Pseudo-random graphs and bit probe schemes with one-sided error

Andrei Romashchenko
CNRS, LIF de Marseille & IITP of RAS (Moscow)

CSR 2011, June 14

The problem under consideration:

bit probe scheme with one-sided error

The problem under consideration:

bit probe scheme with one-sided error

Our technique:

pseudo-random graphs

Bit probe scheme with one-sided error

Bit probe scheme with one-sided error

Given: a set A from universe U

Bit probe scheme with one-sided error

Given: a set A from universe U

$n = |A| \ll m = |U|$,
e.g., $n = m^{0.01}$, $n = \mathrm{poly}\log m$, etc.

Bit probe scheme with one-sided error

Given: a set A from universe U

$n = |A| \ll m = |U|$,
e.g., $n = m^{0.01}$, $n = \text{poly}\log m$, etc.

To construct: a *database* B of size s such that

to answer a query

$$x \in A\,?$$

we need to read *one* bit from the database

Bit probe scheme with one-sided error

Given: a set A from universe U

$n = |A| \ll m = |U|$,
e.g., $n = m^{0.01}$, $n = \mathrm{poly}\log m$, etc.

To construct: a *database* B of size s such that

to answer a query

$$x \in A\ ?$$

we need to read *one* bit from the database

Goal: minimize $s = |B|$

Bit probe scheme with one-sided error

Given: a set A from universe U

$n = |A| \ll m = |U|$,
e.g., $n = m^{0.01}$, $n = \mathrm{poly}\log m$, etc.

To construct: a *database* B of size s such that

to answer a query

$$x \in A\ ?$$

we need to read *one* bit from the database

Goal: minimize $s = |B|$

Remark: $s = \Omega(n \log m)$

static structures for a set: standard solutions

Given: a set A from universe U

How to encode A?

static structures for a set: standard solutions

Given: a set A from universe U

How to encode A?

1. bit vector of size m

static structures for a set: standard solutions

Given: a set A from universe U

How to encode A?

1. bit vector of size m
    - good news: read one bit for a query "$x \in A$?"
    - good news: no randomization
    - bad news: too much memory

static structures for a set: standard solutions

Given: a set A from universe U

How to encode A?

1. bit vector of size m
   - good news: read one bit for a query "$x \in A$?"
   - good news: no randomization
   - bad news: too much memory
2. list of elements

static structures for a set: standard solutions

Given: a set A from universe U

How to encode A?

1. bit vector of size m
   - good news: read one bit for a query "$x \in A$?"
   - good news: no randomization
   - bad news: too much memory
2. list of elements
   - good news: memory $n \log m$
   - good news: no randomization
   - bad news: read too many bits to answer a query

static structures for a set: standard solutions

Given: a set A from universe U

How to encode A?

1. bit vector of size m
   - good news: read one bit for a query "$x \in A$?"
   - good news: no randomization
   - bad news: too much memory
2. list of elements
   - good news: memory $n \log m$
   - good news: no randomization
   - bad news: read too many bits to answer a query
3. Fredman–Komlós–Szemerédi (double hashing):

static structures for a set: standard solutions

Given: a set A from universe U

How to encode A?

1. bit vector of size m
   - good news: read one bit for a query "$x \in A$?"
   - good news: no randomization
   - bad news: too much memory
2. list of elements
   - good news: memory $n \log m$
   - good news: no randomization
   - bad news: read too many bits to answer a query
3. Fredman–Komlós–Szemerédi (double hashing):
   - good news: database of size $O(n \log m)$ bits
   - good news: randomization only to constructe the database
   - bad news: need to read $O(\log m)$ bits to answer a query

Buhrman–Miltersen–Radhakrishnan–Venkatesh [2001]

Two features:

1. a *randomized* algorithm answers a query "$x \in A$?"
2. a scheme based on a highly unbalanced expander

Buhrman–Miltersen–Radhakrishnan–Venkatesh [2001]

Two features:

1. a *randomized* algorithm answers a query "$x \in A$?"
2. a scheme based on a highly unbalanced expander

- good news: read *one* bit to answer a query
- good news: memory $= O(n \log m)$
- bad news: exponential computations
- *some* news: two-sided errors
- bad news: need $\Omega(\frac{n^2 \log m}{\log n})$ for a one-sided error

Bit-probe scheme in this paper:

- read *one* bit to answer a query
- memory $= O(n \log^2 m)$

Bit-probe scheme in this paper:

- read *one* bit to answer a query
- memory $= O(n \log^2 m)$ vs $O(n \log m)$ in [BMRV]
- computations in $\mathrm{poly}(m)$

Bit-probe scheme in this paper:

- read *one* bit to answer a query
- memory $= O(n \log^2 m)$ vs $O(n \log m)$ in [BMRV]
- computations in $\mathrm{poly}(m)$ vs $\exp\{m\}$ in [BMRV]
- one-sided error

Bit-probe scheme in this paper:

- read *one* bit to answer a query
- memory $= O(n \log^2 m)$ vs $O(n \log m)$ in [BMRV]
- computations in $\mathrm{poly}(m)$ vs $\exp\{m\}$ in [BMRV]
- one-sided error vs two-sided in [BMRV]

Bit-probe scheme in this paper:

- read *one* bit to answer a query
- memory $= O(n \log^2 m)$ vs $O(n \log m)$ in [BMRV]
- computations in $\mathrm{poly}(m)$ vs $\exp\{m\}$ in [BMRV]
- one-sided error vs two-sided in [BMRV]
- memory $= O(n \log^2 m)$ better than $\Omega(\frac{n^2 \log m}{\log n})$!

Bit-probe scheme in this paper:

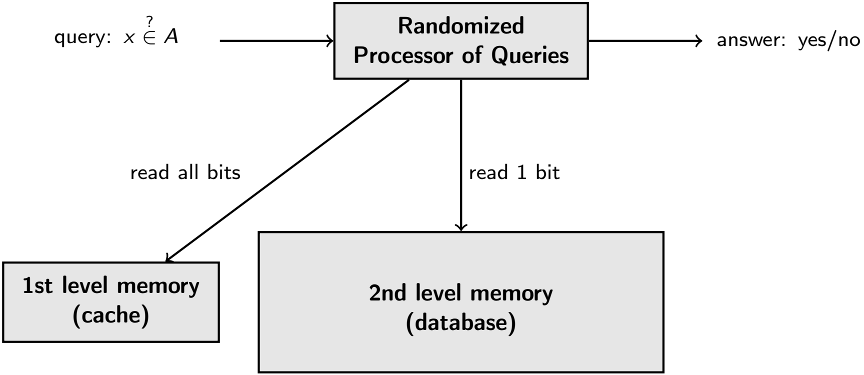
- read *one* bit to answer a query
- memory $= O(n \log^2 m)$ vs $O(n \log m)$ in [BMRV]
- computations in $\mathrm{poly}(m)$ vs $\exp\{m\}$ in [BMRV]
- one-sided error vs two-sided in [BMRV]
- memory $= O(n \log^2 m)$ better than $\Omega(\frac{n^2 \log m}{\log n})$!

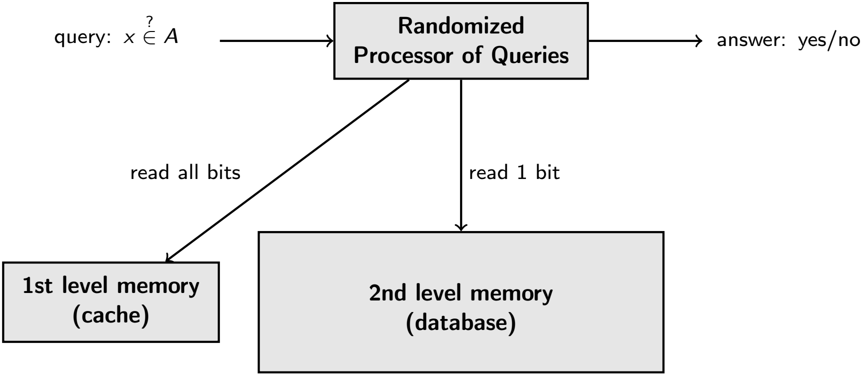
Do we cheat ?

