# Property Testing: Sublinear Algorithms for Promise Problems

Sourav Chakraborty
Chennai Mathematical Institute

# Outline

# Outline

# $k$-colorability of graphs

## Problem

Given an undirected graph $G$ and a parameter $k$ can we color the vertices of $G$ with $k$ colors such that no adjacent vertices are of the same color?

# What do we want?

# What do we want?

- This is a *NP*-complete problem.

## What do we want?

- This is a *NP*-complete problem.
- But we want to answer this problem VERY VERY fast - in less than a function of $k$ (say $f(k)$) number of steps.

## What do we want?

- This is a *NP*-complete problem.
- But we want to answer this problem VERY VERY fast - in less than a function of $k$ (say $f(k)$) number of steps.
- A algorithm that takes time $f(k)$ can only read at most $f(k)$ bits of the input.

# What do we want?

- This is a *NP*-complete problem.
- But we want to answer this problem VERY VERY fast - in less than a function of $k$ (say $f(k)$) number of steps.
- A algorithm that takes time $f(k)$ can only read at most $f(k)$ bits of the input.
- So this is IMPOSSIBLE, as the size of the graph can be way more than $k$.

## What do we want?

- This is a *NP*-complete problem.
- But we want to answer this problem VERY VERY fast - in less than a function of $k$ (say $f(k)$) number of steps.
- A algorithm that takes time $f(k)$ can only read at most $f(k)$ bits of the input.
- So this is IMPOSSIBLE, as the size of the graph can be way more than $k$.

For example, for $k = 2$, say our graph is a cycle on $2n - 1$ vertices.

# What do we want?

- This is a *NP*-complete problem.
- But we want to answer this problem VERY VERY fast - in less than a function of $k$ (say $f(k)$) number of steps.
- A algorithm that takes time $f(k)$ can only read at most $f(k)$ bits of the input.
- So this is IMPOSSIBLE, as the size of the graph can be way more than $k$.

For example, for $k = 2$, say our graph is a cycle on $2n - 1$ vertices.

Now if an algorithm (possibly randomized) looks at a constant number of bits of the graph will not catch the odd cycle. So the algorithm will fail to answer whether the graph is bipartite.

# So... should we give up?

# So... should we give up?

Many times we are interested in knowing if the graph is $k$-colorable or is “far” from being $k$-colorable.

# So... should we give up?

Many times we are interested in knowing if the graph is $k$-colorable or is "far" from being $k$-colorable.

In other words

# So... should we give up?

Many times we are interested in knowing if the graph is $k$-colorable or is “far” from being $k$-colorable.

In other words

- if the graph is $k$-colorable we want our algorithm to ACCEPT, and

## So... should we give up?

Many times we are interested in knowing if the graph is $k$-colorable or is "far" from being $k$-colorable.

In other words

- if the graph is $k$-colorable we want our algorithm to ACCEPT, and
- if it is VERY VERY DIFFERENT from $k$-colorable then we want to REJECT.

And if the graph is not $k$-colorable but $k$-colorable-like we might answer wrongly.

# So... should we give up?

Many times we are interested in knowing if the graph is $k$-colorable or is "far" from being $k$-colorable.

In other words

- if the graph is $k$-colorable we want our algorithm to ACCEPT, and
- if it is VERY VERY DIFFERENT from $k$-colorable then we want to REJECT.

And if the graph is not $k$-colorable but $k$-colorable-like we might answer wrongly.
CAN WE DO SOMETHING NOW?

## So... should we give up?

Many times we are interested in knowing if the graph is $k$-colorable or is “far” from being $k$-colorable.

In other words

- if the graph is $k$-colorable we want our algorithm to ACCEPT, and
- if it is VERY VERY DIFFERENT from $k$-colorable then we want to REJECT.

And if the graph is not $k$-colorable but $k$-colorable-like we might answer wrongly.
CAN WE DO SOMETHING NOW?
ANSWER: YES WE CAN

# YES WE CAN

## $k$-colorability Testing

[Goldreich-Goldwasser-Ron] There is a randomized algorithm $\mathcal{A}$ that looks at only a constant (function depending on $k$) number of bits of the input graph and

- if $G$ is $k$-colorable them $\Pr[\mathcal{A} \text{ accepts}] \geq 2/3$.
- if $G$ is "far" from $k$-colorable then $\Pr[\mathcal{A} \text{ rejects}] \geq 2/3$.

# How is it possible?

# How is it possible?

Lets consider an extremely simple problem:

# How is it possible?

Lets consider an extremely simple problem:

**Equality of strings**

Given two strings $x, y \in \{0,1\}^n$ check if $x = y$, that is, for every $i \in \{1, \dots, n\}$ is $x_i = y_i$.

# How is it possible?

Lets consider an extremely simple problem:

**Equality of strings**

Given two strings $x, y \in \{0,1\}^n$ check if $x = y$, that is, for every $i \in \{1,\dots,n\}$ is $x_i = y_i$.

The goal is to answer it in CONSTANT time and hence can't even read the whole input.

# How is it possible?

Lets consider an extremely simple problem:

**Equality of strings**

Given two strings $x, y \in \{0,1\}^n$ check if $x = y$, that is, for every $i \in \{1, \ldots, n\}$ is $x_i = y_i$.

The goal is to answer it in CONSTANT time and hence can't even read the whole input. —— Not Possible

# How is it possible?

Lets consider an extremely simple problem:

**Equality of strings**

Given two strings $x, y \in \{0,1\}^n$ check if $x = y$, that is, for every $i \in \{1, \ldots, n\}$ is $x_i = y_i$.

The goal is to answer it in CONSTANT time and hence can't even read the whole input. $--$ Not Possible

But, say, the guarantee is that either $x = y$ OR $x$ and $y$ differ at more than $1/4$ fraction of the indices. Then ...

# Simple sampling algorithm for testing of equality

**Algorithm**

Randomly pick 4 indices $\{i_1, i_2, i_3, i_4\}$ uniformly and independently at random. If

$$x_{i_1} = y_{i_1}, x_{i_2} = y_{i_2}, x_{i_3} = y_{i_3}, x_{i_4} = y_{i_4},$$

then ACCEPT otherwise REJECT.

# Simple sampling algorithm for testing of equality

**Algorithm**

Randomly pick 4 indices $\{i_1, i_2, i_3, i_4\}$ uniformly and independently at random. If

$$x_{i_1} = y_{i_1}, x_{i_2} = y_{i_2}, x_{i_3} = y_{i_3}, x_{i_4} = y_{i_4},$$

then ACCEPT otherwise REJECT.

- If $x = y$ then the algorithm always ACCEPTS.

# Simple sampling algorithm for testing of equality

## Algorithm

Randomly pick 4 indices $\{i_1, i_2, i_3, i_4\}$ uniformly and independently at random. If

$$x_{i_1} = y_{i_1}, x_{i_2} = y_{i_2}, x_{i_3} = y_{i_3}, x_{i_4} = y_{i_4},$$

then ACCEPT otherwise REJECT.

- If $x = y$ then the algorithm always ACCEPTS.
- If $x$ and $y$ differ at $1/4$ fraction of the indices then the algorithm ACCEPTS with probability at most $1/3$.

# Formal Definitions: Property and distance

- Let $x \in \{0, 1\}^n$ be an input.

# Formal Definitions: Property and distance

- Let $x \in \{0, 1\}^n$ be an input.
- A property $\mathcal{P}$ is a subset of $\{0, 1\}^n$.

