Sublinear Algorithms

Lectures 1 and 2

Sofya Raskhodnikova

Penn State University

Tentative Topics

Introduction, examples and general techniques.

Sublinear-time algorithms for

- graphs
- strings
- basic properties of functions
- algebraic properties and codes
- metric spaces
- distributions

  Tools: probability, Fourier analysis, combinatorics, codes, …

Sublinear-space algorithms: streaming

Tentative Plan

Introduction, examples and general techniques.

Lecture 1. Background. Testing properties of images and lists.

Lecture 2. Properties of functions and graphs. Sublinear approximation.

Lecture 3-5. Background in probability. Techniques for proving hardness. Other models for sublinear computation.

Motivation for Sublinear-Time Algorithms

Massive datasets

- world-wide web
- online social networks
- genome project
- sales logs
- census data
- high-resolution images
- scientific measurements

Long access time

- communication bottleneck (dial-up connection)
- implicit data (an experiment per data point)

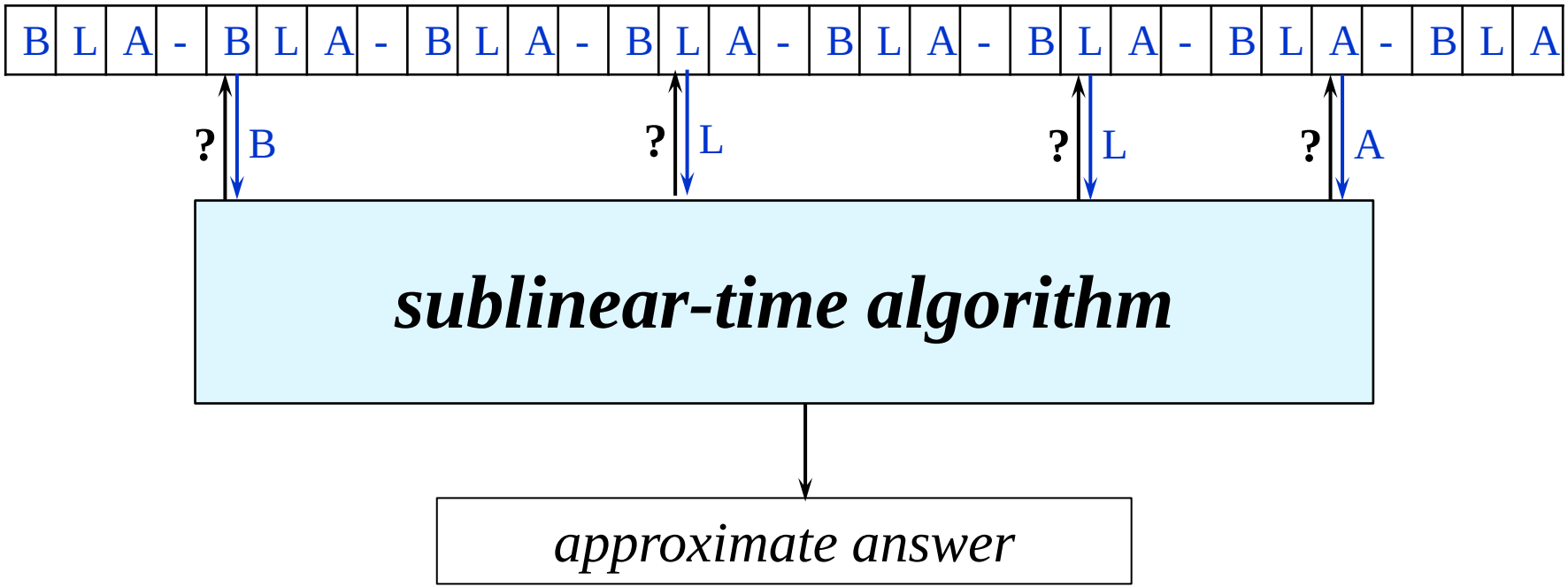
What Can We Hope For?

- What can an algorithm compute if it
  - reads only a **sublinear** portion of the data?
  - runs in **sublinear** time?

- Some problems have exact deterministic solutions

- For most interesting problems algorithms must be
  - approximate
  - randomized

A Sublinear-Time Algorithm



Quality of approximation

vs.

Resources

- number of samples
- running time

Types of Approximation

Classical approximation

- need to compute a value
  - output is close to the desired value
  - examples: average, median values
- need to compute the best structure
  - output is a structure with "cost" close to optimal
  - examples: furthest pair of points, minimum spanning tree

Property testing

- need to answer YES or NO
  - output is a correct answer for a given input,
    or at least some input close to it

Classical Approximation

A Simple Example

Approximate Diameter of a Point Set [Indyk]

Input: m points, described by a distance matrix D

- D_{ij} is the distance between points i and j
- D satisfies triangle inequality and symmetry

(Note: input size is $n = m^2$)

Let i, j be indices that maximize D_{ij}.

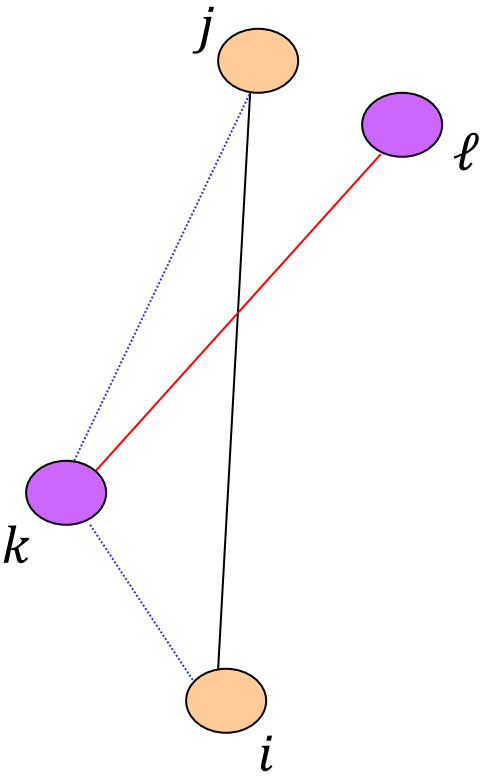
Maximum D_{ij} is the *diameter*.

- Output: (k, ℓ) such that $D_{k\ell} \geq D_{ij}/2$

Algorithm and Analysis

Algorithm (m, D)

1. Pick k arbitrarily
2. Pick ℓ to maximize $D_{k\ell}$
3. Output (k, ℓ)



- Approximation guarantee

$$D_{ij} \le D_{ik} + D_{kj} \text{ (triangle inequality)}$$
$$\le D_{k\ell} + D_{k\ell} \text{ (choice of } \ell \text{ + symmetry of } D)$$
$$\le 2Dk_{\ell}$$

- Running time: $O(m) = O(m = \sqrt{n})$

A rare example of a deterministic sublinear-time algorithm

Property Testing

Property Testing: YES/NO Questions

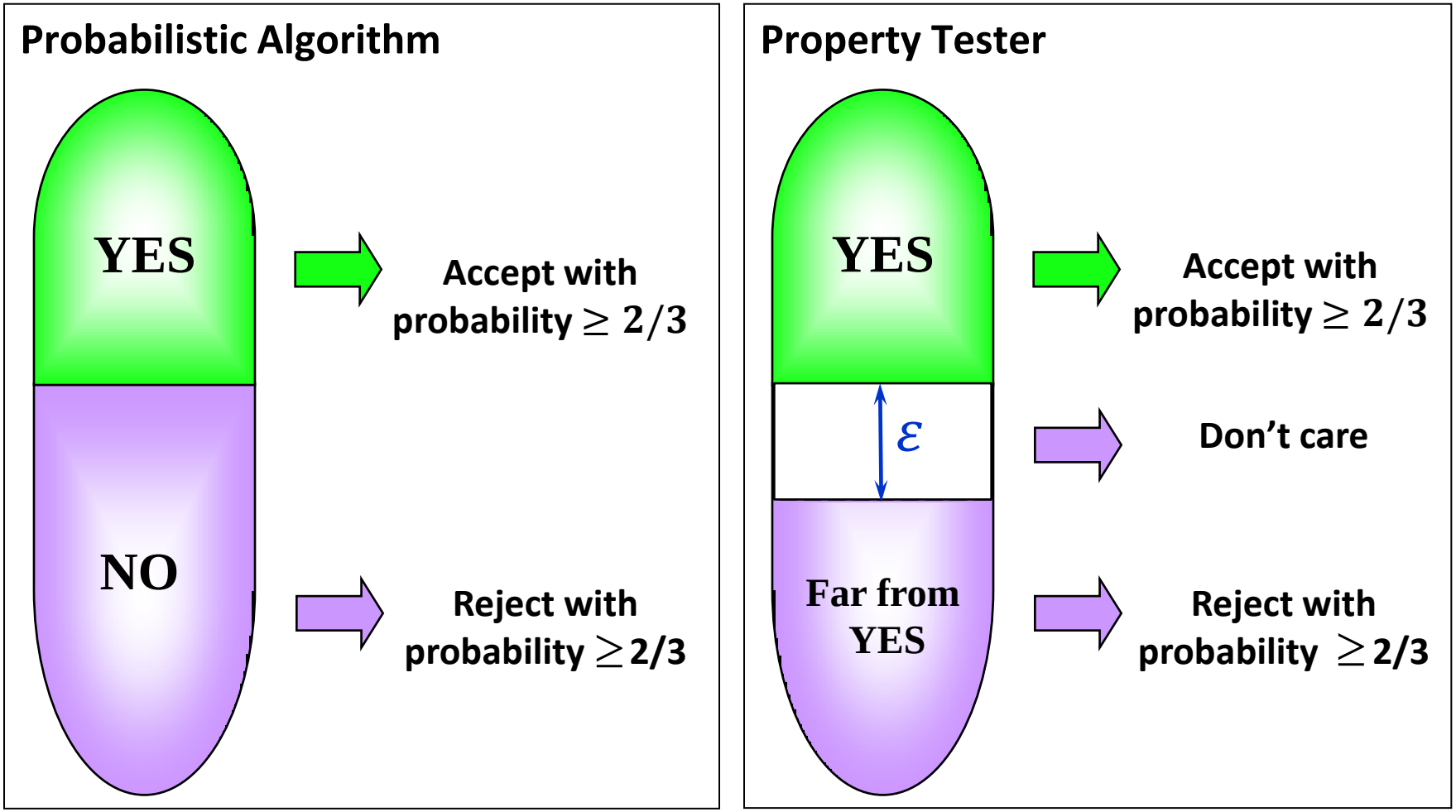
Does the input satisfy some property? (YES/NO)

"in the ballpark" vs. "out of the ballpark"

Does the input satisfy the property or is it far from satisfying it?

