Estimating Graph Parameters from Random Order Streams

Pan Peng

University of Vienna, Austria $\Longrightarrow$ University of Sheffield, UK

Joint work with Christian Sohler (TU Dortmund, Germany)

Graph streams

To analyze the structure of massive and dynamic networks/graphs

Graph streams

To analyze the structure of massive and dynamic networks/graphs

Graph streaming algorithms

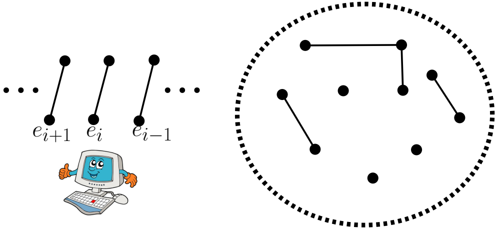
- Input: a sequence of edge insertions and/or deletions
- Goal: using as small space as possible, analyze the structure of the resulting graph.

Graph streams

To analyze the structure of massive and dynamic networks/graphs

Graph streaming algorithms

- Input: a sequence of edge insertions and/or deletions
- Goal: using as small space as possible, analyze the structure of the resulting graph.

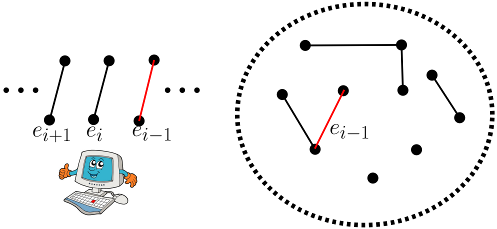


Graph streams

To analyze the structure of massive and dynamic networks/graphs

Graph streaming algorithms

- Input: a sequence of edge insertions and/or deletions
- Goal: using as small space as possible, analyze the structure of the resulting graph.

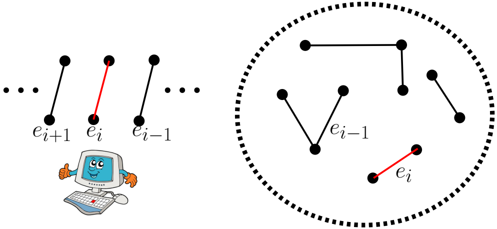


Graph streams

To analyze the structure of massive and dynamic networks/graphs

Graph streaming algorithms

- Input: a sequence of edge insertions and/or deletions
- Goal: using as small space as possible, analyze the structure of the resulting graph.

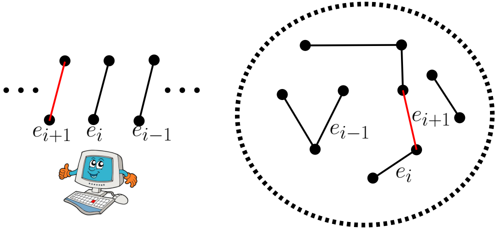


Graph streams

To analyze the structure of massive and dynamic networks/graphs

Graph streaming algorithms

- Input: a sequence of edge insertions and/or deletions
- Goal: using as small space as possible, analyze the structure of the resulting graph.

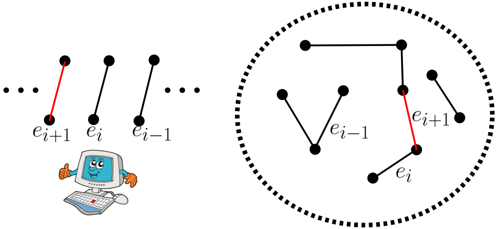


Graph streams

To analyze the structure of massive and dynamic networks/graphs

Graph streaming algorithms

- Input: a sequence of edge insertions and/or deletions
- Goal: using as small space as possible, analyze the structure of the resulting graph.



* This work: insertion-only; single pass

Model: adversarial order streams

Edges arrive in arbitrary order: ($n := \#$ vertices)

1. $\Omega(n)$ space for many basic problems:
   – connectivity [HRR99], diameter, bipartiteness, planarity, etc.

Model: adversarial order streams

Edges arrive in arbitrary order: ($n :=$ # vertices)

1. $\Omega(n)$ space for many basic problems:
   – connectivity [HRR99], diameter, bipartiteness, planarity, etc.

2. *semi-streaming* model [FKMSZ05]: $O(n \cdot \log^{O(1)} n)$ space

Model: adversarial order streams

Edges arrive in arbitrary order: ($n :=$ # vertices)

1. $\Omega(n)$ space for many basic problems:
   – connectivity [HRR99], diameter, bipartiteness, planarity, etc.

2. *semi-streaming* model [FKMSZ05]: $O(n \cdot \log^{O(1)} n)$ space
   – minimum spanning tree, maximal matching, connectivity, spectral/cut sparsifier, etc.

Model: adversarial order streams

Edges arrive in arbitrary order: ($n :=$ # vertices)

1. $\Omega(n)$ space for many basic problems:
   – connectivity [HRR99], diameter, bipartiteness, planarity, etc.

2. *semi-streaming* model [FKMSZ05]: $O(n \cdot \log^{O(1)} n)$ space
   – minimum spanning tree, maximal matching, connectivity, spectral/cut sparsifier, etc.
   – good for dense graphs, while trivial for sparse graphs

Model: adversarial order streams

Edges arrive in arbitrary order: ($n :=$ # vertices)

1. $\Omega(n)$ space for many basic problems:
   – connectivity [HRR99], diameter, bipartiteness, planarity, etc.

2. *semi-streaming* model [FKMSZ05]: $O(n \cdot \log^{O(1)} n)$ space
   – minimum spanning tree, maximal matching, connectivity, spectral/cut sparsifier, etc.
   – good for dense graphs, while trivial for sparse graphs

However: most real networks are sparse!