Bit-probe scheme in this paper:

- read *one* bit to answer a query
- memory = $O(n \log^2 m)$ vs $O(n \log m)$ in [BMRV]
- computations in $\mathrm{poly}(m)$ vs $\exp\{m\}$ in [BMRV]
- one-sided error vs two-sided in [BMRV]
- memory = $O(n \log^2 m)$ better than $\Omega(\frac{n^2 \log m}{\log n})$!

Do we cheat ? Yes, we have changed the model !
We allow *cached* memory of size $\mathrm{poly}(\log m)$.

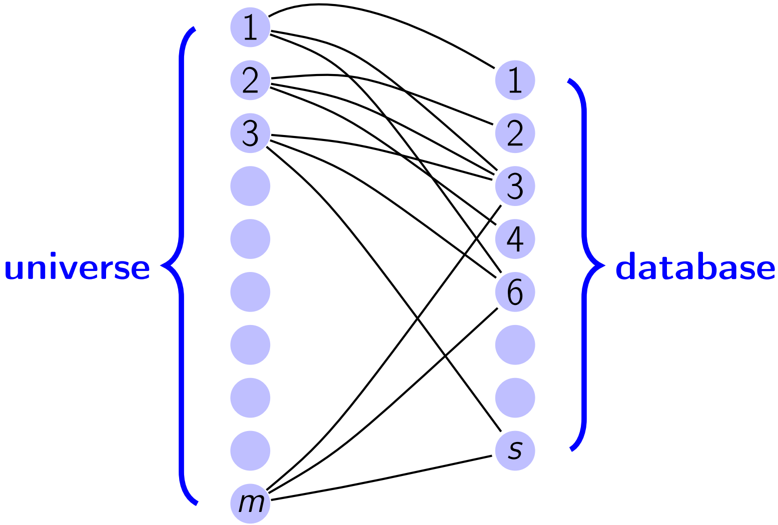

query: $x \stackrel{?}{\in} A$
Randomized
Processor of Queries
answer: yes/no
read all bits
read 1 bit
1st level memory
(cache)
2nd level memory
(database)



Theorem. For any n-element set A from an m-element universe there exists a randomized bit-probe scheme with one-sided error, with cache of size $O(\log^c m)$ and database of size $O(n \log^2 m)$.

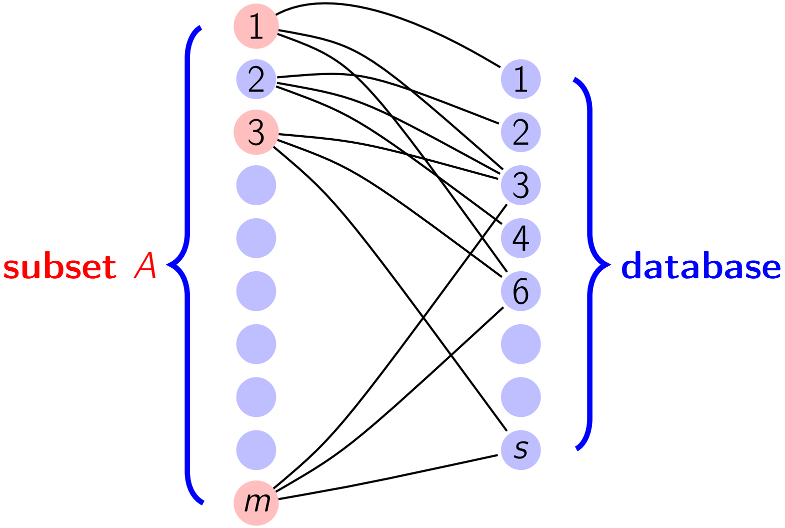
the left part: m vertices; degree $d = O(\log m)$

