Local generation of combinatorial objects

Amartya Shankha Biswas (MIT), Ronitt Rubinfeld (MIT), Anak Yodpinyanee (MIT)

Huge random objects:

How to generate?

Up front?

Locally…on the fly?

Generating large random graph

	1	2	3	4	5	6	7	8	9	10
1			0	1	0		0			
2			0	1	0		0	1		1
3	0	0		0	0	0	0	0	1	
4	1	1	0		0		0	1		
5	0	0	0	0		1	1			
6			0		1				0	
7	0	0	0	0	1					
8		1	0	1						0
9			1			0				
10		1						0		

Generate "on the fly"?

What if *d*-regular? support "next-neighbor" queries?

A challenge:
How to handle dependencies?

Sources of dependencies:

Model, supported queries,…

Models

Two models for random generation of graphs

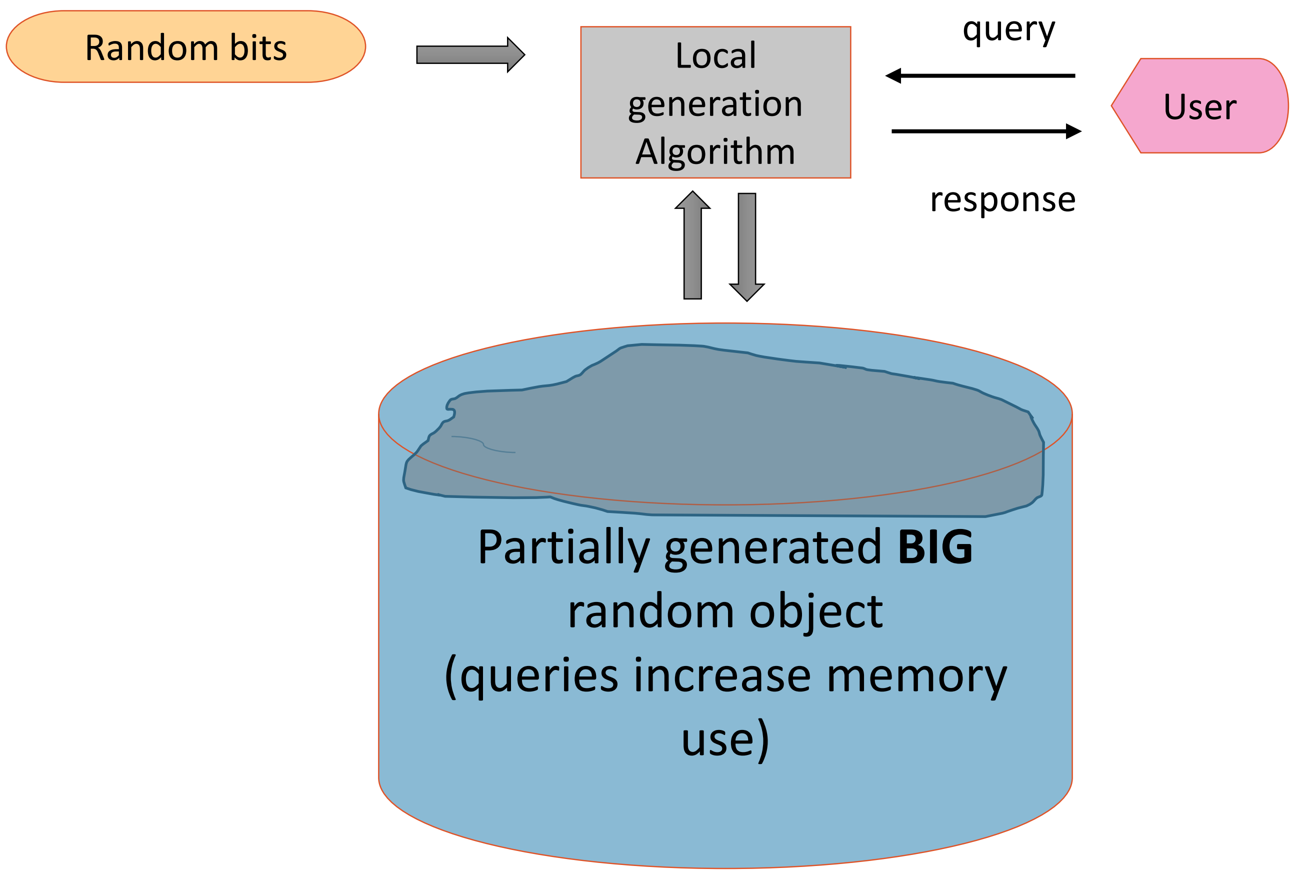
Huge pseudo-random graphs/objects [Goldreich Goldwasser Nussboim]

- Huge = exponential size
- User will not query more than poly locations
- May be sufficient to generate graph that "looks" random to poly time algorithm?