Model: adversarial order streams

Edges arrive in arbitrary order: ($n :=$ # vertices)

1. $\Omega(n)$ space for many basic problems:
   – connectivity [HRR99], diameter, bipartiteness, planarity, etc.

2. *semi-streaming* model [FKMSZ05]: $O(n \cdot \log^{O(1)} n)$ space
   – minimum spanning tree, maximal matching, connectivity, spectral/cut sparsifier, etc.
   – good for dense graphs, while trivial for sparse graphs

However: most real networks are sparse!

Some solutions for sparse graphs

Model: adversarial order streams

Edges arrive in arbitrary order: ($n :=$ # vertices)

1. $\Omega(n)$ space for many basic problems:
   – connectivity [HRR99], diameter, bipartiteness, planarity, etc.

2. *semi-streaming* model [FKMSZ05]: $O(n \cdot \log^{O(1)} n)$ space
   – minimum spanning tree, maximal matching, connectivity, spectral/cut sparsifier, etc.
   – good for dense graphs, while trivial for sparse graphs

However: most real networks are sparse!

Some solutions for sparse graphs

– parameterize the problem;

Model: adversarial order streams

Edges arrive in arbitrary order: ($n :=$ # vertices)

1. $\Omega(n)$ space for many basic problems:
   – connectivity [HRR99], diameter, bipartiteness, planarity, etc.

2. *semi-streaming* model [FKMSZ05]: $O(n \cdot \log^{O(1)} n)$ space
   – minimum spanning tree, maximal matching, connectivity, spectral/cut sparsifier, etc.
   – good for dense graphs, while trivial for sparse graphs

However: most real networks are sparse!

Some solutions for sparse graphs

- parameterize the problem;
- study special class of graphs (planar);

Model: adversarial order streams

Edges arrive in arbitrary order: ($n := \#$ vertices)

1. $\Omega(n)$ space for many basic problems:
   – connectivity [HRR99], diameter, bipartiteness, planarity, etc.

2. *semi-streaming* model [FKMSZ05]: $O(n \cdot \log^{O(1)} n)$ space
   – minimum spanning tree, maximal matching, connectivity, spectral/cut sparsifier, etc.
   – good for dense graphs, while trivial for sparse graphs

However: most real networks are sparse!

Some solutions for sparse graphs

– parameterize the problem;

– study special class of graphs (planar);

– relax the assumption that edges come in arbitrary order

Model: random order streams

Edges arrive in (uniformly) random order

– input stream is chosen u.a.r from the set of all possible permutations of edges

Model: random order streams

Edges arrive in (uniformly) random order

– input stream is chosen u.a.r from the set of all possible permutations of edges

- some problems can be solved using smaller space:
  – matching (size) [KMM12,KKS14]
  – bounded-degree graph property testing [MMPS17]

Model: random order streams

Edges arrive in (uniformly) random order

– input stream is chosen u.a.r from the set of all possible permutations of edges

- some problems can be solved using smaller space:
  – matching (size) [KMM12,KKS14]
  – bounded-degree graph property testing [MMPS17]
- some problems still require large space:
  – $\Omega(n)$ connectivity, $\Omega(n^{1+1/k})$ k-approx. for s, t-distance [CCM08]

Model: random order streams

Edges arrive in (uniformly) random order

– input stream is chosen u.a.r from the set of all possible permutations of edges

- some problems can be solved using smaller space:
  – matching (size) [KMM12,KKS14]
  – bounded-degree graph property testing [MMPS17]

- some problems still require large space:
  – $\Omega(n)$ connectivity, $\Omega(n^{1+1/k})$ k-approx. for s,t-distance [CCM08]

In general, it is unclear if the random-order assumption leads to more space-efficient algorithms

Our result

A new algorithmic technique:

some constant-time approximation algorithms (adjacency list model)	$\Longrightarrow$	constant-space random order streaming algorithms

Our result

A new algorithmic technique:

some constant-time approximation algorithms (adjacency list model)	$\Longrightarrow$	constant-space random order streaming algorithms

$\Downarrow$

(• query access to the adjacency list of the graph
• running time of the algorithm is constant, independent of n)

Our result

A new algorithmic technique:

some constant-time approximation algorithms (adjacency list model) $\Longrightarrow$ constant-space random order streaming algorithms

New random order graph streaming algorithms

approx.	problem	graph class	space
additive εn	number of connected components (CCs)	general	$\left(\frac{1}{\varepsilon}\right)^{O(\frac{1}{\varepsilon^3})}$
$(1+\varepsilon)$	weight of minimum spanning tree (MST)	general connected; edge weights $\{1, \cdots, W\}$	$\left(\frac{1}{\varepsilon}\right)^{\tilde{O}(\frac{W^3}{\varepsilon^3})}$
	size of maximum independent set (MIS)	planar/minor-free	$2^{(\frac{1}{\varepsilon})^{(\frac{1}{\varepsilon})^{\log^{O(1)}(\frac{1}{\varepsilon})}}}$

**with high constant probability

Our result

A new algorithmic technique:

some constant-time approximation algorithms (adjacency list model)	$\Longrightarrow$	constant-space random order streaming algorithms

New random order graph streaming algorithms

approx.	problem	graph class	space
additive εn	number of connected components (CCs)	general	$\left(\frac{1}{\varepsilon}\right)^{O(\frac{1}{\varepsilon^3})}$
$(1+\varepsilon)$	weight of minimum spanning tree (MST)	general connected; edge weights $\{1, \cdots, W\}$	$\left(\frac{1}{\varepsilon}\right)^{\tilde{O}(\frac{W^3}{\varepsilon^3})}$
	size of maximum independent set (MIS)	planar/minor-free	$2^{(\frac{1}{\varepsilon})^{(\frac{1}{\varepsilon})^{\log^{O(1)}(\frac{1}{\varepsilon})}}}$

**with high constant probability

Remark: Adversary order: $\Omega(n^{1-O(\varepsilon)})$ for the first two problems [HP16]

Our result

Further applications from our technique:

Our result

Further applications from our technique:

1. other constant-space random order streaming algorithms

Our result

Further applications from our technique:

1. other constant-space random order streaming algorithms
   - $(1+\varepsilon)$, size of minimum dominating set, planar graphs
   - additive εn, size of maximum matching, bounded average graphs
   - ...

