




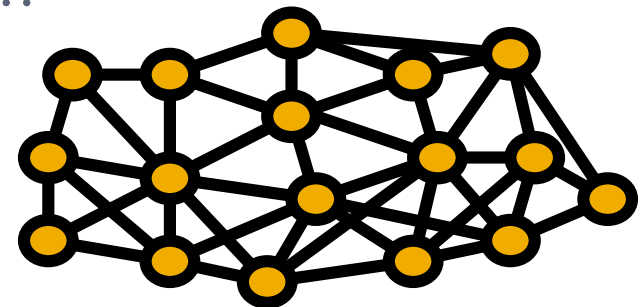
Parameterized and Promised Streaming: Making Big Data More Accessible



Mohammad T. Hajiaghayi
University of Maryland

Overview of My Research

- ▶ Provide **Algorithmic** solutions & provable guarantees
 - ▶ General algorithmic **frameworks** *for any data*
 - ▶ Develop **structural results** to enable these algorithms
 - ▶ Validate our implemented algorithms on **real-world data**, if resources are available.
- ▶ Main subjects of studies are Graphs/Networks
 - ▶ **Nodes** = computers, people, locations, ...
 - ▶ **Edges** = links, relationships, roads, ...



Real-World Networks

► Social networks

- ▶ Google+, Facebook, Twitter, coauthorship, citation, phone calls, ...

► Biological networks

- ▶ Brain connectome, protein interactome, disease networks

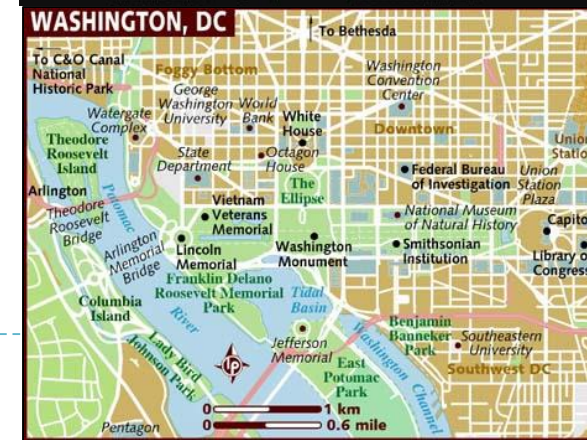
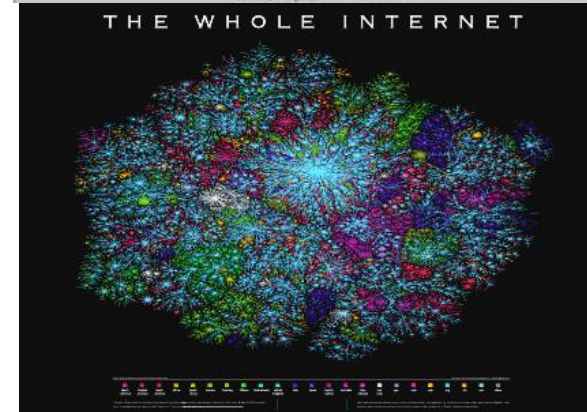
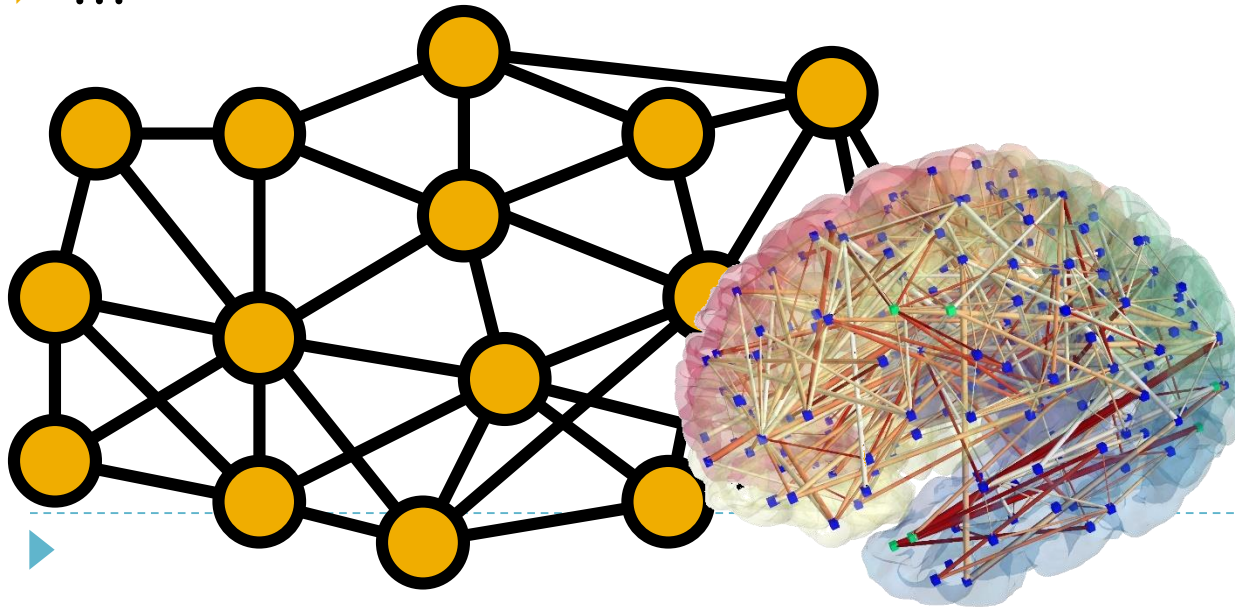
► Computer networks

- ▶ Internet IP, web graph, internet backbone, wireless/sensor net.,...

► Transportation networks

- ▶ road maps, flight tracks, train maps, ...

▶ ...



