Matching and Covering in Streaming Graphs

Graham Cormode

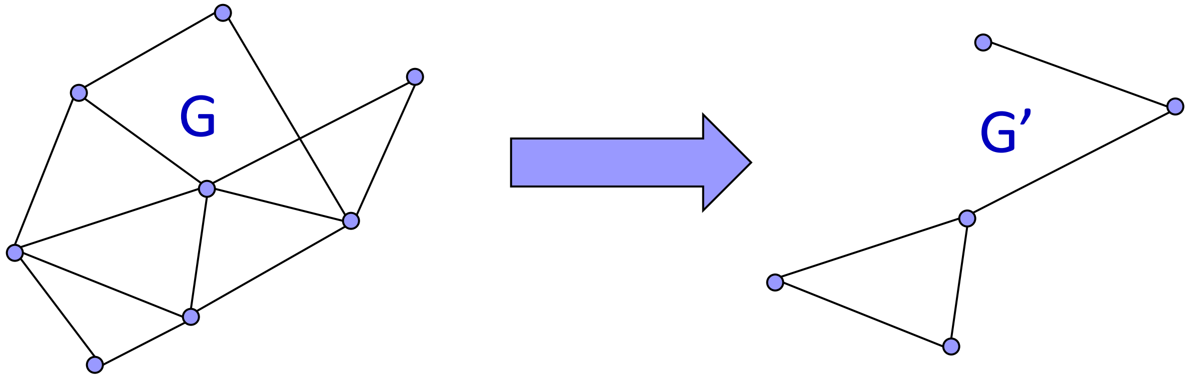
g.cormode@warwick.ac.uk

Joint work with

Rajesh Chitnis, Hossein Esfandiari, MohammadTaghi Hajiaghayi (UMD)

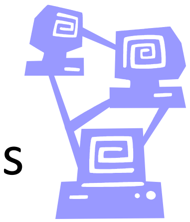
S. Muthukrishnan, Morteza Monemizadeh (Rutgers)

Hossein Jowhari (Warwick)

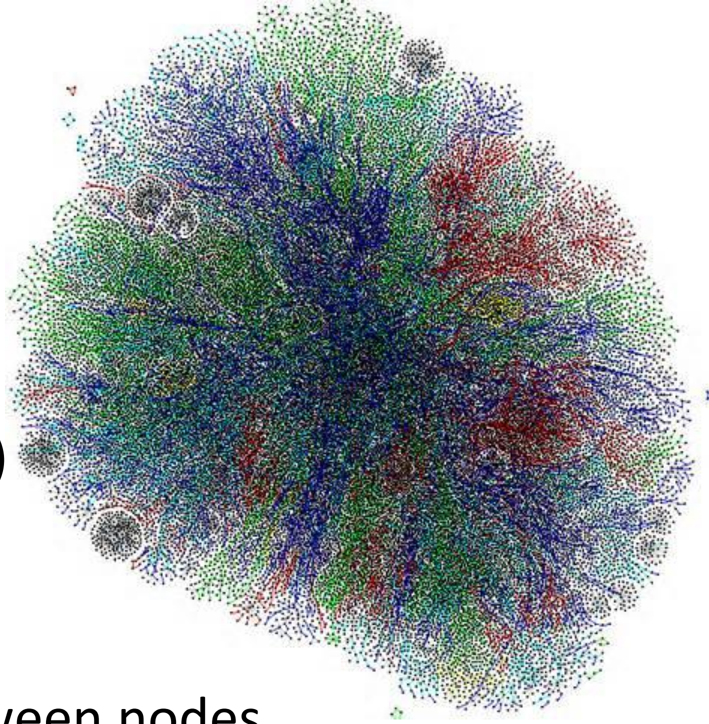


A tale of three graphs

- The telephone call-graph
  - Each edge denotes a call between two phones
  - 2-3 × 10^9 calls made each day in US, maybe 0.5 × 10^9 phones
  - Can store this information (for billing etc.)
- The social graph
  - Each edge denotes a link from one person to another
  - > 10^9 people, > 10^{11} links
  - Store people (nodes) in memory, but maybe not all links
- The IP graph
  - Each edge denotes communication between IP addresses
  - 10^9 packets/hour/router in a large ISP, 2^{32} possible addresses
  - Not feasible to store nodes or edges

