Lecture 5. Graph streams

Ricard Gavaldà

MIRI Seminar on Data Streams, Spring 2014

Contents

Graph streams

Two main models:

- Adjacency model: Stream is a list of edges $(u, v) \in G$ in arbitrary order
- Incidence model: Stream is a list of tuples $(u, v_1, \ldots, v_k)$ where the $(u, v_1), \ldots, (u, v_k)$ are all edges leaving u in G

In fully dynamic models, edge deletions are allowed

Counting subgraphs

Counting triangles

Simplest instance of “counting subgraph occurrences”

Interesting for e.g. “clustering coefficient” and communities

[Bar-Yossef+02]: reduction to computing moments
[Buriol+06] better space & update time bounds

Counting triangles

- T_i ($i = 0 \ldots 3$) = set / number of tuples (u, v, w) for which i out of the 3 possible edges are present
- We want to approximate T_3
- Reduction:
  For every edge (u, v) in stream, produce all tuples (u, v, w)
- Observation: (u, v, w) is generated i times iff it is in T_i

Counting triangles via moments

In the generated stream:

$$F_k = 1 \cdot T_1 + 2^k \cdot T_2 + 3^k \cdot T_3$$

Therefore

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

Invertible matrix

Counting triangles via moments

So T_3 is a linear combination of F_0, F_1, F_2

And we can approximate F_0, F_1, F_2 in $O(\varepsilon^{-2} \ln(|V|^3/\delta))$

Looks like we're done. But there's a glitch

Counting triangles via moments, end

The linear combination is

$$T_3 = F_0 - 1.5F_1 + 0.5F_2$$

So good approximations may cancel into a bad approximation

We need to average $O((T_1 + T_2 + T_3)/T_3)^2$ copies of the algorithm

$\to$ space $O(\frac{1}{\varepsilon^2}(1 + \frac{T_1+T_2}{T_3})^2 \log(|V|/\delta))$

Counting triangles, another solution

Let $m = |E|$, $n = |V|$, and assume $T_3 \geq t$. Note $T_3 \leq m(n-2)$

- Pick an edge $e_i = (u, v)$ at random from stream
- Pick w uniformly at random from $V - \{u, v\}$
- If there are edges $e_j = (u, w)$ and $e_k = (v, w)$, for $j, k > i$ in stream, return $3m(n-2)$, else return 0

- $E[\text{output}] = T_3$
- $Var[\text{output}] = T_3(3m(n-2) - T_3)$
- Repeat $O(\varepsilon^{-2} mn/t)$ times in parallel and average
- Note use of reservoir sampling

Counting arbitrary subgraphs

[Kane+12]

- Stream for graph G on n vertices, t edges
- Fixed graph H, m vertices, k edges
- Want to approximate $\#H(G)$, the number of occurrences of H in G

Counting arbitrary subgraphs

Theorem

For each $\varepsilon > 0$ there is an algorithm that ε-approximates $\#H(G)$ using

- $O\left(\frac{1}{\varepsilon^2}\frac{t^k}{\#H(G)^2}\log n\right)$ bits if $\delta(H) \geq 2$
- $O\left(\frac{1}{\varepsilon^2}\frac{t^k\Delta(G)^k}{\#H(G)^2}\log n\right)$ bits for every H

Here $\Delta()$ and $\delta()$ denote maximum and minimum degrees

(My, not your) homework: understand this algorithm

Counting arbitrary subgraphs

Example: H = undirected triangle, $G = G(n, p)$ random graph

- $E[\#H(G)] \simeq n^3 p^3$
- $E[|E_G|] \simeq n^2 p$
- Space $= O(\varepsilon^{-2} t^3 / \#H(G)^2 \log n) = O(\varepsilon^{-2} p^{-3} \log n)$

Counting arbitrary subgraphs

Init: For each edge $(a, b) \in H$, set $Z_{ab} = 0$

Update((u, v)): For each $(a, b) \in H$,

$$Z_{ab} \mathrel{+}= X_a(u) \cdot X_b(v) \cdot Q^{Y(u)/deg_H(a)} \cdot Q^{Y(v)/deg_H(b)}$$