Our Algorithmic Goals

1. **Correct** (optimal solution)
 2. **Fast** (polynomial time)
 3. **Hard** problems (NP-hard)
- ▶ **Pick any two** (assuming $P \neq NP$)



Classic Algorithms

1. **Correct** (optimal solution)
2. **Fast** (polynomial time)
3. ~~**Hard** problems (NP-hard)~~

- ▶ Most early algorithms
- ▶ Most undergraduate algorithms

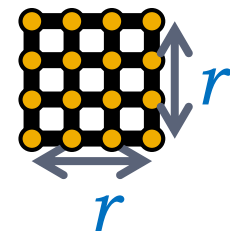


Approximation Algorithms

1. **Correct** (optimal solution)
 2. **Fast** (polynomial time)
 3. **Hard** problems (NP-hard)
- ▶ Guarantee solution is **near optimal**: within some multiplicative factor (1.5, 2, $\log n$, etc.)
 - ▶ **PTAS** (Polynomial Time Approximation Scheme): within factor $1 + \varepsilon$ for *any* specified $\varepsilon > 0$



Fixed-Parameter Algorithms



1. **Correct** (optimal solution)
2. **Fast** (polynomial time)
3. **Hard** problems (NP-hard)

- ▶ Confine exponential growth to **parameter** k (often the optimum solution) other than (and smaller than) input size n
- ▶ Bad: $\exp(n), n^k$
- ▶ Good: $f(k) n^{O(1)}$ e.g. $2^k n$ or even $2^{2^k} n^3$
- ▶ Great: $2^{O(\sqrt{k})} n^{O(1)}$ — **subexponential**

Algorithmic Game Theory

1. **Correct** (optimal solution)
 2. **Fast** (polynomial time)
 3. **Hard** problems (NP-hard)
- ▶ Some of the above goals with a new goal
- *. **Having selfish agents** (agents involved in the algorithm have their own incentives)



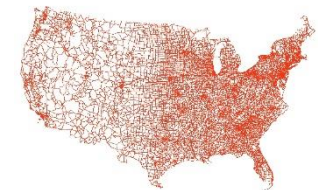
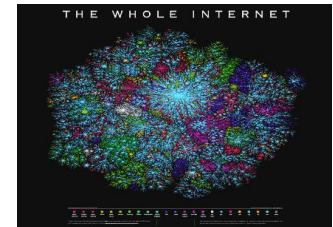
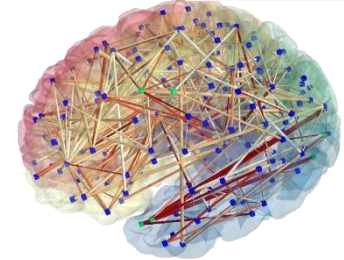
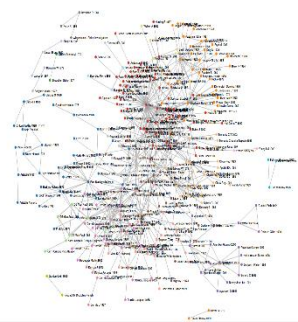
Streaming/Online Algorithms

1. **Correct** (optimal solution)
 2. **Fast** (polynomial time)
 3. **Hard problems** (NP-hard)
- ▶ Some of the above goals with a new goal
- *. **Partial access to input** (often due to BIG DATA the input arrives bit by bit; no access to the whole data)
- ▶ We **focus** more on these types of algorithms **in this talk**



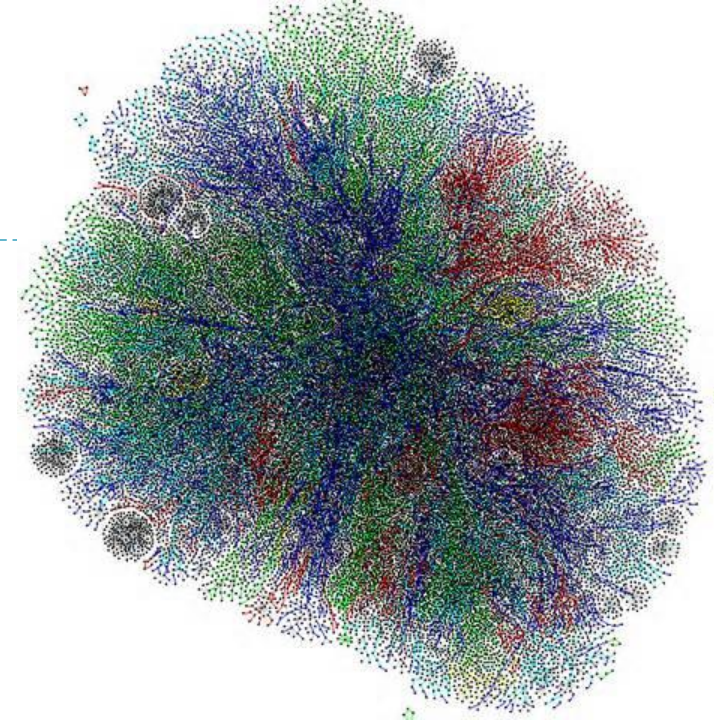
BIG Real-World Networks Are Everywhere

- ▶ **Social** networks: Google+, Facebook, and Twitter, citation network (10^9 nodes)
- ▶ **Biological** networks: brain connectome (10^9 nodes)
- ▶ **Computer** networks: Internet IP/web graph (2^{32} nodes)
- ▶ **Transportation** networks: US road map by GoogleMap (10^8 intersection nodes)



Problems to Solve

- ▶ Many natural problems on big graphs:
 - ▶ Finding botnets/spam detection
 - ▶ Community detection
 - ▶ Reachability/distance between nodes
 - ▶ Summarization/sparsification/visualization
 - ▶
- ▶ ***Traditional and very basic*** optimization problems play critical roles in solving above natural problems:
 - ▶ matching/vertex cover
 - ▶ set cover/hitting set
 - ▶ densest subgraph
 - ▶ cut problems
 - ▶ connectivity problems
 - ▶ ...