the right part: $s = O(n \log^2 m)$ vertices

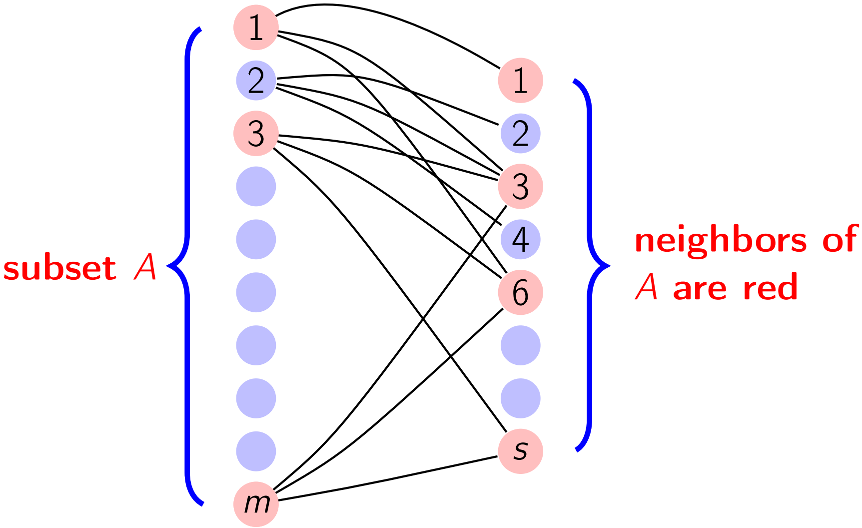


in the left part: set A of n vertices

the right part: $s = O(n \log^2 m)$ vertices



A graph is suitable for A if
for every $x \notin A$ **most** neighbors of x are blue



A graph is called suitable for A if for every $x \notin A$ *most* neighbors of x are blue

A graph is called suitable for A if for every $x \notin A$ *most* neighbors of x are blue

Good news: for every A most graphs are suitable.

Bad news: there is no graph suitable for *every* sets A.

A graph is called suitable for A if for every $x \notin A$ *most* neighbors of x are blue

Good news: for every A most graphs are suitable.

Bad news: there is no graph suitable for *every* sets A.

1st idea: take a *random* graph and *cache* it

A graph is called suitable for A if for every $x \notin A$ *most* neighbors of x are blue

Good news: for every A most graphs are suitable.

Bad news: there is no graph suitable for *every* sets A.

1st idea: take a *random* graph and *cache* it

we cannot, a random graph is too large!

A graph is called suitable for A if for every $x \notin A$ *most* neighbors of x are blue

Good news: for every A most graphs are suitable.

Bad news: there is no graph suitable for *every* sets A.

1st idea: take a *random* graph and *cache* it

we cannot, a random graph is too large!

2nd idea: take a *pseudo-random* graph, cache the *seed*

A graph is called suitable for A if for every $x \notin A$ *most* neighbors of x are blue

Good news: for every A most graphs are suitable.

Bad news: there is no graph suitable for *every* sets A.

1st idea: take a *random* graph and *cache* it

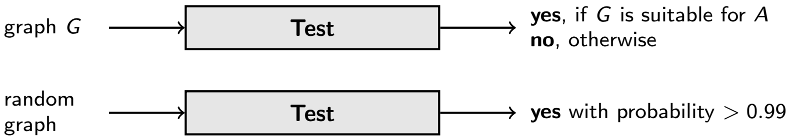
we cannot, a random graph is too large!

