ADGA 2019

D Ellis Hershkowitz

October 15, 2019

These are the notes I took while at the Workshop on Advances in Distributed Graph Algorithms (ADGA) 2019. These notes were written while trying to keep up with the talks and so are not free from errors. Cheers!

Contents

1 Thomas Sauerwald on Random Walks on Dynamic Graphs

Talking about intersection of random + distributed algos

Outline

1. Intro
2. Random Walks on Sequences of Connected Graphs
3. Random walks on Sequences of (Possible) Disconnected Graphs
4. Conclusion

Random Walks

A special class of Markov chains: start at some vertex; jump to randomly chosen neighbor

On static graphs often study hitting time (time to go from u to v) and cover time (expected time to visit **all** vertices); usually work at worst case over all vertices

Classical Results

Hitting time is an $O(\log n)$-approximation of the cover time

Unfortunately not true for dynamic graphs

Worst case: for any graph hitting and covering time at most $O(n^3)$, $O(n^2)$ for regular; lollipop graph shows tightness for $O(n^3)$

Dynamic Graphs

Many networks change dynamically

E.g. wireless mobile networks/social networks/distributed algorithms/particle processes

Lazy Random Walks

Standard assumption that random walk stays with probability 1/2 at current location

Gave example of random walk on dynamic graph

Agenda

Interested in mixing/hitting time on sequence of graphs

For dynamic connected graphs

- If stationary distribution changes over time, we don't have mixing (because no fixed limiting stationary distribution)
- Can we at least say something about hitting times? More or less no; hitting and covering can take exponential time; but if all graphs are connected and regular get same bounds as before up to $O(\log^c n)$ terms

Example of where hitting times can be bad (Sisyphus Graph)

Series of isomorphic graphs; relabel vertices $1, \ldots, n-1$ in a cyclic shift; a fan where center vertex repeatedely pushed out to the outside so to get to n particle has to walk full cycle and so exponential hitting time

Results

Get back $O(n^2)$ mixing and hitting times for dynamic graphs if connected and regular graphs

More generally if a time invariant stationary distribution then $O(n^3)$ mixing and $O(n^3 \log n)$ hitting

Classical Proof Methods

Proof of $O(n^3)$ upper bound on hitting times from FOCS '79

Take an arbitrary spanning tree T in G; bound the hitting times between adjacent vertices; take traversal of spanning tree and add up all the hitting times

Relies on fixed spanning tree

Similar proof shows bound of $O(n^2 \cdot D)$ where do same thing but take shortest paths

Relies on fixed shortest paths

Return Times

Might try and use fact that return time (time for u to u) is $1/\pi(u)$

But not true for dynamic graphs; gave example where swap antipodal vertices in a cycle

Trick is instead of looking at expected time to return, look at return probabilities

Showed example of how probabilities diffuse on the cycle

Can show that even if the graph changes as long as its regular and connected the distance from uniform decays like $1/\sqrt{t}$ where t is the number of time steps

Gave **key lemma** where improvement is larger the further one is from uniform

Proof is quite elementary: if distribution σ is far from uniform then must be that small set of vertices holds much of the probability; there is a short path from large probability to small probability vetex; there has to be some edge then between low and high probability vertices which gives good improvement

Main Result

Result from key lemma: get mixing and hitting of $\tilde{O}(m \log n/\pi_*)$ where π_* is the minimum probability in the uniform distribution

Random Walks on (Possibly) Disconnected Graphs

Mixing time in static graphs always $1/(1-\lambda)$ up to a $\log n$ where λ is the spectral gap

Not true for dynamic graphs; gave example where alternate between 2 cliques and matching between cliques

Get a mixing based on average connectivity theorem

Conclusions

Dichotomy for random walks on dyanmic graphs:

- Stationary distribution does not change makes dynamic graphs look like static graphs
- Otherwise lose many nice properties, (even when changes in the stationary distribution are small)

Bad counter-examples often simulate random walks on directed graphs

In this talk assumed worst-case changes to edge set; could consider:

- Random changes
- Bounded changes

Also could consider vertex set changes

2 Sebastian Brandt on Automatic Round Elimination: A New Approach for Proving Complexity Bounds in the LOCAL Model

New way to prove lower bounds in LOCAL; emerged over last 3 years; seems like proves previously out-of-reach bounds

2016: method used to show no LLL in $o(\log \log n)$ for randomized and no $o(\log n)$ for deterministic