Resource Restrictions and Limitations

- ▶ Often only **partial access** to manipulate data which, e.g.,
 - ▶ Stored in a relatively slow huge external hard-drive/blu-ray disc
 - ▶ Stored somewhere in a network cloud (no space to download)
 - ▶ Comes too fast and is too much to store, e.g., internet routers
- ▶ Potential users may not have **supercomputers (cluster)** to manipulate it as well, e.g.,
 - ▶ Cell phone user may want to verify blockchain
(a 33GB database of all Bitcoin transactions)
 - ▶ E.g., to profile the business's past behavior.
 - ▶ Rare events may need ad-hoc search over heterogeneous data from multiple sources not index for this use
 - ▶ E.g., data from store cameras, social media,... in Boston Marathon Bombing
 - ▶ Internet providers may need statistical data of packets through routers



Simplest Way to Handle BIG Data

- ▶ We investigate the **simplest** but the **most fundamental** way by assuming
 - ▶ Data arrives in a natural order outside the algorithm's control
 - ▶ There is only enough memory to store a small summary of data
 - ▶ Our algorithm can view this stream once or a small (constant) number of passes
- ▶ This model is particularly appropriate for **temporally dynamic data**
 - ▶ e.g., online social networks, Bitcoin transactions, internet routers
- ▶ As a **byproduct** complex network analysis can be performed on a single computer (even a mobile device)

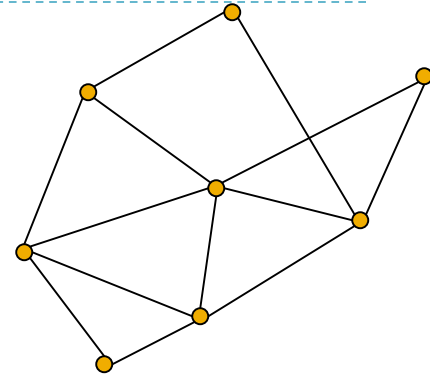


Mapreduce and Other Distributed Models

- ▶ Our solution is often **smart sampling** and **hashing** which are naturally parallel as well
 - ▶ For example, in **MapReduce**, the hash functions can be shared state among all machines
 - ▶ This allows Map function to output each sample under each hash function
- ▶ Making it straightforward to implement in a variety of popular distributed models such as **MapReduce** or **Pregel**.



Streaming Network Model

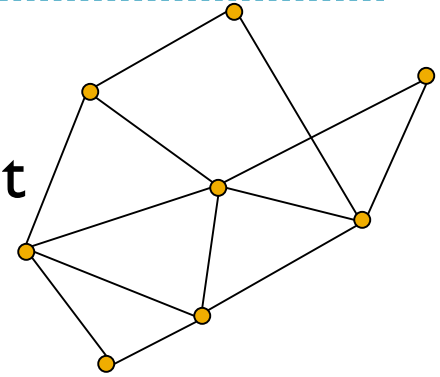


- ▶ The “you get one chance” model:
 - ▶ See each edge once
 - ▶ Space used must be sublinear in the size of the input
 - ▶ Analyze time to process each edge, accuracy of answer, ..
- ▶ Variations within the model:
 - ▶ One pass or a small (constant) number of passes?
(we focus on one pass in this talk)
 - ▶ Insertions only, or edges added and deleted (dynamic)?



Often Streaming Is Hard!

- ▶ With sublinear in n (nodes) space, life is difficult
 - ▶ Cannot remember whether or not a given edge (or even a vertex) was seen
 - ▶ Standard relaxations, specifically randomization, do not help
 - ▶ Formal hardness proved via communication complexity
- ▶ Different relaxations are needed to make any progress
 - ▶ Relax **space**: allow linear in n space
 - ▶ “Semi-streaming”: linear in n (nodes) but sublinear in m (edges)
 - ▶ Make **assumptions about input** – the promised streaming model
 - ▶ “Strictly streaming”: sublinear in n , polynomial or logarithmic



Promised Streaming

- ▶ We know often very **practical** and **reasonable** restrictions/assumptions on the input instance which is **promised** to us, e.g., about
 - ▶ edge density or graph structure (e.g., many real massive graphs are not dense)
 - ▶ cost/size of the solution
- ▶ Some examples
 - ▶ The input graph is a planar or bounded degenerate graph
[Esfandiari, H., Liaghat, Monemizadeh, Onak, SODA 2015]
 - ▶ The edges are coming in a random order or from some distributions
[Esfandiari, H., Monemizadeh, Submitted manuscript]
 - ▶ Parameterized Streaming [detailed in this talk]
[Chitnis, Cormode, H., Monemizadeh, SODA 2015],
[Chitnis, Cormode, Esfandiari, H., Monemizadeh, SPAA 2015],
[Chitnis, Cormode, Esfandiari, H., McGregor, Monemizadeh, Vorotnikova, SODA 2016]
 - ▶ Any combinations of above