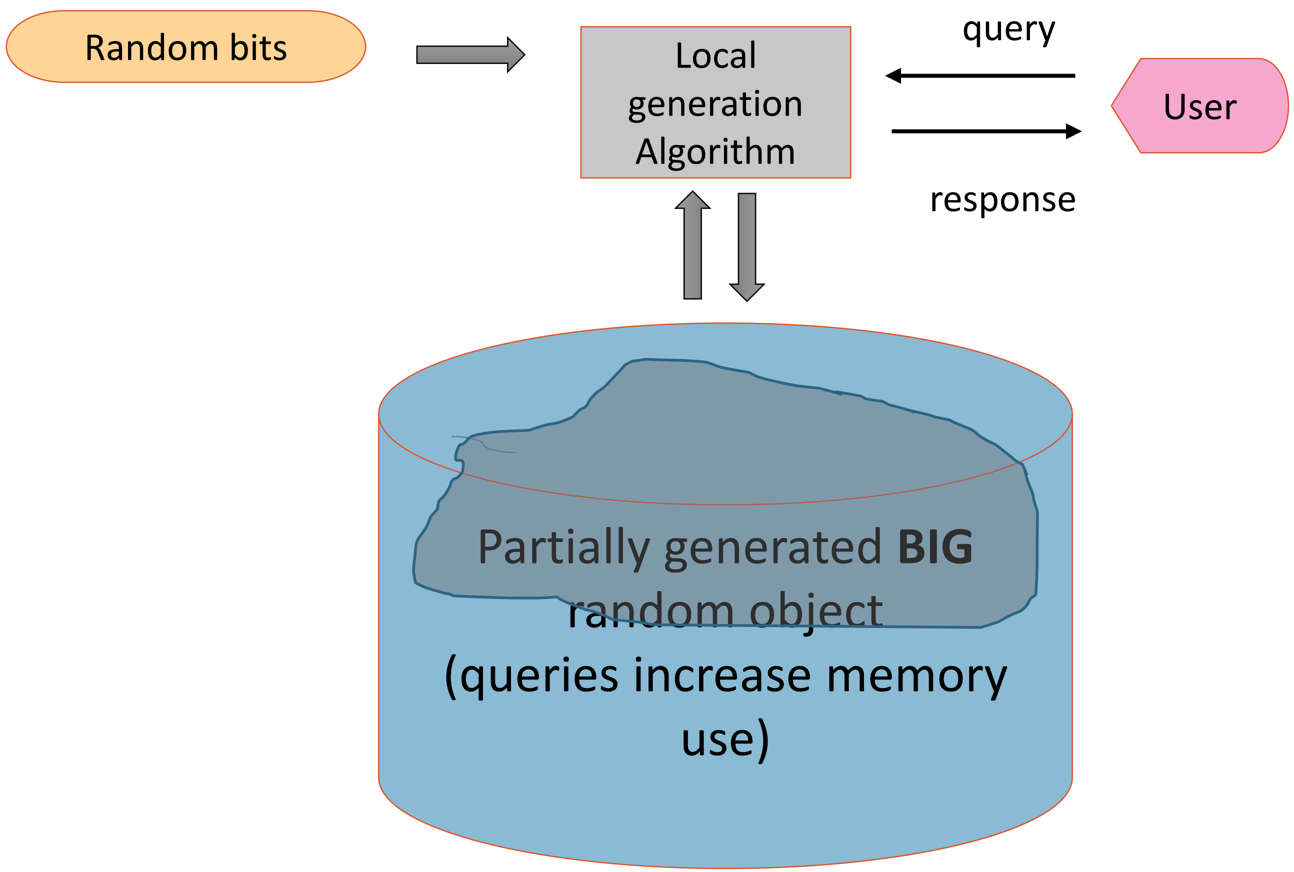
Big random graphs/objects [Even Levi Medina Rosen] [Biswas R Yodpinyanee]

- Big = poly size
- Might eventually write down the whole graph, but don't want to pay cost up-front
- End result should be random according to the claimed process

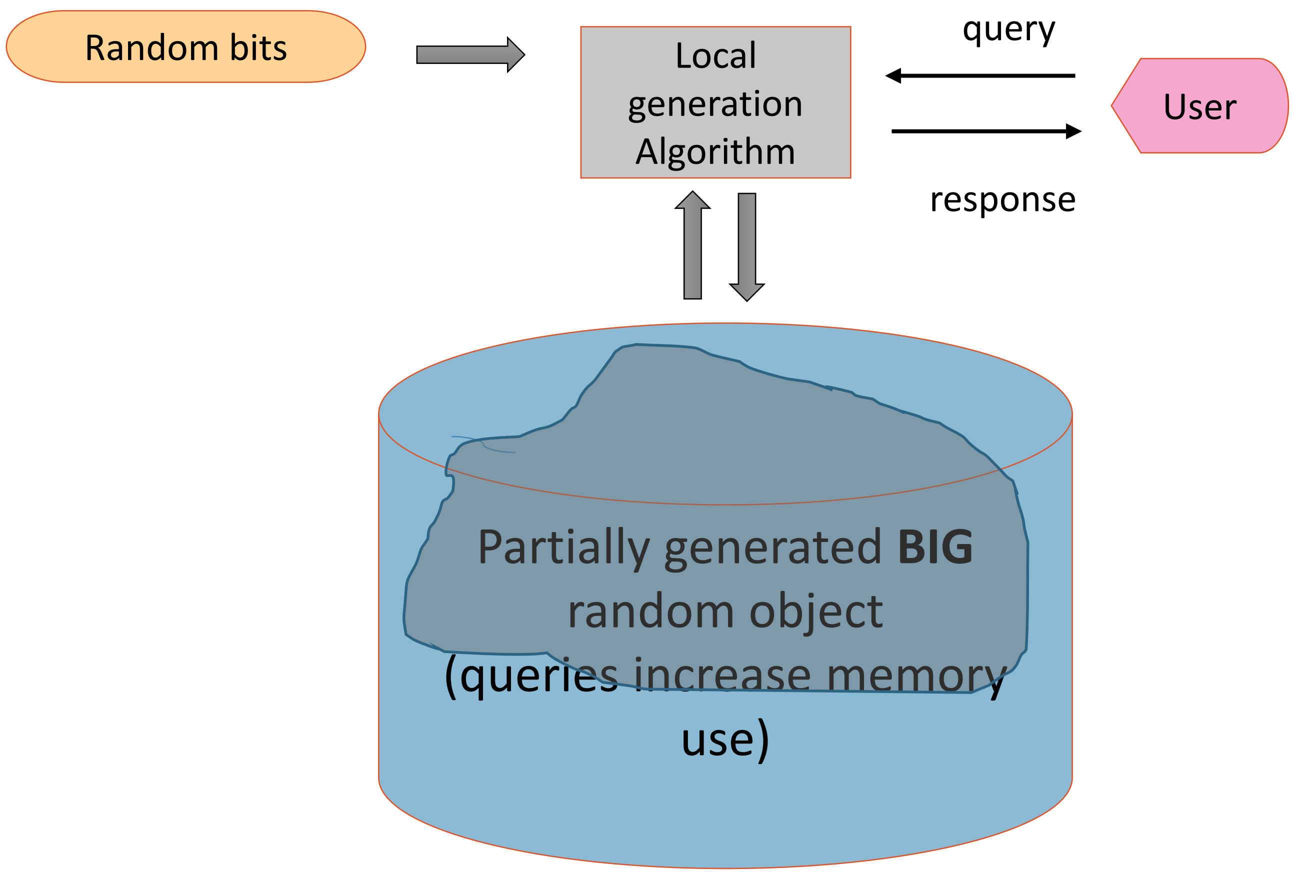
“On the fly” Sampler
[ELMR] [BRY]



“On the fly” Sampler



“On the fly” Sampler



Desiderata:

- Efficiency:
  - Answer in sublinear (polylogarithmic?) time
- Distribution equivalence:
  - Output distribution ϵ-close (ℓ_1-distance) to goal distribution

Possible queries on graphs:

- Vertex-pair (adjacency): Is edge (u,v) present?
- All-Neighbors: What are all neighbors of u?
- Degree: What is degree(u)?
- ith neighbor: What is ith neighbor of u?
- Next-neighbor: What is next neighbor of u?
- Random-neighbor: Output random neighbor of u?

considered by [GGN] [NN]

