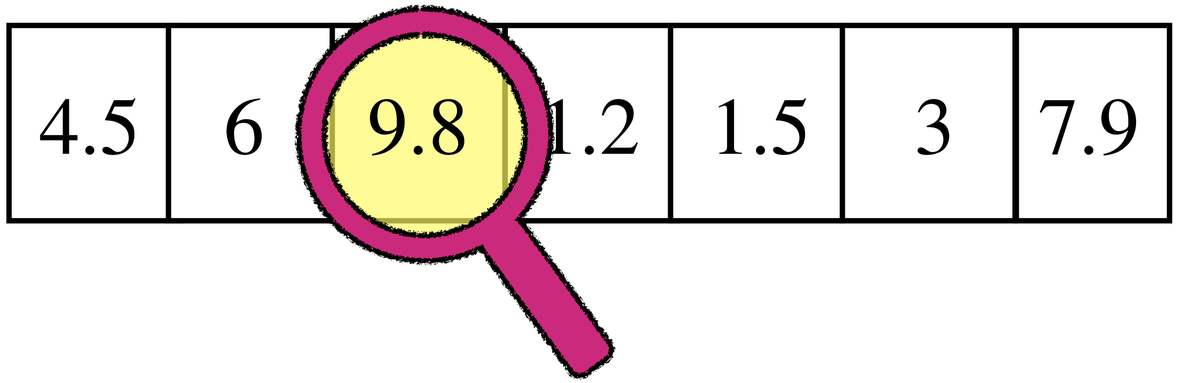
Goal: poly(k) queries and time in some reasonable framework.

Our access to the sequence

Sublinear access

Queries:

What is the value at index i?



Goal: as few queries as possible,
and small runtime

Naive certification of pattern count

Standard worst-case goal:

Accept "SAFE" length-k permutation pattern count $\approx \binom{n}{k} \frac{1}{k!}$

Reject "UNSAFE" length-k permutation pattern count $\not\approx \binom{n}{k} \frac{1}{k!}$

A property-testing style guarantee:
separate every good-count string from
every bad-count string.

Worst-case lower bound

Is count of length-k permutation pattern around $\binom{n}{k}\frac{1}{k!}$?

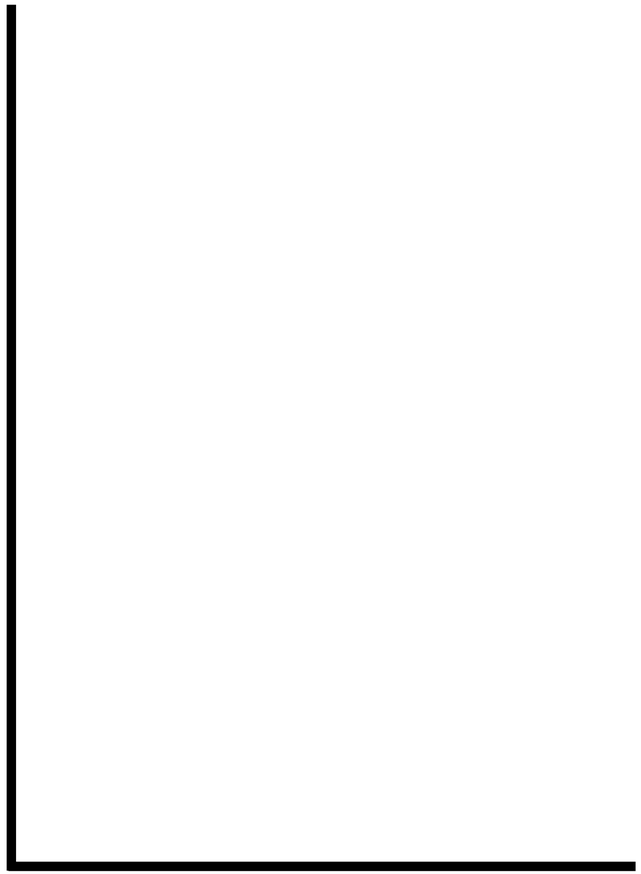
Needs $\exp(\Omega(k))$ queries

Worst-case lower bound

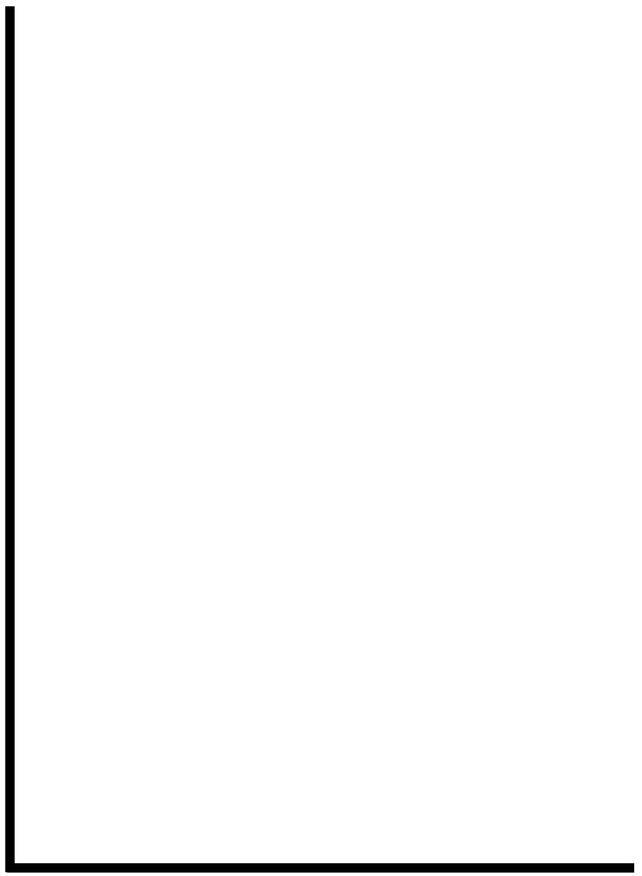
Is count of length-k permutation pattern around $\binom{n}{k}\frac{1}{k!}$?

Needs $\exp(\Omega(k))$ queries

$k = 5$



No copies of length-5 increasing pattern



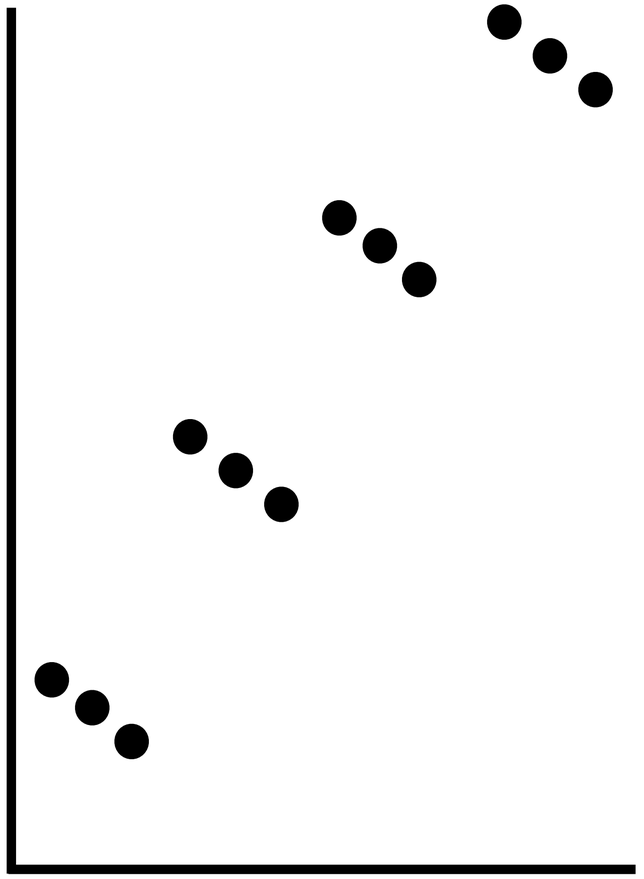
Many copies of length-5 increasing pattern

Worst-case lower bound

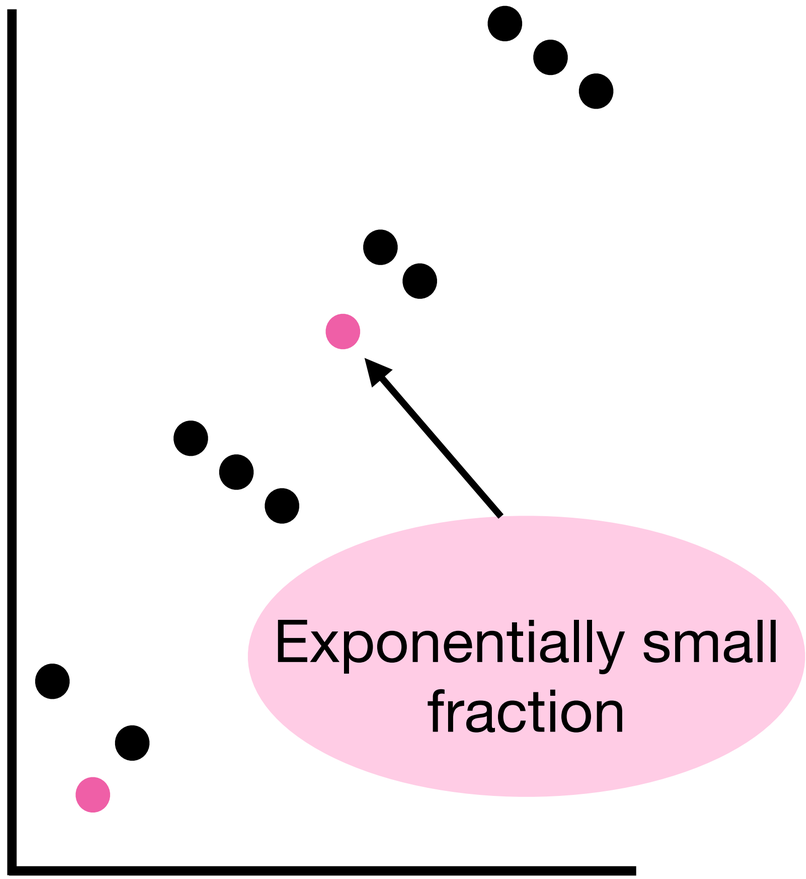
Is count of length-k permutation pattern around $\binom{n}{k}\frac{1}{k!}$?

