Randomized Algorithms

Introduction to Property Testing

Speaker: Chuang-Chieh Lin
Advisor: Professor Maw-Shang Chang
National Chung Cheng University

2006/11/30

NOTICE

- Note that you need to install TeX4PPT to view or edit this powerpoint file.

`$e^{\pi i} + 1 = 0$`

$$e^{\pi i} + 1 = 0$$

Outline

- Introduction
  - Sublinear-time algorithms
  - Notions of approximation
  - Definition of a property tester
- A simple example
  - Testing monotonicity of a list
  - Testing connectivity of a graph
- Further readings

Introduction

- With the recent advances in technology, we are faced with the need to process increasingly larger amounts of data in faster times.
- There are practical situations in which the input is so large, that even taking a linear time in its size to provide an answer is too much.
- Making a decision after reading only a small portion of the input, that is, in *sublinear time*, is thus considered to be an very important issue.

Introduction (cont'd)

- Sublinear time algorithms have received a lot of attention recently.

- Recent results have shown that there are optimization problems whose value can be approximated in sublinear time.

Introduction (cont'd)

- However, most algorithms which run in sublinear time must necessarily use randomization and must give an approximate answer.
- Surprisingly though, there are nontrivial problems for which deterministic exact algorithms exist!
- Let us see the following two examples.

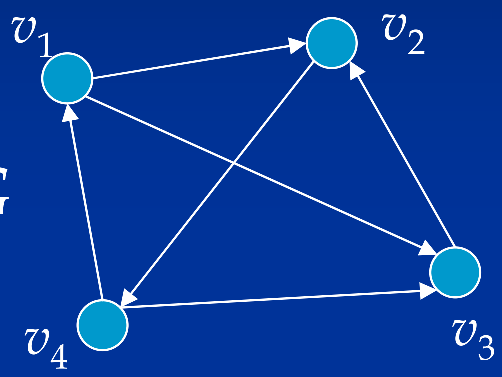
Example 1: Tournament

- A tournament is a digraph such that for each pair of vertices u and v, exactly one of (u, v) and (v, u) is an edge.

- We can interpret the vertices as players such that each pair of players play a match, and an edge from one to another indicates that one player beats another, hence the name tournament.

Tournament (cont'd)

a tournament G



$$M_G = \begin{array}{c} \\ v_1 \\ v_2 \\ v_3 \\ v_4 \end{array} \begin{array}{c} \begin{array}{cccc} v_1 & v_2 & v_3 & v_4 \end{array} \\ \begin{bmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \end{bmatrix} \end{array}$$

- Assume that we have a tournament G on n vertices represented in adjacency matrix form M_G.
- Thus the size of G is $\binom{n}{2}$.

Tournament (cont'd)

- **Input:**
  - a tournament G on n vertices represented in adjacency matrix form M_G .
- **Output:**
  - the source of G if it exists, otherwise output "No source exists". (source: the vertex of out-degree n–1)
- There exists a deterministic algorithm that finds the source of G (a player who beats all others) if it exists in $O(n)$ time.

Tournament (cont'd)

<u>Algorithm-Source-Finding:</u>

1. $S \leftarrow \{v_1, \ldots, v_n\}$;
2. **while** $|S| > 1$ **do**
   (a) Arbitrary pick $v_i, v_j \in S$;
   (b) **if** $M_G[i, j] = 1$ **then** remove v_j from S;
   else remove v_i from S;
3. Denote the remaining vertex in S by v_r;
4. **For** $i = 1$ to n **do**
   if $M_G[r, i] = 0$ then output "No source exists." and return;
5. Return v_r;

<u>End of the Algorithm</u>

Example 2: Diameter

- Assume that we have n points in a metric space.
- The input is an $n \times n$ distance matrix D such that $D(i, j)$ is the distance between i and j.
- We seek a sublinear time algorithm that outputs $\max_{i,j} D(i,j)$, i.e., the diameter.

Diameter (cont'd)

- **Input:**
  - an $n \times n$ distance matrix D such that $D(i, j)$ is the distance between i and j.
- **Output:**
  - diameter of these n points (i.e., $\max_{i,j} D(i,j)$)

- Consider the following simple algorithm.

Diameter (cont'd)

Algorithm-Diameter:

★ Pick a point u arbitrary and output $z := \max_v D(u, v)$.

End of the Algorithm

- Clearly this algorithm runs in $O(n)$ time. Moreover, we argue that z, the value returned by this naïve looking algorithm, is a good approximation for the diameter d of the input.

Diameter (cont'd)

- Claim: $d/2 \leq z \leq d$.

- Proof:
  - Let a and b be two points such that $D(a,b) = d$ and assume that $z = D(u,v)$
  - Since D is a metric space, we have

$$d = D(a,b) \leq D(a,u) + D(u,b) \leq D(u,v) + D(u,v) = 2z.$$

$\blacksquare$

- To study approximation algorithms, we need to define notions of how good an approximation is.

Definitions

Let $\pi(x)$ be the optimal solution of an input x. For $\beta > 1$, we say that A is a *β-multiplicative approximation algorithm* if for all x,

$$\frac{\pi(x)}{\beta} \leq A(x) \leq \beta\pi(x).$$

We say that A is an *α-additive approximation algorithm* if for all x,

$$\pi(x) - \alpha \leq A(x) \leq \pi(x) + \alpha.$$

How to approximate a decision problem?

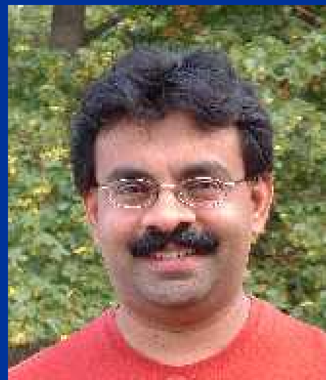
- In addition, *property testing*, an alternative notion of approximation for decision problems, has been applied to give sublinear time algorithms for a wide variety of problems.

- "Still, the study of sublinear time algorithms is very new, and much remains to be understood about their scope." - Ronitt Rubinfeld
  - ACM SIGACT News, Vol. 34, No. 4, 2003.

系統識別號	U0001-1906200616134600
論文名稱(中文)	測試圖的連通性
論文名稱(英文)	Testing Connectivity of Graphs
校院名稱	臺灣大學
系所名稱(中)	資訊工程學研究所
系所名稱(英)	Graduate Institute of Computer Science and Information Engineering
學年度	94
學期	2
出版年	95
研究生(中文)	陳絢昌
研究生(英文)	Hsuan-Chang Chen 陳絢昌
學號	R93922117
學位類別	碩士
語文別	英文
口試日期	2006-06-06
論文頁數	19頁
口試委員	指導教授-呂育道 委員-金國興 委員-戴天時
中文關鍵字	連通圖 近似 測試 圖形演算法
英文關鍵字	Property Testing Graph algorithm Connected graph Approximation
學科別分類	學科別>應用科學>資訊工程

Property testing

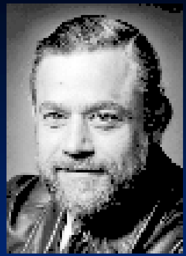
- The notion of property testing was first formulated by Rubinfeld and Sudan.



