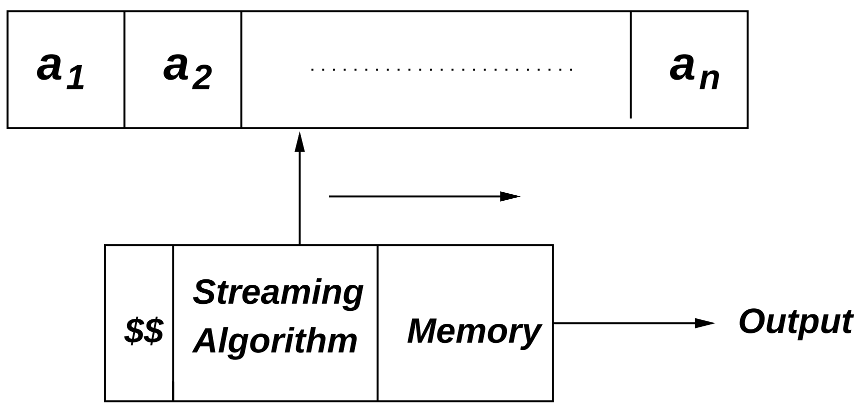
Reductions in Streaming Algorithms, with an Application to Counting Triangles in Graphs

Ziv Bar-Yossef
Computer Science Division
U.C. Berkeley
http://www.cs.berkeley.edu/~zivi

Ravi Kumar
D. Sivakumar
IBM Almaden Research Center

The Streaming Model
[HRR98, AMS96, FKSV99]

- Input data items $a_1, \ldots, a_n \in A$ arrive in a one-way stream and in arbitrary order



- Algorithms are allowed to be randomized and approximate
- Complexity measures: S – space and T – processing time per data item. Ideally, $S, T = poly \log(n, |A|)$

Motivation

- Networking
  - Processing IP packets at ISP routers
- Database
  - One-pass algorithms for large database relations
- Web Information Retrieval
  - Web crawling
  - Processing search engine query logs

Frequency Moments

k-th Frequency Moment:

Input: $a_1, \ldots, a_n \in [m]$

For $j \in [m]$, $\quad f_j = \#$ of occurrences of j in $a_1, \ldots, a_n$

$$F_k(a_1, \ldots, a_n) = \sum_{j=1}^{m} f_j^k$$

- $F_0 = \#$ of distinct data items [FM83]
- $F_1 = n$
- $F_k, k \geq 2 =$ measure of k-wise collision probability

Frequency Moments (cont.)

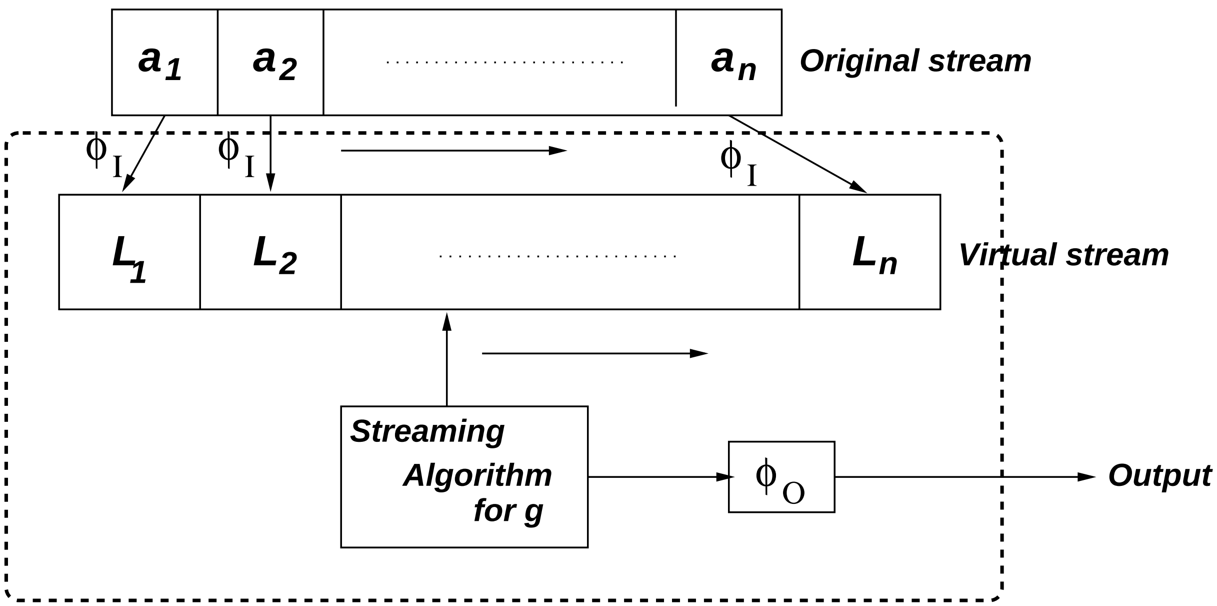
Streaming algorithms for frequency moments [AMS96]:

Function	Approx. Factor	Space & Time per item
F_0	> 2	$O(\log m)$
F_1	exact	$O(\log n)$
F_2	$1 + \epsilon$	$O(\frac{1}{\epsilon^2}(\log \log n + \log m))$
$F_k, k \geq 3$	$1 + \epsilon$	$O(\frac{1}{\epsilon^2} m^{1-\frac{1}{k}} \log \log n)$

A New Paradigm in Streaming Algorithm Design

Definition [stream reduction from f to g]
A pair of mappings (ϕ_I, ϕ_O) satisfying:

v is an approximation of $g(\phi_I(a_1), \ldots, \phi_I(a_n)) \Rightarrow \phi_O(v)$ is an approximation of $f(a_1, \ldots, a_n)$



Caveat: Naive implementation requires $|L_i|$ time to process a_i

List-Efficient Streaming Algorithms

Definition

A list-efficient streaming algorithm:

- Items of the stream are lists of data items
- Time per list should be polynomial in the list's representation size

Crucial point

In stream reductions the lists have succinct representations

$$\Downarrow$$

List-efficiency – essential for stream reductions

List-efficiency is also interesting on its own:

- Find $\#$ of distinct elements in a union of integer intervals

Our Results

1. $(1+\epsilon)$ streaming approximation of F_0 (cf. [FM83],[AMS96])
   (earlier, Gibbons & Tirthapura [GT01]; independently, Trevisan [T01])

2. List-efficient algorithms for all frequency moments

3. $(1+\epsilon)$ streaming approximation of the number of Δ's in a graph
   - First streaming algorithm for a "natural" graph problem
   - Works by reduction to frequency moments

Counting Triangles in a Graph

Objective

Approximate the number of triangles in an undirected graph $G = (V, E)$ that is presented as a stream of edges $e_1, \ldots, e_m$

Applications in:

- Graph data mining
- Database query optimization

Theorem

Any streaming algorithm requires $\Omega(|E|)$ space to decide whether G is triangle-free.

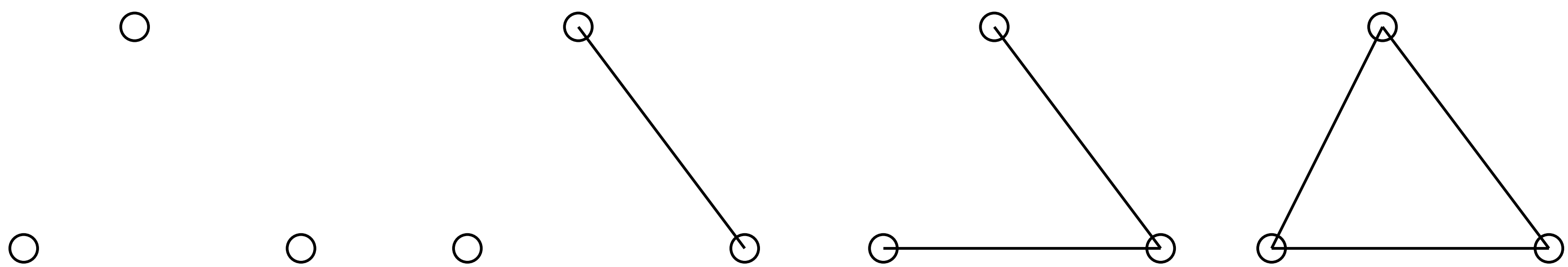
$\Longrightarrow$ problem is hard in general!

Our algorithm works for a restricted class of graphs.