# Formal Definitions: Property and distance

- Let $x \in \{0,1\}^n$ be an input.
- A property $\mathcal{P}$ is a subset of $\{0,1\}^n$.
- For two strings $x, y \in \{0,1\}^n$, $dist(x,y)$ is the fraction of indices where they differ.

$$dist(x,y) = |\{i | x_i \neq y_i\}|/n.$$

# Formal Definitions: Property and distance

- Let $x \in \{0,1\}^n$ be an input.
- A property $\mathcal{P}$ is a subset of $\{0,1\}^n$.
- For two strings $x, y \in \{0,1\}^n$, $dist(x,y)$ is the fraction of indices where they differ.

$$dist(x,y) = |\{i | x_i \neq y_i\}|/n.$$

- For a input $x$ and a property $\mathcal{P}$, $dist(x,\mathcal{P}) = \min_{y \in \mathcal{P}} dist(x,y)$.

# Formal Definitions: Property and distance

- Let $x \in \{0,1\}^n$ be an input.
- A property $\mathcal{P}$ is a subset of $\{0,1\}^n$.
- For two strings $x, y \in \{0,1\}^n$, $dist(x,y)$ is the fraction of indices where they differ.

$$dist(x,y) = |\{i | x_i \neq y_i\}|/n.$$

- For a input $x$ and a property $\mathcal{P}$, $dist(x,\mathcal{P}) = \min_{y \in \mathcal{P}} dist(x,y)$.
- $x$ is $\epsilon$-far from being a property if $dist(x,\mathcal{P}) > \epsilon$.

# Formal Definitions: Property and distance

- Let $x \in \{0,1\}^n$ be an input.
- A property $\mathcal{P}$ is a subset of $\{0,1\}^n$.
- For two strings $x, y \in \{0,1\}^n$, $dist(x,y)$ is the fraction of indices where they differ.

$$dist(x,y) = |\{i | x_i \neq y_i\}|/n.$$

- For a input $x$ and a property $\mathcal{P}$, $dist(x,\mathcal{P}) = \min_{y \in \mathcal{P}} dist(x,y)$.
- $x$ is $\epsilon$-far from being a property if $dist(x,\mathcal{P}) > \epsilon$.

## Promise Problem

For a property $\mathcal{P}$ and a distance parameter $\epsilon$, given an input $x$ distinguish between the two cases:

(a) Is $x \in \mathcal{P}$, OR (b) Is $x$ $\epsilon$-far from $\mathcal{P}$.

# Other Examples of Promise Problem

- **Isomorphism Testing:** Given two objects $\mathcal{O}_1$ and $\mathcal{O}_2$ test if the two are isomorphic OR far-from being isomorphic. (For example: Graph Isomorphism or Function Isomorphism)

# Other Examples of Promise Problem

- **Isomorphism Testing:** Given two objects $\mathcal{O}_1$ and $\mathcal{O}_2$ test if the two are isomorphic OR far-from being isomorphic. (For example: Graph Isomorphism or Function Isomorphism)
- **Connectivity** Given a graph test if it is connected or far-from being connected.

# Other Examples of Promise Problem

- **Isomorphism Testing:** Given two objects $\mathcal{O}_1$ and $\mathcal{O}_2$ test if the two are isomorphic OR far-from being isomorphic. (For example: Graph Isomorphism or Function Isomorphism)
- **Connectivity** Given a graph test if it is connected or far-from being connected.
- **Linearity Testing:** Given a truth-table of a function $f$ test if the function $f$ is linear OR the function has to be changed at at-least $\epsilon$ fraction of the domain to make it linear.

# Other Examples of Promise Problem

- **Isomorphism Testing:** Given two objects $\mathcal{O}_1$ and $\mathcal{O}_2$ test if the two are isomorphic OR far-from being isomorphic. (For example: Graph Isomorphism or Function Isomorphism)
- **Connectivity** Given a graph test if it is connected or far-from being connected.
- **Linearity Testing:** Given a truth-table of a function $f$ test if the function $f$ is linear OR the function has to be changed at at-least $\epsilon$ fraction of the domain to make it linear.
- **Distribution Testing:** Is a given distribution uniform or is the $\ell_1$ distance from uniform more than $\epsilon$?

# Other Examples of Promise Problem

- **Isomorphism Testing:** Given two objects $\mathcal{O}_1$ and $\mathcal{O}_2$ test if the two are isomorphic OR far-from being isomorphic. (For example: Graph Isomorphism or Function Isomorphism)
- **Connectivity** Given a graph test if it is connected or far-from being connected.
- **Linearity Testing:** Given a truth-table of a function $f$ test if the function $f$ is linear OR the function has to be changed at at-least $\epsilon$ fraction of the domain to make it linear.
- **Distribution Testing:** Is a given distribution uniform or is the $\ell_1$ distance from uniform more than $\epsilon$?
- **Branching Program Testing:** Given a truth-table of a function $f$ test if the function is accepted by a constant depth read-once branching program OR is far from being accepted by a constant depth read-once branching program.

# Our Goal ...

- We want to design a randomized algorithm that answers the promise problem correctly with high probability.

# Our Goal ...

- We want to design a randomized algorithm that answers the promise problem correctly with high probability.
- We want to look at a very small portion of the input.

# Our Goal ...

- We want to design a randomized algorithm that answers the promise problem correctly with high probability.
- We want to look at a very small portion of the input.

In the rest of the talk we would not consider the running time of an algorithm but rather the number of bits of the input that is read. Accessing each bit of the input is called a QUERY.

# Property tester

## Definition

Let $\mathcal{P}$ be a property. A tester for $\mathcal{P}$ is a *randomized* algorithm $\mathcal{A}$ with black box access to an input $x$ and satisfies:

- If $x \in \mathcal{P} \Rightarrow \Pr[\mathcal{A} \text{ accepts}] \geq 2/3$.
- If $x$ is $\epsilon$-far from $\mathcal{P} \Rightarrow \Pr[\mathcal{A} \text{ rejects}] \geq 2/3$.

We allow the algorithm to be *adaptive* (queries may depend on the outcome of previous queries).

# Query Complexity

Query complexity for the tester $\mathcal{A}$ is the maximum number of queries queried by the tester on any input.

# Query Complexity

Query complexity for the tester $\mathcal{A}$ is the maximum number of queries queried by the tester on any input.

Query complexity of a property $\mathcal{P}$ is the query complexity of the tester that has the minimum query complexity.

# Query Complexity

Query complexity for the tester $\mathcal{A}$ is the maximum number of queries queried by the tester on any input.

Query complexity of a property $\mathcal{P}$ is the query complexity of the tester that has the minimum query complexity.

Trivial example: let $\mathcal{P}$ be the property "$x \equiv 0$". Then taking $O(1/\epsilon)$ independent samples works w.h.p.