Ronitt Rubinfeld and Madhu Sudan: Robust charaterization of polynomials with applications to program testing, *SIAM Journal on Computing*, 1996, Vol. 25, pp. 252-271.

Property testing (cont'd)

- Due to these two pioneers, plenty results have come out recently.
  - See the "Further readings" for reference.
- Many outstanding scholars have devoted to this topic of research, such as:


Manuel Blum


Madhu Sudan


Ronitt Rubinfeld


Luca Trevisan


Bernard Chazelle


Noga Alon


Dana Ron


Rajeev Motwani


Oded Goldreich


Sanjeev Arora


Eldar Fischer


Carsten Lund


Tugkan Batu


Shafi Goldwasser


Michael Luby


Mario Szegedy


Lance Fortnow


Ravi Kumar


Sampath Kannan


Funda Ergũn

Especially,

- Property testing emerges naturally in the context of program checking and probabilistic checkable proofs (PCP).

Sanjeev Arora

Carsten Lund

Rajeev Motwani

Madhu Sudan

Mario Szegedy

PCP theorem: **NP = PCP**$(O(\log n), O(1))$

- JACM, Vol. 45, 1998.

Roughly speaking, …

- A property tester is an algorithm which
  - accepts with high probability if the input has a certain property, and
  - rejects with high probability if the input is "far" from the property.
    - ✓ That is, the input *cannot be modified slightly* to make it possess the property.

Property testing (cont'd)

- In order to define a property tester, it is important to define a notion of *distance* from having a property.

- Define a language P to be a class of inputs that have a certain property.
  - For example, *connected* graphs, *monotone increasing* integers, …

Property testing (cont'd)

- Let $\Delta(x, y)$ be the distance function between input x and y, with $\Delta(x, y) \in [0, 1]$ and define

$$d(x, P) = \min_{y \in P} \Delta(x, y)$$

Property testing (cont'd)

- For example, the Hamming distance/ #digits of two 0-1 strings with equal length can be a Δ.

$$\Delta(\texttt{01001}_2, \texttt{01110}_2) = 3/5.$$

- Let *P* be a set of 0-1 strings which has fewer 0's than 1's, we can easily have

$$d(\texttt{01001}_2, P) = 1/5.$$

Property testing (cont'd)

- So let us consider the formal definition of a property tester.

Property testing (cont'd)

- A property tester for (P, d) is defined as

  ★ Given input x, $0 < \epsilon < 1$.

  if $x \in P$, then $\mathbf{Pr}[\text{return "}\textbf{Pass}\text{"}] \geq 2/3$.
  if $d(x, P) \geq \epsilon$, then $\mathbf{Pr}[\text{return "}\textbf{Fail}\text{"}] \geq 2/3$.

  Remark:
  If $d(x, P) \geq \epsilon$, we say x is ϵ-far from P.
  If $d(x, P) \leq \epsilon$, we say x is ϵ-close from P.

A simple example

- Consider the following example to figure out the concept of property testing.
- Suppose we have a sequence of n numbers, $x_1, \ldots, x_n$, we would like to determine if the sequence is monotonically increasing.
  - Input: $x_1, \ldots, x_n$
  - Output: Accepts or Rejects.

Testing monotonicity of a list

- Any deterministic decision algorithm runs in $\Omega(n)$ time to read the input and make a decision.

- On the other hand, a property testing algorithm exists such that it
  - accepts, if the sequence is monotonically increasing
  - rejects with probability greater than 2/3, if more than εn of the x_i need to be removed so that the resulting sequence becomes monotonically increasing.

Testing monotonicity of a list (cont'd)

- WLOG, we can assume that all x_i's are distinct.
  - Since we can interpret x_i as (x_i, i), which breaks ties without changing order.

- Consider the following simple approach which can not be ensured to run in sublinear time.

Testing monotonicity of a list (cont'd)

Algorithm 1

★ Select i randomly and test whether $x_i < x_{i+1}$. Then return "Pass" if $x_i < x_{i+1}$, and return "Fail" otherwise.

- Consider the following sequence which is very far from monotonically increasing:

4, 8, 12, 3, 7, 11, 2, 6, 10, 1, 5, 9

PASS

Testing monotonicity of a list (cont'd)

- Generally, such sequence $x_1, x_2, \ldots, x_n$ can be written as the following form:

  $m, 2m, \ldots, km,$

  $m-1, 2m-1, \ldots, km-1, \ldots,$

  $1, m+1, 2m+1, \ldots, (k-1)m+1.$ (thus $n = mk$)

  where m, k are two integers greater than 1.

- For example, when $m = 4$, $k = 3$:

  4, 8, 12, 3, 7, 11, 2, 6, 10, 1, 5, 9

Testing monotonicity of a list (cont'd)

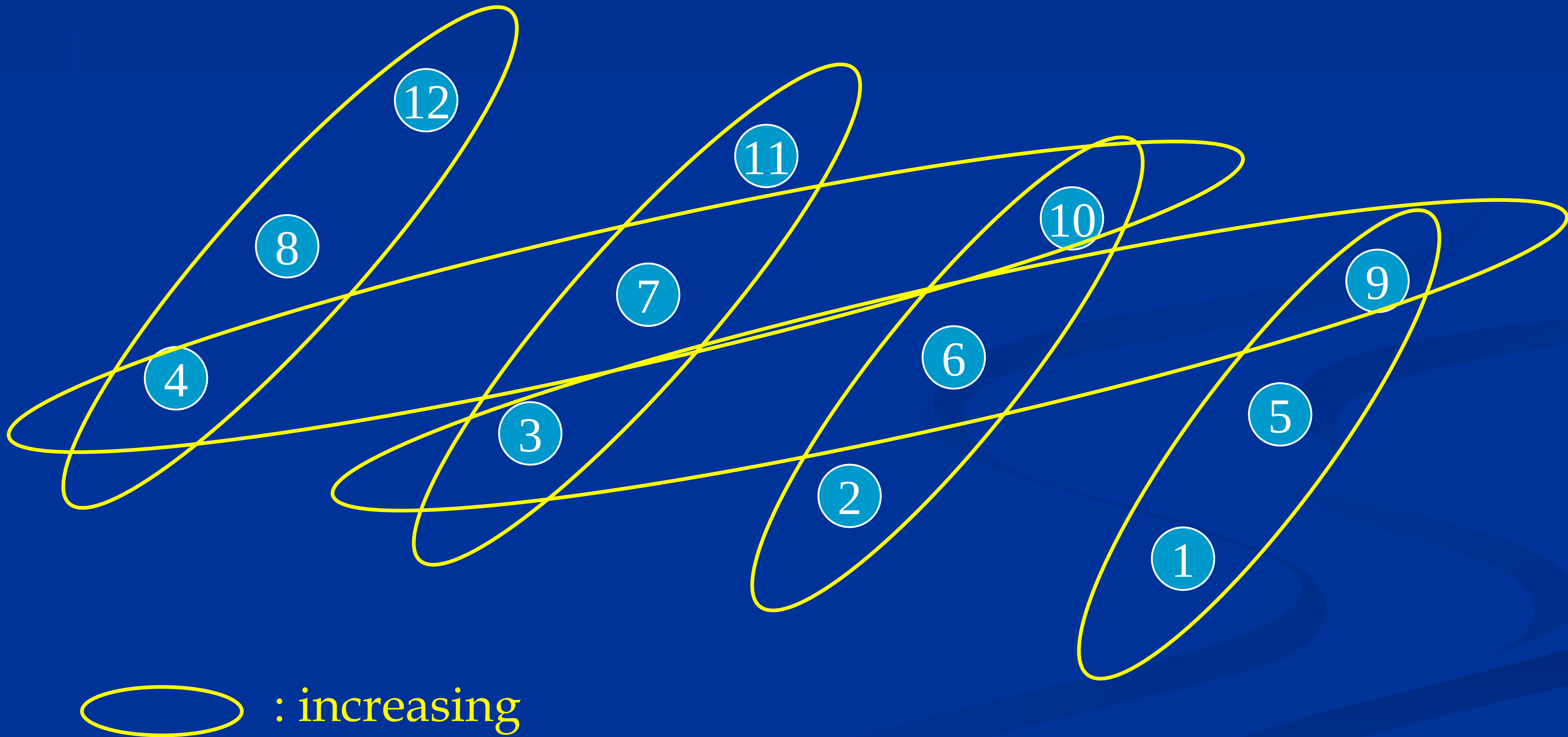
- The distance of such sequence from monotonically increasing is at least ½.
  - WHY?
  - For example,