Needs $\exp(\Omega(k))$ queries

$k = 5$



No copies of length-5 increasing pattern



Many copies of length-5 increasing pattern

Worst-case lower bound

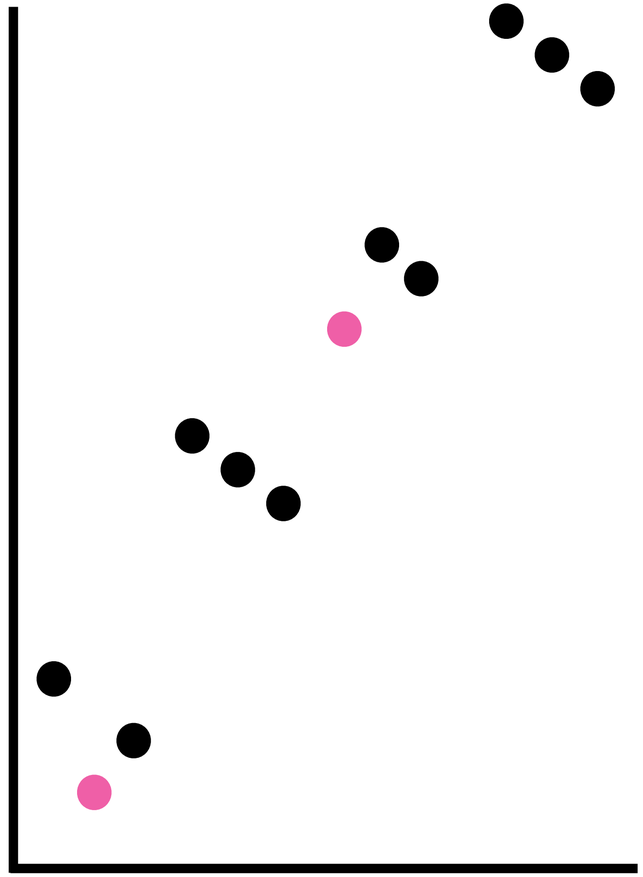
The argument:

Drop τ-fraction of elements to bottom of block

Creates $\approx \tau \left(\frac{n}{k}\right)^k$ increasing k-patterns

Need $\tau \left(\frac{n}{k}\right)^k \gtrapprox \binom{n}{k} \frac{1}{k!} \Rightarrow \tau \approx \frac{1}{k}\left(\frac{e^2}{k}\right)^k$

Distinguishing $\tau = 0$ from $\tau \approx \frac{1}{k}\left(\frac{e^2}{k}\right)^k$ needs $\Omega(k)^k$ queries



Naive certification of pattern count

Standard worst-case goal:

Accept "SAFE" length-k permutation pattern count $\approx \binom{n}{k} \frac{1}{k!}$

Reject "UNSAFE" length-k permutation pattern count $\not\approx \binom{n}{k} \frac{1}{k!}$

Naive certification of pattern count

Standard worst-case goal:

Accept "SAFE" length-k permutation pattern count $\approx \binom{n}{k} \frac{1}{k!}$

Reject "UNSAFE" length-k permutation pattern count $\not\approx \binom{n}{k} \frac{1}{k!}$

An opening: Why should we insist on accepting every adversarially constructed input with the correct count?

Naive certification of pattern count

Standard worst-case goal:

Accept "SAFE" length-k permutation pattern count $\approx \binom{n}{k} \frac{1}{k!}$

Reject "UNSAFE" length-k permutation pattern count $\not\approx \binom{n}{k} \frac{1}{k!}$

An opening: Why should we insist on accepting every adversarially constructed input with the correct count?

Need: inputs generated from the model

Naive certification of pattern count

Standard worst-case goal:

Accept "SAFE" ~~length-k permutation pattern count $\approx \binom{n}{k} \frac{1}{k!}$~~

Reject "UNSAFE" length-k permutation pattern count $\not\approx \binom{n}{k} \frac{1}{k!}$

New goal

Specify distribution p and $k \in \mathbb{N}$

Accept "SAFE" Should accept most i.i.d. instances: $x \sim p^{\otimes n}$

Reject "UNSAFE" length-k permutation pattern count $\not\approx \binom{n}{k} \frac{1}{k!}$

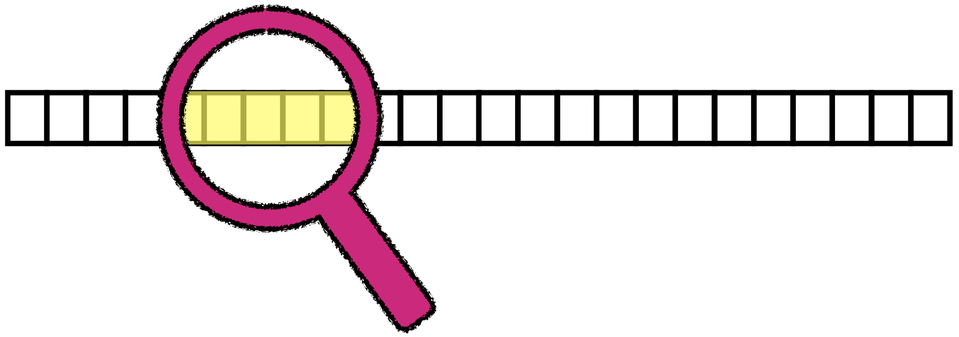
Acceptance is weakened
Rejection stays worst-case

New goal

Specify distribution p and $k \in \mathbb{N}$

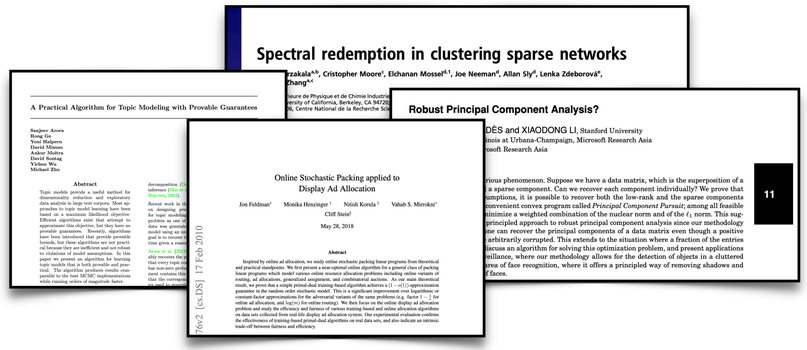
Accept "SAFE" Should accept most i.i.d. instances: $x \sim p^{\otimes n}$

Reject "UNSAFE" length-k permutation pattern count $\not\approx \binom{n}{k} \frac{1}{k!}$



Accept

Reject



UNSAFE

New goal

Specify distribution p and $k \in \mathbb{N}$

Accept "SAFE" Should accept most i.i.d. instances: $x \sim p^{\otimes n}$

Reject "UNSAFE" length-k permutation pattern count $\not\approx \binom{n}{k} \frac{1}{k!}$

Goal: construct algorithm with this guarantee.

Hope: more efficient algorithm?

Our results

Goal: construct algorithm with this guarantee.

Hope: more efficient algorithm?

Yes! For **permutation-pattern counts**,

the worst-case check requires $\exp(\Omega(k))$ queries,

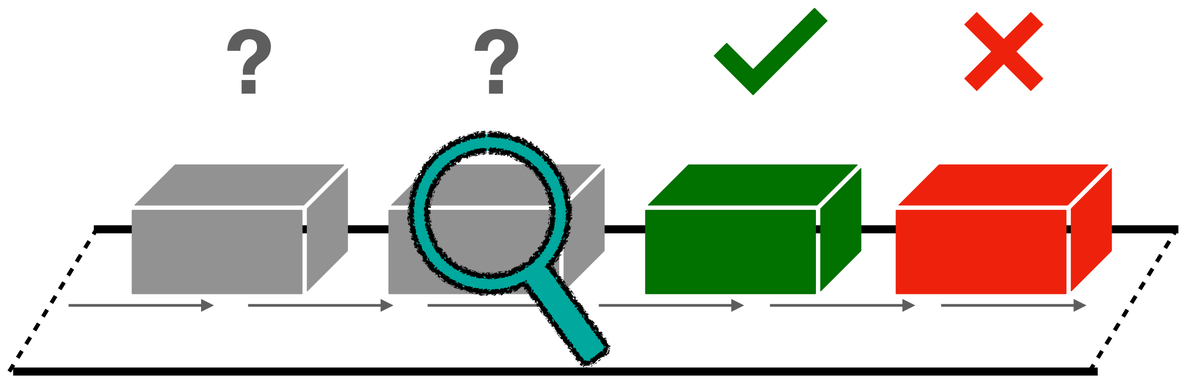
while new goal needs only poly(k) queries and time.