Types of Node Triples

Definition

An (unordered) triple of nodes $u, v, w \in V$ is of type j ($j = 0, 1, 2, 3$) if j of the pairs $\{u, v\}, \{u, w\}, \{v, w\}$ are edges in G.



Type 0 ***Type 1*** ***Type 2*** ***Type 3***

$T_j = \#$ of triples of type j

Our Algorithm

Theorem

There exists a streaming algorithm that $(1+\epsilon)$-approximates T_3 with space

$$S = O\left(\frac{1}{\epsilon^3} \cdot \left(\frac{T_1 + T_2}{T_3}\right)^3 \cdot \log|V|\right)$$

and $poly(S)$ time per edge.

Algorithm works well for dense graphs with $(T_1 + T_2)/T_3$ small (e.g., $G_{n,p}$ with a constant p)

The Reduction

$$\boxed{\begin{array}{l}\text{Counting triangles} \\ \text{in } G\end{array}} \xrightarrow{\text{reduction}} \boxed{\begin{array}{l}\text{Estimating } F_0, F_1, F_2 \\ \text{for a stream of node triples}\end{array}}$$

- Stream $e_1, \ldots, e_m \Longrightarrow$ virtual stream $L = L_{e_1}, \ldots, L_{e_m}$
- For edge $e = \{u, v\}$, $L_e = \{\{u, v, w\} \mid w \in V - \{u, v\}\}$
- L_e has succinct representation

Main observation

$\{u, v, w\}$ is of type $j \Leftrightarrow \{u, v, w\}$ occurs j times in L

The Reduction (cont.)

$\Longrightarrow \quad F_k(L) = T_1 \cdot 1^k + T_2 \cdot 2^k + T_3 \cdot 3^k$

$$\begin{pmatrix} F_0 \\ F_1 \\ F_2 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 2 & 3 \\ 1 & 4 & 9 \end{pmatrix} \begin{pmatrix} T_1 \\ T_2 \\ T_3 \end{pmatrix}$$

$\Longrightarrow \quad T_3 = F_0 - 3/2F_1 + 1/2F_2$

Important Remark

$(1+\epsilon)$-approximation of T_3 requires $(1+\epsilon')$-approximation of F_0, F_1, F_2 with $\epsilon' = O(\epsilon \cdot T_3/(T_1+T_2))$.

$\Longrightarrow$ space and time of our algorithm are polynomial in $(T_1+T_2)/T_3$

What's Missing?

Carrying out the reduction requires:

- $(1+\epsilon)$ streaming approximations of F_0, F_1, F_2
- List-efficient streaming algorithms for F_0, F_1, F_2

Theorem

There exists a streaming algorithm that $(1+\epsilon)$-approximates F_0 with $O((1/\epsilon^3)\log m)$ space and time per item.

Theorem

There exist list-efficient streaming algorithms for F_0, F_1, F_2 with $poly(1/\epsilon, \log n, \log m)$ space and time per item.

(The list-efficiency of the F_2 algorithm is due to Gilbert *et al.* [GKMS01])

Generalizations

- A similar algorithm works for bounded-degree graphs that arrive as a stream of nodes with their neighbors.
- Can generalize to count copies of K_k, $k \geq 4$, but with $O(m^{1-1/k})$ space and time per edge.
- Can generalize to directed graphs

Conclusions and Open Problems

Highlights:

- A novel paradigm for streaming algorithm design
- List-efficient streaming algorithms for frequency moments
- A new, list-efficient, $(1 + \epsilon)$ streaming approximation for the number of distinct elements
- Application: streaming algorithm for counting triangles in a graph

Open problems:

- Better algorithm for F_0 in terms of ϵ
- Tighter lower bounds for counting triangles

$(1+\epsilon)$-Approximation of F_0

Basic Idea

For a random mapping $h : [m] \to [R] \quad (R > F_0)$,

$$E(\#\text{ of distinct items in } a_1, \ldots, a_n \text{ that } h \text{ maps to } 0) = \frac{F_0}{R}$$

The algorithm:

- Use the AMS algorithm to obtain a value $F_0 \leq R \leq 256F_0$
- Pick a pairwise independent hash function $h : [m] \to [R]$
- While streaming through $a_1, \ldots, a_n$, find

$$X = \#\text{ of distinct } a_j\text{'s that } h \text{ maps to } 0$$

- Output $X \cdot R$