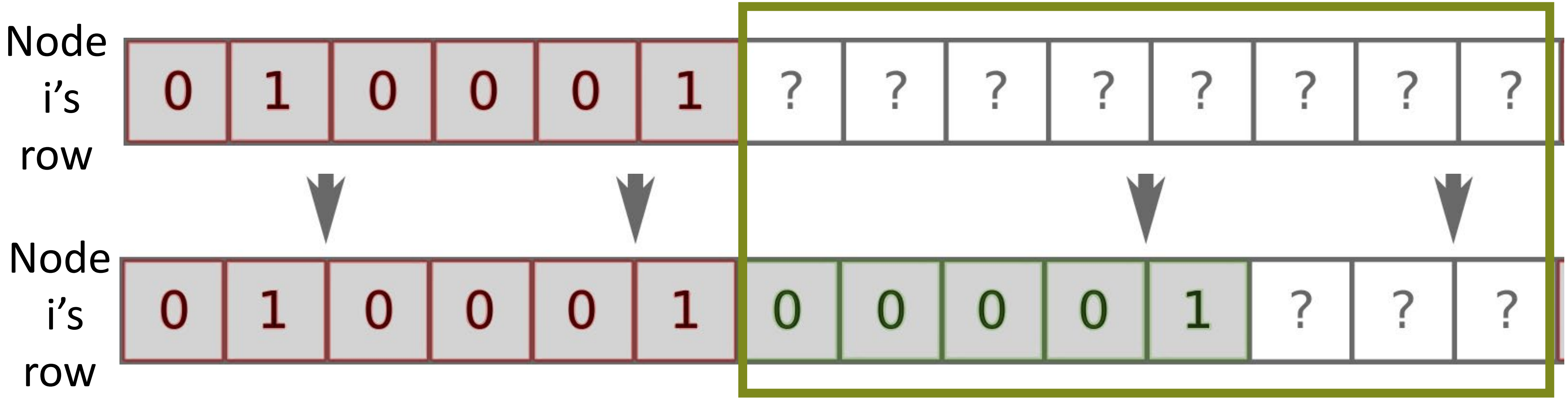
[Even Levi Medina Rosen 2017]

[Biswas R Yodpinyanee]

can take random walk in large degree (random) graph!

$G(n,p)$ graphs

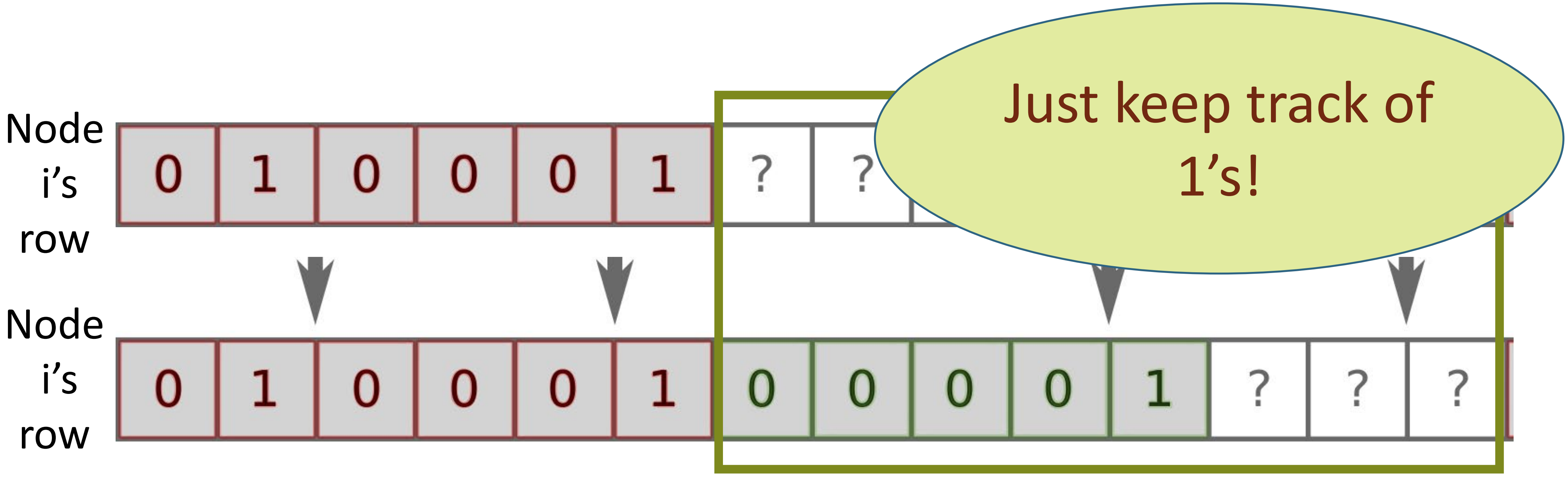
Dense G(n,p) next-neighbor queries:



Algorithm idea:

Toss coins to fill in empty entries until toss a 1

Next-neighbor queries: directed graphs



Algorithm idea:

Pick length of *0*-run according to hypergeometric distribution (via bina

$$\sum_{k=0}^{b-a-1} p\,(1-p)^k \;= 1 \;-(1-p)^{b-a}$$

Fill in next entry *(i, j+k)* with a *1*

need to write all 0s?

Next-Neighbor Query: what is u's next neighbor?

Dense case: $p \geq 1/poly(\log n)$

- Algorithm:
  - Start at last found neighbor
  - Go down row, flipping coins to fill empty entries, until find neighbor.
- Time $O(1/p)$.

Can we do $o(1/p)$ for $p = o(1)$?

Sparse case: $p \leq poly(\log n)/n$

- Algorithm: Use "all neighbor" query [Naor Nussboim 07]
- Time $O(E[degree]) = O(polylog\, n)$