## Different Models

There are different models depending on:

- How the input is represented? For example, is the graph given as adjacency matrix or adjacency list or some other way. [Dense graph model, sparse graph model, orientation model in graph testing]

# Different Models

There are different models depending on:

- How the input is represented? For example, is the graph given as adjacency matrix or adjacency list or some other way. [Dense graph model, sparse graph model, orientation model in graph testing]
- How the queries are made? [Classical, quantum]

# Different Models

There are different models depending on:

- How the input is represented? For example, is the graph given as adjacency matrix or adjacency list or some other way. [Dense graph model, sparse graph model, orientation model in graph testing]
- How the queries are made? [Classical, quantum]
- Do we also want to accept inputs that are “close” to the property? [Tolerant model and Intolerant Model]

# Different Models

There are different models depending on:

- How the input is represented? For example, is the graph given as adjacency matrix or adjacency list or some other way. [Dense graph model, sparse graph model, orientation model in graph testing]
- How the queries are made? [Classical, quantum]
- Do we also want to accept inputs that are "close" to the property? [Tolerant model and Intolerant Model]
- Restricted error. [One-sided error or two-sided error]

# Different Models

There are different models depending on:

- How the input is represented? For example, is the graph given as adjacency matrix or adjacency list or some other way. [Dense graph model, sparse graph model, orientation model in graph testing]
- How the queries are made? [Classical, quantum]
- Do we also want to accept inputs that are "close" to the property? [Tolerant model and Intolerant Model]
- Restricted error. [One-sided error or two-sided error]
- Correct errors. [Self-correction, Reconstruction]

# What kind of questions to ask?

- Given a property $\mathcal{P}$ what is the query complexity for testing $\mathcal{P}$.
  - Design a property tester that tests $\mathcal{P}$ using $O(q)$ number of queries.
  - Prove that no property tester can test using less than $\Omega(q)$ number of queries.

# What kind of questions to ask?

- Given a property $\mathcal{P}$ what is the query complexity for testing $\mathcal{P}$.
  - Design a property tester that tests $\mathcal{P}$ using $O(q)$ number of queries.
  - Prove that no property tester can test using less than $\Omega(q)$ number of queries.
- Classify the set of properties that can be tested using constant number of queries.

# What kind of questions to ask?

- Given a property $\mathcal{P}$ what is the query complexity for testing $\mathcal{P}$.
  - Design a property tester that tests $\mathcal{P}$ using $O(q)$ number of queries.
  - Prove that no property tester can test using less than $\Omega(q)$ number of queries.
- Classify the set of properties that can be tested using constant number of queries.
- Come up with the right model for testing.

# Outline

# 1-sided error testers

## 1-sided-error property tester

Let $\mathcal{P}$ be a property. A 1-sided-error property tester for $\mathcal{P}$ is a *randomized* algorithm $\mathcal{A}$ with black box access to an input $x$ and satisfies:

- (Completeness) If $x \in \mathcal{P} \Rightarrow \Pr[\mathcal{A} \text{ accepts}] = 1$.
- (Soundness) If $x$ is $\epsilon$-far from $\mathcal{P} \Rightarrow \Pr[\mathcal{A} \text{ rejects}] \geq 2/3$.

We allow the algorithm to be *adaptive* (queries may depend on the outcome of previous queries).

# 1-sided-error tester has its hands tied

- The tester has to ACCEPT if the input satisfies the property.

## 1-sided-error tester has its hands tied

- The tester has to ACCEPT if the input satisfies the property.
- Hence, the only way the tester can reject is if it find a PROOF that the input does not satisfy the property.

# 1-sided-error tester has its hands tied

- The tester has to ACCEPT if the input satisfies the property.
- Hence, the only way the tester can reject is if it find a PROOF that the input does not satisfy the property.
- So if the input does not have the property then the tester must find a PROOF/WITNESS with high probability.

# Typical 1-sided-error tester

## 1-sided-error algorithm

Query some bits of the input. The bits to be queried can be either uniformly chosen or chosen in a cleaver co-related fashion.

- If the answers of the queried bits contains a WITNESS that the input is not in the property then REJECT
- Else ACCEPT

Goal is to use some nice structure for the property for making the queries, like

- the Szemeredi's Regularity Lemma for graphs,
- properties of Fourier coefficients for algebraic functions, etc

Usually, the proof of SOUNDNESS is the hard part.

## So what is the success probability of the tester?

Say the tester uses the random string $r$ and queries the bits in $Q_r$ (also say $|Q_r| = q$). Then the probability of success is

$$\Pr_r[Q_r \text{ contains a WITNESS}].$$

# So what is the success probability of the tester?

Say the tester uses the random string $r$ and queries the bits in $Q_r$ (also say $|Q_r| = q$). Then the probability of success is

$$\Pr_r[Q_r \text{ contains a WITNESS}].$$

Thus a 1-sided-error property tester can successfully test a property $\mathcal{P}$ with $q$ queries only if, an input $x$ is "far" from $\mathcal{P}$ implies there is a lots of WITNESS of size $q$ hidden in $x$.

# For Example

**Theorem (Goldreich-Goldwasser-Ron)**

*There exist an 1-sided-error tester for testing k-colorability of graphs using $O(k/\epsilon)$ number of queries.*

# For Example

**Theorem (Goldreich-Goldwasser-Ron)**

*There exist an 1-sided-error tester for testing k-colorability of graphs using $O(k/\epsilon)$ number of queries.*

The above theorem implies that if a graph $G$ is "$\epsilon$-far" from being $k$-colorable then there exists a lot of the subgraphs of $G$, of size $k/\epsilon$, is not $k$-colorable.

# Lower bounds for 1-sided-error testing

# Lower bounds for 1-sided-error testing

$\mathcal{D}_N$ be a distribution on the the set of inputs that are far from $\mathcal{P}$.

# Lower bounds for 1-sided-error testing

$\mathcal{D}_N$ be a distribution on the the set of inputs that are far from $\mathcal{P}$.

The input $x$ is chosen according to the distribution $\mathcal{D}_N$.

# Lower bounds for 1-sided-error testing

$\mathcal{D}_N$ be a distribution on the the set of inputs that are far from $\mathcal{P}$.

The input $x$ is chosen according to the distribution $\mathcal{D}_N$.

And now if one shows that any deterministic algorithms that makes $q$ queries will catch a WITNESS with very low probability then we obtain a lower bound of $q$ on the query complexity for testing $\mathcal{P}$.

# Lower bounds for 1-sided-error testing

$\mathcal{D}_N$ be a distribution on the the set of inputs that are far from $\mathcal{P}$.

The input $x$ is chosen according to the distribution $\mathcal{D}_N$.

And now if one shows that any deterministic algorithms that makes $q$ queries will catch a WITNESS with very low probability then we obtain a lower bound of $q$ on the query complexity for testing $\mathcal{P}$.

For example: Checking whether $f : [n] \to [n]$ is 1-to-1 or 2-to-1 requires at least $\sqrt{n}$ queries. (By Birthday Paradox)

# 2-sided-error property tester

## 2-sided-property tester

Let $\mathcal{P}$ be a property. A 2-sided-error tester for $\mathcal{P}$ is a *randomized* algorithm $\mathcal{A}$ with black box access to an input $x$ and satisfies:

- (Completeness) If $x \in \mathcal{P} \Rightarrow \Pr[\mathcal{A} \text{ accepts}] \geq 2/3$.
- (Soundness) If $x$ is $\epsilon$-far from $\mathcal{P} \Rightarrow \Pr[\mathcal{A} \text{ rejects}] \geq 2/3$.

We allow the algorithm to be *adaptive* (queries may depend on the outcome of previous queries).

## 2-sided-error tester

## 2-sided-error tester

- The tester does not have to find a PROOF/WITNESS to REJECT or ACCEPT.

# 2-sided-error tester

- The tester does not have to find a PROOF/WITNESS to REJECT or ACCEPT.
- The tester can use estimation/approximation as a tool.

For example: Distinguishing whether a string $x \in \{0,1\}^n$ has $n/4$ $1's$ OR $n/3$ $1's$ can be done using CONSTANT number of queries.