2, 4, 1, 3 → 2, 4 or 2, 3 or 1, 3

for monotonically increasing

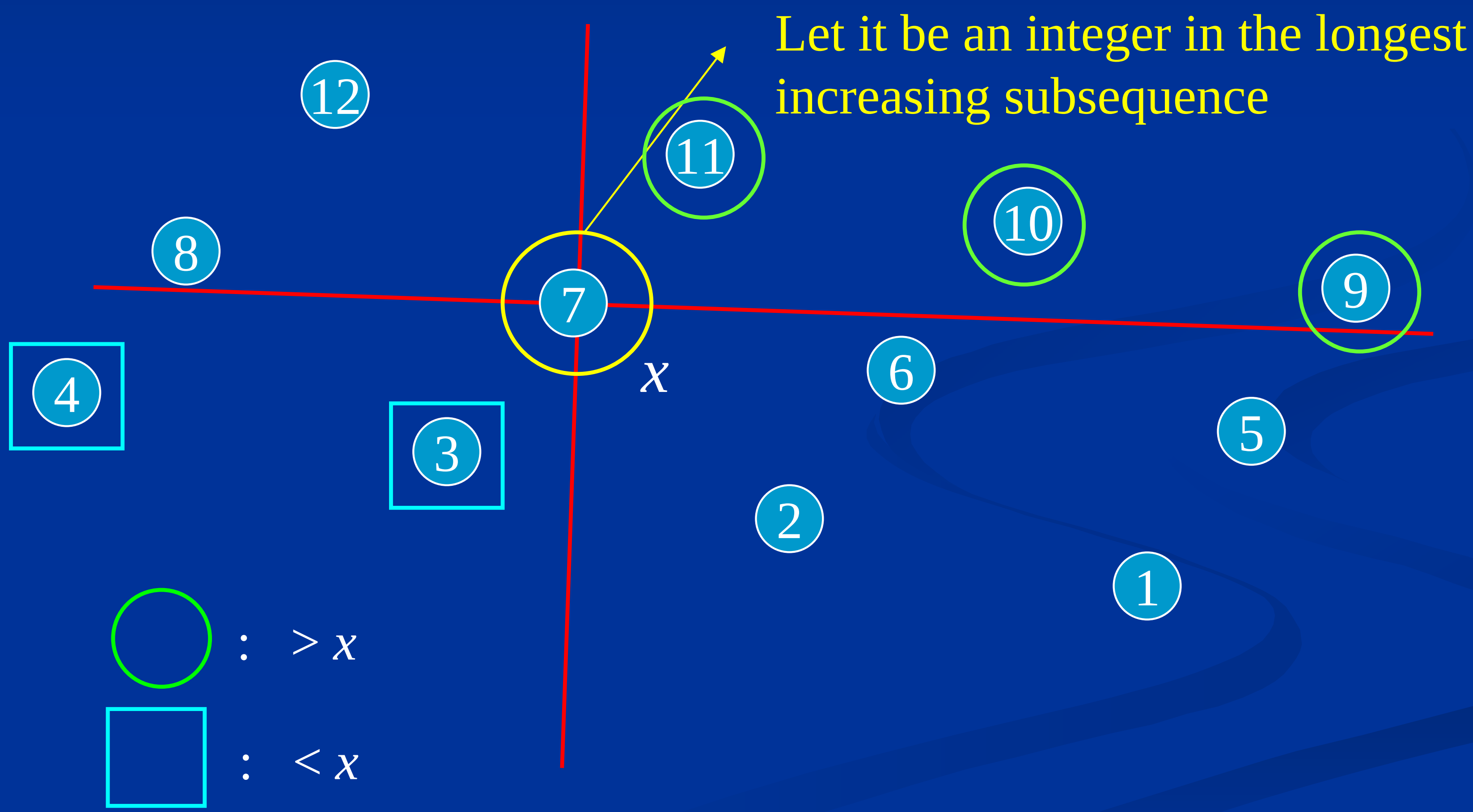
Testing monotonicity of a list (cont'd)

- See the following illustration: (m = 4, k = 3)



Testing monotonicity of a list (cont'd)

- See the following illustration: (m = 4, k = 3)



Testing monotonicity of a list (cont'd)

- We can easily prove that the length of a longest monotonically increasing subsequence in such a sequence must be at most k,
  - Exercise. (Hint: Consult the previous illustration.)

- So the distance of such sequence from monotonically increasing is at least $n - k = (m-1)k$, which is at least ½ of the length of the sequence.
  - For example, 2, 4, 1, 3 → 2, 4 or 2, 3 or 1, 3

Testing monotonicity of a list (cont'd)

$m, 2m, \ldots, km, m-1, 2m-1, \ldots, km-1, \ldots, 1, m+1, 2m+1, \ldots, (k-1)m+1$

- Algorithm 1 does not detect that the sequence is not monotonically increasing as long as it does not query a pair of locations of a yellow integer and its next integer respectively.
- Thus Algorithm 1 will need $\Omega(k)$ queries, that is, repeatedly runs $\Omega(k)$ times.
  - WHY?