2nd idea: take a *pseudo-random* graph, cache the *seed*

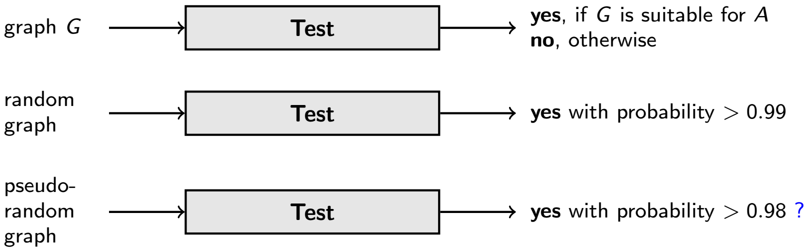
We need a good PRG...

Fix a set A.

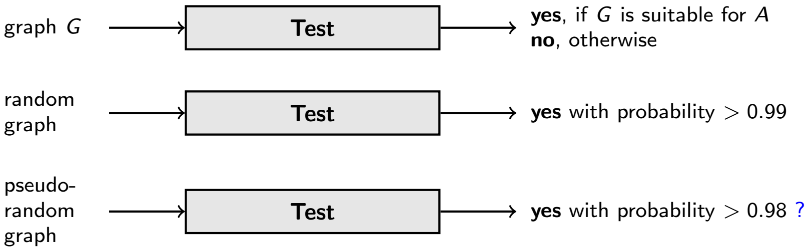
Fix a set A.


graph G
Test
yes, if G is suitable for A
no, otherwise
random graph
Test
yes with probability > 0.99

Fix a set A.

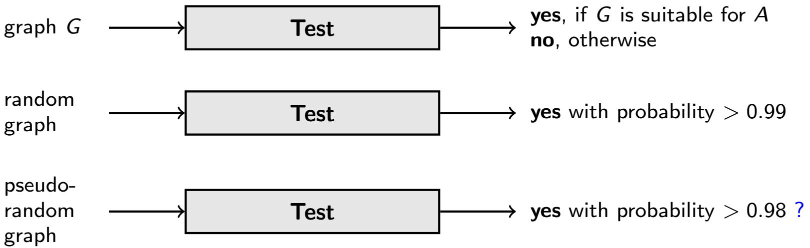

graph G
Test
yes, if G is suitable for A
no, otherwise
random graph
Test
yes with probability > 0.99
pseudo-random graph
Test
yes with probability > 0.98 ?

Fix a set A.



Test is an AC^0-circuit.

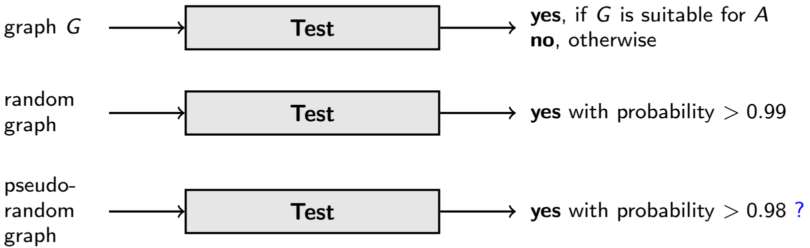
Fix a set A.



Test is an AC^0-circuit.

- Nisan–Wigderson generator is valid

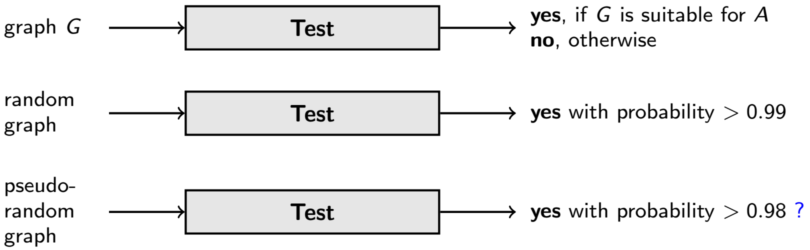
Fix a set A.



Test is an AC^0-circuit.

- Nisan–Wigderson generator is valid
- Braverman: every $(\mathrm{poly} \log m)$-independent function is valid

Fix a set A.