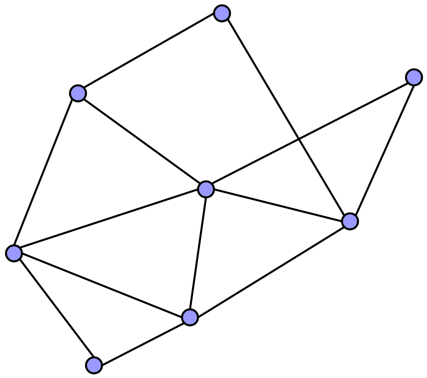


Big Graphs



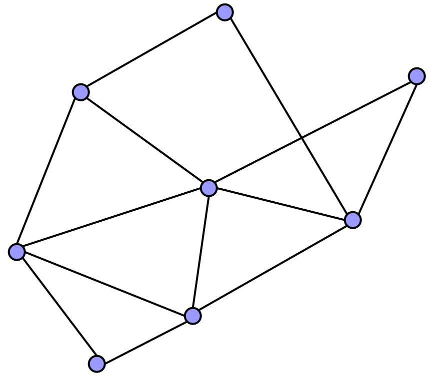
- Increasingly many "big" graphs:
  - Internet/web graph (2^{64} possible edges)
  - Online social networks (10^{11} edges)
- Many natural problems on big graphs:
  - Connectivity/reachability/distance between nodes
  - Summarization/sparsification
  - Traditional optimization goals: vertex cover, maximal matching
- Various models for handling big graphs:
  - Parallel (BSP/MapReduce): store and process the whole graph
  - Sampling: try to capture a subset of nodes/edges
  - Streaming (this talk): seek a compact summary of the graph
    - Ideally, computable by distributed observers

Streaming graph model



- The “you get one chance” model:
  - See each edge only once
  - Space used must be sublinear in the size of the input
  - Analyze costs (time to process each edge, accuracy of answer)
- Variations within the model:
  - See each edge exactly once or at least once?
    - Assume exactly once, this assumption can be removed
  - Insertions only, or edges added and deleted?
  - How sublinear is the space?
    - Semi-streaming: linear in n (nodes) but sublinear in m (edges)
    - “Strictly streaming”: sublinear in n, polynomial or logarithmic

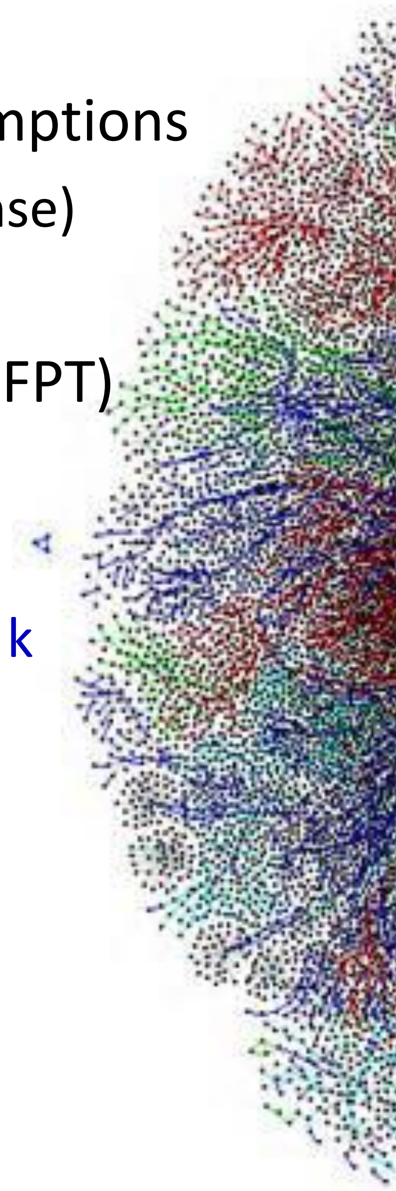
Streaming is hard!