Testing monotonicity of a list (cont'd)

m, $2m$,…, km, m–1, $2m$–1,…, km–1, … , 1, m+1, $2m$+1,…, (k–1)m+1

- The probability that Algorithm 1 doesn't query any yellow integer is larger than 1 – 1/k for each run.
- The probability that Algorithm 1 queries a yellow integer at least once during $c \cdot k$ runs is less than $1 - (1-1/k)^{ck}$.

Testing monotonicity of a list (cont'd)

- $1 - (1-1/k)^{ck} \searrow 1 - 1/e^{c} > 2/3$ when k is large and $c > 1$.
  - That is, if we don't run Algorithm 1 for more than $\Omega(k)$ times, Algorithm 1 will not query any yellow integer with high probability (when k is large and $c > 1$.)
- However, we cannot ensure the probability that Algorithm 1 query a yellow integer at least once during $c \cdot k$ runs is at least 2/3.

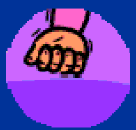
Testing monotonicity of a list (cont'd)

- Thus the time complexity of this algorithm cannot be ensured to be sublinear.
- Try another one!

Testing monotonicity of a list (cont'd)

- Consider another algorithm, which is a little sophisticated.

<u>Algorithm 2</u>



★ Samples the sequence at random points and checks if these random points form a monotonically increasing sequence.

★ Return "Pass" if they do, and return "Fail" otherwise.

Testing monotonicity of a list (cont'd)

- However, consider the following sequence, which is again very far from monotonically increasing.

$$m,\ m-1, \ldots, 1,\ 2m,\ 2m-1, \ldots,\ m+1,\ 3m,\ \ldots,\ 2m+1,\ \ldots$$

- Again, the distance of this sequence from monotonically increasing is at least ½.
- The algorithm detects that this sequence is not monotonically increasing only if two of its query points fall within $[km, (k-1)m+1]$ for some k.

Testing monotonicity of a list (cont'd)

$m, m-1, \ldots, 1, 2m, 2m-1, \ldots, m+1, 3m, \ldots, 2m+1, \ldots$

- However, by the Birthday Paradox, this is unlikely if m is a constant and the number of samples is $o((n/m)^{½}) = o(n^{½})$.
- With high probability, the values of the query points will form a monotonically increasing sub-sequence.
- Thus Algorithm 2 does not work well.

- Can we do better?
  - YES!

F. Ergün, S. Kannan, R. Kumar, R. Rubinfeld and M. Viswanathan proposed a $O((1/\varepsilon) \log n)$ property tester.

- JCSS, Vol. 60, 2000

Testing monotonicity of a list (cont'd)

- Consider the following algorithm. [EKKRV00]

Algorithm 3$((x_1, \ldots, x_n), \epsilon)$

★ Repeat Step 1 to 3 for $O(1/\epsilon)$ times:

1. Pick i uniformly at random from 1 through n.
2. Query x_i.
3. Perform binary search for x_i. If the search does not found x_i, return "Fail" (i.e., Reject).

★ Return "Pass" (i.e., Accept) if all searches are successful.

For example,

```
Begin binary search
```

index	1	2	3	4	5	6	7
value	21	9	1	3	5	8	17

Search for value **1**.

<u>Output:</u> **Fail!**

Another example,

```
Begin binary search
```

index	1	2	3	4	5	6	7
value	21	9	1	3	5	8	17

Search for value **8**.

Output: **Pass!**

Testing monotonicity of a list (cont'd)

- Algorithm 3 runs in time $O((1/\varepsilon) \log n)$ since each binary search takes $O(\log n)$ time.
- If the sequence $\{x_i\}$ is monotonically increasing, then clearly the algorithm accepts.
- We need to show that if at least εn of the sequence need to be removed for it to be monotonically increasing, then the algorithm rejects (resp. accepts) with probability at least 2/3 (resp., less than 1/3).
  - Suppose not, that Algorithm 3 accepts with probability at least 1/3.

Testing monotonicity of a list (cont'd)

- Proof by contradiction:
  - ε-far $\Rightarrow$ accept with probability $< 1/3$
  - accept with probability $\geq 1/3$ $\Rightarrow$ ε-close

- We call index *i* is "*good*" if the binary search for x_i is successful, otherwise we call index *i* is "*bad*".

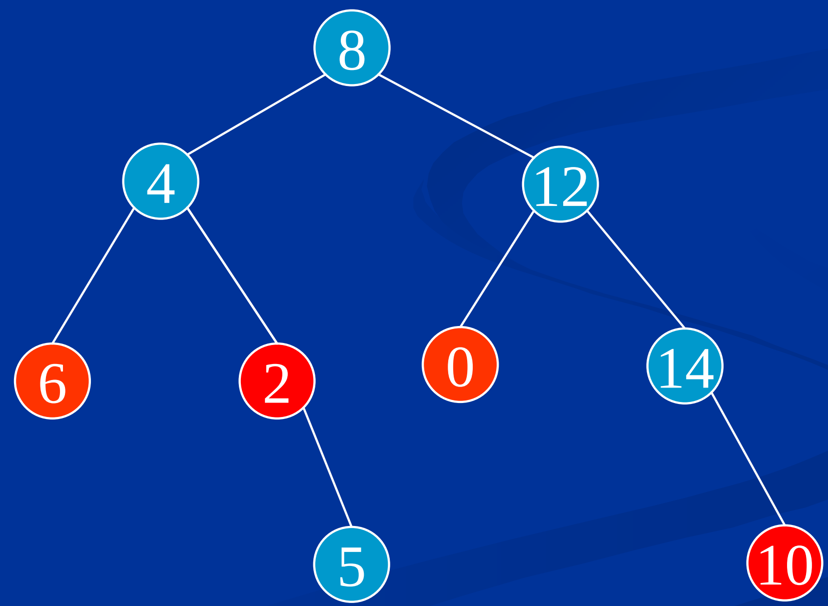
Testing monotonicity of a list (cont'd)