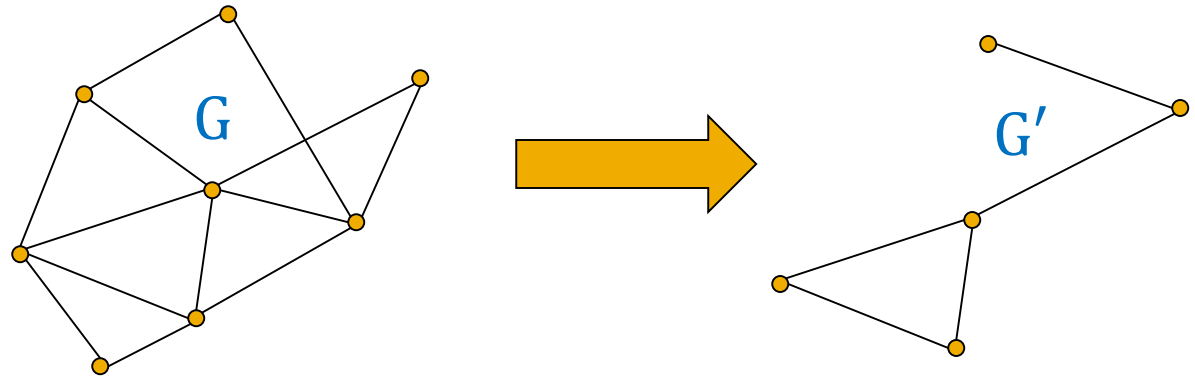


Parameterized Streaming

- ▶ Streaming with **space** being a function of **k**, the solution size (often much smaller than **n**, the number of nodes)
- ▶ More precisely we seek space **$f(k).polylog(n)$**
- ▶ Draw inspiration from **fixed parameter-tractability** (FPT)
 - ▶ For (NP) Hard problems: assume solution has size **k**
 - ▶ Naïve solutions have time **$exp(n), n^k$**
 - ▶ Seek solutions with time **$f(k).poly(n)$** – reasonable for small **k**
 - ▶ Report “no” if size is greater than **k**
- ▶ For simplification, we start with **Vertex Cover** a famous NP-hard problem, though our framework is much more general



Kernelization

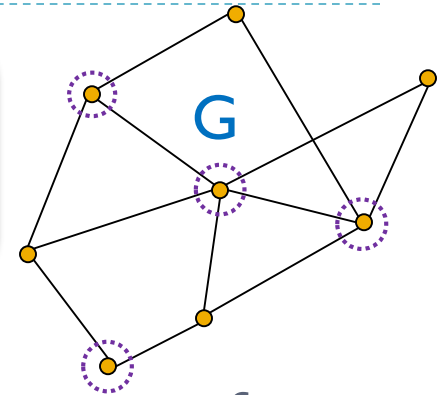


- ▶ A key technique is **kernelization**
 - ▶ Reduce input (graph) G to a smaller (graph) instance G'
 - ▶ Such that solution on G' corresponds to solution on G
 - ▶ Size of G' is **poly(k)** (or even exponential in k)
 - ▶ So naïve (exponential) algorithm on G' is FPT
- ▶ Kernelization is a **powerful technique**
 - ▶ Any problem that is FPT has a kernelization solution



Kernelization on Graph Streams

Vertex cover (*hub detection*): find a set of vertices S so every edge has at least one vertex in S



- ▶ A simple algorithm for **insertions** only
 - ▶ Maintain a matching M (greedily) on the graph seen so far
 - i.e., store an incoming edge in G' if does not share endpoints with previous stored edges of M
 - ▶ For any v in the matching, keep up to k edges incident on v in G'
 - (**Why?** if the degree becomes larger than k , v will be in the vertex cover)
 - ▶ If $|M| > k$, quit
 - (**Why?** since size of matching M is a lower bound on size of vertex cover and thus vertex cover has size more than k)
 - ▶ At any time, run kernelization algorithm on stored edges G'

Why Does It Work?

- ▶ **Proof outline:** argue that kernelization on G' mimics that on G
 - ▶ Every step on G' can be applied to G correspondingly
 - ▶ We keep “enough” edges on a node to test, if it is high-degree
- ▶ As a result we obtain **any** approximation for vertex cover on G' and thus G
- ▶ Guarantees $O(k^2)$ space: at most k edges on $2k$ nodes of M
 - ▶ We prove *tight lower bound* of $\Omega(k^2)$ in the streaming model for vertex cover via *communication complexity*



Kernelization on Dynamic Graph Streams

- ▶ Much more challenging case: **dynamic graph streams**
 - ▶ Edges are inserted or deleted (**friend** or **unfriend** in Facebook)
- ▶ Previous algorithm **breaks**: if a matched edge is deleted we no longer can update the (**maximal**) matching since we **forgot** edges
- ▶ First we solved a **promised** problem that vertex cover at all times is at most size **k** [Chitnis, Cormode, H., Monemizadeh, SODA 2015] (simplified in [Chitnis, Cormode, Esfandiari, H., Monemizadeh, SPAA 2015])
- ▶ **Removing the promise** was an **important open problem** and very recently solved
[Chitnis, Cormode, Esfandiari, H., McGregor, Monemizadeh, Vorotnikova., SODA 2016]
- ▶ Need additional existing technologies, in particular **k-Sparse Recovery** and **k-Sampler** data structures



k -Sparse Recovery and k -Sample Algorithm

- ▶ **k -Sparse Recovery:**
 - ▶ A data structure which accepts **insertions** and **deletions** of n elements
 - ▶ At any moment if the current number of elements stored is at most k , then these can be recovered in full
- ▶ **Deterministic construction:** requires $O(k \text{ polylog } n)$ space
- ▶ **k -Sampler (l_0 sampling):**
 - ▶ A data structure which accepts **insertions** and **deletions** of n elements
 - ▶ At any moment it can provide a sample of size k from the elements stored in it, provided there are at least k elements in it
- ▶ **Randomized construction:** requires $O(k \text{ polylog } n)$ space
(via k -sparse recovery)



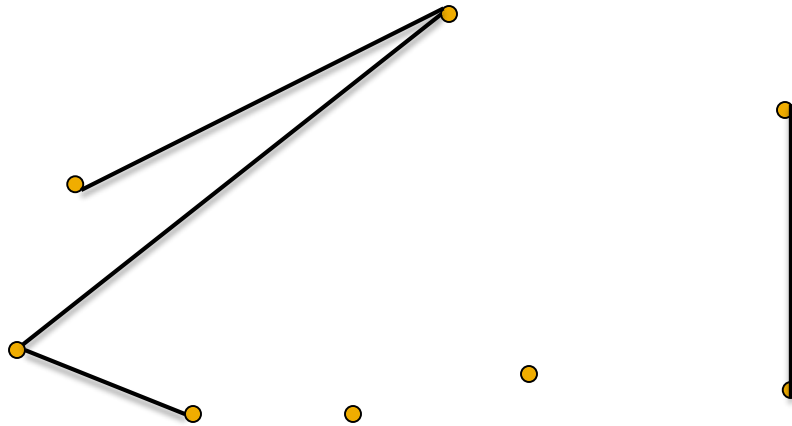
Data Structures for Promised Algorithm

- ▶ Our data structure has two parts
 - ▶ A set of k -sparse recoveries (as well k -samplers)
 - ▶ Each keeps the neighboring edges of a vertex with degree at least k
 - ▶ A set of edges L
 - ▶ These edges are induced by vertices of degree less than k