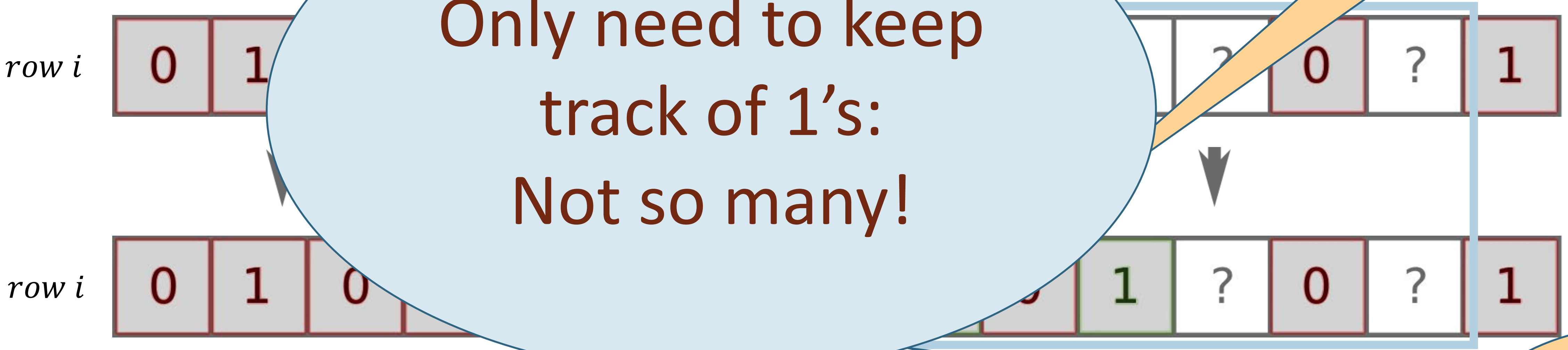
Intermediate case: (e.g. $p = \frac{1}{\sqrt{n}}$)

- "run length encoding" Idea: Sample *length of 0's run* according to hypergeometric distribution $p(1-p)^i$
- Challenge: some entries already filled in!

For next-neighbor queries
Undirected graphs

some are determined by other neighbor?

yields correct distribution?



Only need to keep track of 1's: Not so many!

need to write all 0s?

Algorithm idea:

Pick length of *0*-run ... ution:

$\sum_{k=}^{b}$ ...

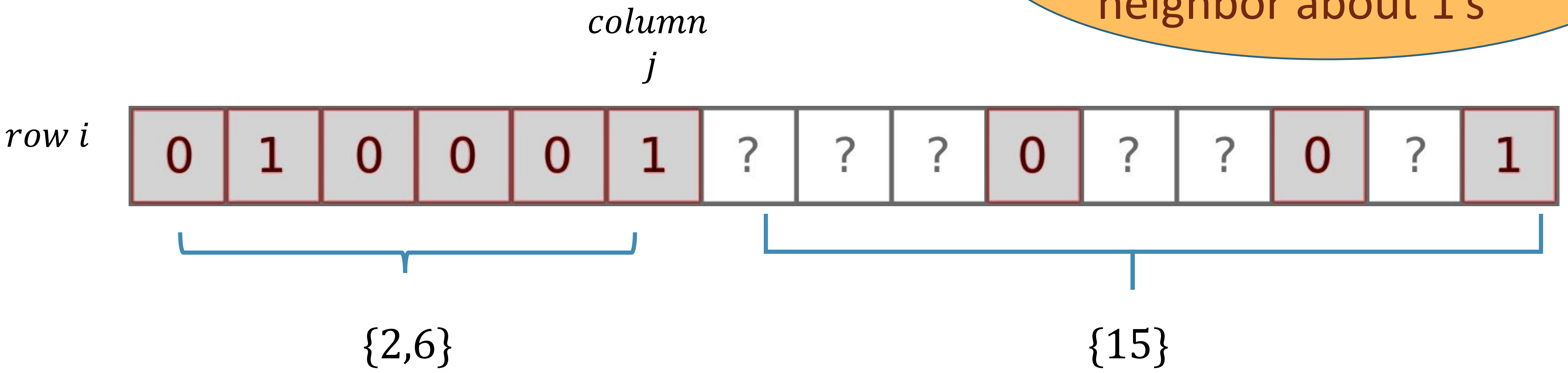
Fill in next entry *(i, j+k)*

Rule: first flip of edge (u,v) is what counts

Implementation of next neighbor queries: (assume no adjacency queries)

- For each node *i* maintain:
    1. last seen neighbor *j* (row entries *1..j* are determined, and *j* is a “1”)
    2. list of “1”s coming before *j (everything else is “0”)*
    3. remaining“1”s via min-heap
    4. Keep track of “0”s on row implicitly

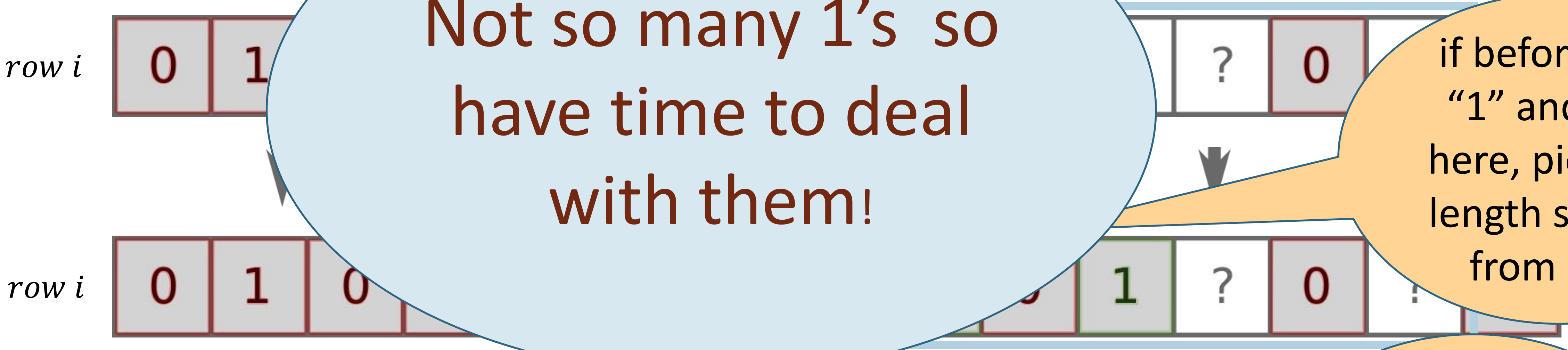
Only keep track of 1’s + notify other neighbor about 1’s



Skip-sampling for next-

some are determined by other neighbor?

if “1”, then neighbor has told i about it



Not so many 1’s so have time to deal with them!

if before next “1” and land here, pick new length starting from here

why correct distribution?

Algorithm:

- pick *k* according to ge
- If *j+k* > next *1* in *i’s*
- else if *(i,j+k)* previou
  - else add *(i, j+k)* to heaps

Rule: first flip of edge (u,v) is what counts

Random-Neighbor Query: output random neighbor of i

Dense case: $p \geq 1/poly(\log n)$

- Algorithm:
  - repeat until find neighbor:
    - pick random j
    - do vertex pair query on (i, j)
- Time $O(1/p)$.

Can we do $o(1/p)$ for $p = o(1)$?