Automatic round elimination makes it easier to apply this technique; used for

- Maximal matching: can't get simultaneously $o(\Delta)$ and $o(\log \log n/ \log \log \log n)$ for determinstic; similar for randomized

Local Model

- Synchronous rounds of communication/computation
- Unlimited message size and computation
- Runtime = number of rounds

_ $O(\log n)$-bit unique IDs (will assume additional symmetry breaking for sake of lower bounds)

WLOG in t-round algo each node collects all information in its t-hop neighborhood

Also will require

- deterministic
- high girth (no node will every see a cycle)

Locally Checkable Problems

Usually what we care about in the local model

Output is defined via local (i.e. $O(1)$-hop constraints)

E.g. vertex coloring just requires checking edges

Round Elimination

Given we can solve a problem in t rounds, what can we do in $t-1$ rounds? Maybe solve a related, easier problem

Sinkless Orientation

Some motivation for round elimination

Orient edges so that no node is a sink

An LLL problem (a lowerbound for SO implies a lower bound for LLL)

Some Intuition

Suppose we have a t-round solution for SO

Question is can we look at a $t-1$ hop radius

If we are trying to orient an edge $e=(u,v)$ then v has stuff v can't see and u has stuff v can't see; but since u and v must agree they can't look at this so really only need $t-1$ hops from v and u

But if we look at all edges in v cannot be that all "extensions" cause all edges into v to be oriented into v; thus if the extensions are independent cannot be one for each edge that causes that edge to be otiented into v

Need high girth and no unique IDs so that extensions are "independent"

Thus, v can determine the orientation of some edge by just looking at its $t-1$-hop neighborhood

Thus a t-round algorithm for sinkless orientation gives a $t-1$-round algorithm

Can iterate this but a 0-round algorithm cannot exist so cannot exist a t-round orientation; So runtime of algorithm cannot be lower than the girth; making firth $\log n$ shows a $\log n$ lowerbound on SO and therefore LLL

Other Problems

Rink coloring and even-degree weak w-coloring

Automatic round elimination theorem states that, given an LCP Π_0 solvable in t rounds can automatically find a LCP Π_1 solvable in $t-1$ rounds

Iterate this, find the first problem in the sequence that can be solved in 0 rounds; if after iterating k times end up with such a problem get a complexity of k

These problems can get very complex descriptions so do a "relaxation step" to make Π_i into Π_i' which is easier to describe

Ideally all Π_i are very similar problems

Apply this to SO

Don't have to do relaxation step for this

Under the Hood

Will use "node/edge" configurations to encode solutions (why need locally checkable)

"Machine" actually produces a $\Pi_{i+.5}$ rounds that's halfway between points

To get halfway problem will produce new node and edge configurations by taking a bunch of combinations consitent in dual ways of previous node and edge configurations

Can relax by adding more constraints

Maximal Matching

An interesting fact about RO: a special vesion for "bipartite" problems; placing a node on each edge gives a bipartite graph for which all this RO machinery still works; also works for hypergraphs

Add an X and Y to white and black configurations; Xs grow linearly and Ys grow quadratically as you iterate this

If number of Xs and Ys doesn't exceed $\Delta/2$ can't solve in 0 rounds which gives a $\sqrt{\Delta}$ lower bound

Problem with this was went to a 0-round solvable problem too quickly; solution is to start with $\sqrt{\Delta}$ Xs already in the configurations (this is maximal $\sqrt{\Delta}$ matching which is just MM where each node can be matched with up to $\sqrt{\Delta}$ other nodes)

For randomized lower bound will argue that error probability grows as you iterate

For deterministic lower bound use the randomized lower bounds; some techniques (e.g. "shattering") to translate from randomized to deterministic

Current Limitations

Ad hoc analysis for determinstic algorithms

Could consider other LOCAL models

Assumed didn't have unique IDs (though other symmetry breaking primitives)

Future

Could try and get upper bounds by making problem harder instead of easier when you relax it

Understand the process / relaxations better

Big goal: complete classiciation of locally checkable problems on high-girth graphs

3 Endre Csóka on Random Local Algorithms from the Graph Limits Perspective

Why Random Graphs?