- sometimes it is the right question (probabilistically checkable proofs (PCPs))
- as good when the data is constantly changing (WWW)
- fast sanity check to rule out inappropriate inputs (airport security questioning)

Property Tester Definition



ε-far = differs in many places ($\geq \varepsilon$ fraction of places)

Randomized Sublinear Algorithms

Toy Examples

Property Testing: a Toy Example

0	0	0	1	…	0	1	0	0

Input: a string $w \in \{0,1\}^n$

Question: Is $w = 00 \ldots 0$?

Requires reading entire input.

Approximate version: Is $w = 00 \ldots 0$ or

does it have $\geq \varepsilon n$ 1's ("errors")?

Test (n, w)

1. Sample $s = 2/\varepsilon$ positions uniformly and independently at random
2. If 1 is found, **reject**; otherwise, **accept**

Analysis: If $w = 00 \ldots 0$, it is always accepted.

Used: $1 - x \leq e^{-x}$

If w is ε-far, Pr[error] = Pr[no 1's in the sample] $\leq (1-\varepsilon)^s \leq e^{-\varepsilon s} = e^{-2} < \frac{1}{3}$

Witness Lemma

If a test catches a witness with probability $\geq p$,

then $s = \frac{2}{p}$ iterations of the test catch a witness with probability $\geq 2/3$.

Randomized Approximation: a Toy Example

Input: a string $w \in \{0,1\}^n$

0	0	0	1	…	0	1	0	0

Goal: Estimate the fraction of 1's in w (like in polls)

It suffices to sample $s = 1 / \varepsilon^2$ positions and output the average
to get the fraction of 1's $\pm\varepsilon$ (i.e., additive error ε) with probability $\geq$ 2/3

Hoeffding Bound

Let $Y_1, \ldots, Y_s$ be independently distributed random variables in [0,1] and let $Y = \sum_{i=1}^{s} Y_i$ (sample sum). Then $\Pr[|Y - E[Y]| \geq \delta] \leq 2e^{-2\delta^2/s}$.

Y_i = value of sample i. Then $E[Y] = \sum_{i=1}^{s} E[Y_i] = s \cdot$ (fraction of 1's in w)

$$\Pr[|(\text{sample average}) - (\text{fraction of 1's in } w)| \geq \varepsilon] = \Pr[|Y - E[Y]| \geq \varepsilon s]$$
$$\leq 2e^{-2\delta^2/s} = 2e^{-2} < 1/3$$

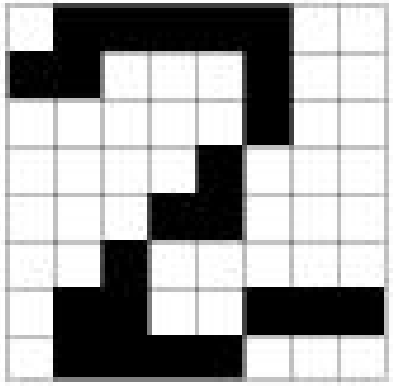
Apply Hoeffding Bound with $\delta = \varepsilon s$

substitute $s = 1 / \varepsilon^2$

Property Testing

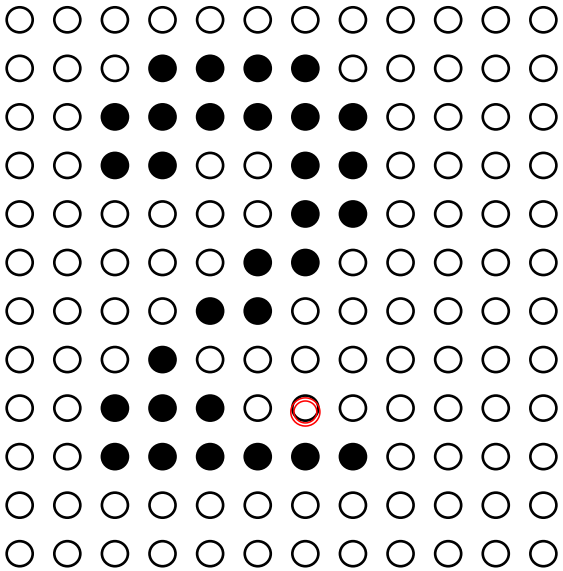
Simple Examples

Testing Properties of Images



Pixel Model

Input: $n \times n$ matrix of pixels
(0/1 values for black-and-white pictures)



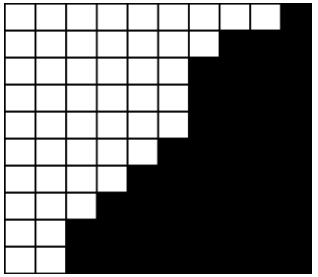
Query: point (i_1, i_2)

Answer: color of (i_1, i_2)

Testing if an Image is a Half-plane [R03]

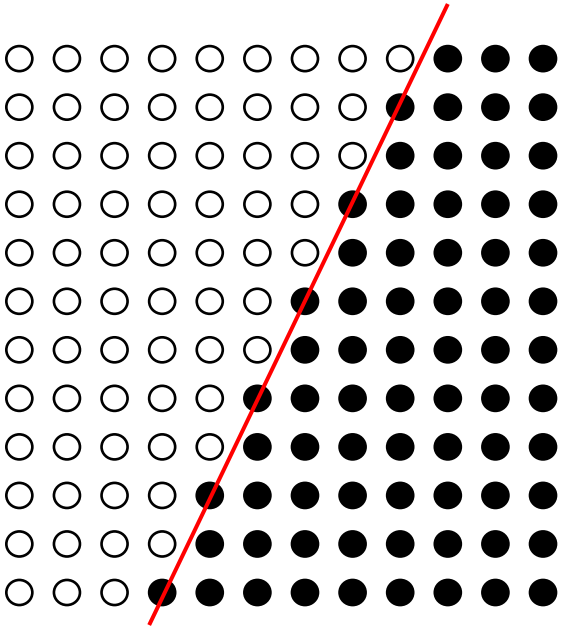
A half-plane or

ε-far from a half-plane?