- For example,

index	1	2	3	4	5	6	7	8	9
value	6	4	2	5	8	0	12	14	10

: good ones

: bad ones



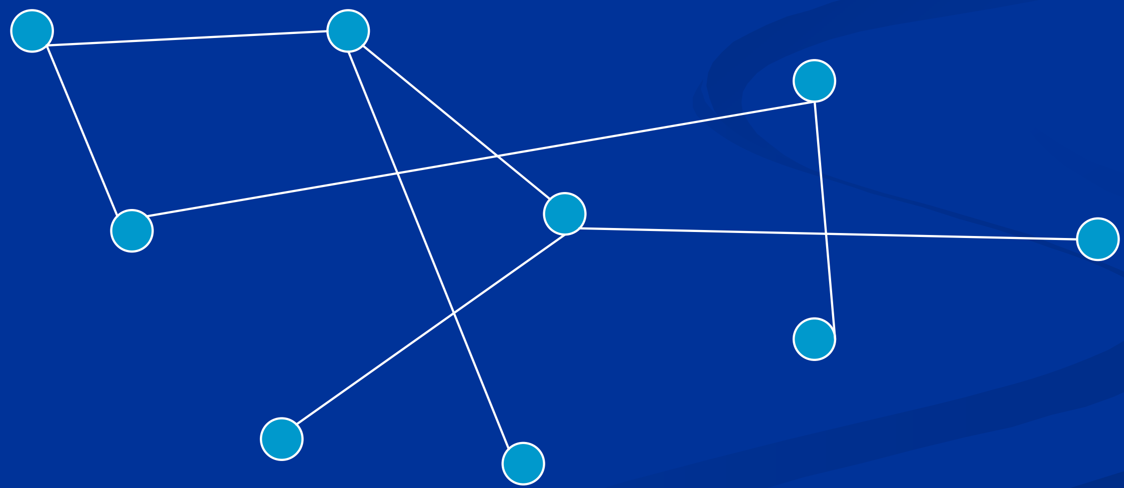
Testing monotonicity of a list (cont'd)

- We claim that less than εn of the indices are bad.
  - Otherwise, each time through the loop, the algorithm finds a bad index with probability at least ε.
  - Then Algorithm 3 accepts with probability at most $(1 - \varepsilon)^{c/\varepsilon} < e^{-c} < 1/3$ for some constant c.
  - A contradiction then occurs.
- Now, the remaining part is to prove that the good points indeed form a monotonically increasing subsequence.

Testing monotonicity of a list (cont'd)

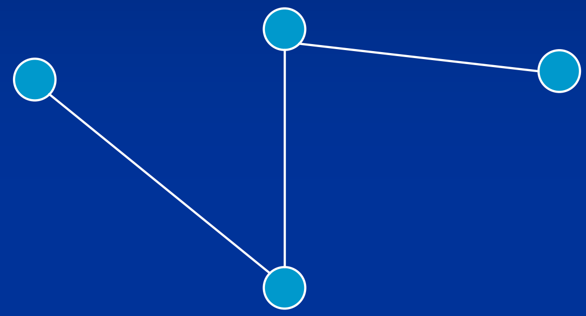
- Consider any two good indices *i*, *j* , where $i < j$.
- Consider the first point in the binary search path where x_i and x_j diverge and assume that point has value *u*.
- Since *i* and *j* are good and $i < j$, we can conclude that $x_i \leq u \leq x_j$. This concludes the proof. ■

- Now, let us consider another problem: Testing connectivity of a graph.

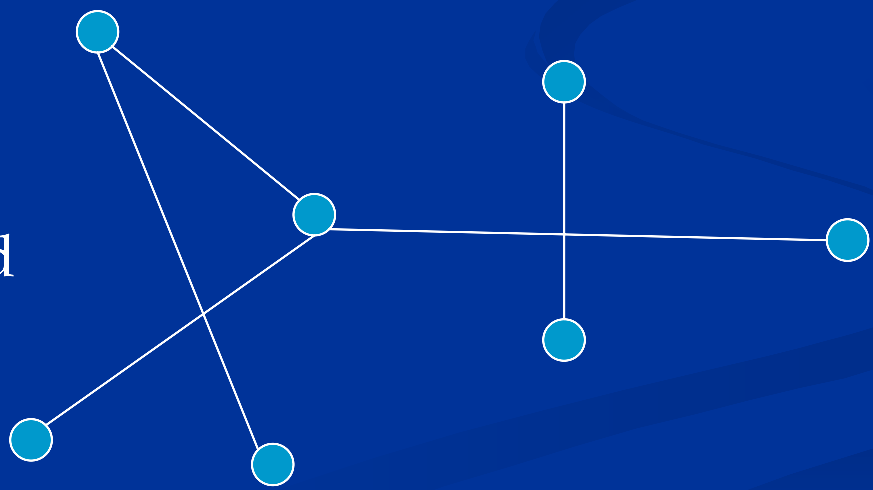


Connected and Disconnected

connected



disconnected



Degree bound

- We say a graph $G(V, E)$ has a degree bound d if for each vertex $v \in V$,

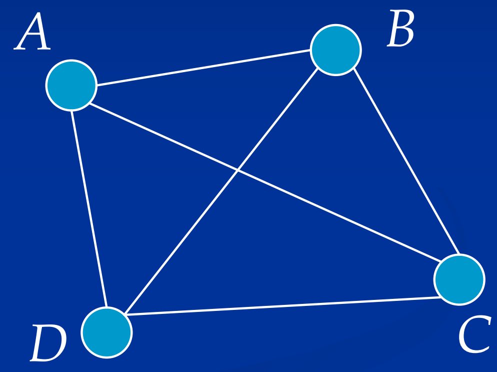
$$\deg(v) \leq d$$

where $\deg(v)$ is the number of vertices adjacent to v in G.

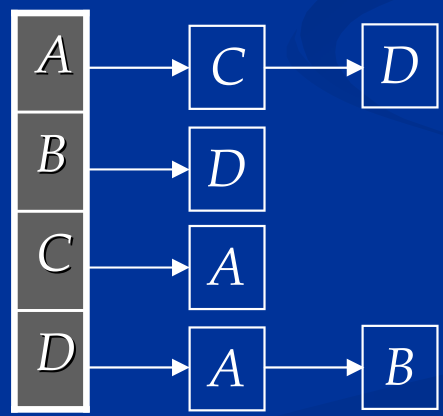
Graph representations

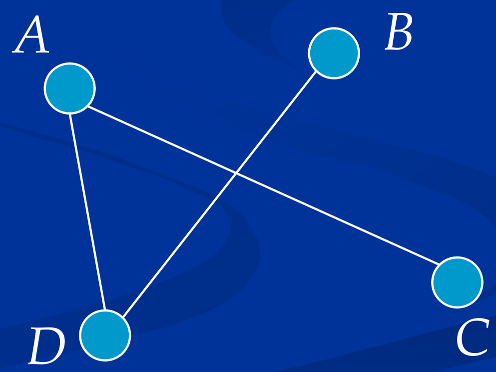
- Adjacency matrix
  - For dense graphs

$$
\begin{array}{c|cccc|}
 & A & B & C & D \\
\hline
A & 0 & 1 & 1 & 1 \\
B & 1 & 0 & 1 & 1 \\
C & 1 & 1 & 0 & 1 \\
D & 1 & 1 & 1 & 0 \\
\hline
\end{array}
$$



- Adjacency list
  - For sparse graphs





Testing connectivity of a graph

- We will adopt the adjacency list model with a given degree bound *d* to proceed with our discussion.
  - The graph possesses $O(dn)$ edges.

Testing connectivity of a graph (cont'd)

★ **Input:** a graph $G(V, E)$ with bounded degree d , given as adjacency list

★ **Desired property:** $P =$ a class of connected graphs with bounded degree d

Let $\mathcal{G}_n^d$ denote the set of graphs of n nodes with a bounded degree d.