Our result

Further applications from our technique:

1. other constant-space random order streaming algorithms
   - $(1+\varepsilon)$, size of minimum dominating set, planar graphs
   - additive εn, size of maximum matching, bounded average graphs
   - ...

2. can be used to derive the following:
   [MMPS17] For graphs with bounded maximum degree, property Π

Π constant-time testable (adjacency list model)	$\Longrightarrow$	Π constant-space testable in random order streams

Our result

Further applications from our technique:

1. other constant-space random order streaming algorithms
   - $(1+\varepsilon)$, size of minimum dominating set, planar graphs
   - additive εn, size of maximum matching, bounded average graphs
   - ...

2. can be used to derive the following:
   [MMPS17] For graphs with bounded maximum degree, property Π

Π constant-time testable (adjacency list model)	$\Longrightarrow$	Π constant-space testable in random order streams

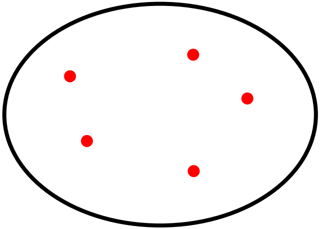
– here "test Π": distinguish if a graph satisfies Π or is "far" from satisfying Π

High-level idea

generic framework for many constant-time algorithms

High-level idea

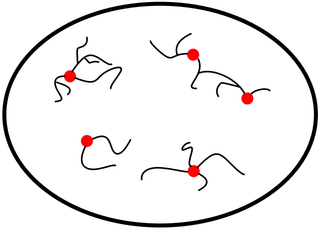
generic framework for many constant-time algorithms



1. sample a set S of constant number of vertices

High-level idea

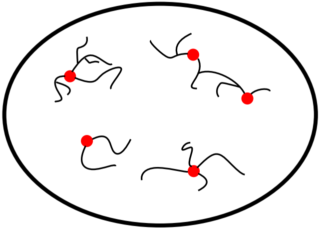
generic framework for many constant-time algorithms



1. sample a set S of constant number of vertices
2. explore the constant-size neighborhood of each $v \in S$ (and ignore "high" degree vertices)

High-level idea

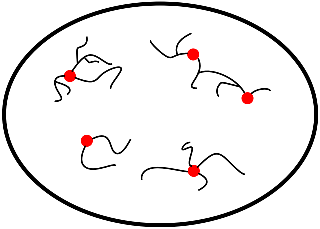
generic framework for many constant-time algorithms



1. sample a set S of constant number of vertices
2. explore the constant-size neighborhood of each $v \in S$ (and ignore "high" degree vertices)
3. draw conclusions from the explored subgraphs

High-level idea

generic framework for many constant-time algorithms

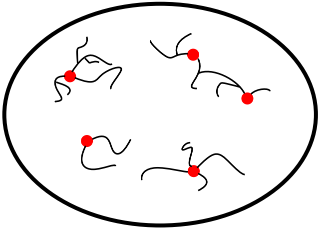


1. sample a set S of constant number of vertices
2. explore the constant-size neighborhood of each $v \in S$ (and ignore “high” degree vertices)
3. draw conclusions from the explored subgraphs

graph exploration: difficult for adversarial order streams

High-level idea

generic framework for many constant-time algorithms



1. sample a set S of constant number of vertices
2. explore the constant-size neighborhood of each $v \in S$ (and ignore "high" degree vertices)
3. draw conclusions from the explored subgraphs

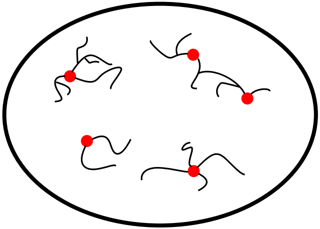
graph exploration: difficult for adversarial order streams

For random order streams:

- with (small) constant probability to see the right exploration

High-level idea

generic framework for many constant-time algorithms



1. sample a set S of constant number of vertices
2. explore the constant-size neighborhood of each $v \in S$ (and ignore "high" degree vertices)
3. draw conclusions from the explored subgraphs

graph exploration: difficult for adversarial order streams

For random order streams:

- with (small) constant probability to see the right exploration
- challenge: to identify when the graph exploration behaves as in the original graph and when it does not.

High-level idea

our technique for random order streams

High-level idea

our technique for random order streams

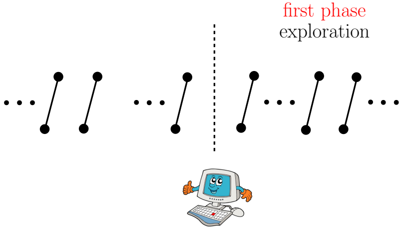
(i) two phases of streaming:

High-level idea

our technique for random order streams

(i) two phases of streaming:

- perform graph exploration in the **first phase**

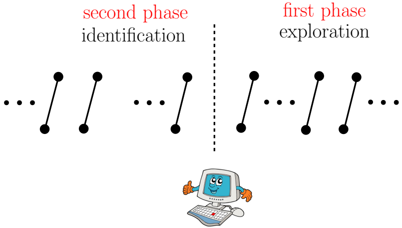


High-level idea

our technique for random order streams

(i) two phases of streaming:

- perform graph exploration in the **first phase**
- identify the right exploration in the **second phase**