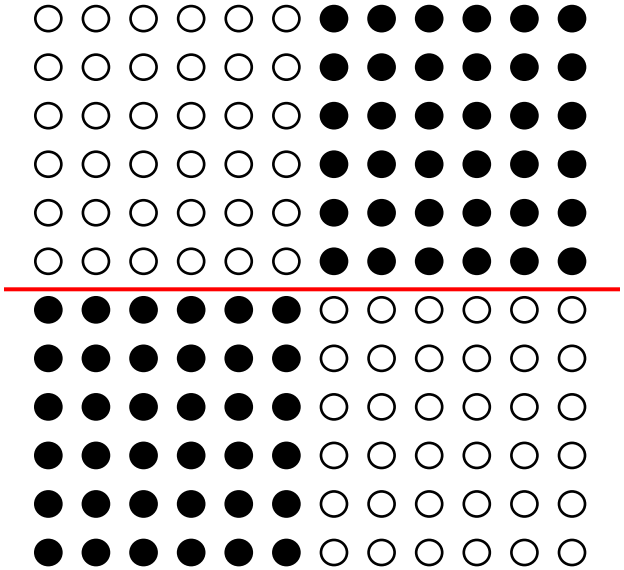


O(1/ε) time

Half-plane Instances

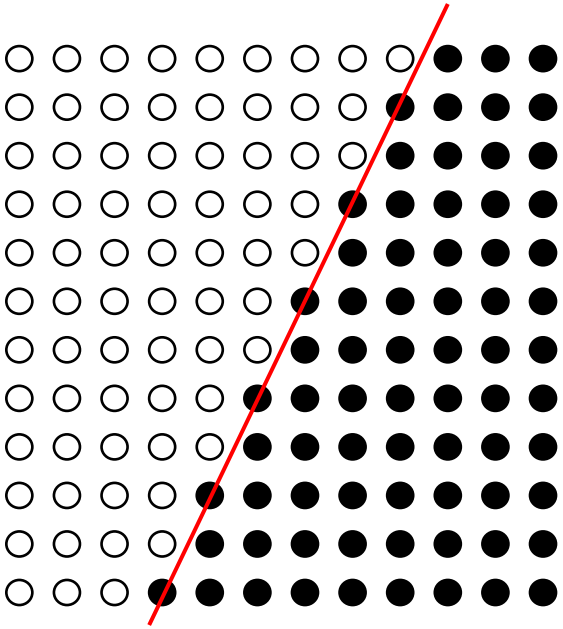


A half-plane

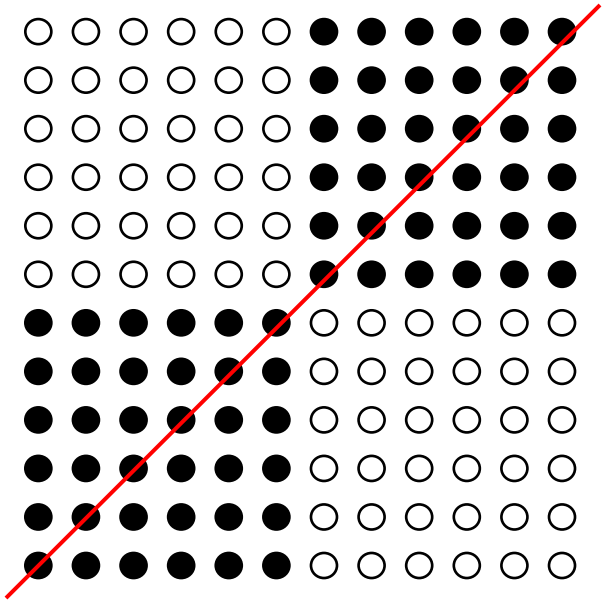


$\frac{1}{4}$-far from a half-plane

Half-plane Instances

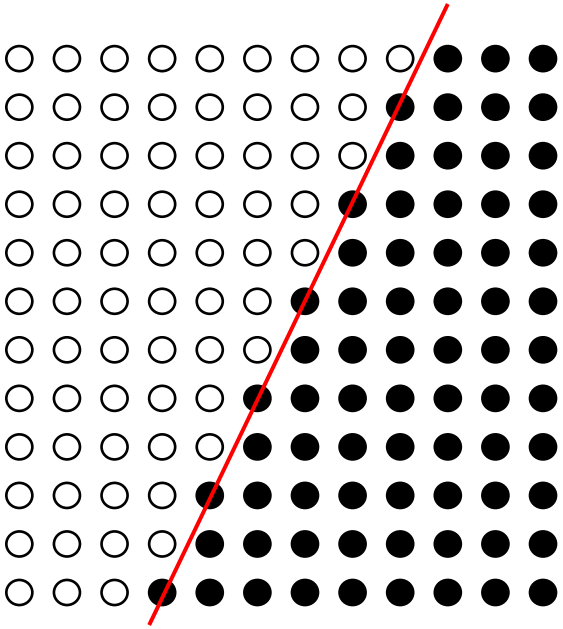


A half-plane

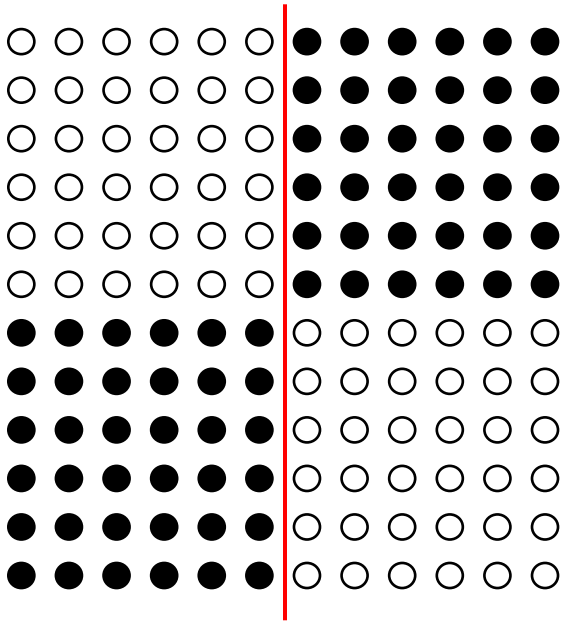


$\frac{1}{4}$-far from a half-plane

Half-plane Instances

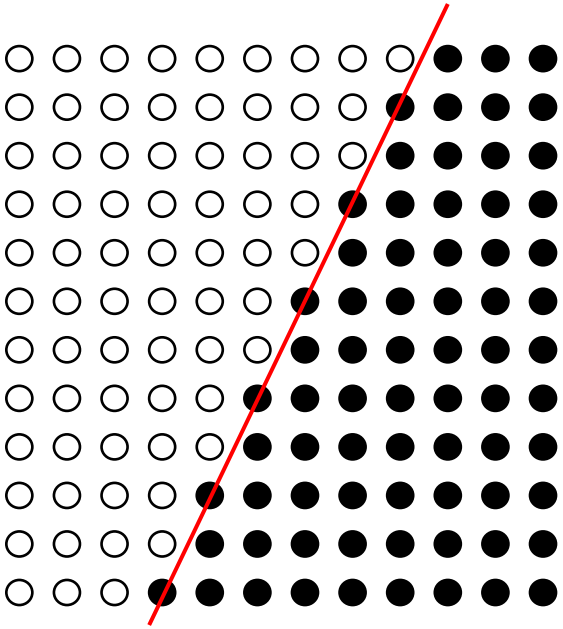


A half-plane

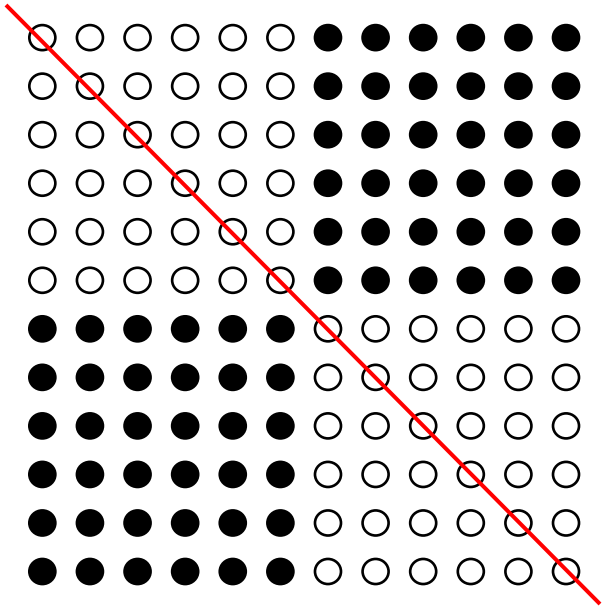


$\frac{1}{4}$-far from a half-plane

Half-plane Instances

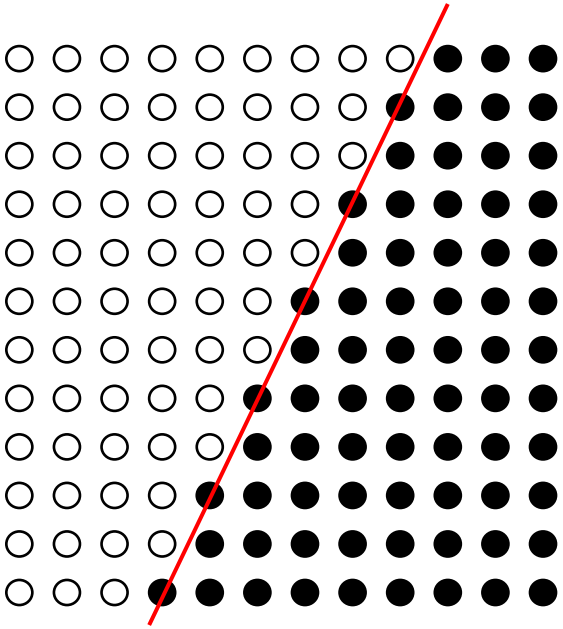


A half-plane

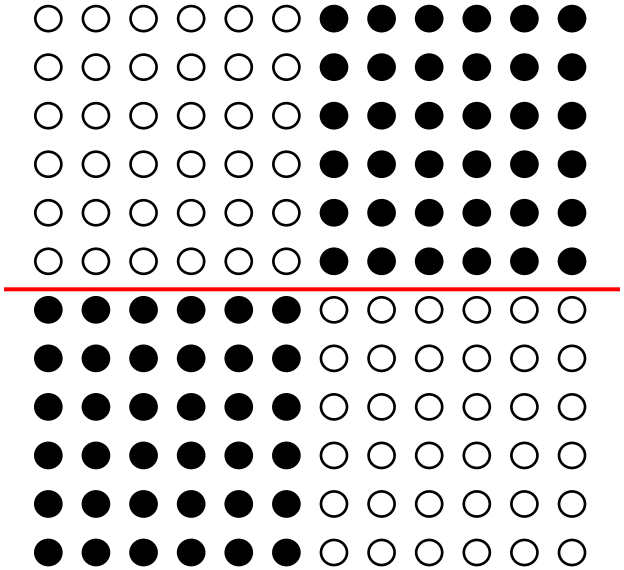


$\frac{1}{4}$-far from a half-plane

Half-plane Instances

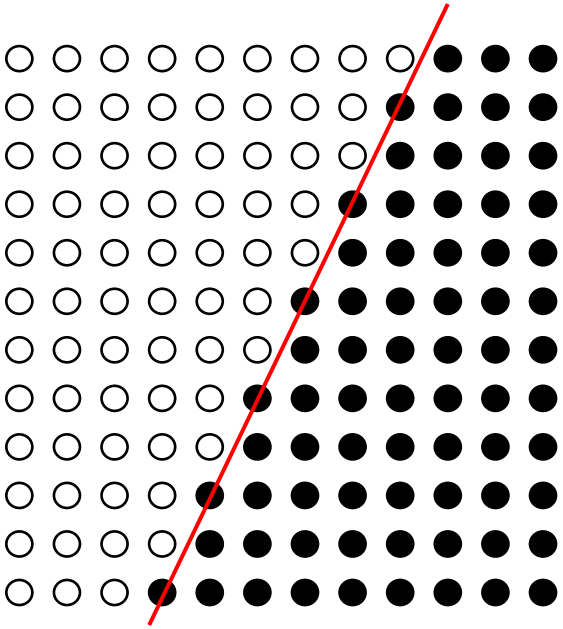


A half-plane

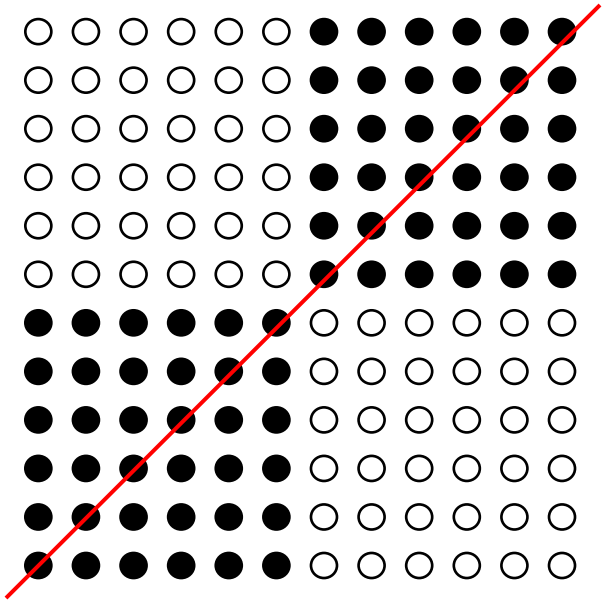


$\frac{1}{4}$-far from a half-plane

Half-plane Instances

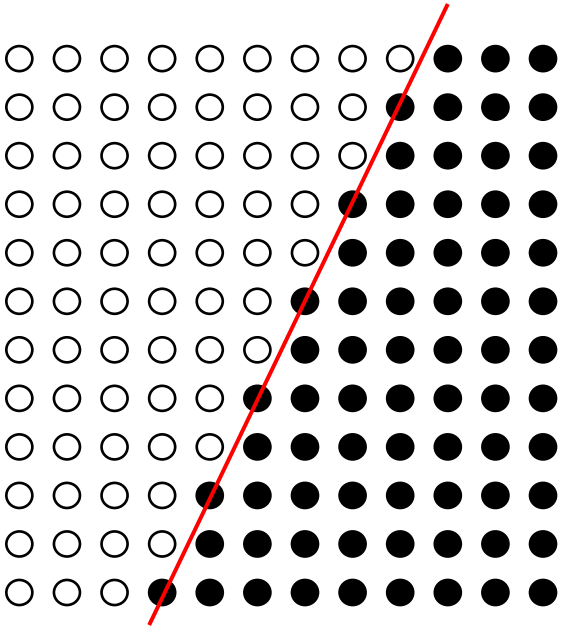


A half-plane

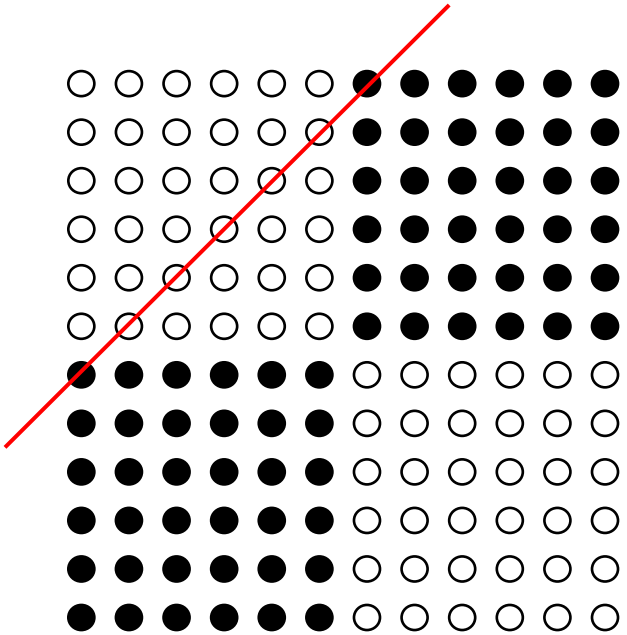


$\frac{1}{4}$-far from a half-plane

Half-plane Instances



A half-plane



$\frac{1}{4}$-far from a half-plane

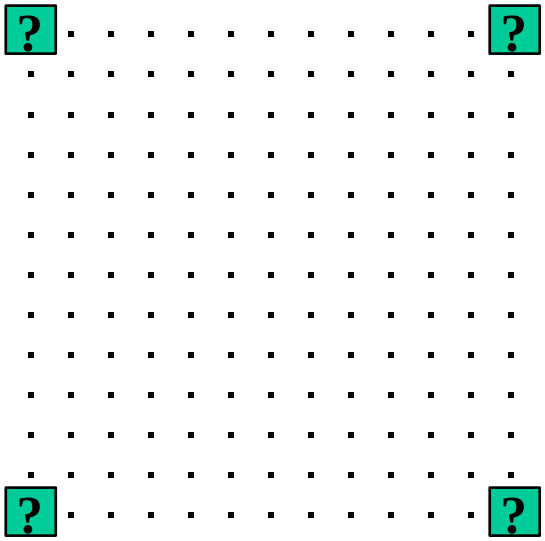
Strategy

"Testing by implicit learning" paradigm

- Learn the outline of the image by querying a few pixels.
- Test if the image conforms to the outline by random sampling, and reject if something is wrong.

Half-plane Test

Claim. The number of sides with different corners is 0, 2, or 4.

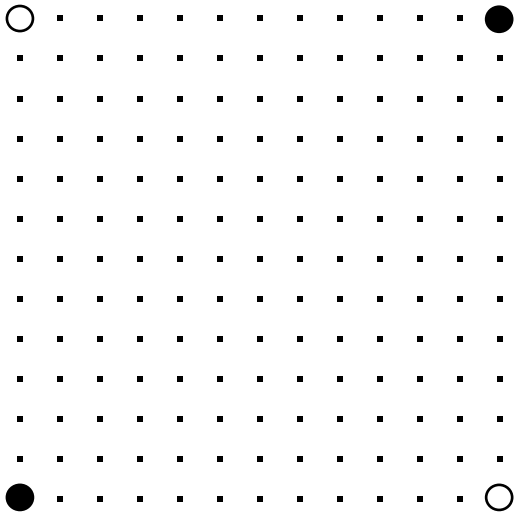


Algorithm

1. Query the corners.

Half-plane Test: 4 Bi-colored Sides

Claim. The number of sides with different corners is 0, 2, or 4.



Analysis

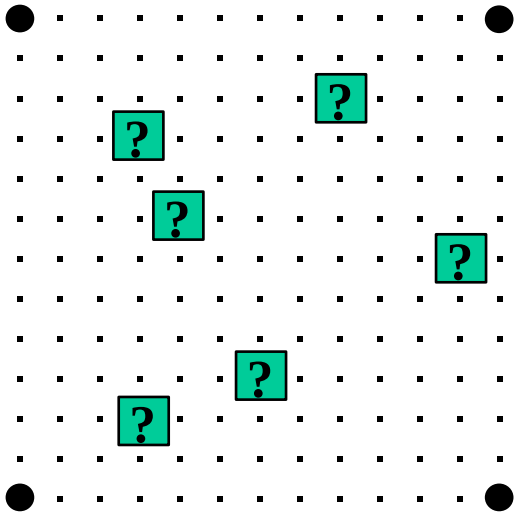
- If it is 4, the image cannot be a half-plane.