Agenda

- Quality control:
  - Permutation counts
  - Broader discussion
- Permutation pattern counts
  - Query-efficient algorithm
  - Runtime-efficient algorithm
- Graphs
- Shared themes and open problems

Broader Framework

Quality Control:

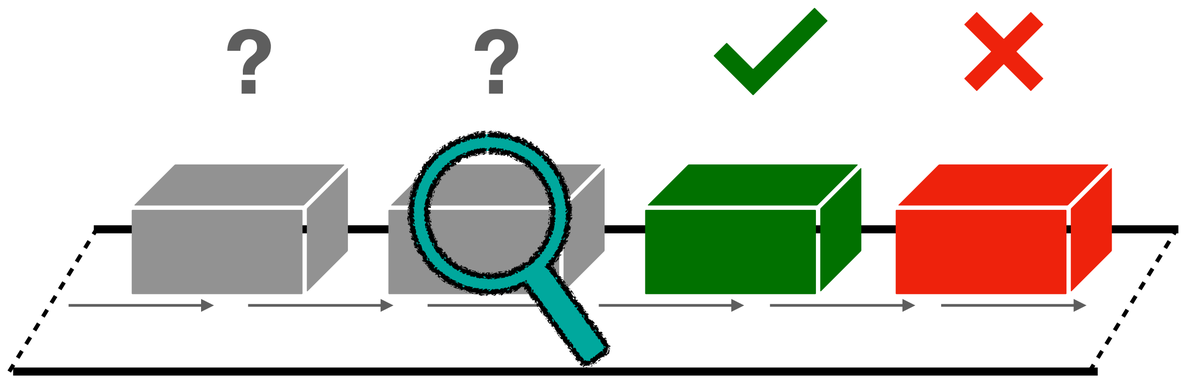


Broader Framework

Quality Control:

HIGH QUALITY (safe) instance ⇒ ACCEPT (usually)

LOW QUALITY (unsafe) instance ⇒ must REJECT



Broader Framework

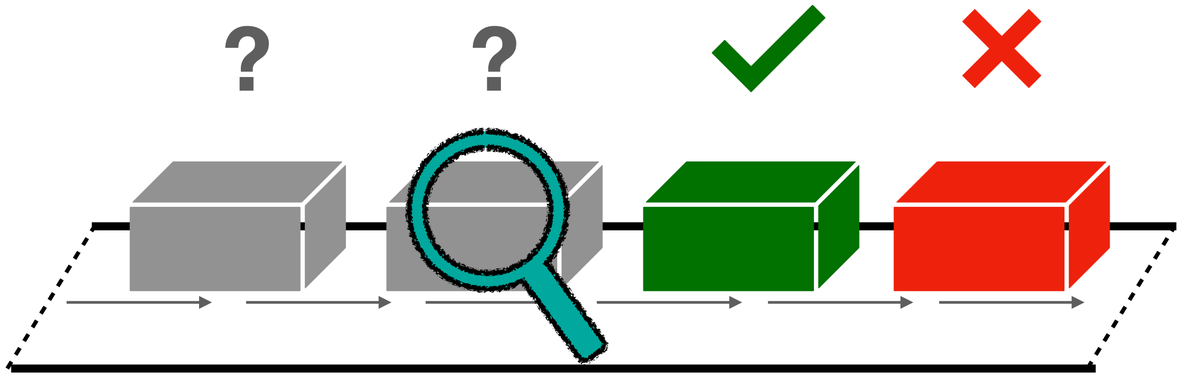
Quality Control:

HIGH QUALITY (safe) instance ⇒ ACCEPT (usually)

LOW QUALITY (unsafe) instance ⇒ must REJECT

Suffices if MOST good instances accepted.

Goal: efficiency.



Quality control

Let Q be a function: on input x, $Q(x)$ represents *quality of* x

$Q \approx 1$ means “high quality product”

“Good distribution” D has $Q(x) \in 1 \pm \varepsilon$ whp for $x \sim D$

Quality control

Let Q be a function: on input x, $Q(x)$ represents *quality of* x

$Q \approx 1$ means "high quality product"

e.g. $C(x)$ = count of a pattern in x

$$Q(x) = C(x)/\mathbb{E}_{y\sim p^{\otimes n}}[C(y)]$$

"Good distribution" D has $Q(x) \in 1 \pm \varepsilon$ whp for $x \sim D$

i.i.d. string: $p^{\otimes n}$

Quality control

Let Q be a function: on input x, $Q(x)$ represents *quality of* x

$Q \approx 1$ means "high quality product"

"Good distribution" D has $Q(x) \in 1 \pm \varepsilon$ whp for $x \sim D$

Definition [M., Rubinfeld, Sudan]**:** A is a (D, Q)-quality control algorithm if

Completeness: $\mathbb{P}_{x \sim D}\left[A(x) \text{ accepts}\right] = 1 - o(1)$

Soundness: $\forall x$ with $Q(x) \notin 1 \pm \varepsilon$, $A(x)$ rejects with probability $\geq 2/3$

Quality control

Let Q be a function: on input x, $Q(x)$ represents *quality of* x

$Q \approx 1$ means "high quality product"

"Good distribution" D has $Q(x) \in 1 \pm \varepsilon$ whp for $x \sim D$

Definition [M., Rubinfeld, Sudan]**:** A is a (D, Q)-quality control algorithm if

Completeness: $\mathbb{P}_{x \sim D}\left[A(x) \text{ accepts}\right] = 1 - o(1)$

"accept (lots of) high-quality instances"

Soundness: $\forall x$ with $Q(x) \notin 1 \pm \varepsilon$, $A(x)$ rejects with probability $\geq 2/3$

"reject low-quality instances"

Quality control

Let Q be a function: on input x, $Q(x)$ represents *quality of* x

$Q \approx 1$ means "high quality product"

"Good distribution" D has $Q(x) \in 1 \pm \varepsilon$ whp for $x \sim D$

Definition [M., Rubinfeld, Sudan]**:** A is a (D, Q)-quality control algorithm if

Completeness: $\mathbb{P}_{x \sim D}\left[A(x) \text{ accepts}\right] = 1 - o(1)$

Soundness: $\forall x$ with $Q(x) \notin 1 \pm \varepsilon$, $A(x)$ rejects with probability $\geq 2/3$

Asymmetric definition and one instance

Comparison to other models

Testable learning [Rubinfeld-Vasilyan ’23]

- Many samples from a distribution
- Accept if learning algorithm safe on distribution, reject otherwise

Testable algorithms [Eden-Rubinfeld-Vasilyan ’26]

- On all graphs with some property, output correct answer
- On all graphs without the property, output correct answer or “reject”

Comparison to other models

Testable learning **[Rubinfeld-Vasilyan ’23]**

- Many samples from a distribution
- Accept **if** learning algorithm **safe on distribution**, reject otherwise

Perfect completeness

Testable algorithms **[Eden-Rubinfeld-Vasilyan ’26]**

- On **all graphs with some property**, output correct answer
- On all graphs without the property, output correct answer or “reject”

Quality control weakens this to: “accept MOST”

Comparison to other models

Testable learning **[Rubinfeld-Vasilyan ’23]**

- Many samples from a distribution
- Accept **if** learning algorithm **safe on distribution**, reject otherwise

Perfect completeness

Testable algorithms **[Eden-Rubinfeld-Vasilyan ’26]**

- On **all graphs with some property**, output correct answer
- On all graphs without the property, output correct answer or “reject”

Quality control weakens this to: “accept MOST”

Plus, connections to:
property testing, distribution testing

Examples in the literature

Many quality control examples studied implicitly in previous work:

Subroutine of average-case algorithms (Dyer–Frieze ’89)

Natural proofs (Razborov–Rudich ’97)

Refutation of random CSPs (Feige ’02)

Strong spectral refutation of k-SAT unsatisfiability
(Allen–O’Donnell–Witmer ’15, Raghavendra–Rao–Schramm ’17)

Algorithmic robust statistics: certifiable subgaussianity
(Kothari–Steurer ’17, Hopkins–Li ’18, Diakonikolas–Hopkins–Pensia–Tiegel ’25)

Untrusted randomness (Girish–Raz–Zhan ’23)

Examples in the literature

Many quality control examples studied implicitly in previous work:

Subroutine of average-

Natural proofs (Razborov

Refutation of random C

Strong spectral refutati
(Allen–O'Donnell–Witmer '15

Algorithmic robust statistics: certifiable subgaussianity
(Kothari–Steurer '17, Hopkins–Li '18, Diakonikolas–Hopkins–Pensia–Tiegel '25)

Untrusted randomness (Girish–Raz–Zhan '23)

We formalize the *general framework*

(and focus on sublinear access models)

Agenda

- Quality control:
  - Permutation counts
  - Broader discussion
- Permutation pattern counts
  - Query-efficient algorithm
  - Runtime-efficient algorithm
- Graphs
- Shared themes and open problems

Our settings

1. Testing independence / counting permutation patterns

Accept i.i.d. sequence
Reject if count of permutation pattern is wrong

2. Testing $\mathscr{G}_{n,p}$ / counting cliques (motifs)

Accept $\mathscr{G}_{n,p}$ = random graph
Reject if count of cliques is wrong

Our settings

1. Testing independence / counting permutation patterns

Accept i.i.d. sequence
Reject if count of permutation pattern is wrong

2. Testing $\mathscr{G}_{n,p}$ / counting cliques (motifs)

Accept $\mathscr{G}_{n,p}$ = random graph
Reject if count of cliques is wrong

Our setting (+ notation)

$(p^{\otimes n}, Q_k)$-quality control

Reminder:

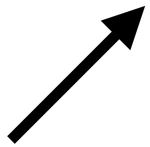
Definition [M., Rubinfeld, Sudan]**:** A is a (D, Q)-quality control algorithm if

Completeness: $\mathbb{P}_{x\sim D}\left[A(x) \text{ accepts}\right] = 1 - o(1)$

Soundness: $\forall x$ with $Q(x) \notin 1 \pm \varepsilon$, $A(x)$ rejects with probability $\geq 2/3$

Our setting (+ notation)

$(p^{\otimes n}, Q_k)$-quality control



i.i.d real-valued sequence

Reminder:

Definition [M., Rubinfeld, Sudan]**:** A is a (D, Q)-quality control algorithm if

Completeness: $\mathbb{P}_{x\sim D}\left[A(x) \text{ accepts}\right] = 1 - o(1)$

Soundness: $\forall x$ with $Q(x) \notin 1 \pm \varepsilon$, $A(x)$ rejects with probability $\geq 2/3$

Our setting (+ notation)

$(p^{\otimes n}, Q_k)$-quality control

i.i.d real-valued sequence

$C_k(x) =$ # copies of a length-k permutation pattern

$$Q_k(x) = \frac{C_k(x)}{\mathbb{E}_{y \sim p^{\otimes n}}\left[C_k(y)\right]} = \frac{C_k(x)}{\binom{n}{k}\frac{1}{k!}}$$

Reminder:

Definition [M., Rubinfeld, Sudan]: A is a (D, Q)-quality control algorithm if

Completeness: $\mathbb{P}_{x \sim D}\left[A(x) \text{ accepts}\right] = 1 - o(1)$

Soundness: $\forall x$ with $Q(x) \notin 1 \pm \varepsilon$, $A(x)$ rejects with probability $\geq 2/3$

$Q_k(x) \approx 1$ for $x \sim p^{\otimes n}$ whp

Our setting (+ notation)

$(p^{\otimes n}, Q_k)$-quality control

i.i.d real-valued sequence

$C_k(x) =$ # copies of a length-k permutation pattern

$$Q_k(x) = \frac{C_k(x)}{\mathbb{E}_{y \sim p^{\otimes n}}\left[C_k(y)\right]} = \frac{C_k(x)}{\binom{n}{k}\frac{1}{k!}}$$

Does the count of a pattern match the i.i.d. case?