Query: return the real part of

$$\frac{t^t}{t! \cdot auto(H)} \cdot \prod_{a,b \in H} Z_{ab}$$

where

- Q is a τ-th root of unity, $\tau = 2^k - 1$
- Y and X are complex-valued $4k$-wise independent hash functions

Counting arbitrary subgraphs

Intuition:

- X maps u, v to random potential images a, b in H
- Y randomly indicates whether the corresponding H edge exists in G
- This is done separately for all edges in H. Presumably all cross-terms cancel by independence
- Basic algorithm with expected value $\#H(G)$ and large variance
- Run many copies and average

Connectivity and distances

Connectivity and spanning forests

Adjacency stream model

Undirected graphs

Spanning trees (forests) solve connectivity problems

- Are vertices u, v connected?
- How many connected components?

Building a spanning forest

$H \leftarrow \emptyset$;

for each edge (u, v) in stream

    add (u, v) to H iff it does not create a cycle

Exercise 5. Prove this claim

At all times, H is a spanning forest of the graph G seen so far. I.e., H is a set of trees and there is a path in G between any two vertices iff there is a path in H

Building minimum weight spanning forests

Consider weighted, undirected graphs

$H \leftarrow \emptyset$;

for each edge (u, v) in stream

add (u, v) to H;

if (H has a cycle) remove heaviest edge in the cycle

Claim

At all times, H is a minimum weight spanning forest of the graph G seen so far

Distances and graph spanners

Consider unweighted, undirected graphs

Each such graph defines a distance between vertices, by shortest paths

If we can compute t-spanners, we can t-approximate distances

Spanner Graph

A graph H is a t-spanner of graph G if for every $u, v \in G$

$$d_G(u, v) \leq d_H(u, v) \leq t \cdot d_G(u, v).$$

(Typically, $V_H = V_G$ and $E_H \subseteq E_G$)

Computing spanners

$H \leftarrow \emptyset$;
For each edge (u, v) in the stream for G
    if $(d_H(u, v) \leq t)$ ignore (u, v), else add it to H

- Suppose there is an edge (u, v) in G
- If we added it to H, fine
- If not, there was (and still is) a path of length $\leq t$ in H
- Hence H is a t-spanner of G

  Note: condition = "adding (u, v) creates a cycle of length $\leq t + 1$ in H"

Computing spanners

How large will H be?

Lemma (see e.g. [McGregor])

A graph H on n nodes with no cycles of length $\leq 2t$ has $O(n^{1+1/t})$ edges

Computing spanners

With this idea and clever data structures, [Baswana08,Elkin08]:

Theorem

There is a algorithm that, given an integer t, and a streamed graph builds a $(2t-1)$-spanner in space $O(n^{1+1/t})$. Time per edge is (amortized) $O(1)$

Note that as $2t-1$ "tends to 1", space tends to $O(n^2)$

Distance distributions

For a directed graph $G = (V, E)$ and $u \in V$, the neighborhood functions

$$B(u,t) \text{ = set of vertices at distance} \leq t \text{ from } u$$
$$N(u,t) = |B(u,t)|$$
$$N(t) \text{ = number of pairs } (u,v) \text{ at (one-way) distance} \leq t$$

Useful, but costly to compute exactly for large G

HyperANF

ANF [Palmer,Gibbons,Faloutsos02]: Memory $O(n \log n)$

- 2 bilion links graph $\rightarrow$ 30 minutes on 90 machines

HyperANF [Boldi,Rosa,Vigna11]: Memory $O(n \log \log n)$

- 15 minutes on a laptop

HyperANF

Key observation 1:

$$B(u,t) = B(u,t-1) \cup \bigcup_{u \to v} B(v,t-1)$$

Obvious algorithm stores sets $B(u,t)$ in disk, repeats passes, random access. Slow

Idea: don't store $B(u,t)$, just an approximation of its cardinality with a HyperLogLog counter

But then, how do we compute cardinality of union with only cardinalities?