Sparse case: $p \leq poly(\log n)/n$

- Algorithm: Use "all neighbor" query [Naor Nussboim 07]
- Time $O(E[degree]) = O(polylog\, n)$

Intermediate case: (e.g. $p = \frac{1}{\sqrt{n}}$)

???

we don't even know degree?

Implementation of Random-Neighbor queries via Bucketing and skip-sampling

Plan: **Equipartition** each row into **contiguous buckets** such that:

Expected # of neighbors in a bucket is a constant
⇒ w.h.p. 1/3 of buckets are non-empty
⇒ w.h.p. no bucket has more than log n neighbors

(drumroll…)
⇒ can write down all $\log n$ neighbors for each bucket! (assuming you can figure them out)

How many buckets?

pn, each of size $1/p$

Note that both size and number of buckets can be big

Random Neighbors with rejection sampling

Bucketing: expected #neighbors in a bucket $= \Theta(1)$ expected, $\leq \mathcal{O}(\log n)$ w.h.p. $\Rightarrow$ #neighbors $\approx$ #buckets

$v :$ $\underbrace{\boxed{\ } \boxed{0} \boxed{\ } \boxed{1} \boxed{\ } \boxed{\ }}$ $\boxed{0} \boxed{\ } \boxed{\ } \boxed{0} \boxed{1} \boxed{\ } \boxed{\ }$ $\cdots$ $\boxed{\ } \boxed{\ } \boxed{1} \boxed{\ } \boxed{\ } \boxed{0} \boxed{\ }$ $\cdots$

Keep list of 1's, then can pick nbr quickly

Step 1 pick a uniform random bucket
"fill" this bucket if needed

$\boxed{0} \boxed{0} \boxed{1} \boxed{0} \boxed{1} \boxed{1} \boxed{0} \boxed{0}$

Step 2 pick a uniform random neighbor u

$\rightarrow$ **return** or **reject**

Step 3 return u with probability $\frac{\#\text{neighbors in the bucket}}{\mathcal{O}(\log n)}$
otherwise, try again

$$\mathbb{P}[\text{return } u] = \frac{1}{\#\text{buckets}} \times \frac{1}{\#\text{neighbors in bucket}} \times \frac{\#\text{neighbors in bucket}}{\mathcal{O}(\log n)} \approx \frac{\Omega(1/\log n)}{\#\text{neighbors of } v}$$

$$\mathbb{P}[\text{return any neighbor}] \approx \Omega(1/\log n) \Rightarrow \mathcal{O}(\log n) \text{ iterations suffice}$$

How to fill a bucket?

- Bucket may be ***indirectly*** filled in certain locations
  - "1" entries reported when created
  - "0" entries not reported but can query from complementary bucket

?	1	?	?	?	0	?	1	0	?	?

- First, fill bucket ignoring existing entries

0	0	1	0	0	1	1	0	0	1	0

- Fix to conform to “first flip”:
  - Re-insert all *indirectly filled* (red) "1" entries: {2,8}
  - For each new (green) "1" entry: remove if coincides with indirectly filled "0" entries

0	1	1	0	0	~~1~~	1	1	0	1	0

Graph models supporting typical graph queries:

$G(n,p)$

Community structure: Stochastic Block Model

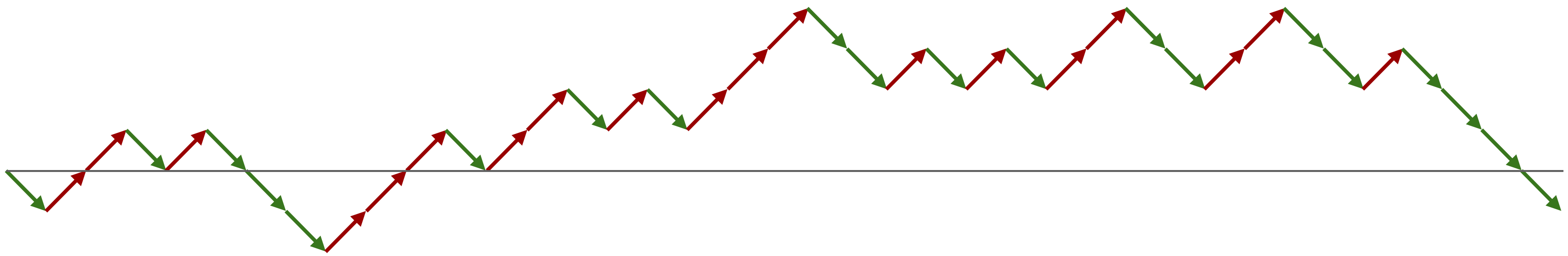
Small world graphs

Random walks

Large 1D Random Walk (on the line)

with probability **1/2**

with probability **1/2**



What if we only care about a few positions?

Query **Height(t)** returns position of walk at time **t**

with probability **1/2**

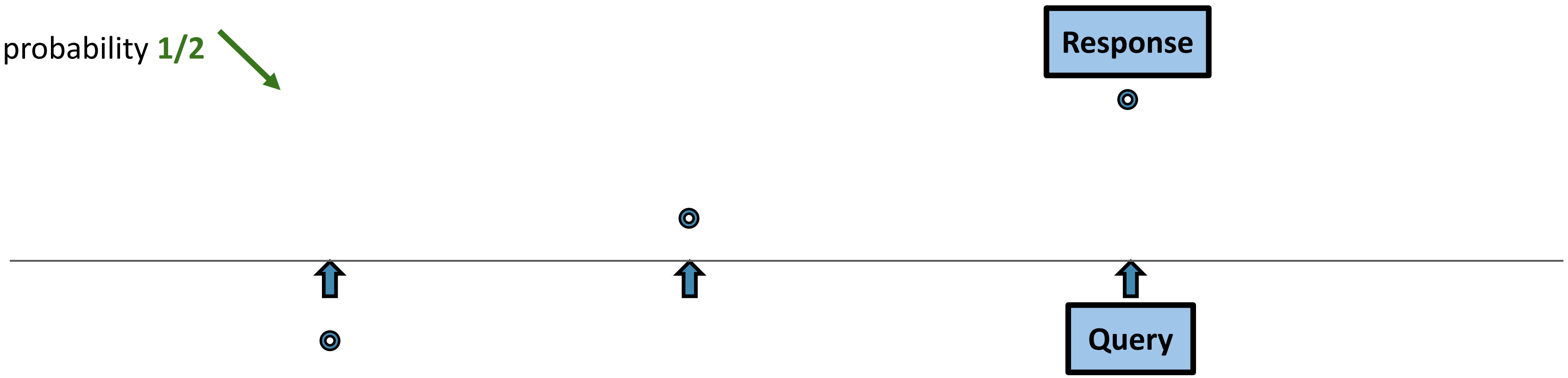
with probability **1/2**

What if we only care about a few positions?