Algorithm

1. Query the corners.
2. If the number of sides with different corners is 4, **reject**.

Half-plane Test: 0 Bi-colored Sides

Claim. The number of sides with different corners is 0, 2, or 4.



Analysis

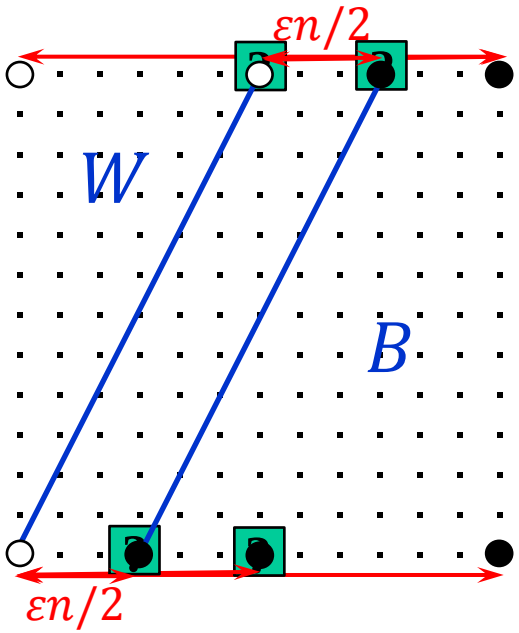
- If all corners have the same color, the image is a half-plane if and only if it is unicolored.

Algorithm

1. Query the corners.
2. If all corners have the same color c, test if all pixels have color c (as in Toy Example 1).

Half-plane Test: 2 Bi-colored Sides

Claim. The number of sides with different corners is 0, 2, or 4.



Analysis

- The area outside of $W \cup B$ has $\leq \varepsilon n^2/2$ pixels.
- If the image is a half-plane, W contains only white pixels and B contains only black pixels.
- If the image is ε-far from half-planes, it has $\geq \varepsilon n^2/2$ wrong pixels in $W \cup B$.
- By Witness Lemma, $4/\varepsilon$ samples suffice to catch a wrong pixel.

Algorithm

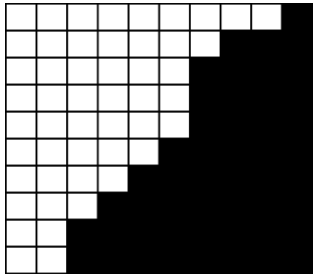
1. Query the corners.
2. If # of sides with different corners is 2, on both sides find 2 different pixels within distance $\varepsilon n/2$ by binary search.
3. Query $4/\varepsilon$ pixels from $W \cup B$
4. **Accept** iff all W pixels are white and all B pixels are black.

Testing if an Image is a Half-plane [R03]

A half-plane or

ε-far from a half-plane?

O(1/ε) time ✓



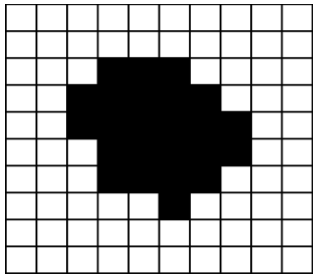
Other Results on Properties of Images

- Pixel Model

  Convexity [R03]

  Convex or ε-far from convex?

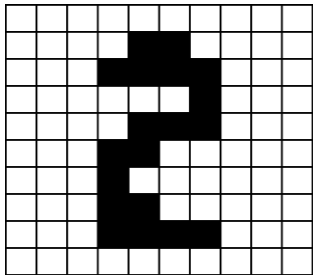
  $O(1/\varepsilon^2)$ time



  Connectedness [R03]

  Connected or ε-far from connected?