In general 2-sided-error algorithms can be very complicated.

# Lower bounds for 2-sided-error testing

# Lower bounds for 2-sided-error testing

Let $\mathcal{D}_N$ be a distribution on the the set of inputs that are far from $\mathcal{P}$ and $\mathcal{D}_Y$ be a distribution on the the set of inputs that satisfy $\mathcal{P}$.

# Lower bounds for 2-sided-error testing

Let $\mathcal{D}_N$ be a distribution on the the set of inputs that are far from $\mathcal{P}$ and $\mathcal{D}_Y$ be a distribution on the the set of inputs that satisfy $\mathcal{P}$. The input $x$ is chosen in the following manner:

## Lower bounds for 2-sided-error testing

Let $\mathcal{D}_N$ be a distribution on the the set of inputs that are far from $\mathcal{P}$ and $\mathcal{D}_Y$ be a distribution on the the set of inputs that satisfy $\mathcal{P}$. The input $x$ is chosen in the following manner:

- With probability $1/2$ the input $x$ is chosen according to the distribution $\mathcal{D}_Y$

## Lower bounds for 2-sided-error testing

Let $\mathcal{D}_N$ be a distribution on the the set of inputs that are far from $\mathcal{P}$ and $\mathcal{D}_Y$ be a distribution on the the set of inputs that satisfy $\mathcal{P}$. The input $x$ is chosen in the following manner:

- With probability $1/2$ the input $x$ is chosen according to the distribution $\mathcal{D}_Y$
- With the other $1/2$ probability the input $x$ is chosen according to the distribution $\mathcal{D}_N$.

# Lower bounds for 2-sided-error testing

Let $\mathcal{D}_N$ be a distribution on the the set of inputs that are far from $\mathcal{P}$ and $\mathcal{D}_Y$ be a distribution on the the set of inputs that satisfy $\mathcal{P}$. The input $x$ is chosen in the following manner:

- With probability $1/2$ the input $x$ is chosen according to the distribution $\mathcal{D}_Y$
- With the other $1/2$ probability the input $x$ is chosen according to the distribution $\mathcal{D}_N$.

And now if one shows that any deterministic algorithms that makes $q$ queries cannot distinguish the two kind of inputs then by Yao's Lemma we obtain a lower bound of $q$ on the query complexity for testing $\mathcal{P}$.

# Lower bounds for 2-sided-error testing

Let $\mathcal{D}_N$ be a distribution on the the set of inputs that are far from $\mathcal{P}$ and $\mathcal{D}_Y$ be a distribution on the the set of inputs that satisfy $\mathcal{P}$. The input $x$ is chosen in the following manner:

- With probability $1/2$ the input $x$ is chosen according to the distribution $\mathcal{D}_Y$
- With the other $1/2$ probability the input $x$ is chosen according to the distribution $\mathcal{D}_N$.

And now if one shows that any deterministic algorithms that makes $q$ queries cannot distinguish the two kind of inputs then by Yao's Lemma we obtain a lower bound of $q$ on the query complexity for testing $\mathcal{P}$.

So, if the distribution of answers to the queries are similar when the input is drawn according to $\mathcal{D}_N$ and when it is drawn according to $\mathcal{D}_Y$ then the query complexity is $\geq q$.

# Outline

# Testing of Function Properties

- The property $\mathcal{P}$ is a set of functions from $\Sigma^n \to \Sigma$. For example: Linear functions, functions that are 1-to-1, functions accepted by a constant width read-once branching program etc.

# Testing of Function Properties

- The property $\mathcal{P}$ is a set of functions from $\Sigma^n \to \Sigma$. For example: Linear functions, functions that are 1-to-1, functions accepted by a constant width read-once branching program etc.
- The input is a truth-table of a function $f : \Sigma^n \to \Sigma$.

# Testing of Function Properties

- The property $\mathcal{P}$ is a set of functions from $\Sigma^n \to \Sigma$. For example: Linear functions, functions that are 1-to-1, functions accepted by a constant width read-once branching program etc.
- The input is a truth-table of a function $f : \Sigma^n \to \Sigma$.
- Queries are of form: $x \in \Sigma^n \longrightarrow f(x)$.

## Property Tester for $\mathcal{P}$

A 1-sided-error tester for $\mathcal{P}$ is a *randomized* algorithm $\mathcal{A}$ that given query access to a truth-table of a function $f$ does the following:

- If $f \in \mathcal{P} \Rightarrow \Pr[\mathcal{A} \text{ accepts}] = 1$.
- If for at least $\epsilon|\Sigma|^n$ number of strings in $\Sigma^n$ the value of $f$ has to be changed so that the property $\mathcal{P}$ is satisfied then $\Pr[\mathcal{A} \text{ rejects}] \geq 2/3$.

# Testing of Linearity

## Linearity testing

Given query access to a Boolean function $f : \{0,1\}^n \to \{0,1\}$ test if $f$ is linear, that is, if for all $x, y \in \{0,1\}^n$, $f(x) \oplus f(y) = f(x \oplus y)$.

# Testing of Linearity

## Linearity testing

Given query access to a Boolean function $f : \{0,1\}^n \to \{0,1\}$ test if $f$ is linear, that is, if for all $x, y \in \{0,1\}^n$,
$f(x) \oplus f(y) = f(x \oplus y)$.

The obvious test is the following: pick two random $x, y \in \{0,1\}^n$ and if $f(x) \oplus f(y) \neq f(x \oplus y)$ then REJECT else ACCEPT.

# Testing of Linearity

## Linearity testing

Given query access to a Boolean function $f : \{0,1\}^n \to \{0,1\}$ test if $f$ is linear, that is, if for all $x, y \in \{0,1\}^n$,
$f(x) \oplus f(y) = f(x \oplus y)$.

The obvious test is the following: pick two random $x, y \in \{0,1\}^n$ and if $f(x) \oplus f(y) \neq f(x \oplus y)$ then REJECT else ACCEPT.

## Linearity Testing [Blum-Luby-Rubinfeld]

The above tester has the following properties:

- If $f$ is linear then the tester always ACCEPTS.
- If $f$ is $\epsilon$-far from linear then the tester REJECTS with high probability. (Proof using Fourier Analysis).

## Generalization of Linearity Testing

Given query access to a function $f : \mathbb{F}^n \to \mathbb{F}$ test if $f$ is a degree $d$ polynomial.

# Generalization of Linearity Testing

Given query access to a function $f : \mathbb{F}^n \rightarrow \mathbb{F}$ test if $f$ is a degree $d$ polynomial.

**Low-degree testing [Babai-Fortnow-Lund, Rubinfeld-Sudan]**

The query complexity for testing degree $d$ polynomials is a function of $|\mathbb{F}|$ and $d$. When $|\mathbb{F}| = 2$ then the query complexity is $2^d$ and when $|\mathbb{F}|$ is around $d$ then the query complexity is $poly(d)$.

# Generalization of Linearity Testing

Given query access to a function $f : \mathbb{F}^n \to \mathbb{F}$ test if $f$ is a degree $d$ polynomial.

**Low-degree testing [Babai-Fortnow-Lund, Rubinfeld-Sudan]**

The query complexity for testing degree $d$ polynomials is a function of $|\mathbb{F}|$ and $d$. When $|\mathbb{F}| = 2$ then the query complexity is $2^d$ and when $|\mathbb{F}|$ is around $d$ then the query complexity is $poly(d)$.