Query **Height(t)** returns position of walk at time **t**

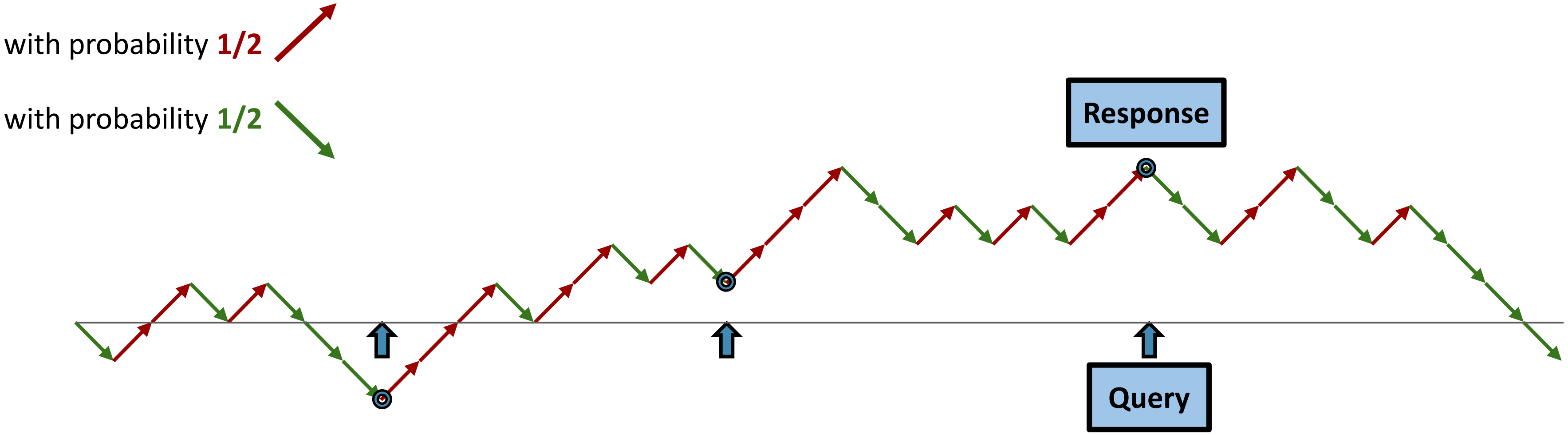
with probability **1/2**

with probability **1/2**



Queries appear in arbitrary order

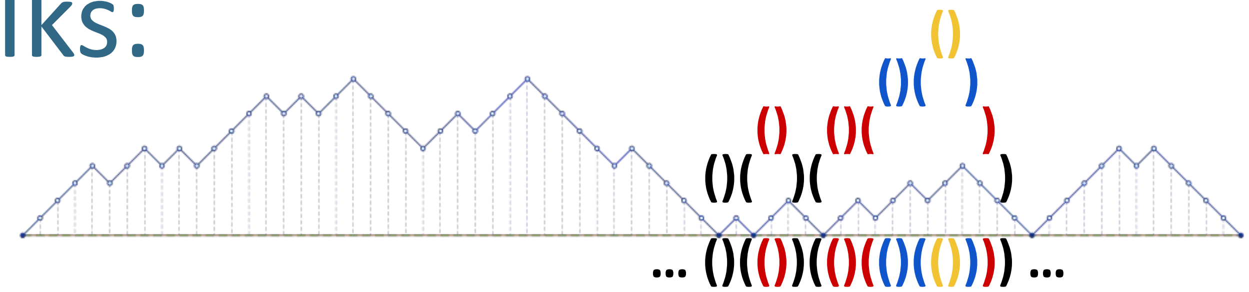
Consistent with Large 1D Random Walk



Queries appear in arbitrary order

local generation of hypergeometric distribution
[Gilbert Ghuha Indyk Kotidis Muthukrishnan Strauss]
[Goldreich Goldwasser Nussboim]

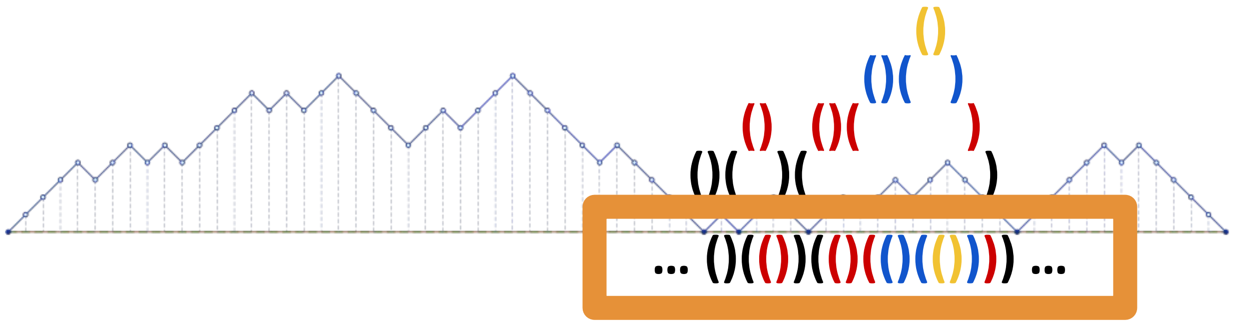
Random walks:



- **Random walks on the line**
- **Random Catalan objects**
  - Random Dyck paths
  - Well bracketed expressions
  - Random Rooted Trees

[Biswas R Yodpinyanee]

Polylogarthmic time queries:

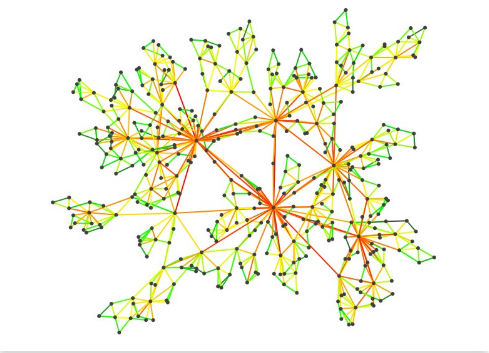


- **Random walks on the line**
- **Random Catalan objects**
  - Random Dyck paths
  - Well bracketed expressions
  - Random Rooted Trees

- **Height** queries
  - **Bracket-Nesting-Depth** queries
- **First-Return** queries
  - **Matching-Bracket** queries

[Biswas R Yodpinyanee]

Random walks on graphs
[Biswas Pyne R]