Insert Edges: Both Endpoints Have Degrees $< k$

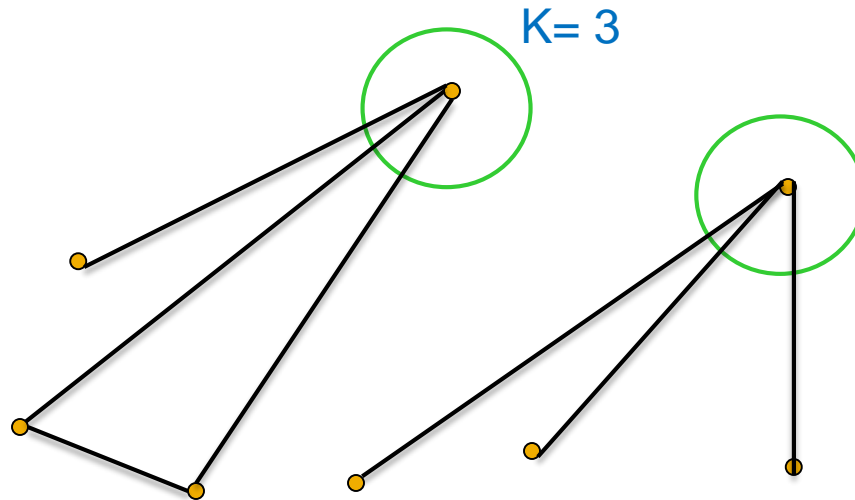
$k = 3$



- ▶ After the insertion both endpoints have degrees less than k :
 - ▶ Add the edge to the set of edges L

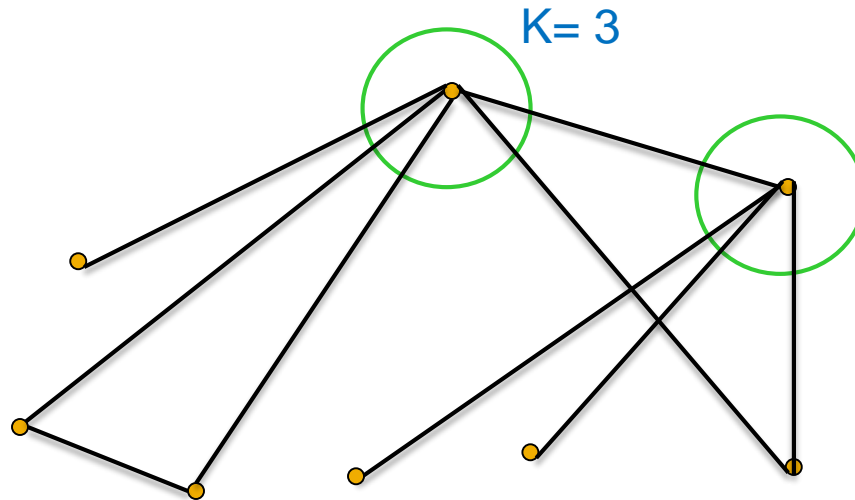


Insert Edge: Endpoint Has Degree k



- ▶ After the insertion an endpoint has degree k :
 - ▶ Initiate an empty k -sparse recovery (as well as k -sampler) for this vertex
 - ▶ Move all of the neighbors of this vertex from L to k -sparse recovery (as well as k -sampler)
 - ▶ Put a **time stamp** on k -sparse recovery

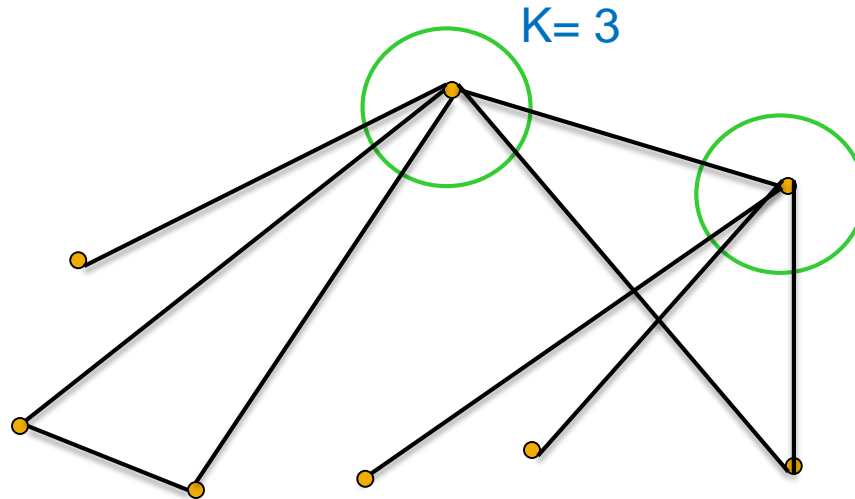
Insert Edge: Endpoint Has k -Sparse Recovery



- ▶ If one of endpoints has a k -sparse recovery (as well as k -sampler)
 - ▶ Just add the edge to the k -sparse recovery (as well as k -sampler)
- ▶ If both endpoints have k -sparse recoveries
 - ▶ Add the edge to the k -sparse recovery (as well as k -sampler) with **smaller** time stamp