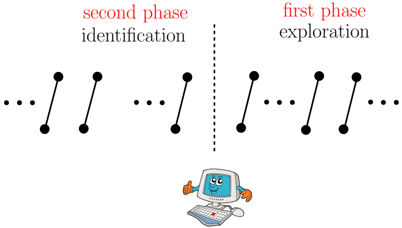


High-level idea

our technique for random order streams

(i) two phases of streaming:

- perform graph exploration in the **first phase**
- identify the right exploration in the **second phase**



(ii) use of conditional probabilities for the analysis

In the rest

- Approximate #CCs
- Approximate the weight of MST

not in this talk

- Approximate the size of MIS in planar graphs and beyond

Approx. #CCs with an additive error εn

$\mathrm{cc}_k :=$ #CCs of size k

– suffices to approx. cc_k with additive error $\varepsilon^2 n$, $k \leqslant 2/\varepsilon$

Approx. $\#$CCs with an additive error εn

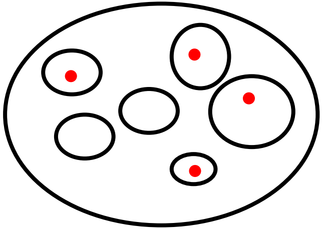
$\mathrm{cc}_k := \#$CCs of size k
– suffices to approx. cc_k with additive error $\varepsilon^2 n$, $k \leqslant 2/\varepsilon$

Approx. cc_k in the adjacency list model [CRZ05, BKMT14]

Approx. #CCs with an additive error εn

$\mathrm{cc}_k :=$ #CCs of size k
– suffices to approx. cc_k with additive error $\varepsilon^2 n$, $k \leqslant 2/\varepsilon$

Approx. cc_k in the adjacency list model [CRZ05, BKMT14]

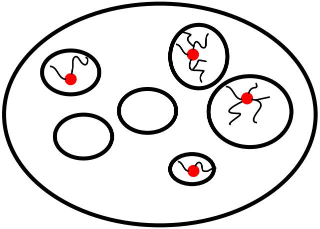


1. sample a set S of vertices

Approx. #CCs with an additive error εn

$\mathrm{cc}_k := $ #CCs of size k
– suffices to approx. cc_k with additive error $\varepsilon^2 n$, $k \leqslant 2/\varepsilon$

Approx. cc_k in the adjacency list model [CRZ05, BKMT14]

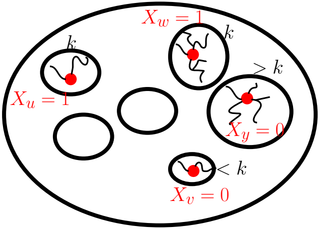


1. sample a set S of vertices
2. for each $v \in S$, perform BFS(v)

Approx. #CCs with an additive error εn

$\mathrm{cc}_k :=$ #CCs of size k
– suffices to approx. cc_k with additive error $\varepsilon^2 n$, $k \leqslant 2/\varepsilon$

Approx. cc_k in the adjacency list model [CRZ05, BKMT14]



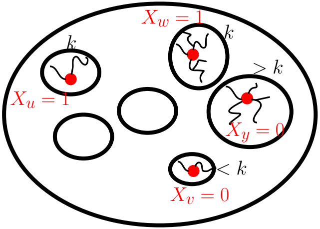
1. sample a set S of vertices
2. for each $v \in S$, perform BFS(v)
   - if a CC of size k is detected, set $X_v = 1$; o.w., $X_v = 0$

Approx. #CCs with an additive error εn

$\mathrm{cc}_k :=$ #CCs of size k

– suffices to approx. cc_k with additive error $\varepsilon^2 n$, $k \leqslant 2/\varepsilon$

Approx. cc_k in the adjacency list model [CRZ05, BKMT14]



1. sample a set S of vertices
2. for each $v \in S$, perform BFS(v)
   - if a CC of size k is detected, set $X_v = 1$; o.w., $X_v = 0$
3. output $\hat{c}_k := \frac{n}{k} \cdot \frac{\sum_{v \in S} X_v}{|S|}$

Perform BFS in random order streams

$$\boxed{\text{BFS}(v) \Longrightarrow \text{StreamBFS}(v)}$$

StreamBFS(v)

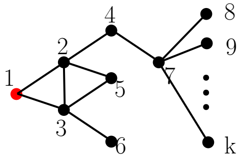
- Initialize $P := \{v\}$. Sequentially add edges e (and vertices) to P if e connects to the current collected subgraph.

Perform BFS in random order streams

$\text{BFS}(v) \Longrightarrow \text{StreamBFS}(v)$

StreamBFS(v)

- Initialize $P := \{v\}$. Sequentially add edges e (and vertices) to P if e connects to the current collected subgraph.

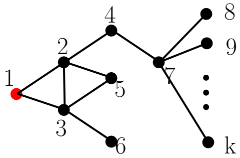


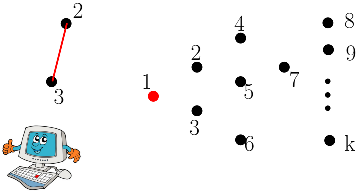
Perform BFS in random order streams

$\text{BFS}(v) \Longrightarrow \text{StreamBFS}(v)$

StreamBFS(v)

- Initialize $P := \{v\}$. Sequentially add edges e (and vertices) to P if e connects to the current collected subgraph.



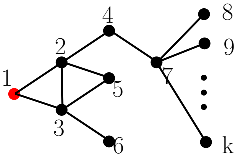


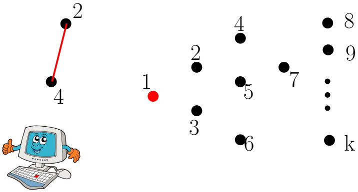
Perform BFS in random order streams

$\boxed{\text{BFS}(v) \Longrightarrow \text{StreamBFS}(v)}$

StreamBFS(v)

- Initialize $P := \{v\}$. Sequentially add edges e (and vertices) to P if e connects to the current collected subgraph.