1. True for random graphs means true for "typical" graphs
2. Random graphs have extremal properties (make good counter e.g.s)
3. Graph limit theory is about separation of structure from randomness

Simplest random graphs: Erdos Renyi; understand them via their structures, e.g. in increasing level of difficulty

- matching ratio
- independence ratio

- chromatic number
- homomorphism numbers

Matching Ratio fr d-Regular Graphs

A "local" algorithm that computes an almost maximum matching on degree $\leq d$ graphs

Cor: random d-regular graphs have an almost perfect matching

Cannot choose 46% fraction of vertices to get an independent set in random 3-regular graphs

Local Algorithms

Constant time distributed algorithm; assign random seed to each vertex independently; each vertex makes a decision based on its local (constant-radius) neighborhood

E.g. each vertex says yes if it has min seed; this gives an $n/4$ size IS for 3-regular graphs

Independence Ratio of 3-Regular Graphs

Gave series of upper and lower bounds

All the lower bounds are based on local algorithms

Graph Limit Theory Motivation

Question: Do d-regular random graphs have no more structure than what can be constructed by local algorithms?

Can rule out independence ratio of $1/2$ by looking at entropy

Why focus on 3-regular graphs? It's the easiest non-trivial case; but actually 3-regular graphs are the most "difficult" because independence ratio is asymptotically twice what can be gound by local algoriths; connections to statistical physics stated re different phases of water

Open Question

Are random graphs among all d-regular large-girth graphs the graphs with the lowest independence ratio?

4 Silvio Lattanzi on Large Scale Algorithms, Clustering, and the MPC Model

More about parallel algorithms than previous talks

Outline

- Models, MapReduce and Simple Examples
- Capacitated Metric Clustering at Scale
- Hierachical Graph Clustering at Scale

Incredible Amount of Online Data

Starting point for talking about parallel algorithms

Lots of tweets, Facebook users etc.

Tough to process this much data on a single machine

Moore's Law

Number of transitors double every two years

But data is growing much faster than hard drive

Another argument for looking at parallel algorithms

Models, MapReduce and Simple Examples

For classic parallel programming simplest model is machines with local memory and interconnect network; but hard to read/understand/write/debug

Instead want an easier abstraction

A simple model for parallel computing around 2003

Main properties assumed

1. Synchronous
2. Complete topology
3. Bounded communication
4. Largish machines
5. Fault-tolerance is transparent to user

This model was **MapReduce**

Important quantities for MR was: # machines, # rounds, amount of communication

Data distributed arbitrarily, each mapper processes its input, sends it to computers by key, each computer processes values collected

MapReduce made theoretical:

- Input Size N
- Num machines $M = O(N^{1-\epsilon})$
- Input size to machines $O(N^{1-\epsilon})$
- Num rounds R

This model has been studied theoretically

Natural to compare this model with other models

PRAM

General idea is you have a shared memory and several processors

- N^c processors
- Input size to machines is $O(1)$

There is a reduction between PRAM and MapReduce algorithms => every exlusive read/exlusive write algorithm can be used in MapReduce with the same number of rounds

But this doesn't give lower bounds for MR; some problems can be solved much faster in MR so seems that MR is more powerful

BSP

- Not assume syhncronization
- Not assume fault-tolerance

Main issue is it is far from practice

Other Distributed Models

LOCAL: restricted topology

CONGEST: limited bandwidth

Is it just PRAM

So may wonder if MR is just PRAM

Prefix sum problem: given a vector compute sum of all the prefixes $\sum_{i \leq j} v[i]$

Can solve in $O(1)$ rounds in MR: suppose vector is sorted and divided across computers; each machine sends its partial sum to all machines corresponding to later prefix sums; this assumed sorting but sorting can be done in $O(1)$ rounds ($\sqrt{N}$ machines each with $\sqrt{N}$ memory; each machine computes $N^{1/2-\epsilon}$-quantiles and sends quantiles to a single machine; this machine then sends back the quantiles which each machine uses as an index to figure out where to send each of its values)

But MapReduce has been deprecated; why is this still interesting?

A higher level system has been build ontop of MapReduce; different systems but same theoretical abstractions work

So now it's called the MPC model

MPC Model

Do actually change a few things

A lot of research focused on more "fine grained complexity" where ϵ matters

Capacitated Metric Clustering

How can we cluster US and world graph?