$Q_k(x) \approx 1$ for $x \sim p^{\otimes n}$ whp

Our setting (+ notation)

$(p^{\otimes n}, Q_k)$-quality control

i.i.d real-valued sequence

Does the count of a pattern match the i.i.d. case?

Why study permutation pattern counts?

Natural test of i.i.d. property,

Connections to quasirandomness,

Q_k expensive to compute/verify

Comparison to worst-case setting

Is $Q_k(x) \in 1 \pm \varepsilon$?

Theorem [M., Rubinfeld, Sudan]
$\exp(\Omega(k))$ queries

$(p^{\otimes n}, Q_k)$-quality control

Theorem [M., Rubinfeld, Sudan]
$\mathrm{poly}(k)$ queries and time

Comparison to worst-case setting

Is $Q_k(x) \in 1 \pm \varepsilon$?

$(p^{\otimes n}, Q_k)$-quality control

Theorem [M., Rubinfeld, Sudan]
$\exp(\Omega(k))$ queries

Theorem [M., Rubinfeld, Sudan]
$\text{poly}(k)$ queries and time

for any continuous-valued distribution p
and any length-k permutation pattern…

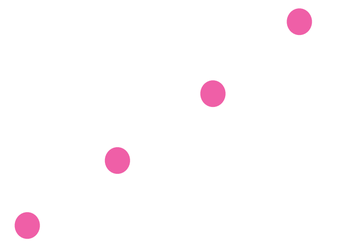
… with the same algorithm!

Agenda

- Quality control:
  - Permutation counts
  - Broader discussion
- Permutation pattern counts
  - Query-efficient algorithm
  - Runtime-efficient algorithm
- Graphs
- Shared themes and open problems

Let's now move to Quality control algorithms

For illustration, let's consider counting the **length-k increasing pattern**



$x_1 < x_2 < x_3 < x_4$

Quality Control Algorithm: Attempt 1

Sample $S \subseteq [n]$ with $|S| = \text{poly}(k)$ randomly

- Check count of length-k increasing pattern in S
- Accept iff count $\approx \binom{|S|}{k} \frac{1}{k!}$

Quality Control Algorithm: Attempt 1

Sample $S \subseteq [n]$ with $|S| = \text{poly}(k)$ randomly

- Check count of length-k increasing pattern in S
- Accept iff count $\approx \binom{|S|}{k} \frac{1}{k!}$

Observation: $\mathbb{E}_S[\text{Count}_k(x|_S)] \propto \text{Count}_k(x)$

But the count is not concentrated for worst-case strings!

How can quality control help?

Allowed to “filter”, as long as accept i.i.d. x and reject low-quality x

How can quality control help?

Allowed to “filter”, as long as accept i.i.d. x and reject low-quality x

Core principle of quality control: Composability

Suppose $x \sim D$ have $P(x) \approx 1$ and $Q(x) \approx 1$ whp.

We can recursively perform (D, P)-quality control
as a filter for (D, Q)-quality control

Composability

What property do we track/control?

The counts of all length-ℓ increasing patterns, $\ell \leq k$

Query efficient algorithm

Pick random set $S \subseteq [n], |S| = s = \text{poly}(k)$

For $\ell = 2,3,\ldots,k$:

Count $C_\ell(S) = \#$ length-ℓ increasing patterns in S

Reject if $C_\ell(S) \not\approx \binom{s}{\ell} \frac{1}{\ell!}$

Else Accept.

Query efficient algorithm

Pick random set $S \subseteq [n], |S| = s = \text{poly}(k)$

For $\ell = 2,3,\ldots,k$:

Count $C_\ell(S) = \#$ length-ℓ increasing patterns in S

Reject if $C_\ell(S) \not\approx \binom{s}{\ell} \frac{1}{\ell!}$

Else Accept.

Composability in action:
Filter out strings with wrong length-ℓ pattern count

Query efficient algorithm

Pick random set $S \subseteq [n]$, $|S| = s = \text{poly}(k)$

How was poly(k) possible?

For $\ell = 2,3,\ldots,k$:

Count $C_\ell(S) =$ # length-ℓ increasing patterns in S

Reject if $C_\ell(S) \approx \binom{s}{\ell} \frac{1}{\ell!}$

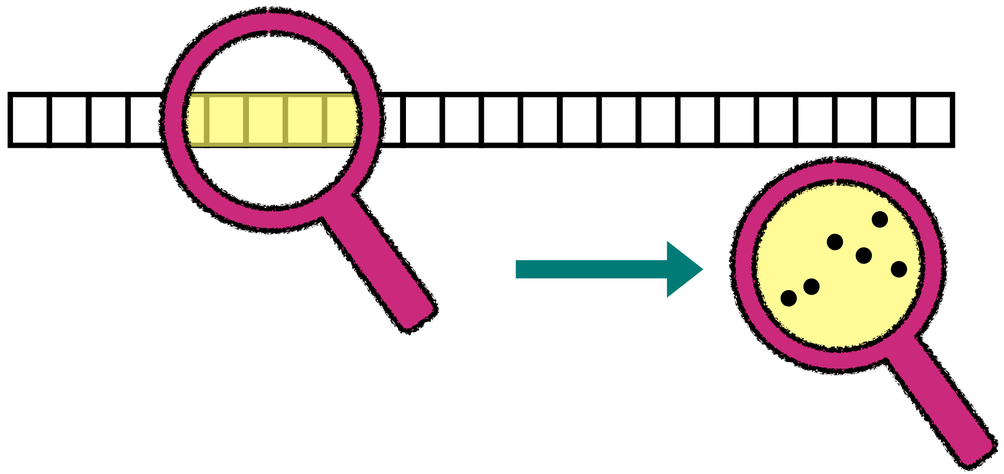
Else Accept.

Composability let us build up a property:

Property $P(\ell)$: "Most" subsequences of x of size s have

$$\approx \binom{s}{\ell} \frac{1}{\ell!}$$ occurrences of length-ℓ increasing pattern

$\Rightarrow$



Can efficiently count length-$(\ell + 1)$ increasing pattern

and see if Property $P(\ell + 1)$ holds

Composability let us build up a property:

Property $P(\ell)$**:** For a $1 - e^{-\Omega(s)}$ fraction of $S : |S| = s$,

$\approx \binom{s}{\ell} \frac{1}{\ell!}$ occurrences of length-ℓ increasing pattern

Composability let us build up a property:

Property $P(\ell)$**:** For a $1 - e^{-\Omega(s)}$ fraction of $S : |S| = s$,

$\approx \binom{s}{\ell} \frac{1}{\ell!}$ occurrences of length-ℓ increasing pattern

We call this property "exponentially robust quasirandomness"

Property that holds for most subsets, so can check on small random sample.

Why this property can be inductively checked: proven via method of bounded differences that hold with high probability.

Why should $C_{\ell+1}(S)$ concentrate?

Property $P(\ell)$: For a $1 - e^{-\Omega(s)}$ fraction of $S : |S| = s$,

$\approx \binom{s}{\ell} \frac{1}{\ell!}$ occurrences of length-ℓ increasing pattern

We will use **bounded differences:**

Changing one element of S changes $C_{\ell+1}(S)$ by at most $\approx C_\ell(S)$

So on a **typical** S:

$$|\Delta C_{\ell+1}| \lessapprox \binom{s}{\ell} \frac{1}{\ell!}$$

Bounded differences, except on rare samples

Ordinary bounded differences (McDiarmid):

If changing one sampled point always chances $f(S)$ by at most c,

then $f(S)$ concentrates.

Our problem: We only know this for *typical* S.

High-probability bounded differences (Combes):

It is enough that the bounded-difference condition holds on a set of probability $1 - \delta$.

The concentration bound only pays an additional additive $O(\delta)$.

Completing the induction

$P(\ell) \Rightarrow$ for all but $e^{-\Omega(s/\mathrm{poly}(\ell))}$ of S, $|\Delta C_{\ell+1}| \lessapprox \binom{s}{\ell} \frac{1}{\ell!}$.

$\Rightarrow C_{\ell+1}(S)$ is concentrated around $\mathbb{E}[C_{\ell+1}(S)]$.

Because $\mathbb{E}[C_{\ell+1}(S)] \propto C_{\ell+1}(x)$,

a wrong global $(\ell+1)$-count forces a random S to reveal it.

So $P(\ell) \Rightarrow$ can check if $P(\ell+1)$ holds.

Back to: query efficient algorithm

Pick random set $S \subseteq [n]$,
$|S| = s = \text{poly}(k)$

For $\ell = 2,3,\ldots,k$:

Count $C_\ell(S) = \#$ length-ℓ increasing patterns in S

Reject if $C_\ell(S) \not\approx \binom{s}{\ell} \frac{1}{\ell!}$

Else Accept.

Back to: query efficient algorithm

Keep sequences when exponentially robustly **quasirandom wrt length-ℓ pattern count**.

Reject sequence **when not**.

Pick random set $S \subseteq [n]$, $|S| = s = \text{poly}(k)$

For $\ell = 2,3,\ldots,k$:

Count $C_\ell(S) = \#$ length-ℓ increasing patterns in S

Reject if $C_\ell(S) \approx \binom{s}{\ell} \frac{1}{\ell!}$

Else Accept.

Back to: query efficient algorithm

Keep sequences when exponentially robustly **quasirandom wrt length-ℓ pattern count**.

Reject sequence **when not**.

Therefore easier to count length-$(\ell+1)$ pattern

Pick random set $S \subseteq [n]$, $|S| = s = \text{poly}(k)$

For $\ell = 2,3,\ldots,k$:

Count $C_\ell(S) = \#$ length-ℓ increasing patterns in S

Reject if $C_\ell(S) \not\approx \binom{s}{\ell} \frac{1}{\ell!}$

Else Accept.

Remark: quasirandomness

Quasirandom properties: *global*, *strong*, implies length-k permutation pattern count matches i.i.d. case

Not efficiently verifiable.

Robust notion of quasirandomness: implies length-k pattern count matches i.i.d. case WHP

Efficiently verifiable.