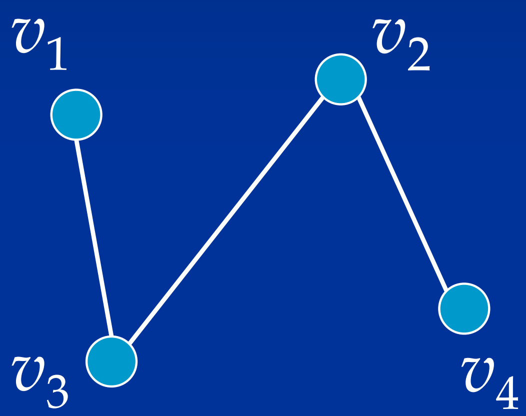
Testing connectivity of a graph (cont'd)

- Let $G \in \mathcal{G}_n^d$, we define the distance of G from connected to be

$$\mathrm{dist}(G, P) = \frac{2\rho_d(G)}{dn}$$

where $\rho_d(G)$ is the minimum number of modifications of edges needed for G to be connected such that the degree bound d is still maintained.

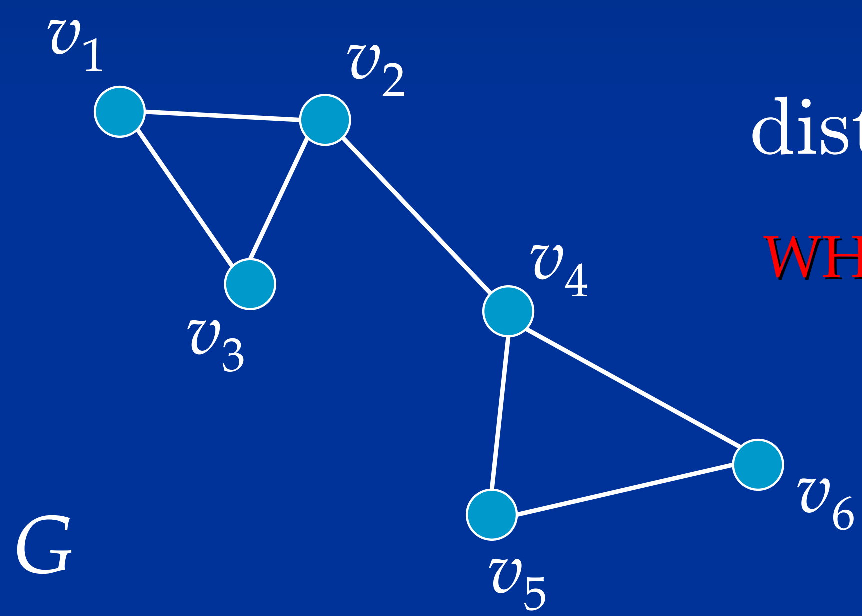
For example, ($d = 2$)



G

$$\mathrm{dist}(G, P) = \frac{2\rho_2(G)}{dn} = \frac{1}{4}.$$

Another example, ($d = 2$)



$$\mathrm{dist}(G, P) = \frac{2\rho_2(G)}{dn} = \frac{1}{2}.$$

WHY?

Idea

- If a graph is far from connected, there must be many components,
  - That in turn implies that there are many small components.
- Consider the following algorithm proposed by O. Goldreich and D. Ron.

\- Algorithmica, Vol. 32, 2002.

Testing connectivity of a graph (cont'd)

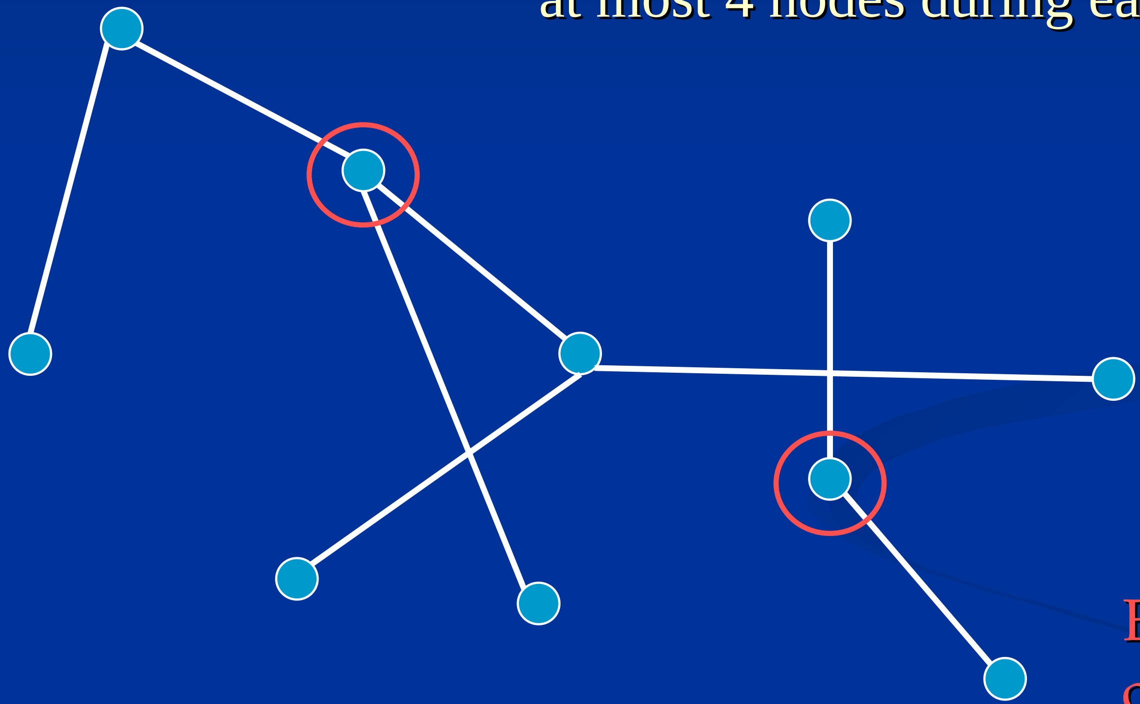
Algorithm GR(G, ϵ) [GR02]

1. Pick $m = O(\frac{1}{\epsilon d})$ nodes of G uniformly at random.
   Let S denote the set of these picked nodes.
2. **For** each node $s \in S$, **do** BFS and stop **if**:
   (a) $\frac{8}{\epsilon d}$ nodes have been reached
   (b) exhaust the component
3. **If** (b) ever happens, return "Fail"; **otherwise**, return "Pass".

(Here we assume that $|V| \geq 8/\epsilon d$.)

An illustration

Pick 2 nodes of the graph, and see at most 4 nodes during each BFS.



STOP

EXAUST the component!

Halt and output: “Fail”

Testing connectivity of a graph (cont'd)

- The running time of Algorithm GR is

$$O(\frac{1}{\epsilon d} \cdot \frac{8}{\epsilon d} \cdot d) = O(\frac{1}{\epsilon^2 d}),$$

which is sublinear.