This tester in also used in Probabilistically Checkable Proofs (PCP) [Arora-Safra, Arora-Lund-Motwani-Sudan-Szegedy]

# Degree $d$ tester, when $\mathbb{F} > d$.

## Algorithm (For $|\mathbb{F}| > d$)

- *Pick a random* $x \in \mathbb{F}^n$
- *Pick a random line through* $x$. *Pick a random* $y \in \mathbb{F}^n$ *and consider all points of form* $x + \lambda y$.
- *Query at all the* $|\mathbb{F}|$ *points.*
- *If* $f$ *is a degree* $d$ *polynomial then restricted to this line it is a degree* $d$ *univariate polynomial in variable* $\lambda$.
- *Use the points* $f(x + \lambda y)$, *when* $\lambda \neq 0$ *to fit a degree* $d$ *polynomial.*
- *If the polynomial evaluated at* $\lambda = 0$ *is equal to* $f(x)$ *then ACCEPT else REJECT.*

# Monotonicity Testing

Given query access to a function $f : \{0,1\}^n \to \mathbb{R}$ test if $f$ is monotone, that is, if $x, y \in \{0,1\}^n$ are such that for all $i \in [n]$ $x_i \leq y_i$ then $f(x) \leq f(y)$.

# Monotonicity Testing

Given query access to a function $f : \{0,1\}^n \to \mathbb{R}$ test if $f$ is monotone, that is, if $x, y \in \{0,1\}^n$ are such that for all $i \in [n]$ $x_i \leq y_i$ then $f(x) \leq f(y)$.

**Monotonicity Testing [Fischer-Lehman-Newman-Raskhodnikova-Rubinfeld-Samorodnitsky,Briet-C-Garcia-Soriano-Matsliah]**

The 1-sided-error query complexity for testing monotonicity with arbitrary range is $O(n^2)$ and $\Omega(n)$.

# Monotonicity Testing

Given query access to a function $f : \{0,1\}^n \to \mathbb{R}$ test if $f$ is monotone, that is, if $x, y \in \{0,1\}^n$ are such that for all $i \in [n]$ $x_i \leq y_i$ then $f(x) \leq f(y)$.

**Monotonicity Testing [Fischer-Lehman-Newman-Raskhodnikova-Rubinfeld-Samorodnitsky,Briet-C-Garcia-Soriano-Matsliah]**

The 1-sided-error query complexity for testing monotonicity with arbitrary range is $O(n^2)$ and $\Omega(n)$.

The upper bound is just a pair-tester where the tester picks $x \in \{0,1\}^n$ and an $i \in \{1,\ldots,n\}$ at random and checks if

$$f(x) \leq f(x \oplus e_i).$$

Repeat it $n^2$ times.

# Testing of combinatorial/complexity measures of functions.

## Testing of BP [Newman]

Given query access to a function $f : \{0,1\}^n \to \{0,1\}$, testing if $f$ is accepted by a width $w$ read-once branching program can be done using $O(w)$ number of queries.

# Testing of combinatorial/complexity measures of functions.

## Testing of BP [Newman]

Given query access to a function $f : \{0,1\}^n \to \{0,1\}$, testing if $f$ is accepted by a width $w$ read-once branching program can be done using $O(w)$ number of queries.

## Testing of Junta [Blais]

Given query access to a function $f : \{0,1\}^n \to \{0,1\}$, testing if $f$ is $k$-junta can be done using $O(k \log k)$ number of queries.

# Testing of combinatorial/complexity measures of functions.

## Testing of BP [Newman]

Given query access to a function $f : \{0,1\}^n \to \{0,1\}$, testing if $f$ is accepted by a width $w$ read-once branching program can be done using $O(w)$ number of queries.

## Testing of Junta [Blais]

Given query access to a function $f : \{0,1\}^n \to \{0,1\}$, testing if $f$ is $k$-junta can be done using $O(k \log k)$ number of queries.

## Testing of Circuit size [Diakonikolas-Lee-Matulef-Onak-Rubinfeld-Servedio, C-Garcia-Soriano-Matsliah]

Given query access to a function $f : \{0,1\}^n \to \{0,1\}$, testing if $f$ is accepted by a circuit of size $s$ has query complexity $s^{\Theta(1)}$.

# Properties of Distributions

A $f : \{1, \ldots, n\} \to \{1, \ldots, k\}$ defines a distribution $\mathcal{D}_f$ on $\{1, \ldots, k\}$, where

$$\Pr_{x \leftarrow \mathcal{D}_f}[x = i] = |f^{-1}(i)|/n.$$

# Properties of Distributions

A $f : \{1, \dots, n\} \to \{1, \dots, k\}$ defines a distribution $\mathcal{D}_f$ on $\{1, \dots, k\}$, where

$$\Pr_{x \leftarrow \mathcal{D}_f}[x = i] = |f^{-1}(i)|/n.$$

**Testing of Uniformity :** Given query access to a function $f : \{1, \dots, n\} \to \{1, \dots, k\}$ test if $\mathcal{D}_f$ is uniform OR $\ell_1$ distance from the uniform distribution is more than $\epsilon$.

# Properties of Distributions

A $f : \{1, \dots, n\} \to \{1, \dots, k\}$ defines a distribution $\mathcal{D}_f$ on $\{1, \dots, k\}$, where

$$\Pr_{x \leftarrow \mathcal{D}_f}[x = i] = |f^{-1}(i)|/n.$$

**Testing of Uniformity :** Given query access to a function $f : \{1, \dots, n\} \to \{1, \dots, k\}$ test if $\mathcal{D}_f$ is uniform OR $\ell_1$ distance from the uniform distribution is more than $\epsilon$.

**Uniformity Testing [Batu-Fortnow-Rubinfeld-Smith-White]**

2-side-error query complexity for testing uniformity is $\tilde{\Theta}(\sqrt{k})$.

# Sketch of Proof for Testing Uniformity

**Uniformity Testing [Batu-Fortnow-Rubinfeld-Smith-White]**

2-side-error query complexity for testing uniformity is $\tilde{\Theta}(\sqrt{k})$.

# Sketch of Proof for Testing Uniformity

**Uniformity Testing [Batu-Fortnow-Rubinfeld-Smith-White]**

2-side-error query complexity for testing uniformity is $\tilde{\Theta}(\sqrt{k})$.

**Proof.**

Upper bound: Take random $\sqrt{k}$ samples and check if they fall in different buckets. If they all fall on distinct buckets estimate the fraction of elements that fall in these buckets.

# Sketch of Proof for Testing Uniformity

**Uniformity Testing [Batu-Fortnow-Rubinfeld-Smith-White]**

2-side-error query complexity for testing uniformity is $\tilde{\Theta}(\sqrt{k})$.

**Proof.**

Upper bound: Take random $\sqrt{k}$ samples and check if they fall in different buckets. If they all fall on distinct buckets estimate the fraction of elements that fall in these buckets.

Lower bound: Distinguishing whether $f$ is uniform with support size $k$ from $f$ is uniform with support size $k/2$ requires $\sqrt{k}$ queries. Just like distinguishing 1-to-1 function from 2-to-1 functions. $\square$

# Outline

# Testing of graph property

- A property $\mathcal{P}$ is a set of graphs. For example: all bipartite graphs, all graphs that is isomorphic to a particular graph, all graphs where there exists a path from vertex 1 to vertex 2, ...

# Testing of graph property

- A property $\mathcal{P}$ is a set of graphs. For example: all bipartite graphs, all graphs that is isomorphic to a particular graph, all graphs where there exists a path from vertex 1 to vertex 2, ...
- How is the graph given as input?