Agenda

- Quality control:
  - Permutation counts
  - Broader discussion
- Permutation pattern counts
  - Query-efficient algorithm
  - Runtime-efficient algorithm
- Graphs
- Shared themes and open problems

So far…

$s =$ **poly**(k) queries

Circumvents $\exp(\Omega(k))$
worst-case lower bound

Algorithm:

Pick random set $S \subseteq [n]$,
$|S| = s =$ poly(k)

For $\ell = 2,3,\ldots,k$:

Count $C_\ell(S) =$ # length-ℓ increasing patterns in S

Reject if $C_\ell(S) \not\approx \binom{s}{\ell} \frac{1}{\ell!}$

Else Accept.

So far…

$s =$ **poly**(k) queries

Circumvents $\exp(\Omega(k))$ worst-case lower bound

$s^k = \exp(\Omega(k))$ runtime to count general patterns in the subsequence

Can we circumvent the $\exp(\Omega(k))$ worst-case lower bound?

Algorithm:

Pick random set $S \subseteq [n]$, $|S| = s =$ poly(k)

For $\ell = 2,3,\ldots,k$:

Count $C_\ell(S) =$ # length-ℓ increasing patterns in S

Reject if $C_\ell(S) \not\approx \binom{s}{\ell} \frac{1}{\ell!}$

Else Accept.

So far…

Need:

Most size-s subsequences of

$$x \sim p^{\otimes n}$$

$$\Rightarrow$$

Property P

$$\Rightarrow$$

All $C_\ell(S) \approx \binom{s}{\ell} \frac{1}{\ell!}$,

$$\ell = 2,3,\ldots,k$$

Algorithm:

Pick random set $S \subseteq [n]$,
$|S| = s = \text{poly}(k)$

For $\ell = 2,3,\ldots,k$:

Count $C_\ell(S) = \#$ length-ℓ increasing patterns in S

Reject if $C_\ell(S) \not\approx \binom{s}{\ell} \frac{1}{\ell!}$

Else Accept.

So far…

Need:

Most size-s subsequences of

$$x \sim p^{\otimes n}$$

$$\Rightarrow$$

Property P

$$\Rightarrow$$

All $C_\ell(S) \approx \binom{s}{\ell} \frac{1}{\ell!}$,

$$\ell = 2,3,\ldots,k$$

Will be:
Another notion of
quasirandomness

We filter out instances
not satisfying Property P
(Composability)

Efficient approximate counting

Theorem: [M., Rubinfeld, Sudan 2026]

Let x be a length-n sequence. There is some property we call jumbledness such that:

i.i.d sequences are jumbled, efficiently checkable, and

If x is jumbled, then, for all $\ell = 2,3,\ldots,k$, the count of every length-ℓ pattern in x is $\approx \binom{n}{\ell} \frac{1}{\ell!}$.

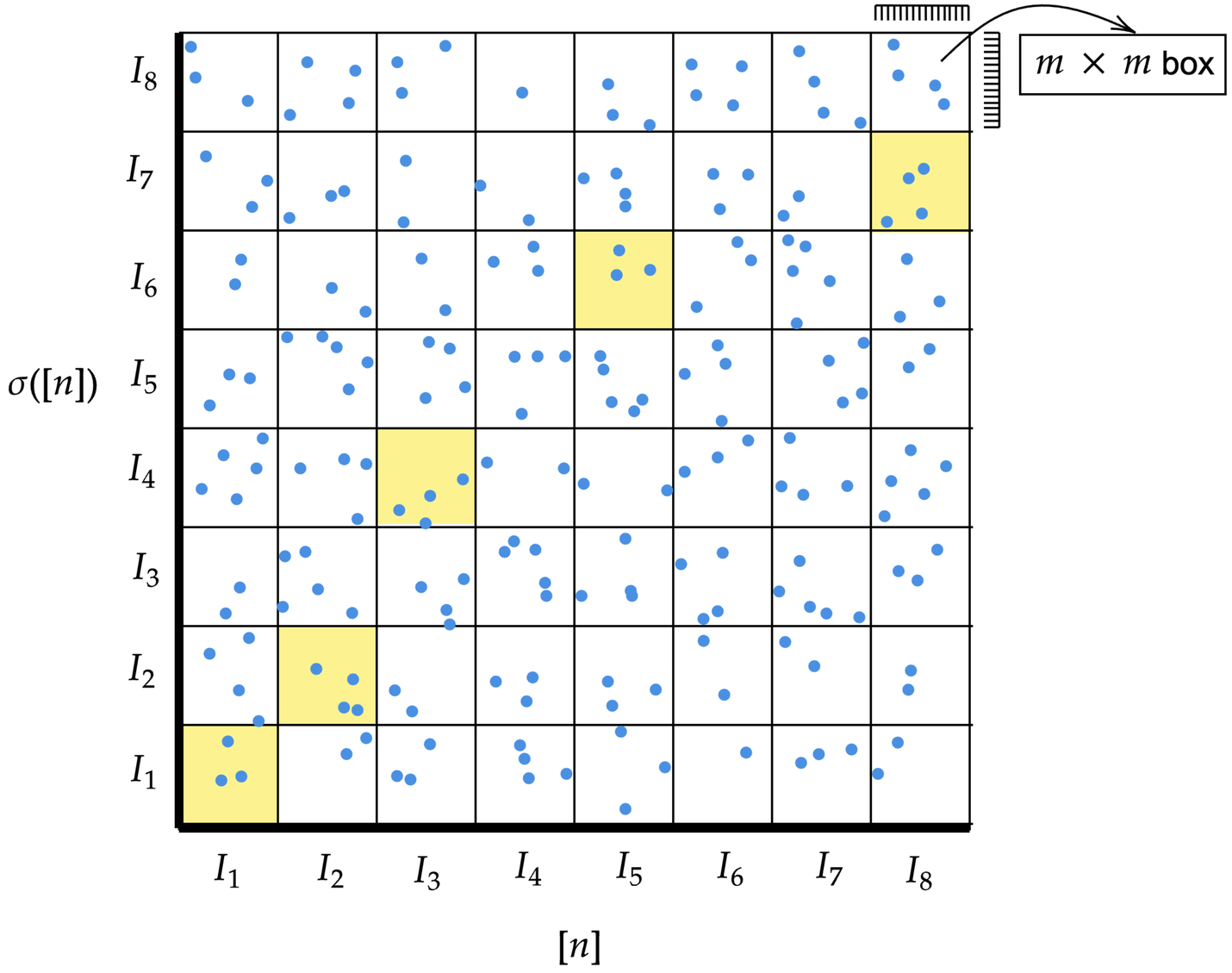
Jumbledness

Jumbled sequence:

1. View the sequence as a permutation (consider relative orderings)

2. Divide into $\frac{n}{m} \times \frac{n}{m}$ grid, $m = \text{poly}(k)$

3. Each block has $\approx \frac{m^2}{n}$ points



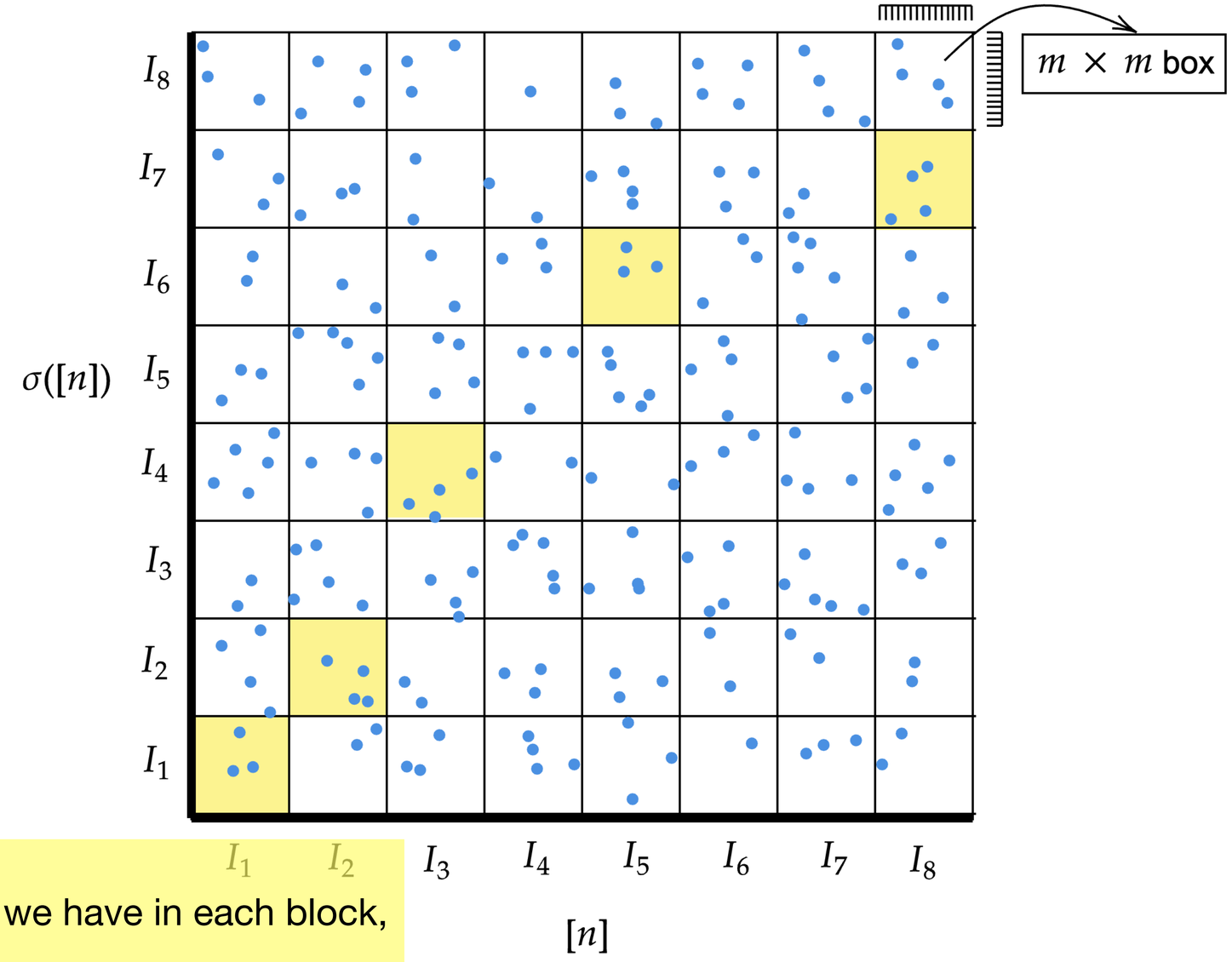
Jumbledness

Jumbled sequence*:

1. View the sequence as a permutation (consider relative orderings)

2. Divide into $\frac{n}{m} \times \frac{n}{m}$ grid,
   $m = \text{poly}(k)$

3. Each block has $\approx \frac{m^2}{n}$ points

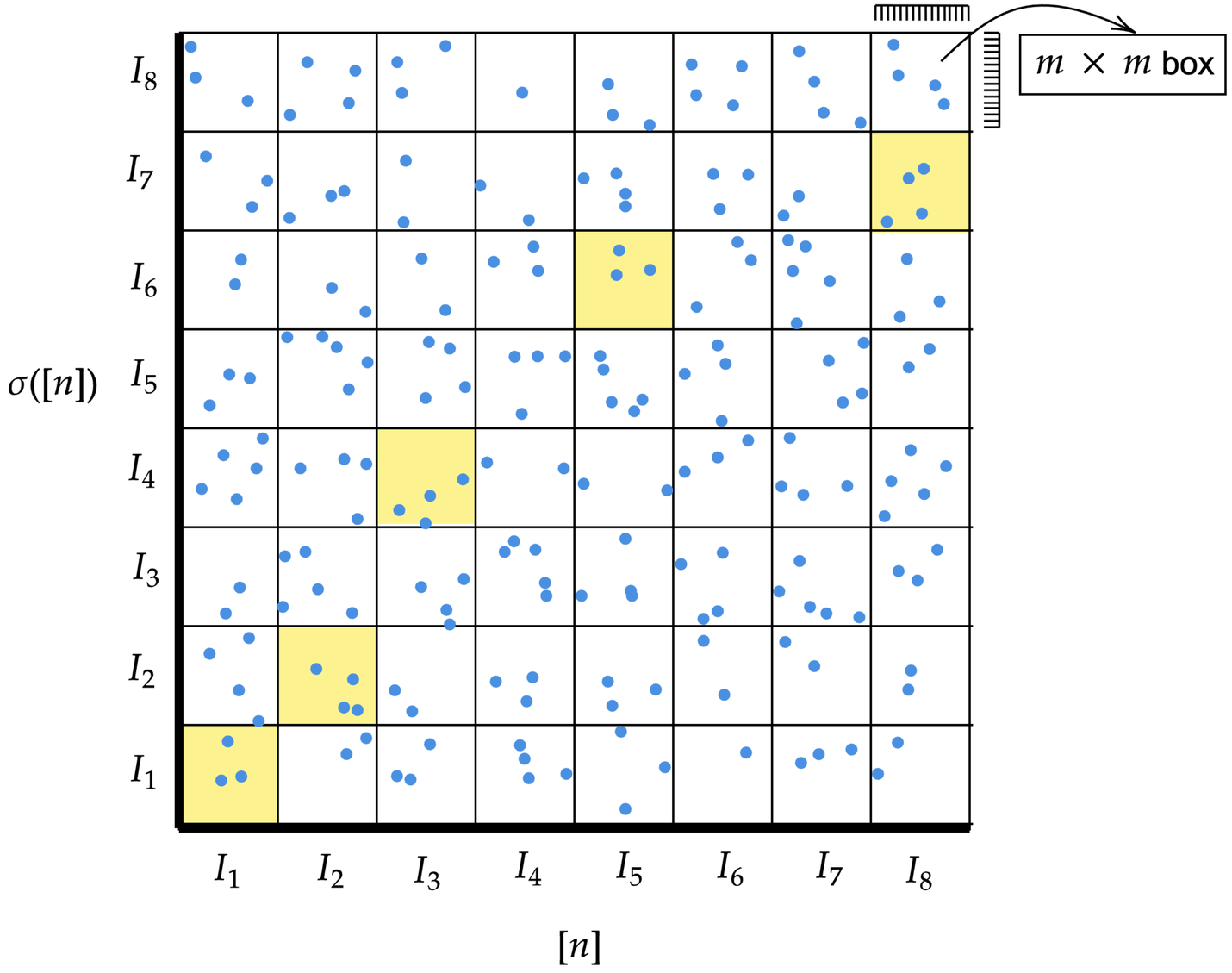


*** Based on a notion defined by Cooper (2004).**
The main difference is in how good of an approximation we have in each block, which implies more fine-grained counting.

Jumbledness

Jumbled sequence:

Implies that all length-ℓ permutation patterns have a count of $\approx \binom{n}{\ell} \frac{1}{\ell!}$.



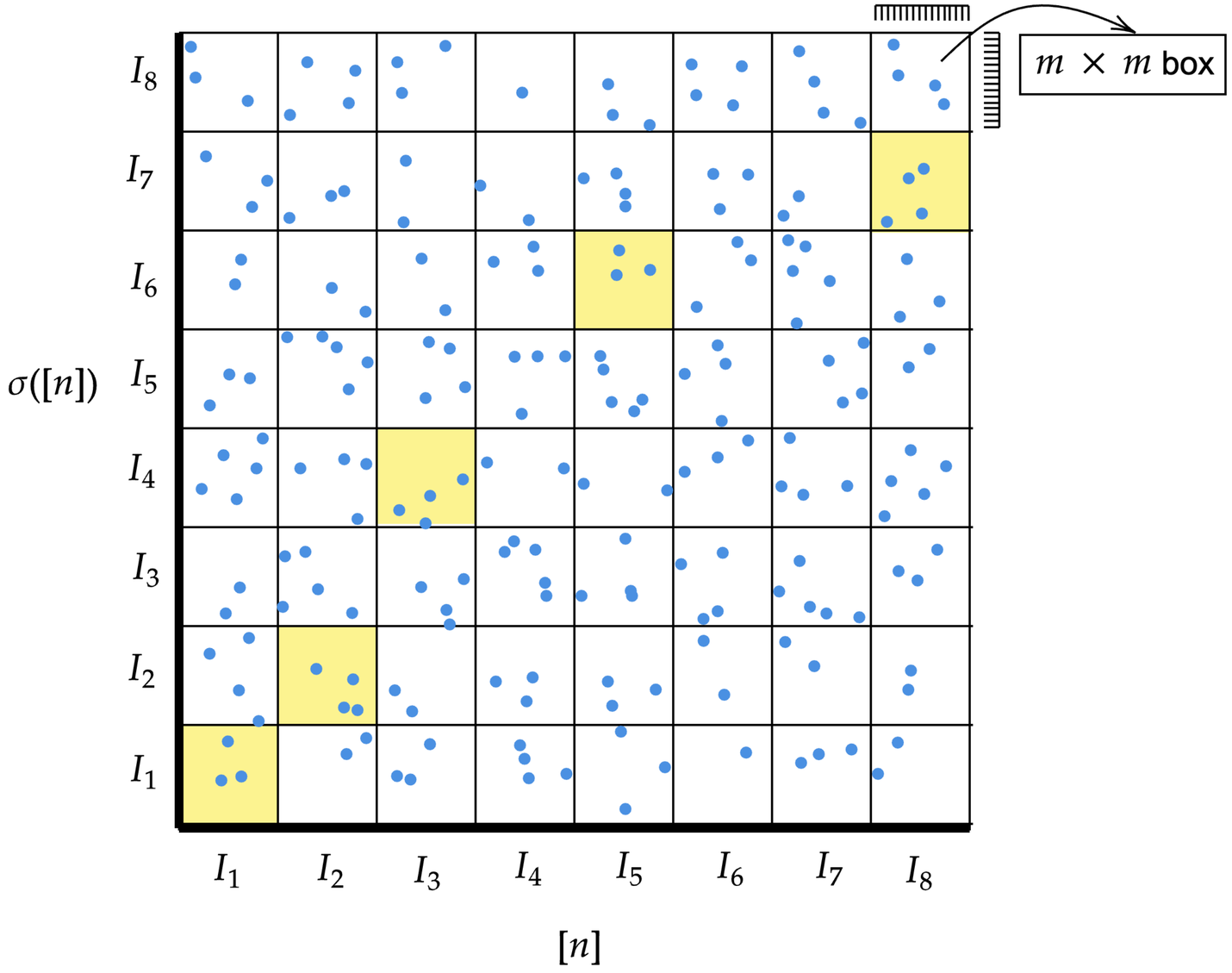
Why jumbledness certifies pattern counts

Counting patterns:

1. Group patterns into coarse cells

2. Jumbledness: know number of points in each cell

3. Sum over increasing cell sequences ⇒ approximately count increasing pattern



Why jumbledness certifies pattern counts

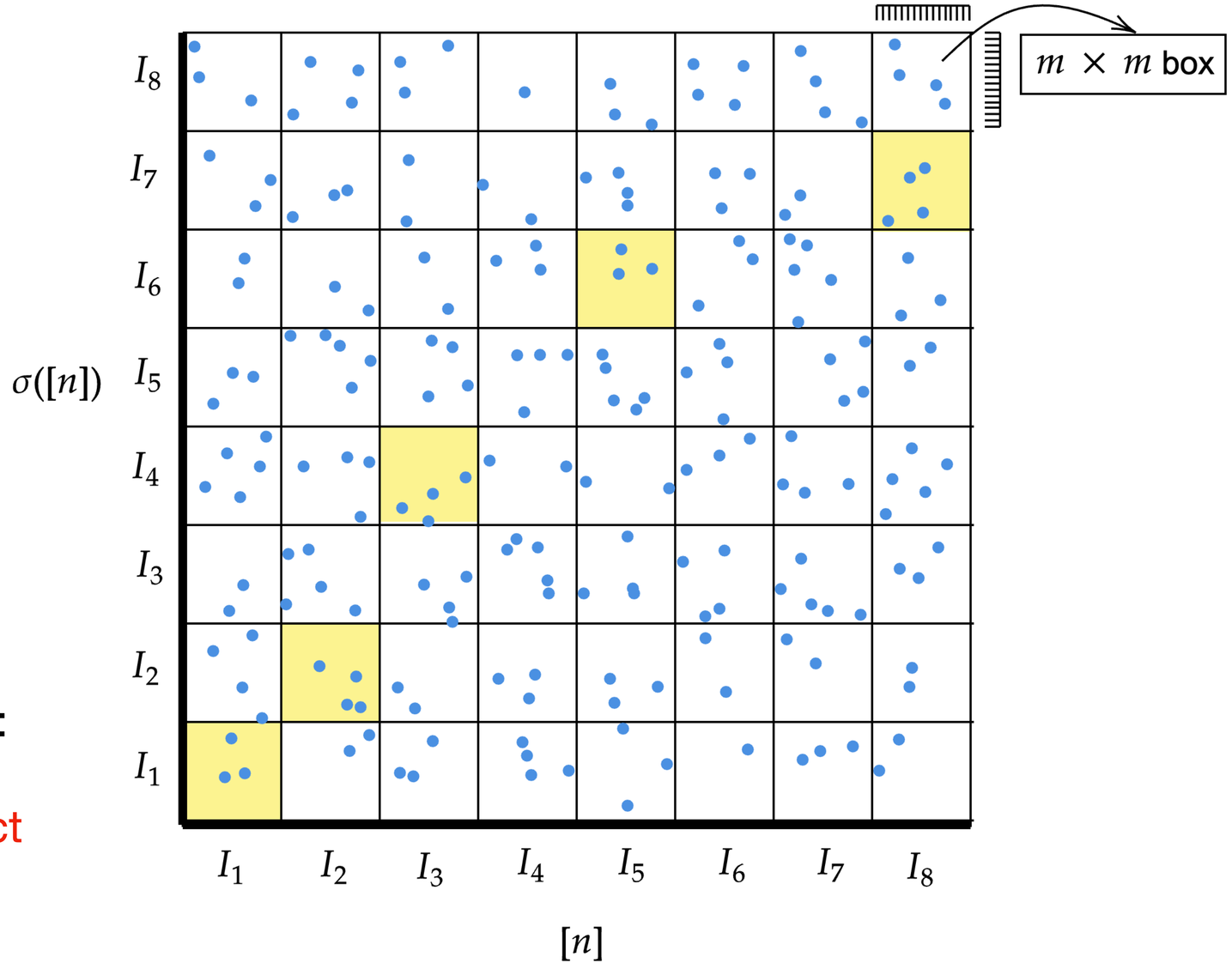
Counting patterns:

Summing within pattern-forming blocks:

$$\underbrace{\binom{n/m}{k}^2}_{\text{Choose blocks}} \underbrace{\left(\frac{m^2}{n}\right)^k}_{\text{Block density}} \approx \underbrace{\binom{n}{k}\frac{1}{k!}}_{\text{i.i.d. case count}}$$

If more than one element chosen in a block:

Too few choices of blocks to make an impact



Final (runtime efficient) algorithm

Pick random set $S \subseteq [n]$, $|S| = s = \text{poly}(k)$

Treat $x|_S$ as a permutation

Reject if any block in $\frac{s}{m} \times \frac{s}{m}$ grid does

not have $\approx \frac{m^2}{s}$ points, $m = \text{poly}(k)$

Else Accept.

Final (runtime efficient) algorithm

Pick random set $S \subseteq [n]$, $|S| = s = \text{poly}(k)$

Treat $x|_S$ as a permutation

Reject if any block in $\frac{s}{m} \times \frac{s}{m}$ grid does not have $\approx \frac{m^2}{s}$ points, $m = \text{poly}(k)$

Else Accept.

Runs quality control for jumbledness on S in $\text{poly}(k)$ time

Final (runtime efficient) algorithm

Pick random set $S \subseteq [n]$, $|S| = s = \text{poly}(k)$

Treat $x|_S$ as a permutation

Reject if any block in $\frac{s}{m} \times \frac{s}{m}$ grid does not have $\approx \frac{m^2}{s}$ points, $m = \text{poly}(k)$

Else Accept.

Runs quality control for jumbledness on S in $\text{poly}(k)$ time

Implies all length-ℓ pattern counts match i.i.d. case, $\ell \leq k$

What we've proven

Theorem:

$(p^{\otimes n}, Q_k)$-quality control in poly(k) queries and time.

Improvement over the $\exp(\Omega(k))$ bound for worst-case sequences.

We also prove: lower bounds, results for value-based patterns

Agenda

- Quality control:
  - Permutation counts
  - Broader discussion
- Permutation pattern counts
  - Query-efficient algorithm
  - Runtime-efficient algorithm
- Graphs
- Shared themes and open problems

Our settings

1. Testing independence / counting permutation patterns

Accept i.i.d. sequence
Reject if count of permutation pattern is wrong

2. Testing $\mathscr{G}_{n,p}$ / counting cliques (motifs)

Accept $\mathscr{G}_{n,p}$ = random graph
Reject if count of cliques is wrong

Our settings

1. Testing independence / counting permutation patterns

Accept i.i.d. sequence
Reject if count of permutation pattern is wrong

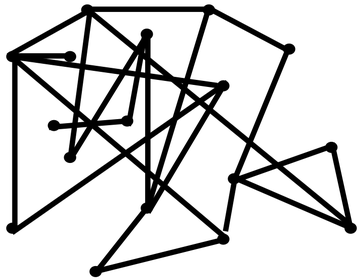
2. Testing $\mathcal{G}_{n,p}$ / counting cliques (motifs)

Accept $\mathcal{G}_{n,p}$ = random graph
Reject if count of cliques is wrong

We'll see a similar story (with some new elements for graphs!)

A common practice

Use a subgraph statistic as certificate of “random-looking”



→ Estimate count of k-cliques → Compare to expected count under $\mathcal{G}_{n,p}$ → Decision: SAFE/UNSAFE

Expected count: $\binom{n}{k} p^{\binom{k}{2}}$

$\mathcal{G}_{n,p}$ = graph with n vertices and each edge included independently with probability p

Examples: [Chung-Graham-Wilson ’88, Bickel-Chen-Levina ’11, Gleich-Owen ’12]

The quality control goal

Accept "SAFE" Should accept most $\mathcal{G}_{n,p}$ instances

Reject "UNSAFE" k-clique count $\not\approx \binom{n}{k} p^{\binom{k}{2}}$

Our setting (+ notation)

$(\mathcal{G}_{n,p}, Q_k)$-quality control

Erdős–Rényi random graphs:
(each edge present wp p)

$$C_k(G) = \#k\text{-cliques in } G$$

$$Q_k(G) = \frac{C_k(G)}{\mathbb{E}_{G' \sim \mathcal{G}_{n,p}}\left[C_k(G')\right]} = \frac{C_k(G)}{\binom{n}{k} p^{\binom{k}{2}}}$$

Reminder:

Definition [M., Rubinfeld, Sudan]: A is a (D, Q)-quality control algorithm if

Completeness: $\mathbb{P}_{x \sim D}\left[A(x) \text{ accepts}\right] = 1 - o(1)$

Soundness: $\forall x$ with $Q(x) \notin 1 \pm \varepsilon$, $A(x)$ rejects with probability $\geq 2/3$

$Q_k(\mathcal{G}_{n,p}) \approx 1$ whp for $p \geq n^{-1/(ck)}$, some c

Our setting (+ notation)

$(\mathcal{G}_{n,p}, Q_k)$-quality control

Erdős–Rényi random graphs:
(each edge present wp p)

Does the k-clique count match $\mathcal{G}_{n,p}$?

Why study clique/motif counts?

Fundamental task in network science

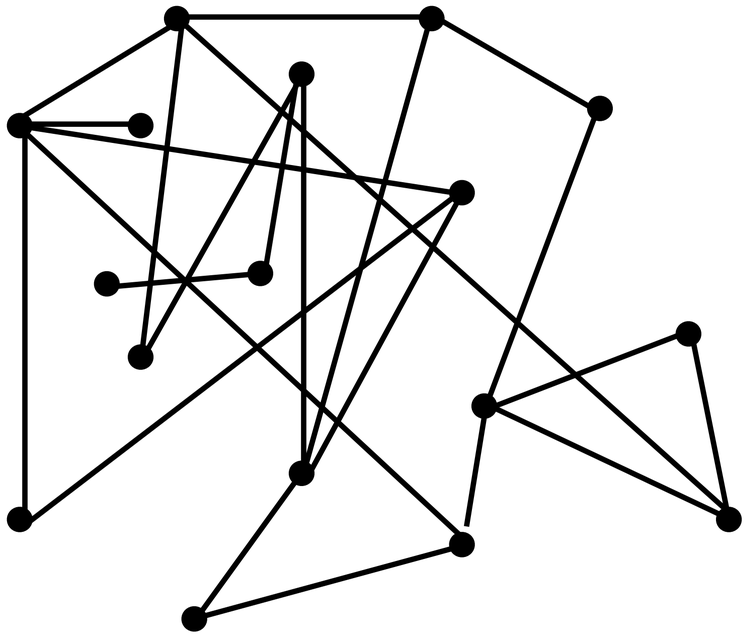
Connections to:
Quasirandomness,
Graph limits,
Moments of a graph*, …

Q_k *expensive to compute/verify*

* Bickel, Chen, Levina 2011. The method of moments and degree distributions for network models.

Our setting

Sublinear graph access



Queries:

Is (u, v) an edge?

Adjacency matrix query

What is the i-th neighbor of u?

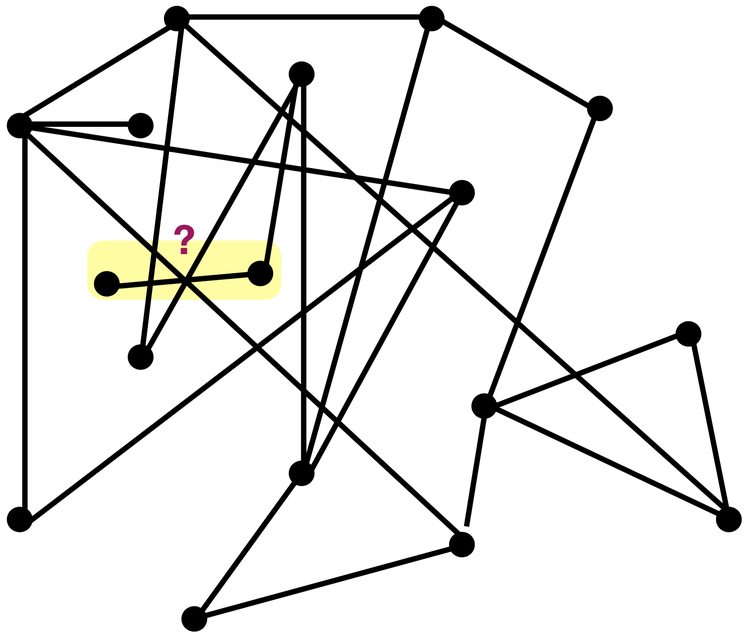
Adjacency list query

What is the degree of u?