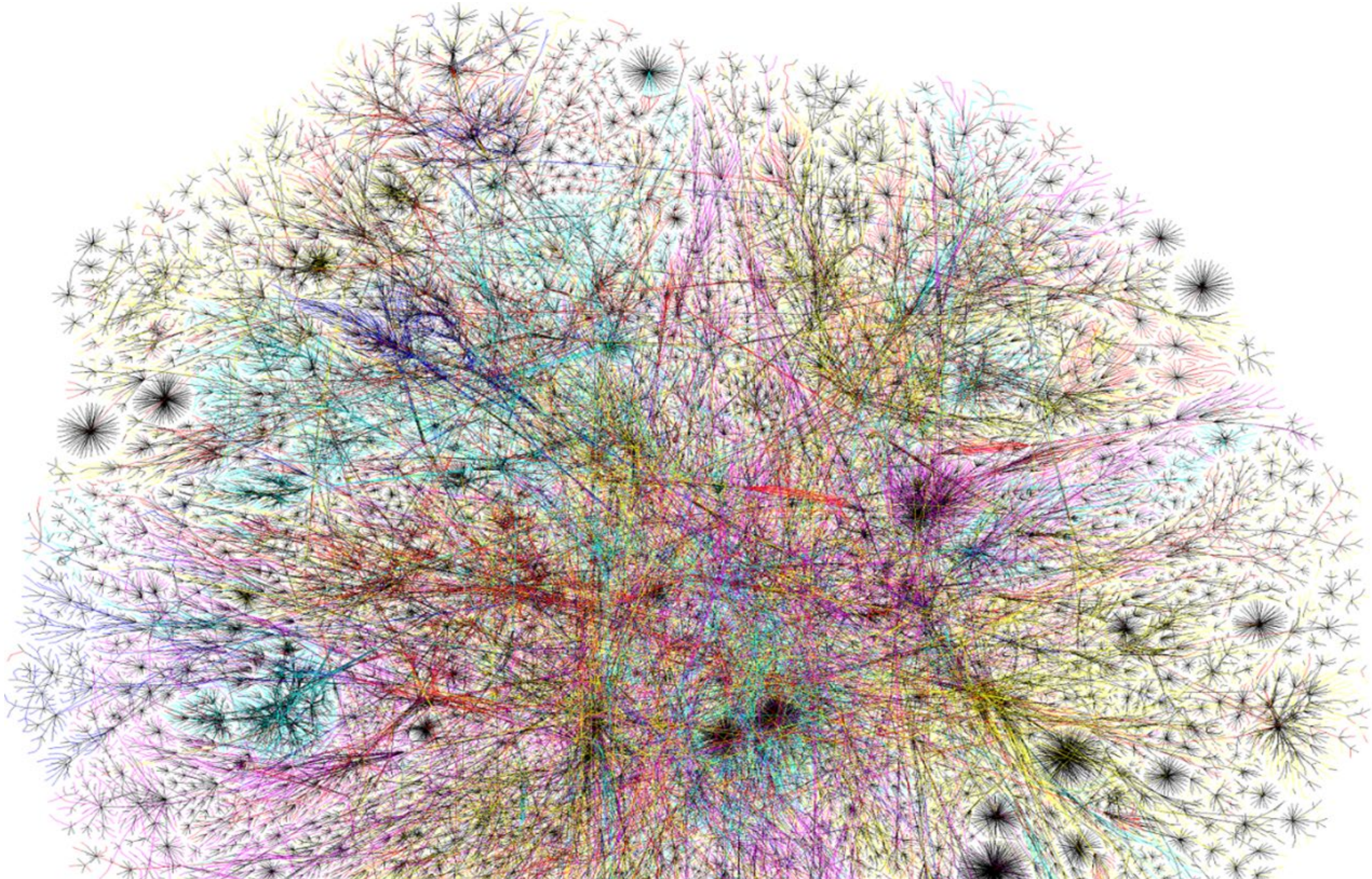
- Given G, start vertex s, what is location of random walk at time t?

- Query time upper bounds:
  - Polylog time for hypercube, cycle, Cayley graphs, structured graphs (tensor and Cartesian products)
  - $\tilde{O}\left(\frac{1}{1-\lambda}\sqrt{n}\right)$ for spectral expansion λ

- Lower bound: $\Omega(\sqrt{n})$ for random graphs

https://graphstream-project.org

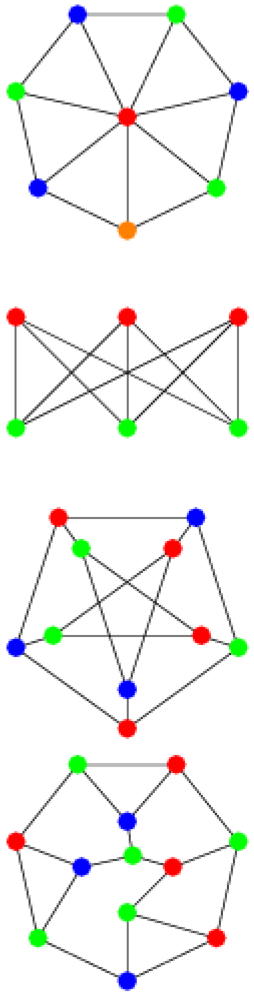
Generating Random Colorings of Large Graphs



Random Colorings of Large Graphs

- Input graph: G
  - Maximum Degree: Δ
  - Number of colors: $q > \Delta$ (here $q > 12\,\Delta$)
- Output: Uniformly random valid coloring of G
- Query: Color of node *v*?

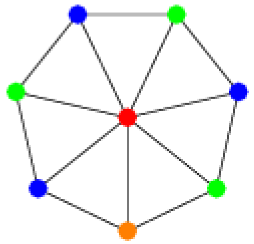
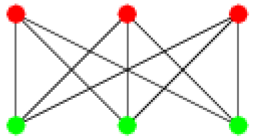
Sublinear probes to G?



First try

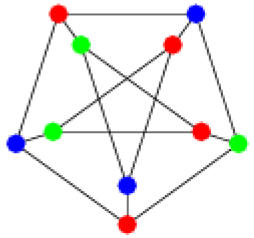
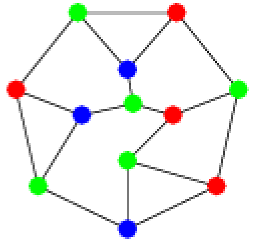
- Basic (sequential) Markov Chain for $q > 2\Delta$ [Jerrum]:
  - Random node *v* picks random color
  - Update *v* to new color if no conflict with neighbors

$O(n \log n)$ steps
(sequential)

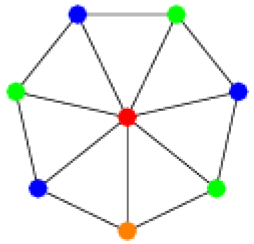



- On query Color(*v*,*t*) = Color of node *v* at time *t*
  - When was v last picked? which color did it choose? Conflict?
  - For all *w* nbr of *v*: color of *w* at that time?
    - Query *w*'s previous random choice
    - Colors of *w*'s neighbors at that time?