  $O(1/\varepsilon^4)$ time

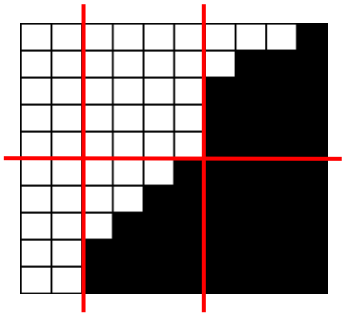


  Partitioning [Kleiner Keren Newman 10]

  Can be partitioned according to a template

  or is ε-far?

  time independent of image size



- Properties of sparse images [Ron Tsur 10]

Testing if a List is Sorted

Input: a list of n numbers $x_1, x_2, \ldots, x_n$

- Question: Is the list sorted?

  Requires reading entire list: $\Omega(n)$ time

- Approximate version: Is the list sorted or ϵ-far from sorted?

  (An ϵ fraction of x_i's have to be changed to make it sorted.)

  [Ergün Kannan Kumar Rubinfeld Viswanathan 98, Fischer 01]: $O((\log n)/\epsilon)$ time

  $\Omega(\log n)$ queries

- Attempts:

  1. Test: Pick a random i and reject if $x_i > x_{i+1}$.

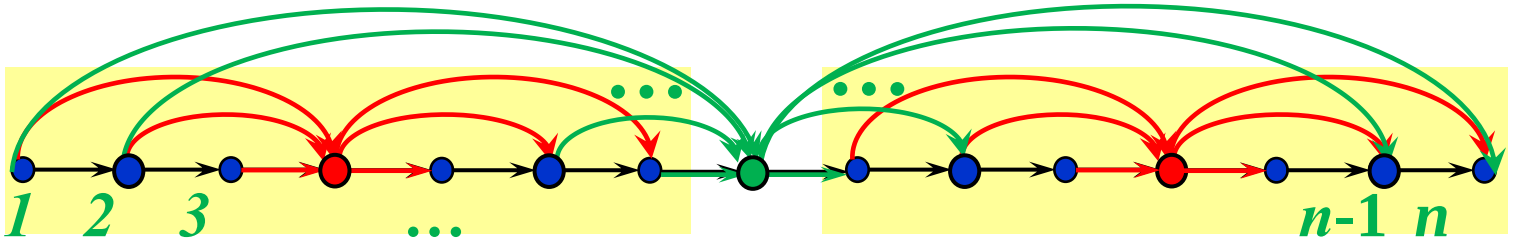
     Fails on: 1 1 1 1 1 1 1 0 0 0 0 0 0 0 ← 1/2-far from sorted

  2. Test: Pick random $i < j$ and reject if $x_i > x_j$.

     Fails on: 1 0 2 1 3 2 4 3 5 4 6 5 7 6 ← 1/2-far from sorted

Is a list sorted or ϵ-far from sorted?

Idea: Associate positions in the list with vertices of the directed line.



Construct a graph (2-spanner)

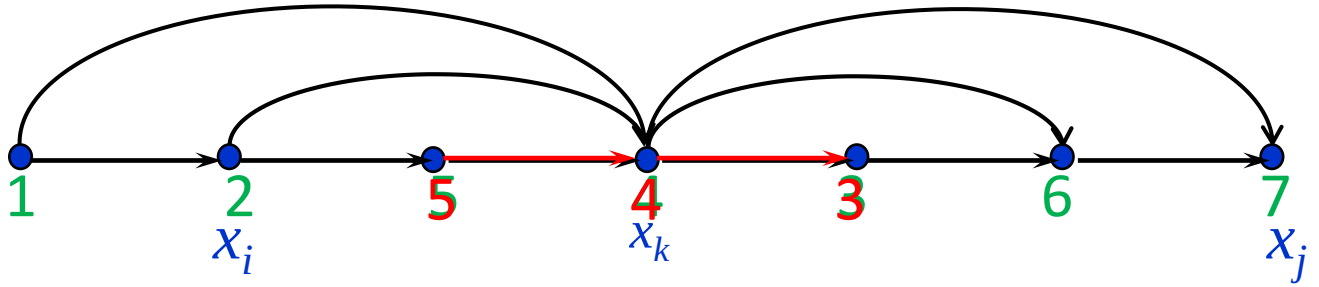
$\leq n \log n$ edges

- by adding a few "shortcut" edges (i, j) for $i < j$
- where each pair of vertices is connected by a path of length at most 2

Is a list sorted or ϵ-far from sorted?

Test [Dodis Goldreich Lehman Raskhodnikova Ron Samorodnitsky 99]

Pick a random edge (x_i,x_j) from the 2-spanner and **reject** if $x_i > x_j$.



Analysis:

- Call an edge (x_i,x_j) **violated** if $x_i > x_j$, and **good** otherwise.
- If x_i is an endpoint of a **violated** edge, call it **bad**. Otherwise, call it **good**.

Claim 1. All **good** numbers x_i are sorted.

Proof: Consider any two good numbers, x_i and x_j.

They are connected by a path of (at most) two **good** edges (x_i,x_k), (x_k,x_j).

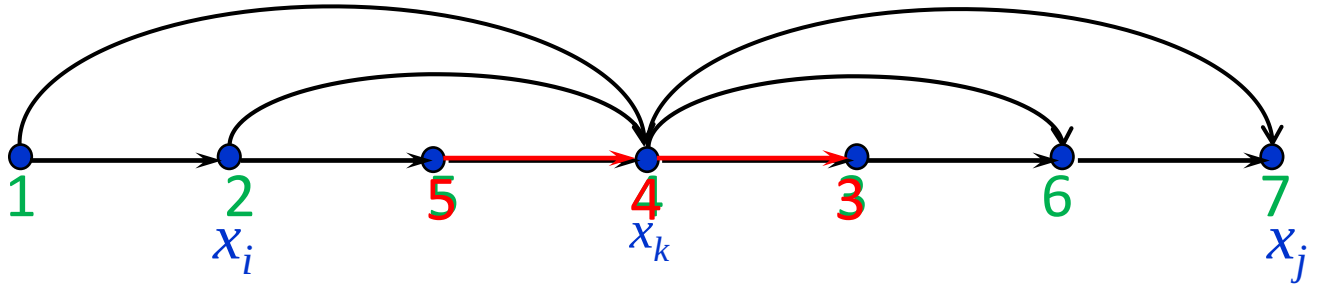
$\Rightarrow$ $x_i \le x_k$ and $x_k \le x_j$

$\Rightarrow$ $x_i \le x_j$

Is a list sorted or ϵ-far from sorted?

Test [Dodis Goldreich Lehman Raskhodnikova Ron Samorodnitsky 99]

Pick a random edge (x_i,x_j) from the 2-spanner and **reject** if $x_i > x_j$.



Analysis:

- Call an edge (x_i,x_j) **violated** if $x_i > x_j$, and **good** otherwise.
- If x_i is an endpoint of a **bad** edge, call it **bad**. Otherwise, call it **good**.

Claim 1. All **good** numbers x_i are sorted.

Claim 2. An ϵ-far list **violates** $\geq \epsilon/(2 \log n)$ fraction of edges in 2-spanner.

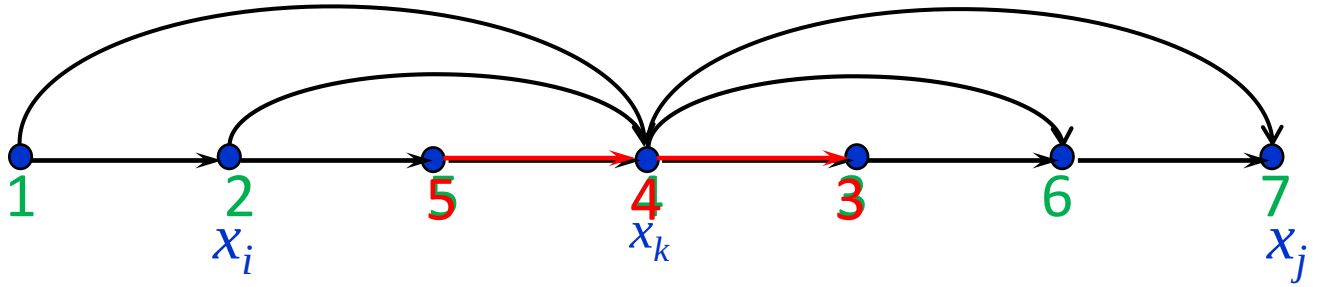
Proof: If a list is ϵ-far from sorted, it has $\geq \epsilon n$ **bad** numbers. (Claim 1)

- Each **violated** edge contributes 2 **bad** numbers.
- 2-spanner has $\geq \epsilon n/2$ **violated** edges out of $\leq n \log n$.

Is a list sorted or ϵ-far from sorted?

Test [Dodis Goldreich Lehman Raskhodnikova Ron Samorodnitsky 99]

Pick a random edge (x_i,x_j) from the 2-spanner and **reject** if $x_i > x_j$.



Analysis:

- Call an edge (x_i,x_j) **violated** if $x_i > x_j$, and **good** otherwise.

Claim 2. An ϵ-far list **violates** $\geq \epsilon/(2 \log n)$ fraction of edges in 2-spanner.

By Witness Lemma, it suffices to sample $(4 \log n)/\epsilon$ edges from 2-spanner.

Algorithm

Sample $(4 \log n)/\epsilon$ edges (x_i,x_j) from the 2-spanner and **reject** if $x_i > x_j$.

Guarantee: All sorted lists are accepted. ✓

All lists that are ϵ-far from sorted are rejected with probability $\geq 2/3$. ✓

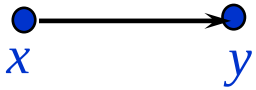
Time: $O((\log n)/\epsilon)$ ✓

Generalization

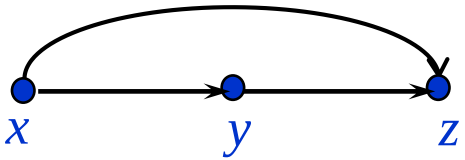
Observation: 

The same test/analysis apply to any edge-transitive property of a list of numbers that allows extension.