- Why does this algorithm work?

Testing connectivity of a graph (cont'd)

- For $G \in \mathcal{G}_n^d$, if $G \in P$, it is obvious that the algorithm must output "Pass".
  - Maybe you don't think that this is trivial. You can prove this claim for an easy exercise.

- So, what if $G \notin P$?
  - We have to prove that if G is far from P, (i.e., G is far from connected with degree bound d) Algorithm GR will output "Fail" with probability at least 2/3.

Testing connectivity of a graph (cont'd)

- Consider the following observation first.

- Observation:

  If $G \in \mathcal{G}_n^d$ is ϵ-far from connected, then G has at least $\epsilon dn/2$ connected components.

- Proof:
  - If G has less than $\varepsilon dn/2$ connected components, we can add less than $\varepsilon dn/2$ edges to make G connected.
  - G is not ε-far from connected. (Because $\varepsilon dn/dn = \varepsilon$) ■

Testing connectivity of a graph (cont'd)

- Lemma 1:

A class of connected graphs with bounded degree d

If $G \in \mathcal{G}_n^d$ is ϵ-far from P, then G has at least $\epsilon dn/4$ connected components.

- Proof: Exercise!
  - Hint: Consider the previous observation and the second example for illustrating $\mathrm{dist}(G, P)$.

Testing connectivity of a graph (cont'd)

- Corollary 1:

  If $G \in \mathcal{G}_n^d$ is ϵ-far from P, then G has at least $\epsilon dn/8$ connected components each containing less than $\frac{8}{\epsilon d}$ nodes.

- Proof:
  - Let $n_<$ be the number of components of size less than $\frac{8}{\epsilon d}$. $\rightarrow$ We call them small components for simplicity.
  - Let $n_>$ be the number of components of size at least $\frac{8}{\epsilon d}$.

Testing connectivity of a graph (cont'd)

- Assume that G is ε-far from P. Then from Lemma 1 we have that G has at least $\varepsilon dn/4$ connected components.

- Since $n_< + n_>$ is the total number of connected components in G, we have $n_< + n_> \geq \varepsilon dn/4$.

- Since $n_> \cdot 8/\varepsilon d \leq n$, we have $n_> \leq \varepsilon dn/8$.
- Therefore, $n_< \geq \varepsilon dn/4 - \varepsilon dn/8 = \varepsilon dn/8$, the corollary immediately follows. ■

Testing connectivity of a graph (cont'd)

- Theorem 1:

  Let $G \in \mathcal{G}_n^d$

  ★ if G is connected, then Algorithm GR return "Pass"

  ★ if G is ϵ-far from P, then $\mathbf{Pr}[\text{Algorithm GR return "Fail"}] \geq \frac{2}{3}$.

- Proof of Theorem 1 is as follows.

Testing connectivity of a graph (cont'd)

- If *G* is connected, Algorithm GR must output "Pass".
  - Trivial.

- Consider the case that *G* is ε-far from *P*.

Testing connectivity of a graph (cont'd)

- By Corollary 1,

$$\begin{aligned}
&\mathbf{Pr}[s \text{ is in a small component}] \\
&= \frac{\text{number of nodes in small components}}{n} \\
&\geq \frac{\text{number of small components}}{n} \\
&\geq \frac{\epsilon d}{8}.
\end{aligned}$$

Each component is of size at least one and they are disjoint.

From Corollary 1.

Testing connectivity of a graph (cont'd)

- Since m is chosen to be $c/\varepsilon d$ for some constant c, we have

$$
\begin{aligned}
& \mathbf{Pr}[\text{no } s \text{ is in a small component}] \\
\leq \quad & (1 - \frac{\epsilon d}{8})^{\frac{c}{\epsilon d}} \\
\leq \quad & e^{-c'} \\
< \quad & \frac{1}{3}.
\end{aligned}
$$

These inequalities holds as long as we pick c large enough (c' is a constant that depends on c).

Therefore, the proof is done. ■

- I think I should finish this talk now.

- Related works on Property testing are listed at “Further readings” as follows.

Further readings

1. [A02] Testing subgraphs in large graphs, N. Alon, *Random Structures and Algorithms*, Vol. 21, 2002, pp. 359-370.
2. [AFKS00] Efficient testing of large graphs, N. Alon, E. Fischer, M. Krivelevich and M. Szegedy, *Combinatorica*, Vol. 20, 2000, pp. 451-476.
3. [AK02] Testing *k*-colorability, N. Alon and M. Krivelevich, *SIAM Journal on Discrete Mathematics*, Vol. 15, 2002, pp. 211-227.
4. [AKKLR03] Testing low-degree polynomials over GF(2), N. Alon, T. Kaufman, M. Krivelevich, S. Litsyn and D. Ron, RANDOM-APPROX'03, pp. 188-199.
5. [AKKR06] Testing triangle-freeness in general graphs, N. Alon, T. Kaufman, M. Krivelevich and D. Ron, SODA'06, pp. 279-288.
6. [AKNS01] Regular languages are testable with a constant number of queries, N. Alon, M. Krivelevich, I. Newman and M. Szegedy, *SIAM Journal on Computing*, Vol. 30, 2001, pp. 1842-1862.
7. [AS05] Every monotone graph property is testable, N. Alon and A. Shapira, STOC'05, pp. 128-137.
8. [AS03a] Testing satisfiability, N. Alon and A. Shapira, *Journal of Algorithms*, Vol. 47, 2003, pp. 87-103.

Further readings (cont'd)

9. [AS03b] Testing subgraphs in directed graphs, N. Alon and A. Shapira, STOC'03, pp. 700-709.
10. [AS04] A characterization of easily testable induced subgraphs, N. Alon and A. Shapira, SODA'04, pp. 935-944.
11. [BEKMRRS03] A sublinear algorithm for weakly approximating edit distance, T. Batu, F. Ergün, J. Kilian, A. Magen, S. Raskhodnikova, R. Rubinfeld and R. Sami, STOC'03, pp. 316-324.
12. [BFFKRW01] Testing random variables for independence and identity, T. Batu, E. Fischer, L. Fortnow, R. Kumar, R. Rubinfeld and P. White, FOCS'01, pp. 442-451.
13. [BFRSW00] Testing that distributions are close, T. Batu, E. Fischer, R. Rubinfeld, W. D. Smith and P. White, FOCS'00, pp. 259-269.
14. [BKR04] Sublinear time algorithms for testing monotone and unimodal distributions, T. Batu, R. Kumar and R. Rubinfeld, STOC'04, pp. 381-390.
15. [BLR93] Self-testing-or-correcting with applications to numerical problems, M. Blum, M. Luby and R. Rubinfeld, *Journal of Computer and System Sciences*, Vol. 47, 1993, pp. 549-595.