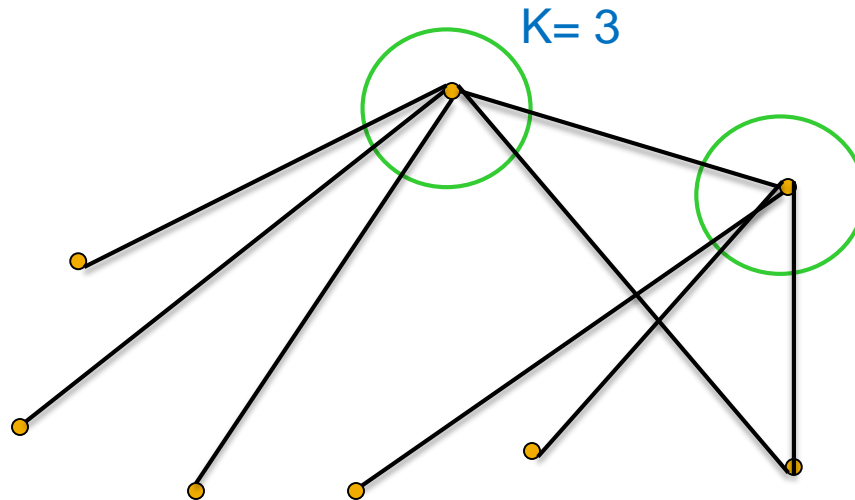


Delete Edge: Edge Is in L



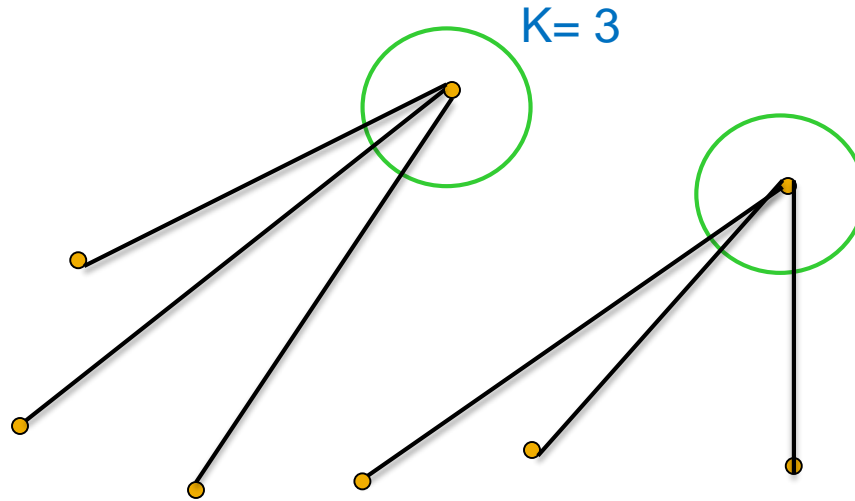
- ▶ If the deleted edge is in the set of edges L
 - ▶ Just delete it from the set of edges L

Delete Edge: Edge Is in One k -Sparse Recovery



- ▶ If one of the endpoints has k -sparse recovery
 - ▶ Just delete the edge from the k -sparse recovery (as well as k -sampler)
- ▶ If both endpoints have k -sparse recoveries
 - ▶ Delete the edge from the k -sparse recovery (as well as k -sampler) with the **smaller** time stamp

k -Sparse Recovery Has $< k$ Edges



- ▶ If after a deletion, a k -sparse recovery has less than k edges
 - ▶ Retrieve edges from the k -sparse recovery and add them to the edge set L
 - ▶ Remove k -sparse recovery (as well as k -sampler)

Proof Outline

- ▶ We have one k -sparse recovery (as well as k -sampler) for each vertex with degree $\geq k$
 - ▶ We know all vertices with degree $\geq k$ (all will be in the vertex cover)
 - ▶ We can sample a set P of k edges from k -samplers
(only use of k -samplers)
- ▶ We know all the other edges in the edge set L
- ▶ $L \cup P$ is enough to create the kernel G' to find a vertex cover



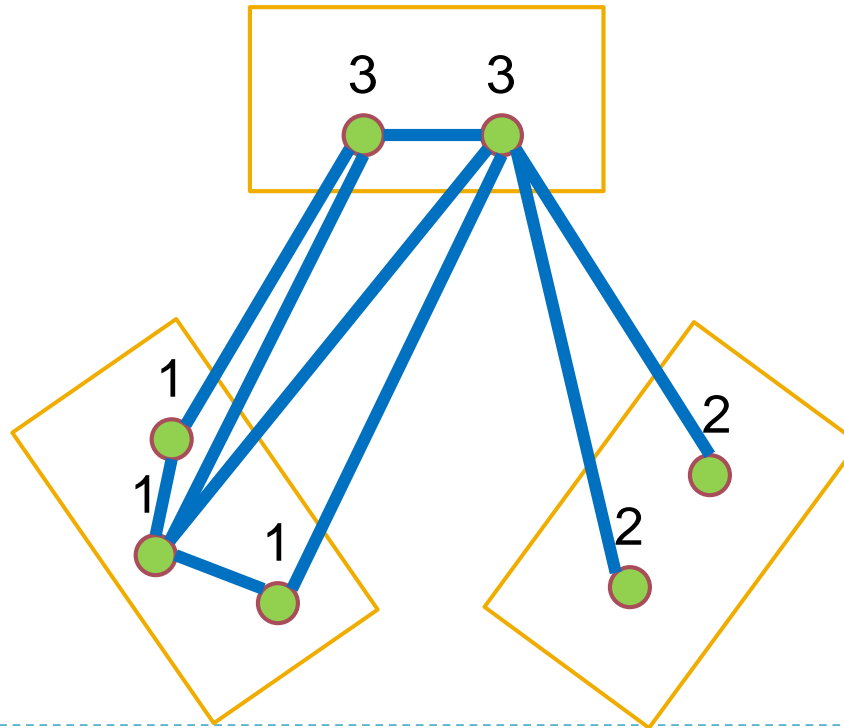
Space Complexity

- ▶ We need $O(k^2 \text{ polylog } n)$ space for k -sparse recoveries (as well as k -samplers) since
 - ▶ Each k -sampler corresponds to a vertex in the vertex cover
 - ▶ Size of the vertex cover is not more than k (by the **promise**)
 - ▶ Thus the number of k -sparse recoveries (as well as k -samplers) is $\leq k$
 - ▶ Each k -sparse recoveries (as well as k -samplers) requires $O(k \text{ polylog } n)$ space
- ▶ We have at most $O(k^2)$ edges in L
 - ▶ Each vertex in the vertex cover covers at most k of these edges
 - ▶ Size of the vertex cover is not more than k
- ▶ In total $O(k^2 \text{ polylog } n)$ space
- ▶ **Very simple to implement...**