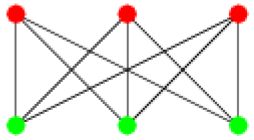
$\Omega(\Delta^t)$? $\Omega\left(\Delta^{\log n}\right)$? $\Omega(n)$?

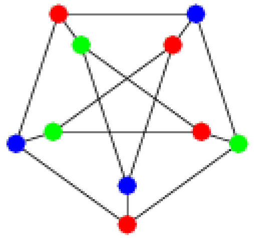
Modified Glauber Dynamics



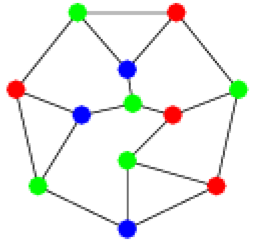
- Distributed Markov Chain round [Feng Sun Yin] [Fischer Ghaffari] [Feng Hayes Yin]:
  - *n* nodes simultaneously choose random colors "proposals"



  - Update color if
    - no conflict with any neighbor's current color or new proposal,



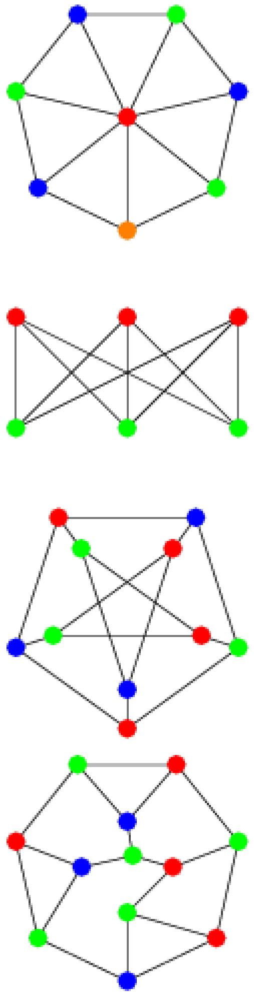
    - no neighbor proposal conflicts with current color



Need O(log n) rounds

Modified Glauber Dynamics

- Distributed Markov Chain round [Feng Sun Yin] [Fischer Ghaffari] [Feng Hayes Yin]:
  - *n* nodes simultaneously choose random colors "proposals"
  - Update color if
    - no conflict with any neighbor's current color or new proposal,
    - no neighbor proposal conflicts with current color



Need O(log n) rounds
[Parnas Ron] → $\Delta^{O(\log n)}$ queries?

Modified Glauber Dynamics

Distributed Markov Chain round [Feng Sun Yin] [Fischer Ghaffari]

- *n* nodes simultaneously choose random colors "proposals"
- Update color if (1) no conflict with any neighbor's current color or new proposal and (2) no neighbor proposal conflicts with current color

Exponent improves with bigger q

Subpolynomial time algorithm: [Biswas R Yodpinyanee]

insight: just make sure that neighbor isn't colored with color c!

- For each neighbor jump back to previous time color c was proposed.
- Increment forward to see if overwritten

Much smaller dependency chains

Some other (prior) works

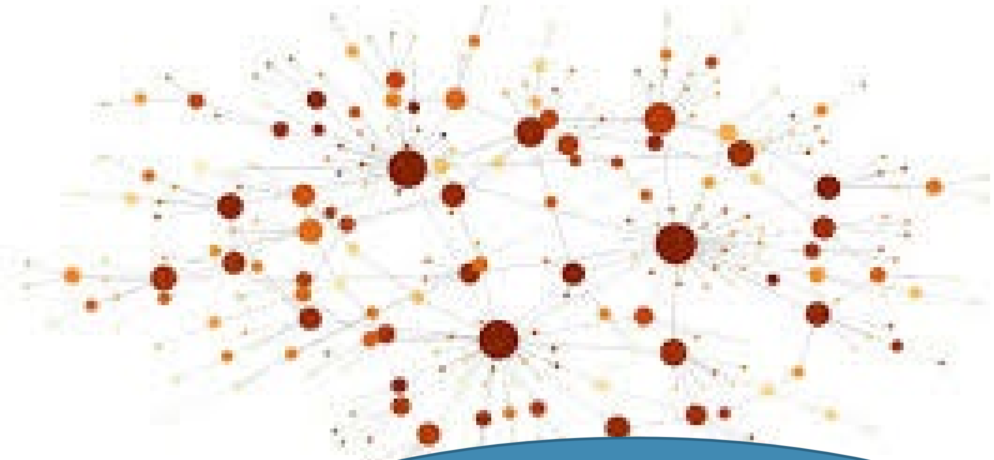
Implementation of Huge Pseudo-Random Objects

- Huge pseudorandom functions/permutations/balls-in-bins [Goldreich-Goldwasser-Micali’86][Luby-Rackoff ‘88][Naor-Reingold ’97][Mansour-Rubinstein-Vardi-Xie ’12]

- Model introduced and formalized in [Goldreich-Goldwasser-Nussboim 2003]
  - Generators for random functions, codes, graphs,…
  - Generators provide queries to random graphs with specified property
    - e.g. Planted Hamiltonian cycle, clique, colorability, connectedness, bipartiteness
    - Focus on *indistinguishability* under small number of queries and poly time. (see also [Naor Nussboim Tromer 05] [Alon Nussboim 07])
  - Give important primitives
    - e.g. Sampling from binomial distribution, interval-sum queries for functions (see also [Gilbert, Guha, Indyk, Kotidis, Muthukrishnan, Strauss 2002]
    - d-regular graph implementations [Naor Nussboim 07]

Locally Implementing Preferential Attachment Graphs [Even-Levi-Medina-Rosen 2017]

- Graphs generated:
  - Highly sequential random process
  - Sparse, but degree not bounded
- Queries:
  - Adjacency
  - Introduce next-neighbor query (implement with polylog(n) resources)
- Guarantee:
  - Close in statistical distance to correct distribution



Give local implementation without reconstructing full history!!

Open problem:

polylogarithmic time for $q \approx 2\Delta$?

Future directions

Other random objects?

Support degree, ith neighbor queries in graphs?

Lower bounds on space?