Want to partition world map so each cluster is smallish (e.g. center distance) and don't want any cluster to have too many nodes

Standard algorithm was LP with n^2 constraints which didn't scale up

Main ingredients to solve:

- Composable core-set: given an input dataset; to reduce its size compute a partial cluster and map points to centers of this partial clustering
- Transform an unbalanced solution to a capacitated one

Composable Core-Set

Let f be a function defined for a subset of Δ. $c(\Delta)$ is an approximate composable core set if $f(c(S_1) \cup \ldots \cup c(S_L))) \approx f(S_1 \cup \ldots \cup S_m))$

E.g. for k-center: solve k-center on each part of a partition; if solve on the resulting centers get a 4 approximation; can see cost of each instance of k center is no more than twice the global opt; thus lose a factor of 2 in mapping to centers; then can see that cost of k-center on the centers is at most 2 the optimal so overall lose a 4

Capcitated k-Clustering

The algorithm has to run using space proportional to the compressed instance

Use sequential algorithm to get an $O(1)$-approximation

If a cluster is too big, add a new cluster with center closest to old center; opens at most k extra centers for a bicriteria of 2

Experiments

Had to subsample by $1/300$ and $1/1000$ for US and world map

If k is very large don't know how to solve this problem

Hierarchical Clustering

Possibility to do a lot of work

No parallel algorithms for Dasgupta result

Density based Clustering

Detecting dense structure in a graph is well-studied with practical applications; e.g.

- Community detection
- Spam detection
- Computational biology

Can think about min vs average degree

Both are in P

K-Core Definition

A K-core is a maximal subgraph of minimum degree K

Gives a hierarchacl structure of 1, 2, 3 core etc.

A simple sequential algorithm to compute the coreness number of every node: remove nodes of degree 1, when no more then remove nodes of degree 2 etc

However, slow for gigantic graphs

Could relax this notion to a $(1-\epsilon)$-approximate K-core

Subsample graph; if a node has high degree you have concentration and you can estimate coreness for high degree nodes well; but what about low degree nodes? Can just delete the high degree nodes and iterate

Can we do better?

Sample vertices instead of edges

Instead of sampling edges just partition vertices randomly; in each machine keep just the induced graph on its vertex set; main advantage is each edge present with probability p^2 if each vertex sampled with probability p

Get an exponential speedup in the number of rounds

Gave experiments where this worked in practice

Conclusion and Future Work

Nice model; captures rea world scenarios

Active area of research

Many open problems

5 Thatchaphol Saranurak on Expander Decomposition: Applications to Dynamic and Distributed Algorithms

Goal of Talk

1. Motivate dynamic algorithms
2. Expander decompositions through dynamic graph applications
3. How it is also used for centralized and distributed algorithms
4. Survey of applications

Dynamic Algorithms

Road networks and social network communities change over time

Common theme: solve same problem repeatedely but input keeps changing slightly

Dynamic algorithms: how not to compute things from scratch

Example

Insert/delete a number in a set S

Want to maintain a minmum number

Recomputing takes $O(|S|)$ time but could do in $O(\log |S|)$ with a BST

But for graph problems a lot of open questions: e.g. compute if G is connected; best known algorithm is poly $\log n$

Dynamic algorithms are used inside static algorithms e.g. in shortest paths or Kruskal's algorithm

Adaptive VS Non-Adaptive

User is non-adaptive if all the updates are fixed from the beginning; adaptive user can update in ways that depend on prevous answers from algorithm

Usually when using dynamic algorithm as a subroutine in a static algorithm you are in adaptive case

Spanning Forest

Defined spanning forests

Goal: minimize update time

Why this problem can be hard? Few edges connecting almost clique; when an update only interesting if you delete a tree-edge across crossing edge need to repair your tree by finding a new crossing edge; it would be bad if you scanned all the clique edges

Turns out you can do $\sqrt{n}$; long gap where not improved but then improved to poly $\log n$ by King et al. SODA 13 but assumes user is not adaptive

Got adaptive case down to $n^{o(1)}$ using expander decompositions

First try to solve problem in expanders

Intuition is expanders are well-connected

A grpah is an expander if number of edges leaving every cut is large wrt to all edges incident to the set (i.e. the volume of the set), namely $1/\text{poly} \log n$

Expander paradigm

1. Solve problem on expanders

2. Combine solutions on expanders

Warm up

Suppose graph is an expander and there is just one update and you want to maintain a spanning tree T; think of cut induced by cut tree edge; then sample an edge with an endpoint in S (can be done in $O(\log n)$ time with e.g. Euler tour or link-cut tree)

By expansion guarantee get an edge crossing S with \$1/ poly$\log$ n\$time

Thus just repeating $\tilde{O}(1)$ times works w.h.p.