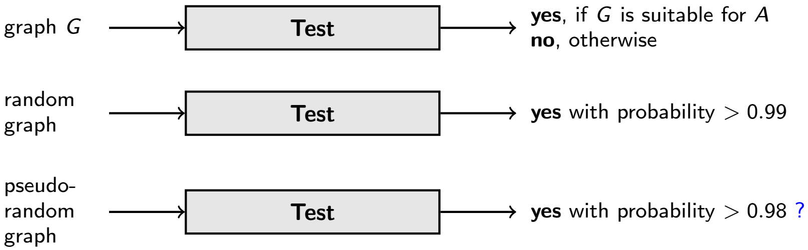
Test is an AC^0-circuit.

- Nisan–Wigderson generator is valid
- Braverman: every $(\mathrm{poly} \log m)$-independent function is valid

Test is a FSM with small memory.

- Nisan's generator is valid

Fix a set A.



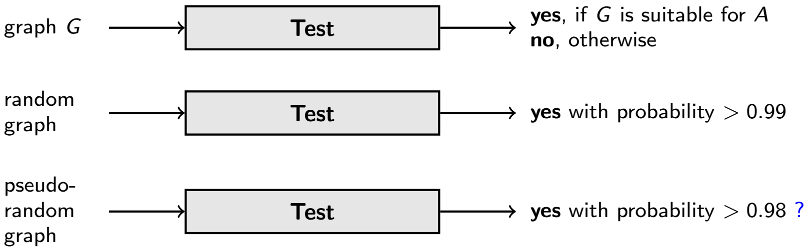
Test is an AC^0-circuit.

- Nisan–Wigderson generator is valid
- Braverman: every $(\mathrm{poly} \log m)$-independent function is valid

Test is a FSM with small memory.

- Nisan's generator is valid

Fix a set A.



Test is an AC^0-circuit.

- Nisan–Wigderson generator is valid
- Braverman: every $(\mathrm{poly} \log m)$-independent function is valid

Test is a FSM with small memory.

- Nisan's generator is valid

Size of the seed $= \mathrm{poly}(\log m)$.

Conclusion: a **bit-probe scheme:**

- read *one* bit to answer a query
- one-sided error
- 1-st level "cached" memory $= \mathrm{poly}\log m$
- 2-nd level memory $= O(n \log^2 m)$

Conclusion: a **bit-probe scheme:**

- read *one* bit to answer a query
- one-sided error
- 1-st level "cached" memory $= \mathrm{poly}\log m$
- 2-nd level memory $= O(n\log^2 m)$
- database is prepared in time $\mathrm{poly}(m)$

Conclusion: a **bit-probe scheme:**

- read *one* bit to answer a query
- one-sided error
- 1-st level "cached" memory $= \mathrm{poly}\log m$
- 2-nd level memory $= O(n \log^2 m)$
- database is prepared in time $\mathrm{poly}(m)$

Beyond this talk: combine our construction with Guruswami–Umans–Vadhan

- read *two* bit to answer a query
- one-sided error
- 1-st level "cache" memory $= \mathrm{poly}\log m$
- 2-nd level memory $= n^{1+\delta}\mathrm{poly}\log m$
- computations in time $\mathrm{poly}(n, \log m)$

What this talk is about:

- a pseudo-random graph can be better than an explicit one (better expansion parameters, etc.)

What this talk is about:

- a pseudo-random graph can be better than an explicit one (better expansion parameters, etc.)
- a pseudo-random graph can be better than a "truly random" graph (shorter description)

What this talk is about:

- a pseudo-random graph can be better than an explicit one (better expansion parameters, etc.)
- a pseudo-random graph can be better than a "truly random" graph (shorter description)
- a small "cache" may be useful in static structures

What this talk is about:

- a pseudo-random graph can be better than an explicit one (better expansion parameters, etc.)
- a pseudo-random graph can be better than a "truly random" graph (shorter description)
- a small "cache" may be useful in static structures

Thank you! Questions?