# Dense Graph Model

A property $\mathcal{P}$ is a set of graphs. For example: all bipartite graphs, all graphs that is isomorphic to a particular graph, all graphs where there exists a path from vertex 1 to vertex 2, ...

# Dense Graph Model

A property $\mathcal{P}$ is a set of graphs. For example: all bipartite graphs, all graphs that is isomorphic to a particular graph, all graphs where there exists a path from vertex 1 to vertex 2, ...
The graph is given as an adjacency matrix. The input size is $\binom{|V|}{2}$.

# Dense Graph Model

A property $\mathcal{P}$ is a set of graphs. For example: all bipartite graphs, all graphs that is isomorphic to a particular graph, all graphs where there exists a path from vertex 1 to vertex 2, ...
The graph is given as an adjacency matrix. The input size is $\binom{|V|}{2}$.

A query is of form: Is there an edge between vertex $i$ and $j$?

# Dense Graph Model

A property $\mathcal{P}$ is a set of graphs. For example: all bipartite graphs, all graphs that is isomorphic to a particular graph, all graphs where there exists a path from vertex 1 to vertex 2, ...
The graph is given as an adjacency matrix. The input size is $\binom{|V|}{2}$.

A query is of form: Is there an edge between vertex $i$ and $j$?

## Definition

A 1-sided-error tester for $\mathcal{P}$ is a *randomized* algorithm $\mathcal{A}$ that given query access to a graph $G$ does the following:

- If $G \in \mathcal{P} \Rightarrow \Pr[\mathcal{A} \text{ accepts}] = 1$.
- If at least $\epsilon\binom{|V|}{2}$ number of entries of the adjacency matrix has to be changed so that the property $\mathcal{P}$ is satisfied then $\Pr[\mathcal{A} \text{ rejects}] \geq 2/3$.

# Testing of Bipartiteness in the dense graph model

Given query access to the adjacency matrix of a graph $G$, test if $G$ is bipartite of one has to remove $\epsilon\binom{|V|}{2}$ edges to make it bipartite.

# Testing of Bipartiteness in the dense graph model

Given query access to the adjacency matrix of a graph $G$, test if $G$ is bipartite of one has to remove $\epsilon\binom{|V|}{2}$ edges to make it bipartite.

**Algorithm**

- Pick $O(1/\epsilon^2 \log(1/\epsilon))$ number of vertices at random.
- Query all the pairs of selected vertices.
- If the induced graph is not bipartite REJECT else ACCEPT

**Proof:** If the graph is bipartite the algorithm always accept.

So now we have to prove that if $G$ is $\epsilon$-far from being bipartite then the induced graph is not bipartite with high probability.

# Proof of Soundness of the Algorithm for Testing Bipartiteness

Since it is a 1-sided-error algorithm for every possible bipartition of the vertex set we should catch a violating edge, that is edges within the same part.

# Proof of Soundness of the Algorithm for Testing Bipartiteness

Since it is a 1-sided-error algorithm for every possible bipartition of the vertex set we should catch a violating edge, that is edges within the same part.

If the graph is $\epsilon$-far from being bipartite then any bipartition of the vertex set will have at least $\epsilon|V|^2$ violating edges.

# Proof of Soundness of the Algorithm for Testing Bipartiteness

Since it is a 1-sided-error algorithm for every possible bipartition of the vertex set we should catch a violating edge, that is edges within the same part.

If the graph is $\epsilon$-far from being bipartite then any bipartition of the vertex set will have at least $\epsilon|V|^2$ violating edges.

Note that given a particular bipartition by randomly sampling of $O(1/\epsilon^2)$ edges we would catch a violation for that bipartition with high probability. But we have to catch for all the bipartitions with high probability. Unfortunately, simple union bound does not give the math as the number of such bipartitions is $2^{|V|}$.

# Proof of Soundness of the Algorithm for Testing Bipartiteness (contd...)

So we think of the selected vertices as two sets $V_A$ and $V_B$. Vertices $V_A$ induces the subgraph $G_A$.

After we have queried the subgraph $G_A$ we show only a “small” number of partitions survive with high probability.

And then we can say, using union bound, that the second set $V_B$ helps to catch the violations for the small number of surviving bipartitions.

# Various other Graph Properties

- $k$-colorability of graphs $--$ $O(k/\epsilon)$.

# Various other Graph Properties

- $k$-colorability of graphs $--$ $O(k/\epsilon)$.
- Is there a clique of size $\rho n$ $--$ $O(1/\epsilon)$ number of queries. (2-sided-error)

## Various other Graph Properties

- $k$-colorability of graphs $--$ $O(k/\epsilon)$.
- Is there a clique of size $\rho n$ $--$ $O(1/\epsilon)$ number of queries. (2-sided-error)
- Triangle free-ness $--$ $tower(1/\epsilon)$. (Using Regularity Lemma)

## What all can be tested?

Can we characterize all the graph properties that can be tested by a 1-sided-error tester using constant number of queries.

# What all can be tested?

Can we characterize all the graph properties that can be tested by a 1-sided-error tester using constant number of queries.

**Theorem (Alon-Shapira)**

*A graph property is called monotone if it is closed under removal of edges and vertices. Every monotone graph property is testable with constant number of queries.*

# What all can be tested?

Can we characterize all the graph properties that can be tested by a 1-sided-error tester using constant number of queries.

**Theorem (Alon-Shapira)**

*A graph property is called monotone if it is closed under removal of edges and vertices. Every monotone graph property is testable with constant number of queries.*

The proof uses Szemeredi's Regularity Lemma.

# What all can be tested?

Can we characterize all the graph properties that can be tested by a 1-sided-error tester using constant number of queries.

**Theorem (Alon-Shapira)**

*A graph property is called monotone if it is closed under removal of edges and vertices. Every monotone graph property is testable with constant number of queries.*

The proof uses Szemeredi's Regularity Lemma.
The proof roughly based on the idea that testing monotone graph properties can be reduced to testing whether the graph has a regular-partition with certain parameters.

# What all can be tested?

Can we characterize all the graph properties that can be tested by a 1-sided-error tester using constant number of queries.

**Theorem (Alon-Shapira)**

*A graph property is called monotone if it is closed under removal of edges and vertices. Every monotone graph property is testable with constant number of queries.*

The proof uses Szemeredi's Regularity Lemma.
The proof roughly based on the idea that testing monotone graph properties can be reduced to testing whether the graph has a regular-partition with certain parameters.
And testing whether a graph has a regular-partition can be tested with constant number of queries.

# In the dense-graph-model its all about regularity

**Theorem (Alon-Fischer-Newman-Shapira)**

*A graph property $P$ can be tested with a constant number of queries if and only if testing $P$ can be reduced to testing the property of satisfying one of finitely many Szemeredi-partitions.*

# Testing of Connectivity

**Problem**

*Can we test whether in a graph is connected?*

# Testing of Connectivity

**Problem**

*Can we test whether in a graph is connected?*

Actually, ... this problem does not make much sense - in the dense graph model.
All graphs are just $|V|$ changes away from being connected and hence all graphs are $\epsilon$-close to being connected.

# Testing of Connectivity

## Problem

*Can we test whether in a graph is connected?*

Actually, ... this problem does not make much sense - in the dense graph model.
All graphs are just $|V|$ changes away from being connected and hence all graphs are $\epsilon$-close to being connected.

So we need some other models for sparse-graph-properties.

# Sparse Graph Model

- The input is a graph with $m$ edges.