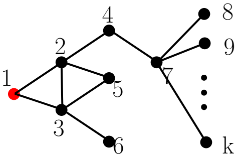


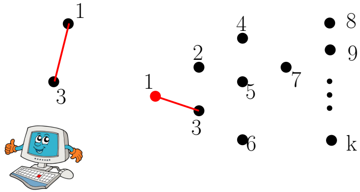
Perform BFS in random order streams

$$\boxed{\text{BFS}(v) \Longrightarrow \text{StreamBFS}(v)}$$

StreamBFS(v)

– Initialize $P := \{v\}$. Sequentially add edges e (and vertices) to P if e connects to the current collected subgraph.



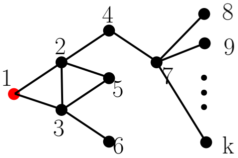


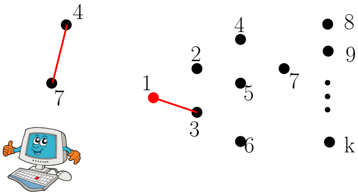
Perform BFS in random order streams

BFS(v) $\Longrightarrow$ StreamBFS(v)

StreamBFS(v)

– Initialize $P := \{v\}$. Sequentially add edges e (and vertices) to P if e connects to the current collected subgraph.



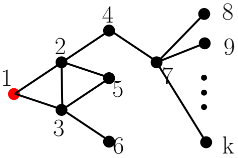


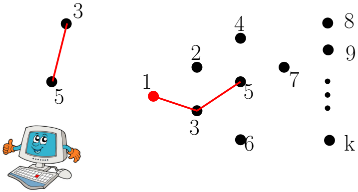
Perform BFS in random order streams

$\text{BFS}(v) \Longrightarrow \text{StreamBFS}(v)$

StreamBFS(v)

– Initialize $P := \{v\}$. Sequentially add edges e (and vertices) to P if e connects to the current collected subgraph.



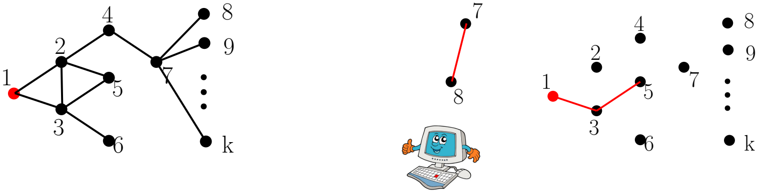


Perform BFS in random order streams

$\boxed{\text{BFS}(v) \Longrightarrow \text{StreamBFS}(v)}$

StreamBFS(v)

– Initialize $P := \{v\}$. Sequentially add edges e (and vertices) to P if e connects to the current collected subgraph.

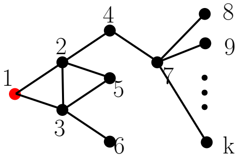


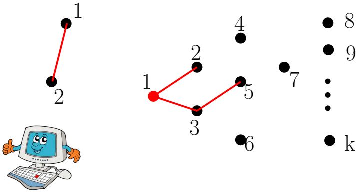
Perform BFS in random order streams

$$\boxed{\text{BFS}(v) \Longrightarrow \text{StreamBFS}(v)}$$

StreamBFS(v)

- Initialize $P := \{v\}$. Sequentially add edges e (and vertices) to P if e connects to the current collected subgraph.



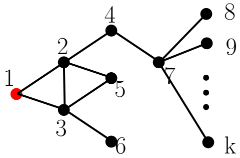


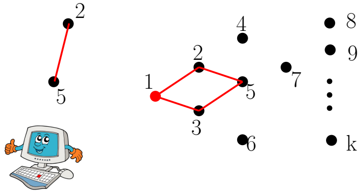
Perform BFS in random order streams

$$\boxed{\text{BFS}(v) \Longrightarrow \text{StreamBFS}(v)}$$

StreamBFS(v)

- Initialize $P := \{v\}$. Sequentially add edges e (and vertices) to P if e connects to the current collected subgraph.



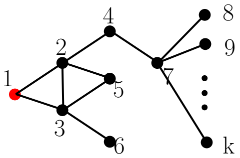


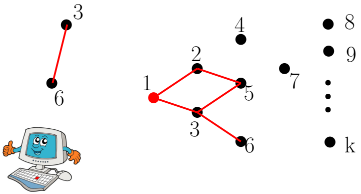
Perform BFS in random order streams

$$\boxed{\text{BFS}(v) \Longrightarrow \text{StreamBFS}(v)}$$

StreamBFS(v)

- Initialize $P := \{v\}$. Sequentially add edges e (and vertices) to P if e connects to the current collected subgraph.



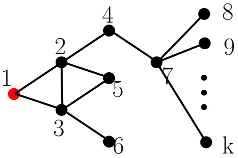


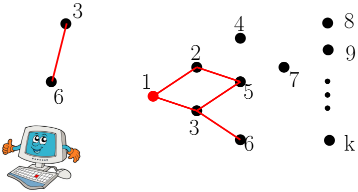
Perform BFS in random order streams

$\text{BFS}(v) \Longrightarrow \text{StreamBFS}(v)$

StreamBFS(v)

- Initialize $P := \{v\}$. Sequentially add edges e (and vertices) to P if e connects to the current collected subgraph.





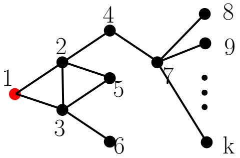
A property: if $|C_v| = k$, $\Pr[\text{StreamBFS}(v) = \text{BFS}(v)] = \Omega(1)$

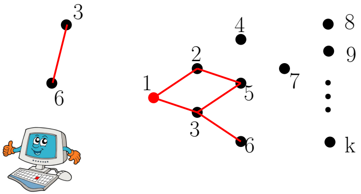
Perform BFS in random order streams

$\text{BFS}(v) \Longrightarrow \text{StreamBFS}(v)$

StreamBFS(v)

– Initialize $P := \{v\}$. Sequentially add edges e (and vertices) to P if e connects to the current collected subgraph.