Further readings (cont'd)

16. [BOT02] A linear lower bound on the query complexity of property testing algorithms for 3-coloring in bounded-degree graphs, A. Bogdanov, K. Obata and L. Trevisan, FOCS'02, pp. 93-102.
17. [BR02] Testing properties of directed graphs: acyclicity and connectivity, M. Bender and D. Ron, *Random Structures and Algorithms*, Vol. 20, 2002, pp. 184-205.
18. [BRW05] Fast approximate PCPs for multidimensional bin-packing problems, T. Batu, R. Rubinfeld and P. White, *Information and Computation*, Vol. 196, 2005, pp. 42-56.
19. [BT02] Lower bounds for testing bipartiteness in dense graphs, A. Bogdanov and L. Trevisan, *Electronic Colloquium on Computational Complexity*, Vol. 64, 2002.
20. [CG04] A lower bound for testing juntas, H. Chockler and D. Gutfreund, *Information Processing Letters*, Vol. 90, 2004, pp. 301-305.
21. [CS01a] Property testing with geometric queries, A. Czumaj and C. Sohler, Proceedings of the 9th Annual European Symposium on Algorithms (ESA), 2001, pp. 266-277.

Further readings (cont'd)

22. [CS01b] Testing hypergraph coloring, A. Czumaj and C. Sohler, *Theoretical Computer Science*, Vol. 331, 2001, pp. 37-52.
23. [CS02] Abstract combinatorial programs and efficient property testers, A. Czumaj and C. Sohler, FOCS'02, pp. 83-92.
24. [CSZ00] Property testing in computational geometry, A. Czumaj, C. Sohler and M. Ziegler, Proceedings of the 8th Annual European Symposium on Algorithms (ESA), 2000, pp. 155-166.
25. [DGLRRS99] Improved testing algorithms for monotonicity, Y. Dodis, O. Goldreich, E. Lehman, S. Raskhodnikova, D. Ron and A. Samorodnitsky, RANDOM-APPROX'99, pp. 97-108.
26. [EKKRV00] Spot-Checkers, F. Ergün, S. Kannan, R. Kumar, R. Rubinfeld and M. Vishwanathan, *Journal of Computer and System Sciences*, Vol. 60, 2000, pp. 717-751.
27. [EKR03] Fast approximate probabilistic checkable proofs, F. Ergün, R. Kumar and R. Rubinfeld, *Information and Computation*, Vol. 189, 2004, pp. 135-159.
28. [F01] On the strength of comparisons in property testing, E. Fischer, *Electronic Colloquium on Computational Complexity*, Vol. 8, 2001.

Further readings (cont'd)

29. [F04] On the strength of comparisons in property testing, E. Fischer, *Information and Computation*, Vol. 189, 2004, pp. 107-116.
30. [F05] Testing graphs for colorability properties, E. Fischer, *Random Structures and Algorithms*, Vol. 25, 2005, pp. 289-309.
31. [FKRSS04] Testing juntas, E. Fischer, G. Kindler, D. Ron, S. Safra and A. Samorodnitsky, *Journal of Computer and System Sciences*, Vol. 68, 2004, pp. 103-112.
32. [FLNRRS02] Monotonicity testing over general poset domains, E. Fischer, E. Lehman, I. Newman, S. Raskhodnikova, R. Rubinfeld and A. Samorodnitsky, STOC'02, pp. 474-483.
33. [FM06] Testing graph isomorphism, E. Fischer and A. Matsliah, SODA'06, pp. 299-308.
34. [FN01] Testing of matrix properties, E. Fischer and I. Newman, STOC'01, pp. 286-295.
35. [GGLRS00] Testing monotonicity, O. Goldreich, S. Goldwasser, E. Lehman, D. Ron and A. Samorodnitsky, *Combinatorica*, Vol. 20, 2000, pp. 301-337.
36. [GGR98] Property testing and its connection to learning and approximation, O. Goldreich, S. Goldwasser and D. Ron, *Journal of the ACM*, Vol. 45, 1998, pp. 653-750.

Further readings (cont'd)

37. [GR02] Property Testing in Bounded Degree Graphs, O. Goldreich and D. Ron, *Algorithmica*, Vol. 32, 2002, pp. 302-343.
38. [GR99] A Sublinear Bipartiteness Tester for Bounded Degree Graphs, O. Goldreich and D. Ron, *Combinatorica*, Vol. 19, 1999, pp. 335-373.
39. [GR04] On estimating the average degree of a graph, *Electronic Colloquium on Computational Complexity*, Vol. 11, 13, 2004.
40. [GT03] Three theorems regarding testing graph properties, O. Goldreich and L. Trevisan, *Random Structures and Algorithms*, Vol. 23, 2003, pp. 23-57.
41. [HK03] Distribution-free property testing, S. Halevy and E. Kushilevitz, RANDOM-APPROX'03, pp. 302-317.
42. [KKR04] Tight Bounds for Testing Bipartiteness in General Graphs, T. Kaufman, M. Krivelevich and D. Ron, *SIAM Journal on Computing*, Vol. 33, 2004, pp. 1441-1483.
43. [KMS03] Approximate testing with error relative to input size, M. Kiwi, F. Magniez and M. Santha, *Journal of Computer and System Sciences*, Vol. 66, 2003, pp. 371-392.
44. [KR00] Testing problems with sub-learning sample complexity, M. Kearns and D. Ron, *Journal of Computer and System Sciences*, Vol. 61, 2000, pp. 428-456.

Further readings (cont'd)

45. [KR00] Testing problems with sub-learning sample complexity, M. Kearns and D. Ron, *Journal of Computer and System Sciences*, Vol. 61, 2000, pp. 428-456.
46. [N02] Testing Membership in Languages that Have Small Width Branching Programs, I. Newman, *SIAM Journal on Computing*, Vol.31, 2002, pp. 251-258.
47. [PR02] Testing the diameter of graphs, M. Parnas, D. Ron, *Random Structures and Algorithms*, Vol. 20, 2002, pp. 165-183.
48. [PR03] Testing metric properties, M. Parnas and D. Ron, *Information and Computation*, Vol. 187, 2003, pp. 155-195.
49. [PRR03] Testing parenthesis languages, M. Parnas, D. Ron, R. Rubinfeld, *Random Structures and Algorithms*, Vol. 22, 2003, pp. 98-138.
50. [PRR03] On Testing Convexity and Submodularity, M. Parnas, D. Ron and R. Rubinfeld, *SIAM Journal on Computing*, Vol. 32, 2003, pp. 1158-1184.
51. [PRS02] Testing basic Boolean formulas, M. Parnas, D. Ron and A. Samorodnitsky, *SIAM Journal on Discrete Mathematics*, Vol. 16, 2002, pp. 20-46.

Further readings (cont'd)

- Some good surveys are available on the following website:
  - http://theory.lcs.mit.edu/%7Eronitt/sublinear.html

- This powerpoint file can be downloaded from the following hyperlink:
  - http://www.cs.ccu.edu.tw/~lincc/research/randalg/slides/IntroductionToPropertyTesting.ppt

Thank you.