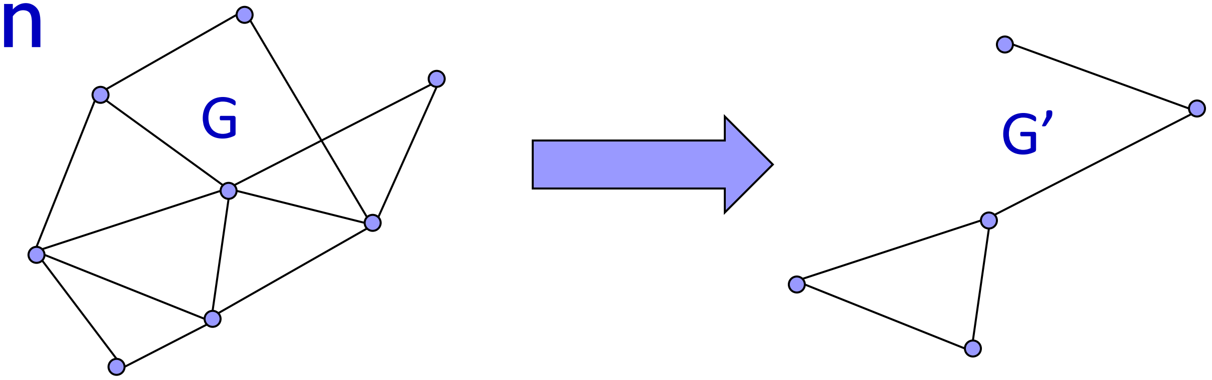
- With sublinear in n (nodes) space, life is difficult
  - Cannot remember whether or not a given edge was seen
  - Therefore, cannot determine (e.g.) whether graph is connected
  - Standard relaxations, specifically randomization, do not help
  - Formal hardness proved via communication complexity
- Different relaxations are needed to make any progress
  - Relax space: allow linear in n space – semi-streaming model
  - Make assumptions about input
    - Solution is not too large: parameterized streaming model
    - Graph has some additional structure: e.g. sparsity assumptions

Parameterized Streaming

- For many “real life” graphs we can make such assumptions
  - About edge density (few real massive graphs are dense)
  - **About cost/size of the solution**
- Draw inspiration from fixed parameter-tractability (FPT)
  - For (NP) Hard problems: assume solution has size k
  - Naïve solutions have cost exp(n)
  - Seek solutions with cost poly(n)exp(k) – OK for small k
  - Report “no” if solution size is greater than k

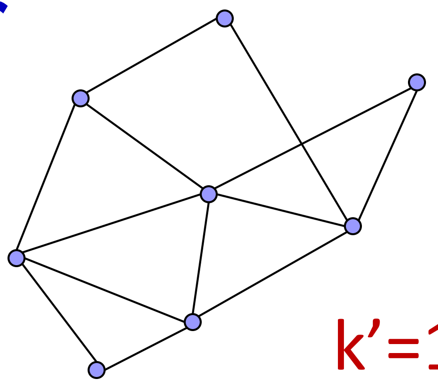


Kernelization



- 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)
  - So naïve (exponential) algorithm on G’ is FPT
- Kernelization is a powerful technique
  - Any problem that is FPT has a kernelization solution

Kernelization for Vertex Cover



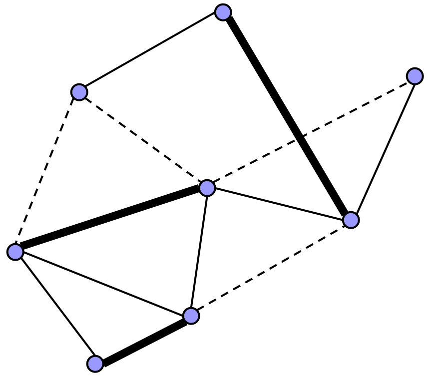
k’=1

Vertex cover: find a set of vertices S so every edge has at least one vertex in S

- Set k'=k, desired size of vertex cover
- Repeat till neither of the following rules can be applied
  1. There is a vertex v in G with degree > k'. v must be in any cover. Remove v and all edges incident on v from G, decrease k' by one.
  2. There is an isolated vertex v in G. Remove v from G.
- If neither rule can be applied, but $m>k'^2$ then G does not have a vertex cover of size at most k’.
- Else, G’ is a kernel with at most $2k'^2$ nodes and k'^2 edges
  - Can run exponential time algorithm on G’ to test for vertex cover

J. F. Buss and J. Goldsmith. Nondeterminism within P, 1993

Kernelization on Graph Streams



- A simple algorithm for insertions only
  - Maintain a matching M (greedily) on the graph seen so far
  - For any v in the matching, keep up to k edges incident on v as G_M
  - If $|M|>k$, quit: any vertex cover must have more than k nodes
  - At any time, run kernelization algorithm on the stored edges G_M
- Key insight: size of M is a lower bound on size of vertex cover
- Proof outline: argue that kernelization on G_M mimics that on G
  - Every step on G_M can be applied to G correspondingly
  - We keep "enough" edges on a node to test if it is high-degree
- Guarantees $O(k^2)$ space: at most k edges on 2k nodes
  - Lower bound of $\Omega(k^2)$ in the streaming model for Vertex Cover
  - Can run with distributed observers, then merge and prune