A property: if $|C_v| = k, \Pr[\text{StreamBFS}(v) = \text{BFS}(v)] = \Omega(1)$

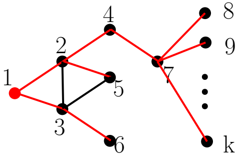
A difficulty: if $|C_u| = |C_v| = k$, $\Pr[\text{StreamBFS}(u) = C_u]$ might be different from $\Pr[\text{StreamBFS}(v) = C_v]$

Perform BFS in random order streams

canonical BFS (CBFS) tree: BFS tree + lexicographic order of vertices

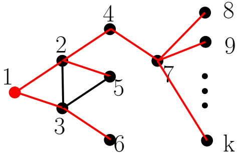
Perform BFS in random order streams

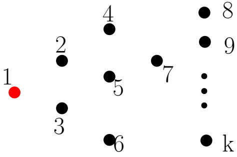
canonical BFS (CBFS) tree: BFS tree + lexicographic order of vertices



Perform BFS in random order streams

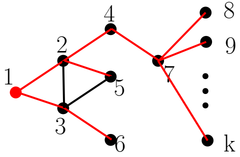
canonical BFS (CBFS) tree: BFS tree + lexicographic order of vertices

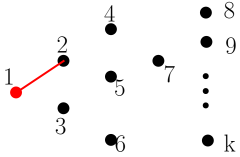




Perform BFS in random order streams

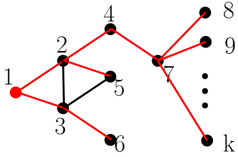
canonical BFS (CBFS) tree: BFS tree + lexicographic order of vertices

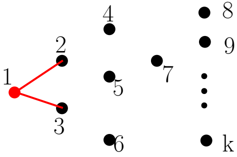




Perform BFS in random order streams

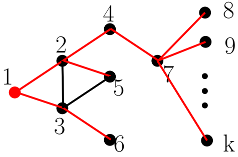
canonical BFS (CBFS) tree: BFS tree + lexicographic order of vertices

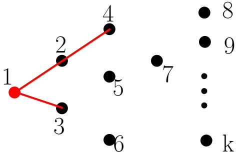




Perform BFS in random order streams

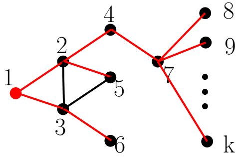
canonical BFS (CBFS) tree: BFS tree + lexicographic order of vertices

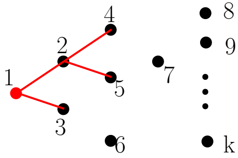




Perform BFS in random order streams

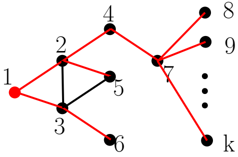
canonical BFS (CBFS) tree: BFS tree + lexicographic order of vertices

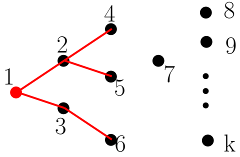




Perform BFS in random order streams

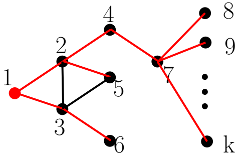
canonical BFS (CBFS) tree: BFS tree + lexicographic order of vertices

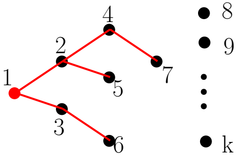




Perform BFS in random order streams

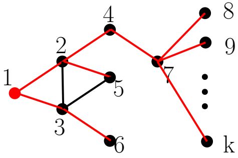
canonical BFS (CBFS) tree: BFS tree + lexicographic order of vertices

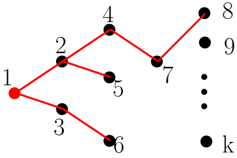




Perform BFS in random order streams

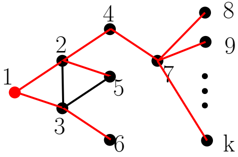
canonical BFS (CBFS) tree: BFS tree + lexicographic order of vertices

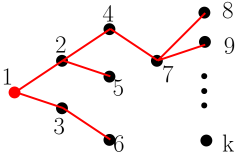




Perform BFS in random order streams

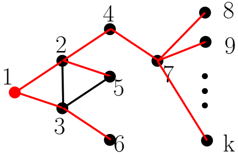
canonical BFS (CBFS) tree: BFS tree + lexicographic order of vertices

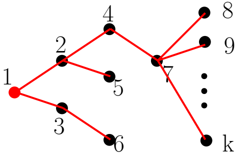




Perform BFS in random order streams

canonical BFS (CBFS) tree: BFS tree + lexicographic order of vertices





From v, there is a unique CBFStree CT_v of C_v

Perform BFS in random order streams

canonical BFS (CBFS) tree: BFS tree + lexicographic order of vertices

From v, there is a unique CBFStree CT_v of C_v

StreamBFS(v) $\Longrightarrow$ StreamCanoBFS(v)

StreamCanoBFS(v)

- Perform StreamBFS(v) w.r.t. lexicographic order of vertices to collect CT_v

Perform BFS in random order streams

canonical BFS (CBFS) tree: BFS tree + lexicographic order of vertices

From v, there is a unique CBFStree CT_v of C_v

$$\text{StreamBFS}(v) \Longrightarrow \text{StreamCanoBFS}(v)$$

StreamCanoBFS(v)