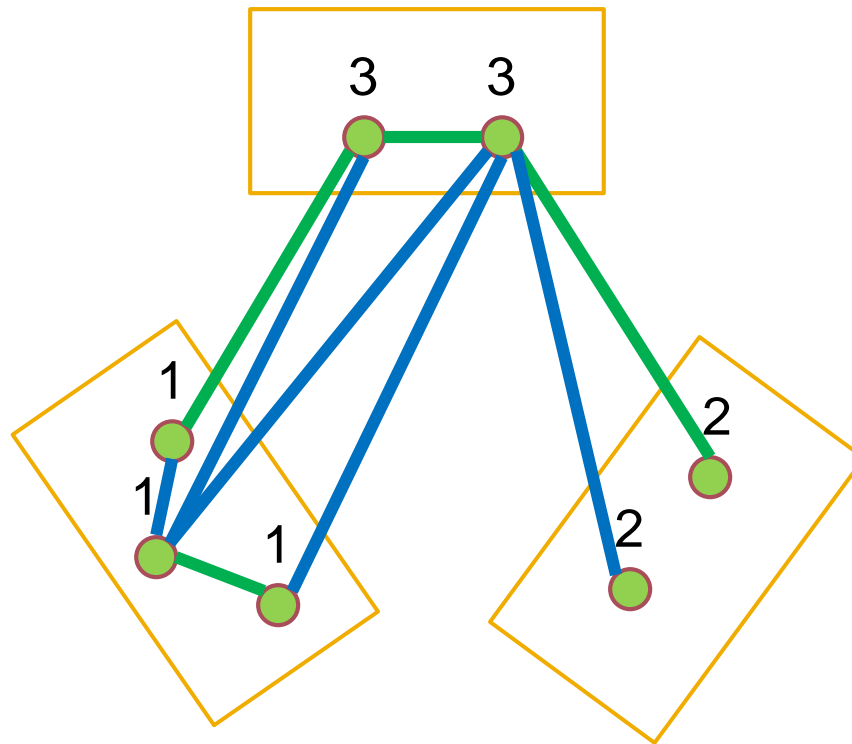
Removing the Promise

- ▶ Let k be the size of vertex cover at the end
(before k was the max size at ALL times)
- ▶ Needs a more sophisticated **vertex sampling**
- ▶ Hash each vertex to a number in $[1, k]$



Removing the Promise

- ▶ For each pair of numbers $(i, j) \in ([1, k], [1, k])$, maintain one edge of data stream, if any exists



Algorithm Summary

- ▶ We run this procedure $O(\log k)$ times in parallel
- ▶ Let G' be the graph containing all stored edges

Assuming that size of the vertex cover is k at the end, w.h.p. G' and G have the same set (and size) vertex cover (and matching)

- ▶ Thus we only need to run any approximation algorithm for vertex cover in G' (to find one in G)
- ▶ We use $O(k^3 \text{polylog } n)$ space in total since
 - ▶ One k -sampler for each pair $(i, j) \in ([1, k], [1, k])$ ($O(k^2)$ in total)
- ▶ It can be improved to $O(k^2 \text{polylog } n)$ with more sophistication
- ▶ The results are very general and work for optimization of any property preserved under “vertex contraction” including:
 - ▶ matching, b -matching, hitting set, set cover, k -colorable subgraph, several maximum subgraph problems, etc.



Experimental Results for BIG DATA:

Brain and World Web Graphs

Brain graphs:

<http://mrbrain.cs.jhu.edu/disa/download/>

World web graph:

<http://webdatacommons.org/hyperlinkgraph>

- ▮ Some raw data size are over 1.3 Tb
- ▮ Refined data size is 200 Gb

Implementation Difficulties

- ▮ A typical graph needs 200 GB space
 - ▮ Most classical algorithms (i.e BFS, DFS, etc) cannot be implemented without additional hard drives
 - ▮ Most editors crash while opening the data
- ▮ 25m to lookup an entry
- ▮ 1h,40m to sort the data

Experimental Results

▮ World Graph:

▮ Vertex: **subdomain/host**.

▮ Edge: **hyperlink** between subdomain/hosts

▮ $n = 101,717,775$ $m = 2,043,203,933$

Experimental Results

👉 Brain Graph (KKI):

👉 Vertex: Brain cell

👉 Edge: Connections between cells

👉 $n = 34,143,521$ $m = 2,679,090,553$

Experimental Results

👉 Brain Graph (MRN):

👉 Vertex: Brain cell

👉 Edge: Connections between cells

👉 $n = 74,256,292$ $m = 6,516,647,674$






Implemented Algorithms

Dataset	Time	Memory	Matching Size
Webgraph	2h,15m	1.8 MB	24,432,528
KKI	3h,10m	2 MB	6,302,146
MRN	7h,25m	1.3 MB	27,189,228

Running time on Samsung Laptop with an external hard-drive

Codes and Data Available at BigDND Website

← → ↺  projects.csail.mit.edu/dnd/

 Apps  Bookmarks  EDAS (258776 - haji...  OneStop  Inbox (2,879)  megabus.com  md.speedtest.rcn.net  Customer Service  2012 Elect

BigDND: Big Dynamic Network Data

[Erik Demaine](#) (MIT) & [MohammadTaghi Hajiaghayi](#) (UMD)

Networks are everywhere, and there is an increasing amount of data about networks viewed as graphs: nodes and edges/connections. But this data typically ignores a third key component of networks: time. This repository provides **free, big datasets for real-world networks** viewed as a dynamic (multi)graph, with two types of temporal data:

1. A timeseries of **instantaneous edge events**, such as messages sent between people. Many such events can occur between the same pair of nodes.
2. Timestamped **edge insertions and edge deletions**, such as friending and defriending in a social network. Generally only one such edge can exist at any specific time, but the same edge can be added and deleted multiple times.