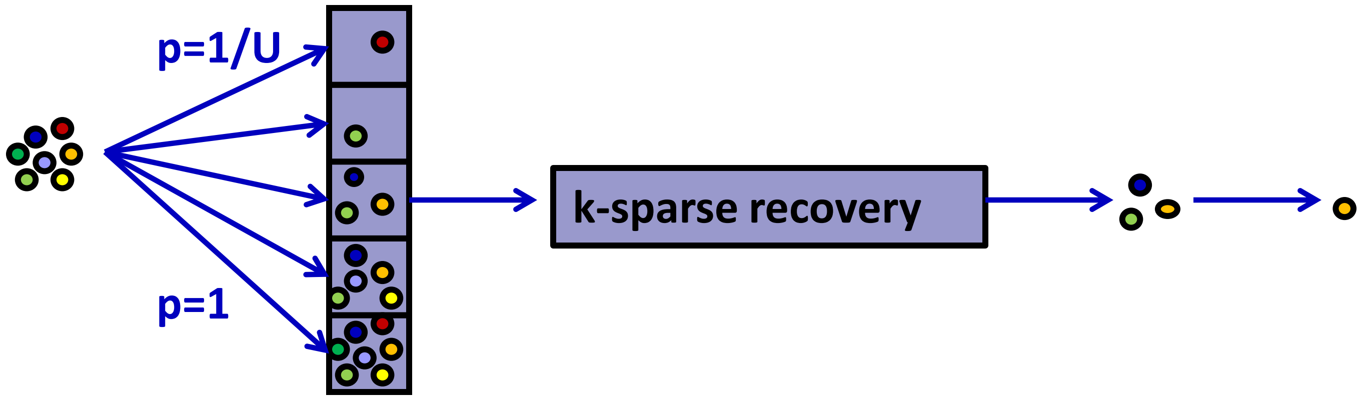
Kernelization on Dynamic Graph Streams

- More challenging case: dynamic graph streams
  - Edges are inserted and deleted, over distributed observers
- Previous algorithm breaks: deleting a matched edge means we no longer have a maximal matching
- Study promise problem that max matching always at most size k
- Need some additional technology: L_0 sampling
  - Allows us to deal with high degree nodes
  - A sketch algorithm: maintains linear transform of input
    - Allows inserts and deletes to be analyzed easily
    - Mergeable: sketches can be "added" to sketch union of inputs

L_0 Sampling

- Goal: sample (near) uniformly from items with non-zero frequency
- General approach: [Frahling, Indyk, Sohler 05, C., Muthu, Rozenbaum 05]
  - Consider input to define a vector of frequencies
  - Sub-sample all items (present or not) with probability p
  - Generate a sub-sampled vector of frequencies f_p
  - Feed f_p to a *k-sparse recovery* data structure
    - Allows reconstruction of f_p if number of non-zero entries $< k$
  - If vector f_p is k-sparse, sample from reconstructed vector
  - Repeat in parallel for exponentially shrinking values of p

Sampling Process



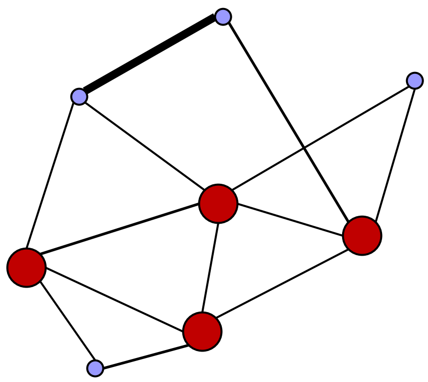
- Exponential set of probabilities, $p=1, ½, ¼, 1/8, 1/16\ldots 1/U$
  - Let $N = F_0 = |\{ i : f_i \neq 0\}|$
  - Want there to be a level where k-sparse recovery will succeed
  - At level p, expected number of items selected S is Np
  - Pick level p so that $k/3 < Np \leq 2k/3$
- Chernoff bound: with probability exponential in k, $1 \leq S \leq k$
  - Pick $k = O(\log 1/\delta)$ to get $1-\delta$ probability

k-Sparse Recovery

- Given vector x with at most k non-zeros, recover x via sketching
  - A core problem in compressed sensing/compressive sampling
- Randomized construction: hash elements to O(k) buckets
  - Elements are probably isolated in each bucket
  - Keep count of items and sum of item identifiers in each cell
  - Sum/count will reveal item id
  - Avoid false positives: keep fingerprint of items in each cell
- Can keep a sketch of size O(k log U) to recover up to k items