– Perform StreamBFS(v) w.r.t. lexicographic order of vertices to collect CT_v

A property: if $|C_u| = |C_v| = k$:
$\Pr[\text{StreamCanoBFS}(u) = CT_u] = \Pr[\text{StreamCanoBFS}(v) = CT_v]$

Approx. cc_k in random order streams

Another difficulty: false positives

Approx. cc_k in random order streams

Another difficulty: false positives

– divide the stream into two phases to rule out most "false positives"

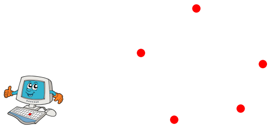
Approx. cc_k in random order streams

Another difficulty: false positives

– divide the stream into two phases to rule out most “false positives”

The algorithm

1. sample a set S of vertices

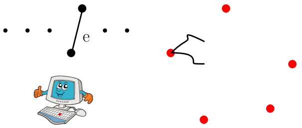


Approx. cc_k in random order streams

Another difficulty: false positives

– divide the stream into two phases to rule out most "false positives"

The algorithm



1. sample a set S of vertices
2. **first phase** (i.e., first λm edges):

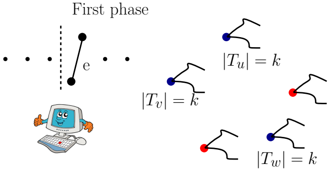
   for each $v \in S$, perform a StreamCanoBFS

Approx. cc_k in random order streams

Another difficulty: false positives

– divide the stream into two phases to rule out most "false positives"

The algorithm



1. sample a set S of vertices
2. **first phase** (i.e., first λm edges):

   for each $v \in S$, perform a StreamCanoBFS

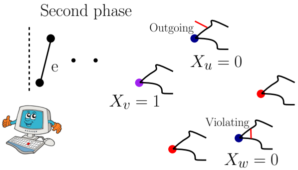
   – v good: $v \in S$ & $|\text{StreamCanoBFS}(v)| = k$.

Approx. cc_k in random order streams

Another difficulty: false positives

– divide the stream into two phases to rule out most “false positives”

The algorithm



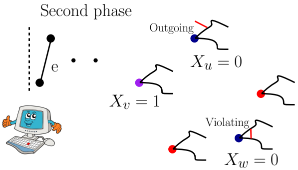
1. sample a set S of vertices
2. **first phase** (i.e., first λm edges):
   for each $v \in S$, perform a StreamCanoBFS
   – v good: $v \in S$ & $|\text{StreamCanoBFS}(v)| = k$.
3. **second phase**:
   for each good v, check if StreamCanoBFS(v) has an outgoing or “violating” edge
   – If so, set $X_v = 0$; else set $X_v = 1$

Approx. cc_k in random order streams

Another difficulty: false positives

– divide the stream into two phases to rule out most "false positives"

The algorithm



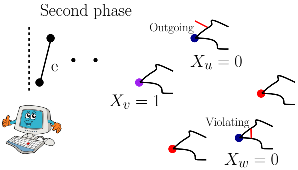
1. sample a set S of vertices
2. **first phase** (i.e., first λm edges):
   for each $v \in S$, perform a StreamCanoBFS
   – v good: $v \in S$ & $|\text{StreamCanoBFS}(v)| = k$.
3. **second phase**:
   for each good v, check if StreamCanoBFS(v) has an outgoing or "violating" edge
   – If so, set $X_v = 0$; else set $X_v = 1$
4. output $\hat{c}_k := \frac{n}{k} \cdot \frac{1}{\gamma_k} \cdot \frac{\sum_{v:\mathrm{good}} X_v}{|S|}$

Approx. cc_k in random order streams

Another difficulty: false positives

– divide the stream into two phases to rule out most "false positives"

The algorithm



1. sample a set S of vertices
2. **first phase** (i.e., first λm edges):

   for each $v \in S$, perform a StreamCanoBFS

   – v good: $v \in S$ & $|\text{StreamCanoBFS}(v)| = k$.
3. **second phase**:

   for each good v, check if StreamCanoBFS(v) has an outgoing or "violating" edge

   – If so, set $X_v = 0$; else set $X_v = 1$
4. output $\hat{c}_k := \frac{n}{k} \cdot \frac{1}{\gamma_k} \cdot \frac{\sum_{v:\text{good}} X_v}{|S|}$

γ_k:=Pr[any set T of $k-1$ edges appears in the lexico. order in the first phase]

The analysis

A simple but useful conditional probability

$T \subseteq E, |T| = k-1, e \in E \setminus T; F :=$ set of edges in the first phase:

$$\Pr[e \in F | T \subseteq F] \sim \lambda$$

The analysis

A simple but useful conditional probability

$T \subseteq E, |T| = k-1, e \in E \setminus T; F :=$ set of edges in the first phase:

$$\Pr[e \in F | T \subseteq F] \sim \lambda$$

$\Longrightarrow$ for any v with $|C_v| \geqslant k$: $\Pr[\text{false positive}] \ll \gamma_k$

The analysis

A simple but useful conditional probability

$T \subseteq E, |T| = k-1, e \in E \setminus T; F :=$ set of edges in the first phase:

$$\Pr[e \in F | T \subseteq F] \sim \lambda$$

$\implies$ for any v with $|C_v| \geqslant k$: $\Pr[\text{false positive}] \ll \gamma_k$

Our guarantee

- if $|C_v| < k$, $\Pr[X_v = 1] = 0$

The analysis

A simple but useful conditional probability

> $T \subseteq E, |T| = k - 1, e \in E \setminus T; F :=$ set of edges in the first phase:
>
> $$\Pr[e \in F | T \subseteq F] \sim \lambda$$

$\Longrightarrow$ for any v with $|C_v| \geqslant k$: $\Pr[\text{false positive}] \ll \gamma_k$