- A property is edge-transitive if
  1) it can be expressed in terms conditions on ordered pairs of numbers



  2) it is transitive: whenever (x, y) and (y, z) satisfy (1), so does (x, z)



- A property allows extension if
  3) any function that satisfies (1) on a subset of the numbers can be extended to a function with the property

Lipschitz Continuous Functions

A function $f : D \to R$ has Lipschitz constant c
if for all x,y in D,
$distance_R(f(x),f(y)) \leq c \cdot distance_D(x,y)$.

A fundamental notion in

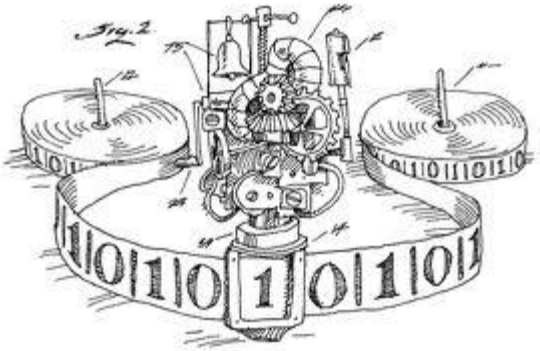
- *mathematical analysis*
- *theory of differential equations*

Example uses of a Lipschitz constant c of a given function f

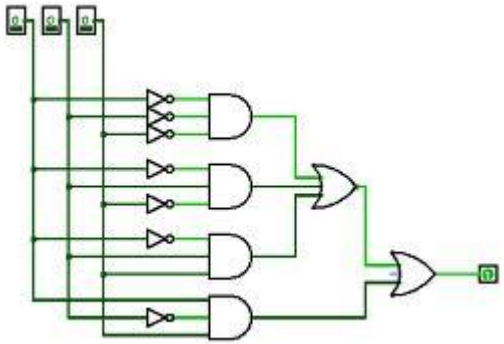
- probability theory: in tail bounds via McDiarmid's inequality
- program analysis: as a measure of robustness to noise
- data privacy: to scale noise added to preserve differential privacy

Computing a Lipschitz Constant?

- Infeasible



- Undecidable to even verify if f computed by a TM has Lipschitz constant c
- NP-hard to verify if f computed by a circuit has Lipschitz constant c
  - even for finite domains



Question: Can we test if a function has Lipschitz constant c or is ε-far from any such function?

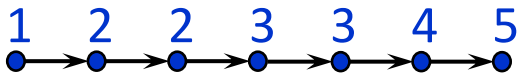
Image sources: http://www.ecs.syr.edu/faculty/fawcett/handouts/webpages/coretechnologies.htm
http://www.augustana.ab.ca/~mohrj/courses/2004.fall/csc110/assignments/lab2.html

Testing if a Function is Lipschitz [Jha R]

A function $f : D \to R$ is Lipschitz if it has Lipschitz constant c:
that is, if for all x,y in D,
$distance_R(f(x),f(y)) \leq distance_D(x,y)$.

- can rescale by $1/c$ to get a Lipschitz function from a function with Lipschitz constant c

Consider $f : \{1,\ldots,n\} \to \mathbb{R}$:



nodes = points in the domain; edges = points at distance 1

node labels = values of the function

The Lipschitz property is *edge-transitive*:

1. a pair (x,y) is *good* if $|f(y)-f(x)| \leq |y-x|$
2. (x,y) and (y,z) are *good* $\Rightarrow$ (x,z) is *good*

It also allows extension for the range $\mathbb{R}$.

Testing if a function $f : \{1,\ldots,n\} \to \mathbb{R}$ is Lipschitz takes $O((\log n)/\epsilon)$ time.

Does the spanner-based test apply if the range is $\mathbb{R}^2$ with Euclidean distances? $\mathbb{Z}^2$ with Euclidean distances?

Properties of a List of n Numbers

- Sorted or ε-far from sorted?

- Lipschitz (does not change too drastically)
  or ε-far from satisfying the Lipschitz property?

O(log n/ε) time 

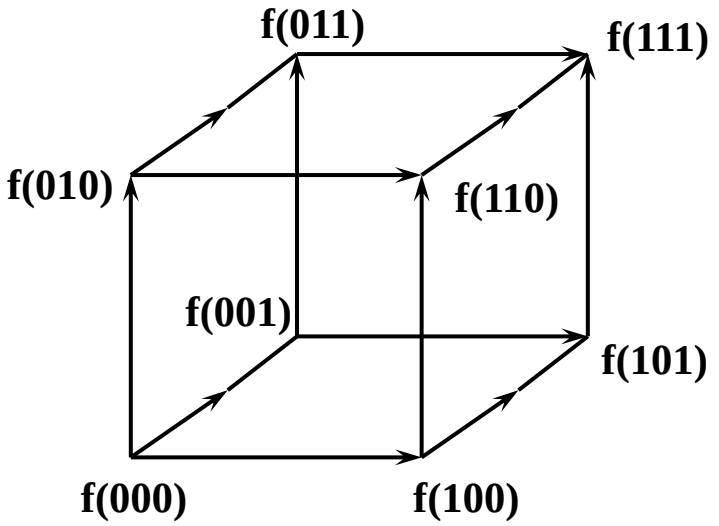
Open: can it be improved?

Basic Properties of Functions

Boolean Functions $f : \{0,1\}^n \rightarrow \{0,1\}$

Graph representation:

n-dimensional hypercube



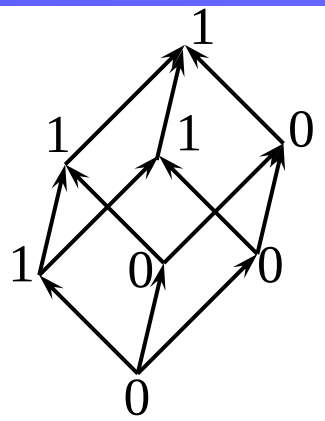
- vertices: bit strings of length n
- edges: (x, y) is an edge if y can be obtained from x by increasing one bit from 0 to 1

x	001001
y	011001

- each vertex x is labeled with $f(x)$

Monotonicity of Functions

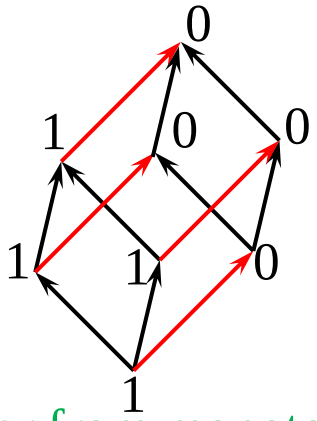
[Goldreich Goldwasser Lehman Ron Samorodnitsky,
Dodis Goldreich Lehman Raskhodnikova Ron Samorodnitsky]



monotone

- A function $f : \{0,1\}^n \to \{0,1\}$ is monotone
  if increasing a bit of x does not decrease $f(x)$.

- Is f monotone or ε-far from monotone
  (f has to change on many points to become monontone)?
  - Edge $x \to y$ is violated by f if $f(x) > f(y)$.



$\frac{1}{2}$-far from monotone

Time:

- $O(n/\varepsilon)$, logarithmic in the size of the input, 2^n
- $\Omega(\sqrt{n}/\varepsilon)$ for restricted class of tests

Monotonicity Test [GGLRS, DGLRRS]

Idea: Show that functions that are far from monotone violate many edges.

EdgeTest (f, ε)

1. Pick $2n/\varepsilon$ edges (x, y) uniformly at random from the hypercube.
2. **Reject** if some (x, y) is violated (i.e. $f(x) > f(y)$). Otherwise, **accept**.

Analysis

- If f is monotone, EdgeTest always accepts.
- If f is ε-far from monotone, by Witness Lemma, it suffices to show that $\geq \varepsilon/n$ fraction of edges (i.e., $\frac{\varepsilon}{n} \cdot 2^{n-1}n = \varepsilon 2^{n-1}$ edges) are violated by f.
  - Let $V(f)$ denote the number of edges violated by f.

  Contrapositive: If $V(f) < \varepsilon\, 2^{n-1}$,
  f can be made monotone by changing $< \varepsilon\, 2^n$ values.

Repair Lemma

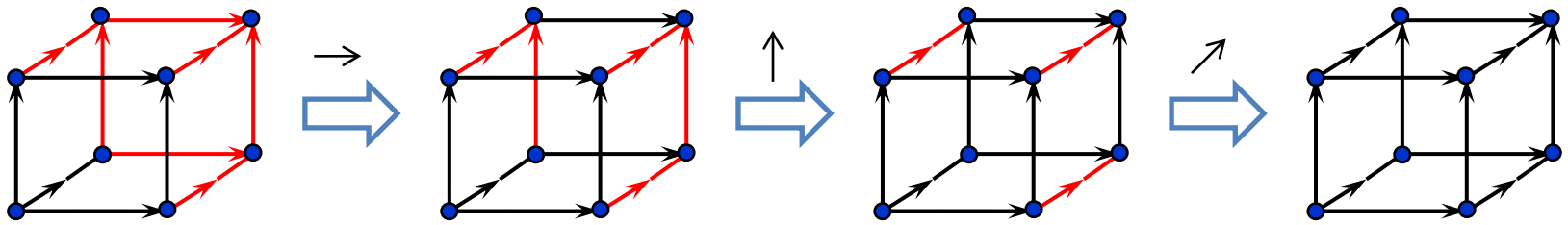
f can be made monotone by changing $\leq 2 \cdot V(f)$ values.