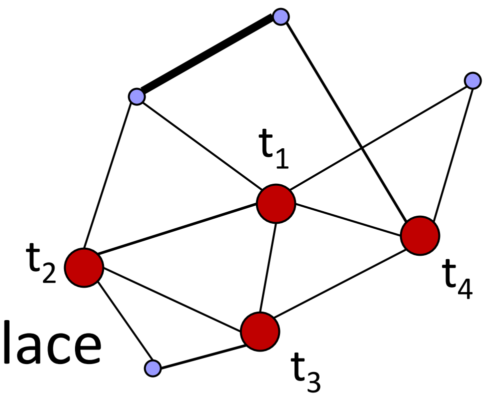
Sum, $\sum_{i\,:\,h(i)=j} i$
Count, $\sum_{i\,:\,h(i)=j} x_i$
Fingerprint, $\sum_{i\,:\,h(i)=j} x_i r^i$

Neighborhood sampling



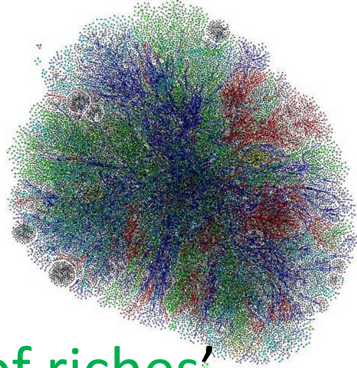
- Back to maximal matchings and vertex cover
  - Algorithm outline: keep information about the graph in L and H
  - H: set of high degree nodes (degree > 2k)
    - Keep an L_0 sketch of the neighbourhood of each node in H
  - L: set of edges neither of whose endpoints is in H
- Given L and H, we can find a maximal matching
  - Recover edges from sketches of H (at most k+1 from each node)
  - Combine with L and greedily find a matching on this set
- Proof outline. We need to argue:
  1. We can maintain L and H correctly
  2. The matching found is good

Maintaining L and H



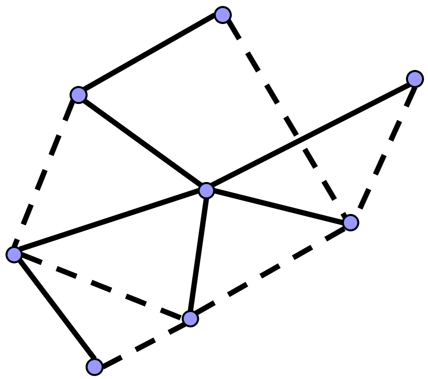
- Invariant: Every edge is stored in exactly one place
  - Use timestamps on nodes becoming heavy to break ties
  - If a light node becomes heavy, put all its edges into a sketch
  - If a heavy node becomes light, can recover all its edges
    - And put these into L
  - Edge deletions delete the edge from the one place it was stored
- Space Analysis:
  - Cannot be more than $2k+1$ high degree nodes in H
    - Else, could find a matching larger than k between them
  - Cannot be more than $4k^2$ edges in L
    - Else could find a larger matching as nodes in L are low-degree
  - Consequently, space used is $O(k^2 \text{ polylog}(k))$

Correctness of the algorithm



- Key point: for high degree nodes, we have a 'surfeit of riches'
  - Doesn't matter which edges we remember, there are enough to match this node somehow
  - So can match all nodes in H using the recovered edges
  - L consists of all edges not incident on H, so have these exactly
  - Hence can greedily find a maximal matching for the graph
- Summary: can find a maximal matching in $O\tilde{}(k^2)$ space
  - Under the promise that the matching is always at most k in size
  - Centralized: need to track membership of L and H
  - Use the maximal matching in an FPT vertex cover algorithm
- Can remove the limitations with a hash/sampling based approach
  - See SODA'16 paper with McGregor and Vorotnikova