Our hope is that these datasets will promote new research into the dynamics of complex networks, improving our understanding of their behavior, and helping the community to experimentally evaluate their big-data algorithms: approximation, fixed-parameter, external-memory, streaming, and network-analysis algorithms.

Help us:

- If you have a **dynamic network dataset**, email us at dnd@csail.mit.edu with a brief description about the data, its format, its license, and how/where to download it. We will link to it with appropriate credit/citation.
- If you have interesting **visualizations and/or analysis** of these data sets, email us at dnd@csail.mit.edu and we will post it with appropriate credit/citation.

DBLP Data

News: [Ranking of CS departments based on the number of DBLP papers in theoretical computer science has just been released. A similar ranking for other areas of CS and a general CS ranking is coming in the future.](#)

The computer science bibliography [DBLP](#) offers its [entire dataset of bibliography entries in XML format](#) under the [Open Data Commons Attribution License \(ODC-BY 1.0\)](#). The data is updated daily, and includes years with each publication, making for timeseries data. As of October 2014, it consists of 4,215,613 papers and 9,086,030 edges between papers and authors.

We have developed [free software](#) to compute [timestamped graph data](#) for this DBLP data.

Social Network Data from MPI-SS

The Max Planck Institute for Software Systems has gathered [several large dynamic network datasets](#) in a variety of social networks. This data is publicly available by emailing Alan Mislove at amislove@mpi-sws.org.

- **Facebook**: 60,290 users, 1,545,686 friendships, 838,092 timestamped wall posts. [[Viswanath, Mislove, Cha, Gummadi 2009](#)]
- **Flickr**: 1,620,392–2,570,535 users, 11,195,144 photos, 17,034,807–33,140,018 timestamped links, 34,734,221 timestamped favorite markings, from November 2–December 3, 2006 and February 3–May 18, 2007. [[Cha, Mislove, Gummadi 2009](#)]
- **YouTube, LiveJournal, and Orkut** data also available

Google+ Social Network Data with Node Attributes

UC Berkeley has published [four snapshots taken at four times of the same subset of the Google+ social network](#). Thus each object has a coarse notion of timestamp, between 1 and 4. The network starts with 4,693,129 nodes and 47,130,325 edges, and grows to 28,942,911 nodes and 462,994,069 edges. Nodes additionally have optional attributes of employer, school, major and places lived.

Twitter Data

The Max Planck Institute for Software Systems has gathered [Twitter data](#) encompassing 54,981,152 user accounts, 1,963,263,821 follow links (based on a snapshot in August 2009, no timestamps), and 1,755,925,520 timestamped tweets. This data is publicly available by emailing twitter-contact@mpi-sws.org.

Paper Citation Data

In these networks, nodes represent papers/publications and directed edges represent citations.

- **arXiv.org HEP-PH** (high energy physics phenomenology). 34,546 timestamped papers and 421,578 citations, from January 1993 to April 2003. Available for download from [SNAP](#). [From [KDD Cup 2003](#)]
- **arXiv.org HEP-TH** (high energy physics theory). 27,770 timestamped papers and 352,807 citations, from January 1993 to April 2003. Available for download from [SNAP](#). [From [KDD Cup 2003](#)]
- **USA Patents**. 3,774,768 timestamped patents and 16,518,948 citations, from 1975 to 1999. Available for download from [SNAP](#). [From [National Bureau of Economic Research](#)]

HUGE: Brain Connectome Data

Johns Hopkins University's [Open Connectome Project](#) has gathered [a huge amount of brain network/connectome data](#). All data is available for download.

HUGE: Web Graph Data

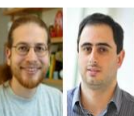
University of Mannheim has gathered a [Web Hyperlink Graph](#) based on the [Common Crawl](#) 2012 web corpus, featuring 3,500,000,000 webpages and 128,000,000,000 hyperlinks. All data is available for download.

Codes

We have provided two codes for measuring quantitative properties of graphs. These algorithms are specifically designed to run on large graphs.

1. [matching.cgo](#) approximates the size of the largest matching of a graph based on the [work](#) of Esfandiari et al.
2. [dense.cgo](#) approximates the size of the densest subgraph of a graph based on the [work](#) of Esfandiari et al.

About Us



We have been designing graph algorithms for our whole lives, and collaborating together since 2001 with over 60 joint publications. **Erik Demaine** is a MacArthur Fellow, Sloan Fellow, Guggenheim Fellow, Presburger Award recipient, and Polyá Lecturer; he has published over 400 papers with over 340 co-authors, and has given over 225 plenary and invite Investigator recipient, Google Faculty Research Award (twice); he has published over 165 papers with over 160 co-authors, has given over 45 invited talks around the world, and has over 10 granted or filed patents.

Our work is currently supported in part by DARPA GRAPHS grant FA9550-12-1-0423, NSF grant CCF-1161626, NSF CAREER award 1053605, and ONR YIP award N000141110662.

Looking Forward

- ▶ Can other parameterized-algorithm ideas inspire new streaming algorithms and vice versa?
- ▶ Implementation in MapReduce and other distributed models
- ▶ **More algorithms with provable guarantees specifically when we know real-world assumptions on input**
- ▶ **More algorithmic frameworks (meta-algorithms) for streaming**
- ▶ Special thanks to Google Faculty Research Award (*twice*), NSF IIS (BIGDATA), DARPA GRAPHS, ONR Young Investigator Award, NSF CAREER, NSF CCF (Medium)

Thank you!