Repair Lemma: Proof Idea

Repair Lemma

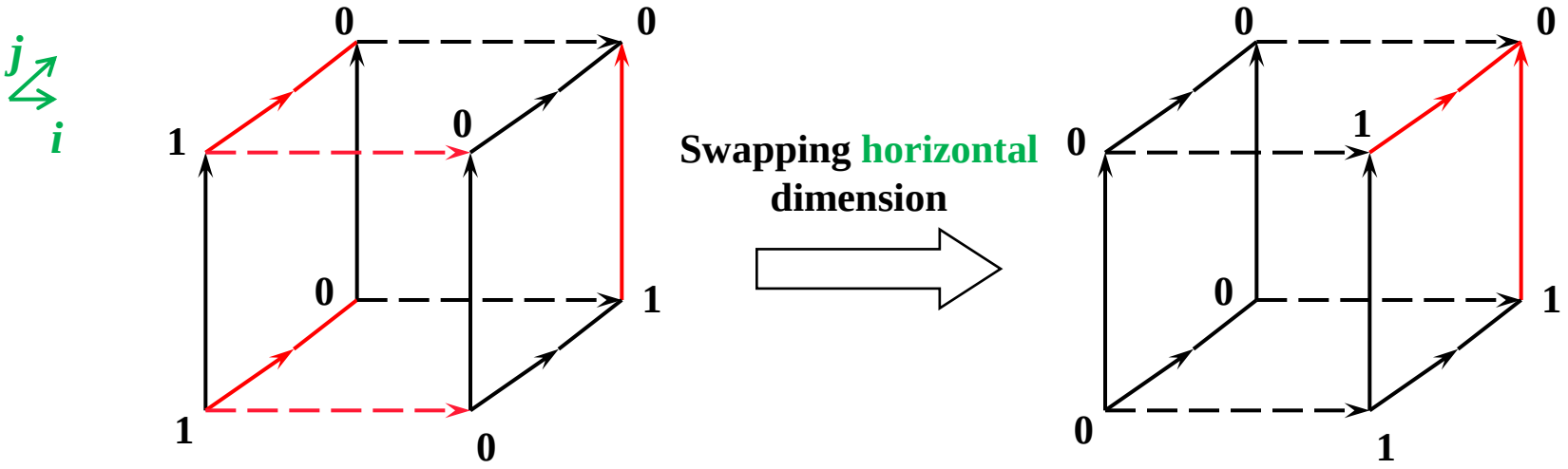
f can be made monotone by changing $\leq 2 \cdot V(f)$ values.

Proof idea: Transform *f* into a monotone function by repairing edges in one dimension at a time.



Repairing Violated Edges in One Dimension

Swap violated edges $1 \to 0$ in one dimension to $0 \to 1$.



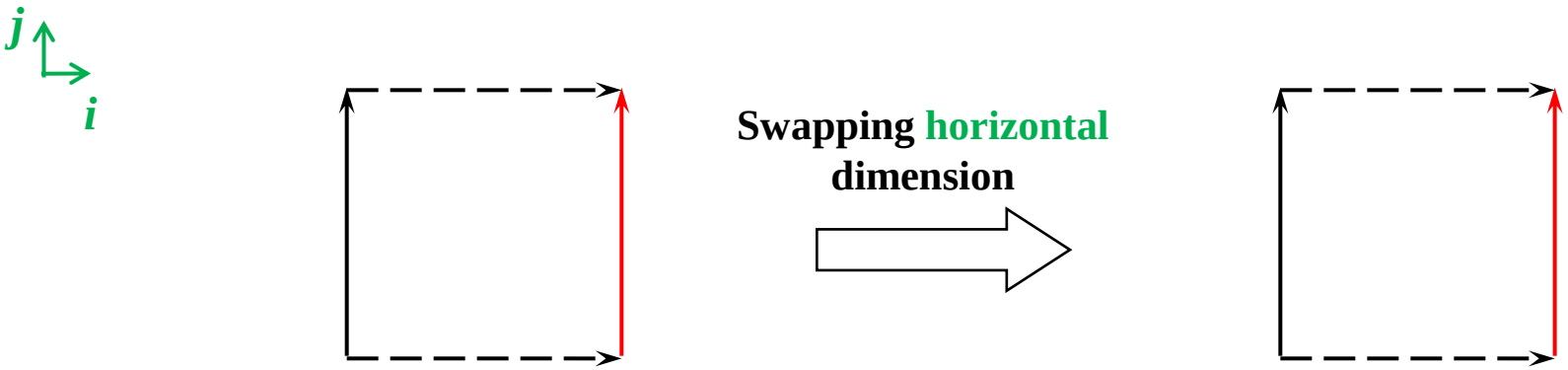
Let V_j = # of violated edges in dimension j

Claim. Swapping in dimension i does not increase V_j for all dimensions $j \neq i$

Enough to prove the claim for squares

Proof of The Claim for Squares

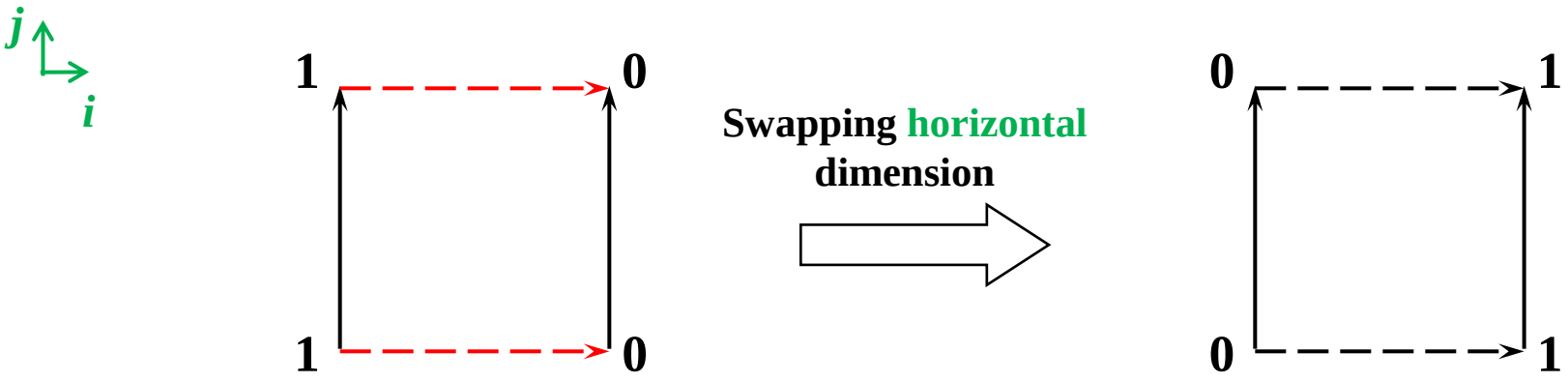
Claim. Swapping in dimension i does not increase V_j for all dimensions $j \neq i$



- If no horizontal edges are violated, no action is taken.

Proof of The Claim for Squares

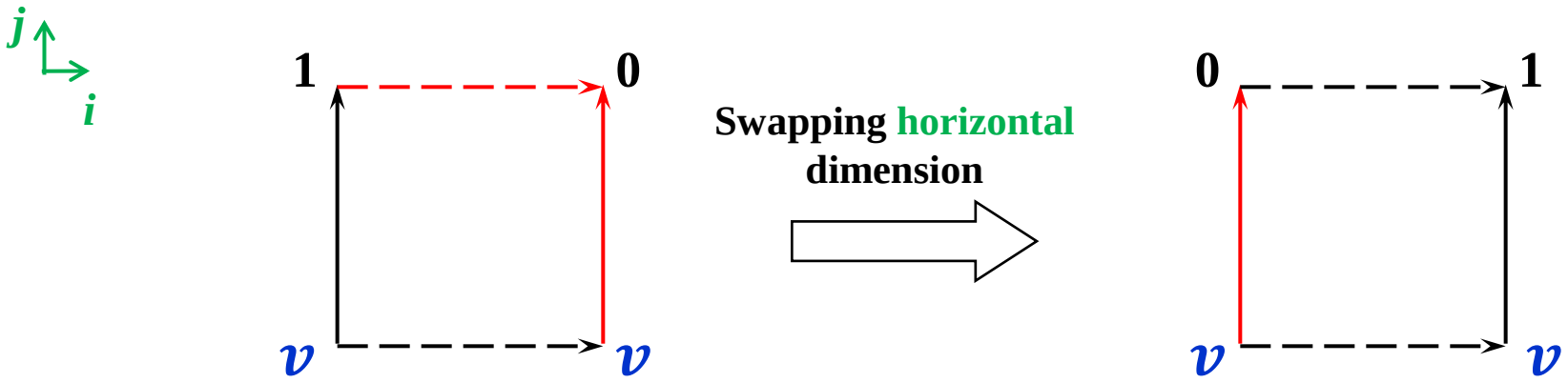
Claim. Swapping in dimension i does not increase V_j for all dimensions $j \neq i$



- If both horizontal edges are violated, both are swapped, so the number of vertical violated edges does not change.

Proof of The Claim for Squares

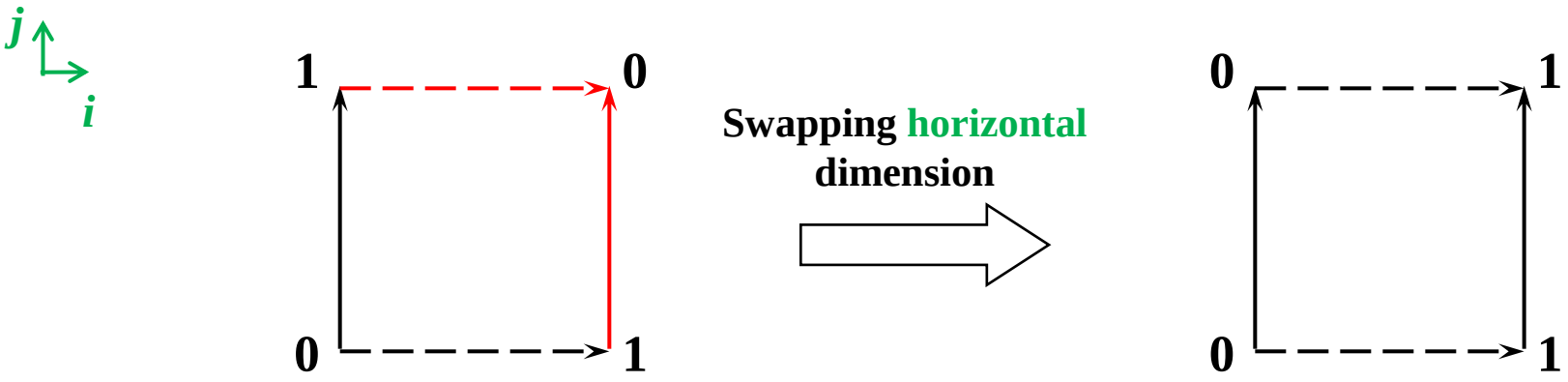
Claim. Swapping in dimension i does not increase V_j for all dimensions $j \neq i$



- Suppose one (say, top) horizontal edge is violated.
- If both bottom vertices have the same label, the vertical edges get swapped.