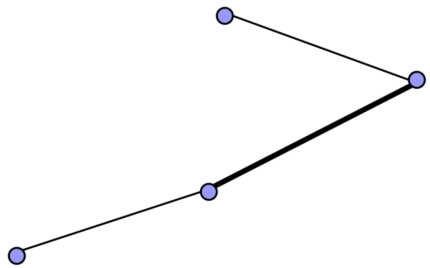
Matching under sparsity



- Many graphs (phone, web, social) are ‘sparse’
  - Asymptotically fewer than $O(n^2)$ edges
- Characterize sparsity by bounded arboricity c
  - Edges can be partitioned into at most c forests
  - Equivalent to the largest local density, $|E(U)|/(|U|-1)$ for $U \subseteq V$
    - $E(U)$ is the number of edges in the subgraph induced by U
  - E.g. planarity corresponds to 3-bounded arboricity
- Use structural properties of sparse graphs to give results

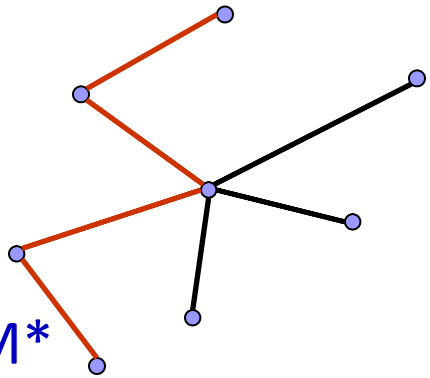
α -Goodness

- Define an edge in a stream to be α-good if neither of its endpoints appears more than α times in the suffix of the input
  - Intuition: This definition sparsifies the graph but approximately preserves the matching
  - Estimating the number of α-good edges is easier than finding the matching itself



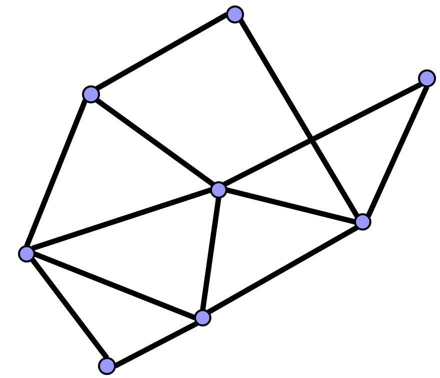
Edge is 1-good if at most 1 edge on each endpoint arrives later

Easy case: trees (c=1)



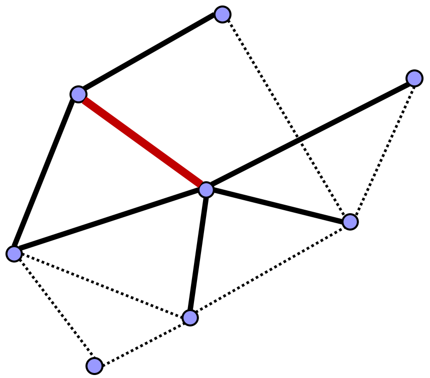
- Consider a tree T with maximum matching size M^*
- $|E_1| \leq 2M^*$: The subgraph E_1 has degree at most 2, no cycles
  - So can make a matching for T from E_1 using at least half the edges
- $|E_1| \geq M^*$: Proof by induction on number of nodes n
  - Base case: $n=2$ is trivial
  - Inductive case: add an edge (somewhere in the stream) that connects a leaf to an internal node
    - Either M^* and $|E_1|$ stay the same, or $|E_1|$ increases by 1 and M^* increases by at most 1
    - At most 1 edge is ejected from E_1, but the new edge replaces it

General case



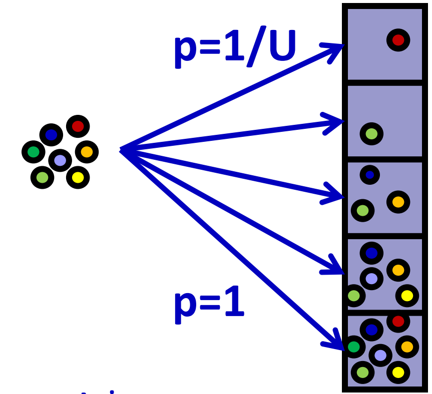
- Upper bound: $|E_{6c}| \leq (22.5c + 6)/3\ M^*$
  - E_α has degree at most $\alpha+1$, and invoke a bound on M^* [Han 08]
- Lower bound: $M^* \leq 3|E_{6c}|$
  - Break nodes into low L and high degree H classes (as before)
  - Relate the size of a maximum matching to number of high degree nodes plus edges with both ends low degree
  - Define HH: the nodes in H that only link to others in H
    - There must still be plenty of these by a counting argument
  - Use bounded arboricity to argue that half the nodes in HH have degree less than 6c (averaging argument)
  - These must all have a 6c-good edge (not too many neighbors)
- Combine these to conclude $M^* \leq 3|E_{6c}| \leq (22.5c + 6)M^*$