Our guarantee

- if $|C_v| < k$, $\Pr[X_v = 1] = 0$
- if $|C_v| = k$, $\Pr[X_v = 1] \sim \gamma_k$

The analysis

A simple but useful conditional probability

$T \subseteq E, |T| = k-1, e \in E \setminus T; F :=$ set of edges in the first phase:

$$\Pr[e \in F | T \subseteq F] \sim \lambda$$

$\Longrightarrow$ for any v with $|C_v| \geqslant k$: $\Pr[\text{false positive}] \ll \gamma_k$

Our guarantee

- if $|C_v| < k$, $\Pr[X_v = 1] = 0$
- if $|C_v| = k$, $\Pr[X_v = 1] \sim \gamma_k$
- if $|C_v| > k$, $\Pr[X_v = 1] \ll \gamma_k$

The analysis

A simple but useful conditional probability

$T \subseteq E, |T| = k-1, e \in E \setminus T; F :=$ set of edges in the first phase:

$$\Pr[e \in F | T \subseteq F] \sim \lambda$$

$\Longrightarrow$ for any v with $|C_v| \geqslant k$: $\Pr[\text{false positive}] \ll \gamma_k$

Our guarantee

- if $|C_v| < k$, $\Pr[X_v = 1] = 0$
- if $|C_v| = k$, $\Pr[X_v = 1] \sim \gamma_k$
- if $|C_v| > k$, $\Pr[X_v = 1] \ll \gamma_k$

$\Longrightarrow \mathrm{E}[\hat{c}_k] \sim \mathrm{cc}_k$

$(1+\varepsilon)$-approx. the weight of MST

Input: connected G, edge weights $\in \{1, \cdots, W\}$

- M: weight of MST;

$(1+\varepsilon)$-approx. the weight of MST

Input: connected G, edge weights $\in \{1, \cdots, W\}$

- M: weight of MST; connected $\Longrightarrow M \geqslant n-1$

$(1+\varepsilon)$-approx. the weight of MST

Input: connected G, edge weights $\in \{1, \cdots, W\}$

- M: weight of MST; connected $\Longrightarrow M \geqslant n-1$

A relation between MST weight and #CCs

- $G^{(t)}$: subgraph of G consisting of edges of weight $\leqslant t$
- $c^{(t)}$: #CCs of $G^{(t)}$

$(1+\varepsilon)$-approx. the weight of MST

Input: connected G, edge weights $\in \{1, \cdots, W\}$

- M: weight of MST;   connected $\Longrightarrow M \geqslant n-1$

A relation between MST weight and #CCs

- $G^{(t)}$: subgraph of G consisting of edges of weight $\leqslant t$
- $c^{(t)}$: #CCs of $G^{(t)}$

$\Longrightarrow M = n - W + \sum_{t=1}^{W-1} c^{(t)}$ [CRZ05]

$(1+\varepsilon)$-approx. the weight of MST

Input: connected G, edge weights $\in \{1, \cdots, W\}$

- M: weight of MST; connected $\Longrightarrow M \geqslant n-1$

A relation between MST weight and #CCs

- $G^{(t)}$: subgraph of G consisting of edges of weight $\leqslant t$
- $c^{(t)}$: #CCs of $G^{(t)}$

$\Longrightarrow M = n - W + \sum_{t=1}^{W-1} c^{(t)}$ [CRZ05]

– to $(1+\varepsilon)$-approx. M: suffices to approx. each $c^{(t)}$ with additive error $\frac{\varepsilon n}{4W}$

$(1+\varepsilon)$-approx. the weight of MST

Input: connected G, edge weights $\in \{1, \cdots, W\}$

- M: weight of MST; connected $\Longrightarrow M \geqslant n-1$

A relation between MST weight and #CCs

- $G^{(t)}$: subgraph of G consisting of edges of weight $\leqslant t$
- $c^{(t)}$: #CCs of $G^{(t)}$

$\Longrightarrow M = n - W + \sum_{t=1}^{W-1} c^{(t)}$ [CRZ05]

– to $(1+\varepsilon)$-approx. M: suffices to approx. each $c^{(t)}$ with additive error $\frac{\varepsilon n}{4W}$

The algorithm

1. for each t from 1 to $W-1$, approx. #CCs of $G^{(t)}$ to obtain $\hat{c}(t)$
2. output $\hat{M} := n - W + \sum_{t=1}^{W-1} \hat{c}(t)$

Conclusions and open problems

Summary:

A new algorithmic technique:

some constant-time approximation algorithms (adjacency list model)	$\Longrightarrow$	constant-space random order streaming algorithms

Conclusions and open problems

Summary:

A new algorithmic technique:

some constant-time approximation algorithms (adjacency list model)	$\Longrightarrow$	constant-space random order streaming algorithms

Questions:

Conclusions and open problems

Summary:

A new algorithmic technique:

some constant-time approximation algorithms (adjacency list model)	$\Longrightarrow$	constant-space random order streaming algorithms

Questions:

1. Lower bounds in random order streams

   – our conjecture for approx. #CCs: $\exp(\Omega(1/\varepsilon))$.

Conclusions and open problems

Summary:

A new algorithmic technique:

some constant-time approximation algorithms (adjacency list model)	$\Longrightarrow$	constant-space random order streaming algorithms

Questions:

1. Lower bounds in random order streams
   – our conjecture for approx. $\#$CCs: $\exp(\Omega(1/\varepsilon))$.
2. Anything between uniformly random ordering and worst-case ordering?

Conclusions and open problems

Summary:

A new algorithmic technique:

some constant-time approximation algorithms (adjacency list model)	$\Longrightarrow$	constant-space random order streaming algorithms

Questions:

1. Lower bounds in random order streams
   – our conjecture for approx. #CCs: $\exp(\Omega(1/\varepsilon))$.
2. Anything between uniformly random ordering and worst-case ordering?

Thanks!