HyperANF

Key observation 2:
HyperLogLog is well-behaved w.r.t unions

Fix a number of registers r in HyperLogLog.
The HyperLogLog counter associated to $S_1 S_2$ is obtained maximizing the counters for S_1 and S_2, register for register

HyperANF

Other ideas:

- Broadword programming: Resisters are short than machine words. Pack several in a word and use bitwise opserations to speedup maximumization
- Try to maximize only changed counters. Large savings near the end, when most counters have stabilized.
- Systolic computation: A modified counter *signals* its predecessors that they must update

HyperANF: applications

Distinguishing Web-like and social-network-like networks [Boldi+11]

- Shortest-path-index of dispersion: Variance-to-mean ratio of distances
- < 1 for social networks, > 1 for web-like networks

Diameter of the Facebook graph [Backstrom+11]

- 720M active users, 69B friendship links
- Average distance is 4.74 (= 3.74 degrees of separation)
- 92% of users are at distance ≤ 5
- 10 hours on 256Gb RAM machine

Clustering

k-center clustering: The problem

Just a taster of a large body of work on geometric problems ...

k-center clustering

Fix a metric space (X, d)

Input: an integer k, a stream of points $S = x_1, x_2, \ldots$

Output: a set $Y \subseteq X$, $|Y| \leq k$, minimizing

$$\max_i \min_{y \in Y} d(x_i, y)$$

k-center clustering: Greedy algorithm

Suppose we know optimal value *OPT* with k centers. Then:

$r = 2OPT$;
repeat over the stream:
  wait for a point y at distance $> r$ from all previous centers
  add y as new center

Claim: This algorithm uses space k and returns a solution with value $\leq 2OPT$

BTW, $(2 - \varepsilon)$-approximation is impossible if P $\neq$ NP (even non-streaming)

k-center clustering: Greedy algorithm

Why does this work?

- Each center is at distance $> r$ from previous ones
- Suppose the value of returned solution is $> r$
- $\therefore$ One point in stream is still at distance $> r$ from all k centers
- We have $k+1$ points at distance $> r = 2OPT$ from each other
- X cannot be covered with any k balls of radius OPT

k-center clustering: Streaming algorithm

[McCutchen-Khuller08], [Guha09]

- Now we don't know OPT
- We could get approximation $(1+\varepsilon)$ if we knew $OPT(1 \pm \varepsilon)$
- Let's run parallel copies of with guesses $OPT \leq (1+\varepsilon)^i$, $i = 0, 1, 2, \ldots$
- $\ldots$ carefully not to exceed space bounds

k-center clustering: Streaming algorithm

- Cluster first $k + 1$ points in S; gives a lower bound $a \leq OPT$
- Run parallel copies with radius $(1 + \varepsilon)^i a$,
  - i so that radius ranges from a to a/ε
- While k centers suffice, the smallest radius that goes well is a $2(1 + \varepsilon)$-approximation

k-center clustering: Streaming algorithm

- We have a problem when the algorithm tries to open a $(k+1)$-th center, after picking say $y_1, \dots y_k$
- This is because x_{j+1} is at distance $g > a/\varepsilon$ from existing centers
- We realize we should have guessed $OPT > g$

k-center clustering: Streaming algorithm

But we have not worked in vain:

Claim

If $OPT(x_1, \ldots x_j, x_{j+1}, \ldots) = OPT$, then
$OPT(y_1, \ldots y_k, x_{j+1}, \ldots) \leq OPT + 2g$

Forget all previous point but the y_i's, restart again with $a = g$, seeds y_i's

k-center clustering: Streaming algorithm

- Deterministic!
- $2(1+\varepsilon)$ approximation algorithm
- Space & update time: $O(k/\varepsilon \cdot \log(1/\varepsilon))$
  - run i copies, with $(1+\varepsilon)^i a = a/\varepsilon$
  - $i \simeq (1/\varepsilon) \cdot \log(1/\varepsilon)$