Proof of The Claim for Squares

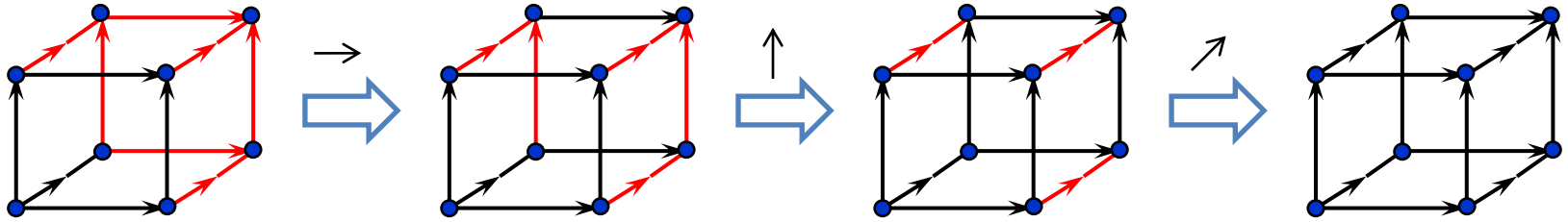
Claim. Swapping in dimension i does not increase V_j for all dimensions $j \neq i$



- Suppose one (say, top) horizontal edge is violated.
- If both bottom vertices have the same label, the vertical edges get swapped.
- Otherwise, the bottom vertices are labeled $0 \rightarrow 1$, and the vertical violation is repaired.

Proof of The Claim for Squares

Claim. Swapping in dimension i does not increase V_j for all dimensions $j \neq i$



After we perform swaps in all dimensions:

- f becomes monotone
- # of values changed:

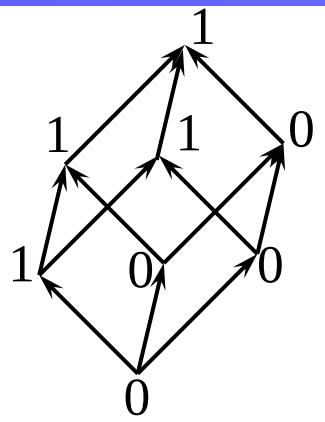
$$2 \cdot V_1 + 2 \cdot (\#\text{ violated edges in dim 2 after swapping dim 1}) + 2 \cdot (\#\text{ violated edges in dim 3 after swapping dim 1 and 2}) + \ldots = 2 \cdot V_1 + 2 \cdot V_2 + \cdots 2 \cdot V_n = 2 \cdot V(f)$$

Repair Lemma

f can be made monotone by changing $\leq 2 \cdot V(f)$ values.

Improve the bound by a factor of 2.

Testing if a Functions $f : \{0,1\}^n \to \{0,1\}$ *is monotone*



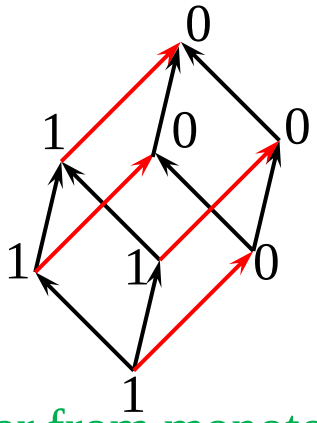
monotone

Monotone or

ε-far from monotone?

O(n/ε) time ✓

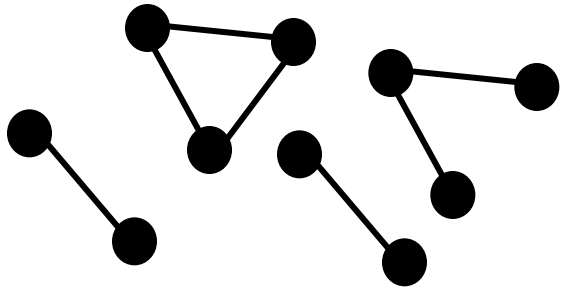
(logarithmic in the size of the input)



$\frac{1}{2}$-far from monotone

Graph Properties

Testing if a Graph is Connected [Goldreich Ron]



Input: a graph $G = (V, E)$ on n vertices

- in adjacency lists representation
  (a list of neighbors for each vertex)
- maximum degree *d*, i.e., adjacency lists of length *d* with some empty entries

Query (v, i), where $v \in V$ and $i \in [d]$: entry i of adjacency list of vertex v

Exact Answer: $\Omega(dn)$ time

- Approximate version:

Is the graph connected or ϵ-far from connected?

$$\text{dist}(G_1, G_2) = \frac{\#\ of\ entires\ in\ adjacency\ lists\ on\ which\ G_1\ and\ G_2\ differ}{dn}$$

Time: $O\left(\frac{1}{\varepsilon^2 d}\right)$ today

No dependence on n!

+ improvement on HW

Testing Connectedness: Algorithm

Connectedness Tester(G, d, ε)

1. **Repeat** s=16/εd times:
2. pick a random vertex u
3. determine if connected component of u is small:
   perform BFS from u, stopping after at most 8/εd new nodes
4. **Reject** if a small connected component was found, otherwise **accept.**

Run time: $O(d/\varepsilon^2 d^2)=O(1/\varepsilon^2 d)$

Analysis:

- Connected graphs are always accepted.
- Remains to show:

If a graph is ϵ-far from connected, it is rejected with probability $\geq \frac{2}{3}$

Testing Connectedness: Analysis

Claim 1

If G is ε-far from connected, it has $\geq \frac{\varepsilon dn}{4}$ connected components.

Claim 2

If G is ε-far from connected, it has $\geq \frac{\varepsilon dn}{8}$ connected components of size at most $8/\varepsilon d$.

- If Claim 2 holds, at least $\frac{\varepsilon dn}{8}$ nodes are in small connected components.
- By Witness lemma, it suffices to sample $\frac{2 \cdot 8}{\varepsilon dn/n} = \frac{16}{\varepsilon d}$ nodes to detect one from a small connected component.

Testing Connectedness: Proof of Claim 1

Claim 1

If G is ε-far from connected, it has $\geq \frac{\varepsilon dn}{4}$ connected components.

Proof: We prove the contrapositive:

If G has $< \frac{\varepsilon dn}{4}$ connected components, one can make G connected by modifying $< \varepsilon$ fraction of its representation, i.e., $< \varepsilon dn$ entries.

- If there are no degree restrictions, k components can be connected by adding k-1 edges, each affecting 2 nodes. Here, $k < \frac{\varepsilon dn}{4}$, so $2k-2 < \varepsilon dn$.
- What if adjacency lists of all vertices in a component are full, i.e., all vertex degrees are d?

Freeing up an Adjacency List Entry

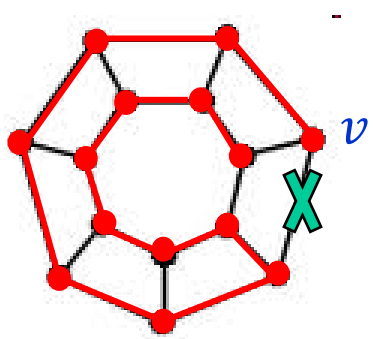
Claim 1

If G is ε-far from connected, it has $\geq \frac{\varepsilon dn}{4}$ connected components.

Proof:

What if adjacency lists of all vertices in a component are full,

i.e., all vertex degrees are d?



- Consider an MST of this component.
- Let v be a leaf of the MST.
- Disconnect v from a node other than its parent in the MST.
- Two entries are changed while keeping the same number of components.
- Thus, k components can be connected by adding 2k-1 edges, each affecting 2 nodes. Here, $k < \frac{\varepsilon dn}{4}$, so $4k-2 < \varepsilon dn$.

Testing Connectedness: Proof of Claim 2

Claim 1

If G is ε-far from connected, it has $\geq \frac{\varepsilon dn}{4}$ connected components.

Claim 2

If G is ε-far from connected, it has $\geq \frac{\varepsilon dn}{8}$ connected components of size at most $8/\varepsilon d$.

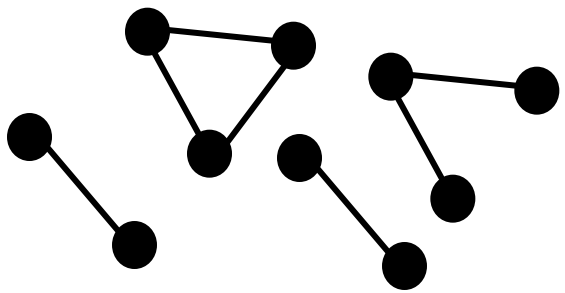
Proof of Claim 2:

- If Claim 1 holds, there are at least $\frac{\varepsilon dn}{4}$ connected components.
- Their average size $\leq \frac{n}{\varepsilon dn/4} = \frac{4}{\varepsilon n}$.
- By an averaging argument (or Markov inequality), at least half of the components are of size at most twice the average.

Testing if a Graph is Connected [Goldreich Ron]

Input: a graph $G = (V, E)$ on n vertices

- in adjacency lists representation
  (a list of neighbors for each vertex)
- maximum degree d



Connected or

ε-far from connected?

$O\left(\frac{1}{\varepsilon^2 d}\right)$ time ✓

(no dependence on n)