Degree query

Our setting

Sublinear graph access



Queries:

Only need these

Is (u, v) an edge?

Adjacency matrix query

What is the i-th neighbor of u?

Adjacency list query

What is the degree of u?

Degree query

Comparison to worst-case setting

Is $Q_k(G) \in 1 \pm \varepsilon$?

$1/p^{\Theta(k^2)}$ queries/time
for worst-case graphs
[Eden, Ron, Seshadhri 2018]

Comparison to worst-case setting

Is $Q_k(G) \in 1 \pm \varepsilon$?

$1/p^{\Theta(k^2)}$ queries/time
for worst-case graphs
[Eden, Ron, Seshadhri 2018]

$(\mathcal{G}_{n,p}, Q_k)$-quality control

Theorem [M., Rubinfeld, Sudan]
$1/p^{\Theta(k)}$ queries/time*

*Tight up to universal constants in the exponent

Comparison to worst-case setting

Is $Q_k(G) \in 1 \pm \varepsilon$?

$1/p^{\Theta(k^2)}$ queries/time
for worst-case graphs
[Eden, Ron, Seshadhri 2018]

$(\mathcal{G}_{n,p}, Q_k)$-quality control

Theorem [M., Rubinfeld, Sudan]
$1/p^{\Theta(k)}$ queries/time*

$\exists\ k, p, c$ such that for $n \approx p^{-ck}$
[ERS18] requires $\Omega(n^k)$ time, and
we require $o(n)$.

*Tight up to universal constants in the exponent

Main results

Theorem: [M., Rubinfeld, Sudan]

1. $(\mathcal{G}_{n,p}, Q_k)$-quality control in $O\left((1/p)^{ck}\right)$ queries and time

1'. Tight up to c

Main results

Theorem: [M., Rubinfeld, Sudan]

1. $(\mathcal{G}_{n,p}, Q_k)$-quality control in $O\left((1/p)^{ck}\right)$ queries and time

1'. Tight up to c

2. (General version of 1) Quality control for count of motif with max degree Δ compared to $\mathcal{G}_{n,p}$: $O\left((1/p)^{c\Delta}\right)$ queries and time

2'. Tight up to c

Query-efficient algorithm

Pick random set $S \subseteq [n]$,
$|S| = s = 1/p^{O(k)}$

For $\ell = 2,3,\ldots,k$:

Count $C_\ell(S) = \#\ \ell$-cliques in S

Reject if $C_\ell(S) \not\approx \binom{s}{\ell} p^{\binom{\ell}{2}}$

Else Accept.

Query-efficient algorithm

Similar principles of **exponentially robust quasirandomness** wrt ℓ -cliques.

How do we make this time-efficient?

Pick random set $S \subseteq [n]$, $|S| = s = 1/p^{O(k)}$

For $\ell = 2,3,\ldots,k$:

Count $C_\ell(S) = \#\ \ell$-cliques in S

Reject if $C_\ell(S) \not\approx \binom{s}{\ell} p^{\binom{\ell}{2}}$

Else Accept.

Efficient approximate counting

Theorem: [Thomason '87]

Let G be an s-vertex graph such that all $\deg(u) \approx ps$ and $\text{codeg}(u, v) \approx p^2 s$.

Then, for all $\ell = 2,3,\ldots,k$, the number of ℓ-cliques in G is $\approx \binom{s}{\ell} p^{\binom{\ell}{2}}$.

codeg(u, v) = # shared neighbors of u and v

Efficient approximate counting

Theorem: [Thomason '87]

Let G be an s-vertex graph such that all $\deg(u) \approx ps$ and $\text{codeg}(u, v) \approx p^2 s$.

Then, for all $\ell = 2,3,\ldots,k$, the number of ℓ-cliques in G is $\approx \binom{s}{\ell} p^{\binom{\ell}{2}}$.

Relies on:
"jumbledness"

codeg(u, v) = # shared neighbors of u and v

Jumbled graphs

Theorem: [Thomason '87]

Let G be an s-vertex graph such that all $\deg(u) \approx ps$ and $\text{codeg}(u,v) \approx p^2 s$.

$\Rightarrow$ jumbled $\Rightarrow$

Then, for all $\ell = 2,3,\ldots,k$, the number of ℓ-cliques in G is $\approx \binom{s}{\ell} p^{\binom{\ell}{2}}$.

Definition: (Jumbled graph) [Thomason '87] $G = (V,E)$ is (p,β)-jumbled if

$$\forall X, Y \subseteq V(G), \left| e(X,Y) - p|X||Y| \right| \leq \beta\sqrt{|X||Y|}.$$

Jumbledness in quality control

Most subgraphs of $\mathcal{G}_{n,p}$ of size s

$\Rightarrow$

Degrees $\approx ps$, codegrees $\approx p^2 s$

$\Rightarrow$

Subgraph jumbled

$\Rightarrow$

Clique count matches $\mathcal{G}_{n,p}$

Final (runtime efficient) algorithm

Pick random set $S \subseteq [n]$, $|S| = s = 1/p^{O(k)}$

Reject if any $\deg_S(u) \not\approx ps$ or $\text{codeg}_S(u, v) \not\approx p^2 s$

Else Accept.

Final (runtime efficient) algorithm

Pick random set $S \subseteq [n]$, $|S| = s = 1/p^{O(k)}$

Reject if any $\deg_S(u) \not\approx ps$ or $\text{codeg}_S(u, v) \not\approx p^2 s$

Else Accept.

Runs quality control for jumbledness on S, $O(s^3)$ time

Final (runtime efficient) algorithm

Pick random set $S \subseteq [n]$, $|S| = s = 1/p^{O(k)}$

Reject if any $\deg_S(u) \not\approx ps$ or $\text{codeg}_S(u, v) \not\approx p^2 s$

Else Accept.

Runs quality control for jumbledness on S, $O(s^3)$ time

Implies all ℓ-clique count matches $\mathcal{G}_{n,p}$, $\ell \leq k$

Agenda

- Quality control:
  - Permutation counts
  - Broader discussion
- Permutation pattern counts
  - Query-efficient algorithm
  - Runtime-efficient algorithm
- Graphs
- Shared themes and open problems

Shared themes

Key feature of quality control:

Composability —can rule out “abnormal” inputs based on other properties

Shared themes

Key feature of quality control:

Composability —can rule out “abnormal” inputs based on other properties

Query complexity:

Robust tests of quasirandomness of lower-level objects

Runtime:

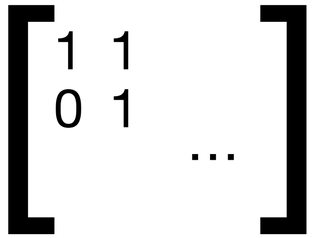
Notions of quasirandomness that imply efficient counting of patterns or motifs

Open problems

Different distributions
and quality functions

Open problems

Different distributions
and quality functions



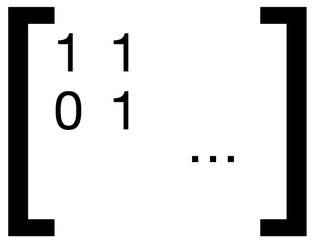
& counts of sub matrices

Other distributions $\mathscr{D}$

and quality functions ρ

Open problems

Different distributions
and quality functions



& counts of sub matrices

Other distributions $\mathscr{D}$

and quality functions ρ

When does quality control yield
more efficient algorithms?

Open problems

Different distributions and quality functions

$$\begin{bmatrix} 1 & 1 & \\ 0 & 1 & \\ & & \ddots \end{bmatrix}$$ & counts of sub matrices

Other distributions $\mathscr{D}$

and quality functions ρ

When does quality control yield more efficient algorithms?

Characterizations of quality control

Open problems

Different distributions and quality functions

$$\begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \quad \begin{matrix} \\ \\ \cdots \end{matrix}$$

& counts of sub matrices

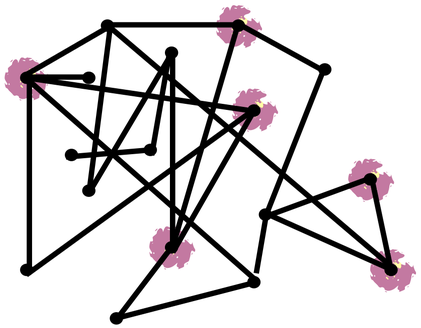
Other distributions $\mathscr{D}$

and quality functions ρ

When does quality control yield more efficient algorithms?

Characterizations of quality control

When are there simple tests using random samples?



Open problems

Different distributions and quality functions

$$\begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \quad \cdots$$

& counts of sub matrices

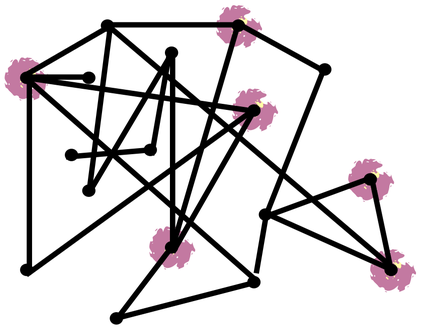
Other distributions $\mathscr{D}$

and quality functions ρ

When does quality control yield more efficient algorithms?

Characterizations of quality control

When are there simple tests using random samples?



Combinatorial characterizations?

Conclusion

Quality control: Accept $x \sim D$ whp and reject when $Q(x) \not\approx 1$

i.i.d. and length-k permutation patterns: $\text{poly}(k)$ queries/runtime

$\mathcal{G}_{n,p}$ **and k-cliques:** $1/p^{\Theta(k)}$ queries/runtime

Key tools: 1) Composability
2) New and robust versions of “quasirandomness”

Future directions: Other distributions and quality functions; characterizations

Thank you!