Testing edges for α-Goodness



- To estimate matching size, count number of α-good edges
- Follow a sampling strategy similar to L_0 sampling
  - Uniformly sample an edge (u, v) from the stream (easy to do)
  - Count number of subsequent edges incident on u and v
  - Terminate procedure if more than α incident edges
- Need to sample many times in parallel to get result
  - Sample rate too low: no edges found are α-good
  - Sample rate too high: space too high
    - But we can drop the instances that fail
- Goldilocks effect: We can find a sample rate that is just right
  - And bound the space of the over-sampling instances

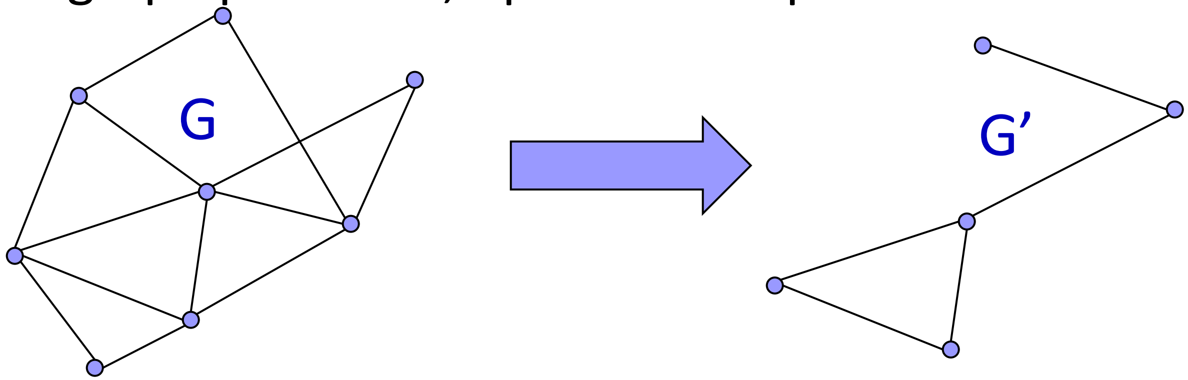
Parallel guessing



- Make parallel guesses of sampling rates p_i
  - Run $1/\varepsilon \log n$ guesses with sampling rates $p_i = (1+\varepsilon)^{-i}$
  - Terminate level i if more than $O(\alpha^2 \log n/\varepsilon^2)$ guesses are active
- Estimate: Use lowest non-terminated level to make estimate
- Correctness: there is a 'good' level that will not be terminated
  - E_α might go up and down as we see more edges
  - But the matching size only increases as the stream goes on
  - Use the previous analysis relating E_α to matching size to bound
  - Also argue that using other levels to estimate is OK
- Result: use $O(c/\varepsilon^2 \log n)$ space to $O(c)$ approximate M^*

Open Problems

- More consideration to the distributed case
  - Many of the pieces can be easily distributed (e.g. sketches)
  - But some pieces (e.g. a-good definition) are inherently centralized
- Other notions of structure/sparsity beyond arboricity?
- Extend to the weighted matching case: some recent results here
- Connections between the streaming and online models?
- Other problems for which kernelization/FPT makes sense?
  - Hypergraph problems, optimization problems…



Concluding Remarks

- Use of l_0 sketches has arisen in several recent graph algorithms
  - Streaming graph connectivity in O(n polylog) space [Ahn, Guha, McGregor 12]
  - Dynamic graph connectivity in polylogarithmic worst-case time [Kapron, King, Mountjoy 13]
- Prompts several natural questions:
  - Can other streaming ideas inspire new (distributed) graph algorithms?
  - Can streaming (bounded space) lead to dynamic (fast updates)?
  - Can the primitives (l_0 sampling) be engineered for practical use?
  - Can assumptions (promises on input) be removed or weakened?

Thank you!