# Sparse Graph Model

- The input is a graph with $m$ edges.
- Queries are of the form: What is the $i$th neighbor of vertex $v$?

# Sparse Graph Model

- The input is a graph with $m$ edges.
- Queries are of the form: What is the $i$th neighbor of vertex $v$?
- If the degree of $v$ is less than $i$ then the answer to query is "NONE".

# Sparse Graph Model

- The input is a graph with $m$ edges.
- Queries are of the form: What is the $i$th neighbor of vertex $v$?
- If the degree of $v$ is less than $i$ then the answer to query is "NONE".

## Property Tester for Bounded Degree Model

A 1-sided-error tester for $\mathcal{P}$ is a *randomized* algorithm $\mathcal{A}$ that given query access to a graph $G$ does the following:

- If $G \in \mathcal{P} \Rightarrow \Pr[\mathcal{A} \text{ accepts}] = 1$.
- If at least $\epsilon m$ number of edges has to be added or removed so that the property $\mathcal{P}$ is satisfied then $\Pr[\mathcal{A} \text{ rejects}] \geq 2/3$.

# Testing Connectivity in Sparse Graph Model

**Observation**

*If a graph $G$ is $\epsilon$-far (in the sparse-graph-model) from being connected then it has more than $\epsilon m + 1$ connected components. And thus it must have at least $(\epsilon/2)m$ number of components of size at most $2n/\epsilon m$.*

# Testing Connectivity in Sparse Graph Model

## Observation

*If a graph $G$ is $\epsilon$-far (in the sparse-graph-model) from being connected then it has more than $\epsilon m + 1$ connected components. And thus it must have at least $(\epsilon/2)m$ number of components of size at most $2n/\epsilon m$.*

## Algorithm

- *Randomly pick $4n/\epsilon m$ vertices.*

# Testing Connectivity in Sparse Graph Model

## Observation

*If a graph G is $\epsilon$-far (in the sparse-graph-model) from being connected then it has more than $\epsilon m + 1$ connected components. And thus it must have at least $(\epsilon/2)m$ number of components of size at most $2n/\epsilon m$.*

## Algorithm

- *Randomly pick $4n/\epsilon m$ vertices.*
- *Do a BFS from each of the selected vertices till you find $2n/\epsilon m$ vertices.*

# Testing Connectivity in Sparse Graph Model

## Observation

*If a graph $G$ is $\epsilon$-far (in the sparse-graph-model) from being connected then it has more than $\epsilon m + 1$ connected components. And thus it must have at least $(\epsilon/2)m$ number of components of size at most $2n/\epsilon m$.*

## Algorithm

- *Randomly pick $4n/\epsilon m$ vertices.*
- *Do a BFS from each of the selected vertices till you find $2n/\epsilon m$ vertices.*
- *If you find a component of size less than $2n/\epsilon m$ then REJECT, else ACCEPT.*

# Other problems that has constant query complexity in the sparse graph model

- Cycle-freeness,
- Eulerianess,
- subgraph freeness

All the above has similar algorithms to connectivity testing.

# Testing of *st*-connectedness

**Problem**

*Can we test whether in a graph there is a path from a given vertex $s$ to another given vertex $t$?*

# Testing of *st*-connectedness

**Problem**

*Can we test whether in a graph there is a path from a given vertex $s$ to another given vertex $t$?*

Actually, ... this problem does not make much sense - in the sparse graph model also.

All graphs are just 1 change away from having *st*-connectivity and hence all graphs are $\epsilon$-close to being *st*-connected.

# Testing of *st*-connectedness

**Problem**

*Can we test whether in a graph there is a path from a given vertex $s$ to another given vertex $t$?*

Actually, ... this problem does not make much sense - in the sparse graph model also.
All graphs are just 1 change away from having *st*-connectivity and hence all graphs are $\epsilon$-close to being *st*-connected.

So we need some other models for this.

# Orientation Model

- The input graph is a directed graph.

## Orientation Model

- The input graph is a directed graph.
- The underlying un-directed graph is known in advance.

## Orientation Model

- The input graph is a directed graph.
- The underlying un-directed graph is known in advance.
- Queries are of the form: What is the orientation of the edge $e$?

# Orientation Model

- The input graph is a directed graph.
- The underlying un-directed graph is known in advance.
- Queries are of the form: What is the orientation of the edge $e$?

## Property Tester for Orientation Model

A 1-sided-error tester for $\mathcal{P}$ is a *randomized* algorithm $\mathcal{A}$ that given query access to a graph $G$ does the following:

- If $G \in \mathcal{P} \Rightarrow \Pr[\mathcal{A} \text{ accepts}] = 1$.
- If at least $\epsilon m$ number of edges has to be re-oriented so that the property $\mathcal{P}$ is satisfied then $\Pr[\mathcal{A} \text{ rejects}] \geq 2/3$.

# Testing in the orientation model

## *st*-connectivity [C-Fischer-Lachish-Matsliah-Newman]

There is a 1-sided-error tester that makes $2^{2^{2^{O(1/\epsilon)}}}$ number of queries and tests for *st*-connectivity in the orientation model ($\epsilon$ is the distance parameter).

# Testing in the orientation model

**$st$-connectivity [C-Fischer-Lachish-Matsliah-Newman]**

There is a 1-sided-error tester that makes $2^{2^{2^{O(1/\epsilon)}}}$ number of queries and tests for $st$-connectivity in the orientation model ($\epsilon$ is the distance parameter).

- Other properties like Eulerianness has also been studied in this model. But their query complexity is not constant.
- Not many properties are known to have constant query complexity in the orientation model.
- Even proving that a constant size witness exist is also hard. For example: If $G$ is $\epsilon$ far from being $s$-to-all connected then does there exist a constant size witness?

# Characterization in the Sparse-Graph-Model and Orientation-Model

# Characterization in the Sparse-Graph-Model and Orientation-Model

Characterization of properties that can be tested using constant number of queries in the Sparse-Graph-Model. —— OPEN

# Characterization in the Sparse-Graph-Model and Orientation-Model

Characterization of properties that can be tested using constant number of queries in the Sparse-Graph-Model. —— OPEN

Characterization of properties that can be tested using constant number of queries in the Orientation-Model. —— OPEN

# Outline

# Testing of Graph Isomorphism

Let $H$ be a fixed graph. Then given query access to the adjacency matrix of a graph $G$ test if $G$ is isomorphic to $H$ or if $G$ is $\epsilon$-far from being isomorphic to $H$.

# Testing of Graph Isomorphism

Let $H$ be a fixed graph. Then given query access to the adjacency matrix of a graph $G$ test if $G$ is isomorphic to $H$ or if $G$ is $\epsilon$-far from being isomorphic to $H$.

**Graph Isomorphism Testing [Fischer-Matsliah]**

The 1-sided-query complexity for testing isomorphism to a fixed graph is $\tilde{\Theta}(|V|)$, whereas the 2-sided-error query complexity is $\tilde{\Theta}(\sqrt{|V|})$.

# Testing of Graph Isomorphism

Let $H$ be a fixed graph. Then given query access to the adjacency matrix of a graph $G$ test if $G$ is isomorphic to $H$ or if $G$ is $\epsilon$-far from being isomorphic to $H$.

## Graph Isomorphism Testing [Fischer-Matsliah]

The 1-sided-query complexity for testing isomorphism to a fixed graph is $\tilde{\Theta}(|V|)$, whereas the 2-sided-error query complexity is $\tilde{\Theta}(\sqrt{|V|})$.

## GI Testing with constant number of queries [Fischer]

The query complexity for testing isomorphism to a fixed graph is constant iff the given graph is close to a graph that is generated by a constant number of cliques.

# Hyper-graph isomorphism testing and its generalizations

**Hyper-Graph Isomorphism testing:** Let $H$ be a fixed $d$-refular-hypergraph. Then given query access to the adjacency matrix of a $d$-regular-hypergraph $G$ test if $G$ is isomorphic to $H$ or if $G$ is $\epsilon$-far from being isomorphic to $H$.

# Hyper-graph isomorphism testing and its generalizations

**Hyper-Graph Isomorphism testing:** Let $H$ be a fixed $d$-refular-hypergraph. Then given query access to the adjacency matrix of a $d$-regular-hypergraph $G$ test if $G$ is isomorphic to $H$ or if $G$ is $\epsilon$-far from being isomorphic to $H$.

**Testing Isomorphism under Group Operations:** Let $\mathcal{G}$ be a primitive subgroup of $S_n$. Let $x \in \{0,1\}^n$ be a fixed string. Then given a string $y \in \{0,1\}$ test if $x$ is isomorphic to $y$ under permutation of the indices by elements of the group $\mathcal{G}$, that is, is there a $\pi \in \mathcal{G}$ such that for all $i$, $x_i = y_{\pi(i)}$, OR for all $\pi \in (G)$ for at least $\epsilon n$ indices $i$, $x_i \neq y_{\pi(i)}$.

**Testing Isomorphism under Group Operations:** Let $\mathcal{G}$ be a primitive subgroup of $S_n$. Let $x \in \{0,1\}^n$ be a fixed string. Then given a string $y \in \{0,1\}$ test if $x$ is isomorphic to $y$ under permutation of the indices by elements of the group $\mathcal{G}$, that is, is there a $\pi \in \mathcal{G}$ such that for all $i$, $x_i = y_{\pi(i)}$, OR for all $\pi \in (G)$ for at least $\epsilon n$ indices $i$, $x_i \neq y_{\pi(i)}$.

### Testing Isomorphism under Group Operations [Babai-C]

The query complexity for test isomorphism under primitive group operation is $\tilde{\Theta}(\log|G|)$. This implies the query complexity for testing $d$-regular hypergraph isomorphism is $\tilde{\Theta}(|V|)$.
For 2-sided-error the bounds are $\tilde{\Theta}(\sqrt{\log|G|})$ and $\tilde{\Theta}(\sqrt{|V|})$ respectively.

# Boolean Function Isomorphism Testing

Let $f : \{0,1\}^n \to \{0,1\}$ be a fixed function. Then given query access to the truth-table of a function $g$ test if $g$ is isomorphic to $f$ upto a permutation of its variable, that is, does there exist a permutation $\pi \in S_n$ such that for all $x$, $f(x^\pi) = g(x)$, where $x_i^\pi = x_{\pi(i)}$.

# Boolean Function Isomorphism Testing

Let $f : \{0,1\}^n \to \{0,1\}$ be a fixed function. Then given query access to the truth-table of a function $g$ test if $g$ is isomorphic to $f$ upto a permutation of its variable, that is, does there exist a permutation $\pi \in S_n$ such that for all $x$, $f(x^\pi) = g(x)$, where $x_i^\pi = x_{\pi(i)}$.

For example:

- Is the function $g$ a dictator function? $--$ Constant query complexity.
- Is the function a parity on $k$ variable? $--$ Query complexity $O(k \log k)$ and $\Omega(k)$
- Is the function isomorphic to Majority? $--$ Constant Query Complexity.

# Boolean Function Isomorphism Testing

Let $f : \{0,1\}^n \to \{0,1\}$ be a fixed function. Then given query access to the truth-table of a function $g$ test if $g$ is isomorphic to $f$ upto a permutation of its variable, that is, does there exist a permutation $\pi \in S_n$ such that for all $x$, $f(x^\pi) = g(x)$, where $x_i^\pi = x_{\pi(i)}$.

**Boolean FI testing [Alon-Blais, C-Garcia-Soriano-Matsliah]**

The 1-sided-error query complexity for testing isomorphism to a $k$-junta is $\Theta(k \log n)$ where as the 2-sided-error query complexity is $O(k \log k)$.

# Characterization?

Just like in case of graph isomorphism Fischer proved that query complexity is constant iff the graph is generated by a constant number of cliques, can we say something like that for function isomorphism.

# Characterization?

Just like in case of graph isomorphism Fischer proved that query complexity is constant iff the graph is generated by a constant number of cliques, can we say something like that for function isomorphism.

We know isomorphism to $k$-junta takes only $k \log k$ queries. Also isomorphism to a symmetric function takes constant number of queries. Can we combine to say something like -

# Characterization?

Just like in case of graph isomorphism Fischer proved that query complexity is constant iff the graph is generated by a constant number of cliques, can we say something like that for function isomorphism.

We know isomorphism to $k$-junta takes only $k \log k$ queries. Also isomorphism to a symmetric function takes constant number of queries. Can we combine to say something like -

**Conjecture**

*If $f(x)$ depends on $|X|$ and at most $k$ indices then the query complexity for testing isomorphism to $f$ is $O(k \log k)$ and $\Omega(\log k)$.*

# Outline

# Conclusion

# Conclusion

- Function Property Testing
  - Linearity, low degree, constant-width-read-once-BP, $k$-juntas have constant query complexity
  - Monotonicity - query complexity is $\Omega(n)$ and $O(n^2)$
  - Testing distribution - uniformity testing has query complexity $\tilde{\Theta}(\sqrt{|Range|})$.

# Conclusion

- Function Property Testing
  - Linearity, low degree, constant-width-read-once-BP, $k$-juntas have constant query complexity
  - Monotonicity - query complexity is $\Omega(n)$ and $O(n^2)$
  - Testing distribution - uniformity testing has query complexity $\tilde{\Theta}(\sqrt{|Range|})$.
- Graph Property Testing
  - $k$-colorability in dense graph model is testable with $O(k)$ queries,
  - Dense Graph Model - Testing is all about regularity,
  - Sparse Graph Model - testing of connectivity
  - Orientation Model - testing of $s$-connectivity

# Conclusion

- Function Property Testing
  - Linearity, low degree, constant-width-read-once-BP, $k$-juntas have constant query complexity
  - Monotonicity - query complexity is $\Omega(n)$ and $O(n^2)$
  - Testing distribution - uniformity testing has query complexity $\tilde{\Theta}(\sqrt{|Range|})$.
- Graph Property Testing
  - $k$-colorability in dense graph model is testable with $O(k)$ queries,
  - Dense Graph Model - Testing is all about regularity,
  - Sparse Graph Model - testing of connectivity
  - Orientation Model - testing of $s$-connectivity
- Isomorphism Testing
  - Generalization of GI testing
  - Isomorphism to $k$-junta can be tested with $O(k \log k)$ queries.

# Future Work

- Characterizations of testable properties under various models.
- Reduce the query complexity for the properties testable with constant number of queries.
- Connections to other areas of research like coding theory, self-correction, Additive Combinatorics, Cryptography, Learning Theory, and (possibly Kernelization?).

# Surveys and Books

- Property Testing (Current Research and Surveys), by Oded Goldreich.
- Algorithmic and Analysis Techniques in Property Testing, by Dana Ron.
- Sublinear-time Algorithms, by Ronitt Rubinfeld.