What if there are more updates? deleting things might make the graph not an expander

General Tool: Expander Pruning [NSW'17]

Suppose you have you have an expander G_0 and you remove an edge e_i to get graph G_i; you update your set P_i such that the complement remains an expander; the non-expanding part grows slighly; non-trivial to see that this is possible

Guarantees

1. Time to update is $n^{o(1)}$
2. $vol(P_i) = i \cdot n^{o(1)}$ (grows slowly)
3. $G_i[V - P_i]$ is a $\frac{1}{n^{o(1)}}$-expander

Now Try to Solve Problem on Expanders

Will solve relaxed problem of dynamic spanning subgraph (keep any sparse—i.e. $\tilde{O}(n)$ edges—subgraph)

Suppose have expander G but many updates; use expander pruning to maintain set P; maintain spanning tree in $G[V - P]$ union with $E(P, V)$

Update time is $n^{o(1)}$: updating $E(P, V)$ possible by expander pruning; updating T works by random sampling

Now done with solving problem on expanders, but graph might not be an expander; now how to combine solutions; important tool is **expander decomposition**

Expander Decompositions

Given graph $G = (V, E)$

Output a partition such that each part induces an expander and there are at most $m/2$ edges crossing parts

So given any graph can get disjoint expander with just a "few" edges

Computable in near-linear time

Repeated Expander Decomposition

Given graph, compute expander decomposition, get graph by contracting parts, take new expander decomposition etc…

After repeating $\log m$ times get down to constant number of edges crossing parts

Dynamic Spanning Subgraph: General Graphs

1. Do repeated expander decomposition
2. For each expander G' maintain a spanning subgraph H'; to get H' just apply our algorithm for getting a sparse spanning subgraph for an expander
3. Union of all H' is a spanning subgraph of G with $O(n \log n)$ edges

Part 2: Centralized Algorithms

Suppose want to compute a k-spanner

Defined k-spanner

Apply same paradigm

Spanners of Expanders

A shortest path tree in G is a poly $\log n$-spanner of G because it is a subtree for sure and for any edge (u, v) their distance in the tree is at most through r and the diameter of an expander is at most poly $\log n$

Spanners of General Graphs

Do same thing: repeated expander decomposition solve on each expander

Same analysis gives cut and spectral sparsifiers

Part 3: Distributed Algorithms

Consider CONGEST model

Defined CONGEST

Last important tool

Expander Routing

Node u can exchange $\deg_G(u)$ messages with any set of nodes in $n^{o(1)}$ rounds in an expander

So expanders allow global communication with small overhead

This tool lets you import CONGEST CLIQUE algorithms into CONGEST

Expander decomps can be done in poly $\log n$ time

Applications

Expander Paradigm is key in following:

- Laplacian system solvers
- Spectral sparsifiers
- Approx max flow
- Approx vertex max flow
- Bipartite matching, shortest path, max flow

New construction of expander decomposition in $m^{1+o(1)}$ deterministic time which gives better results for all of the above

Frontier

Don't know how to break barriers for adaptive to non-adaptive

Open: improve $n^{o(1)}$ to poly $\log n$ in expander pruning

6 Rotem Oshman on Distributed Property Testing — Progress and Challenges

Missed the first few minutes of the talk...

Distributed Subgraph-Freeness

Given constant size subgraph, want to determine if graph contains it E.g.:

- odd-cycle freeness requires $\tilde{\Omega}(n)$ rounds and cycle requires $\tilde{\Omega}(\sqrt{n})$ rounds; same for even cycles
- some graphs H of size k require $n^{2-o(1/k)}$

Natural then to look for ways to relax problem

Property Testing

General way to relax a decision problem where if trying to decide some property P only have to be correct if object has property or is "far" from having property

Defined for

- Graphs
- Boolean functions
- Distributions: e.g. is this distribution uniform?

Property Testing in Graphs

1. Dense graphs: $\Theta(n^2)$ edges, adjacency matrix
2. Sparse graphs: degree $O(1)$, adjacency list
3. General graphs (hard): lower bound of $\sqrt{n}$ queries to check if graph is triangle free

Distributed Subgraph Freeness

Testing H-freeness

- Yes: G is H-free
- No: at least $\epsilon \cdot |E|$ edges must be removed to eliminate al copies of H

Testers for the dense graph model work as is

In general graph moel can do better than emulating centralized testers

Current State of Affairs

$O(1/\epsilon^2)$ rounds for triangle-freeness

$O(1/\epsilon^2)$ rounds for all 4-vertex subgraphs

$O(1/\epsilon)$ rounds for any H containing an edge $\{u, v\}$ whose removal eliminates all cycles

This is the best we know for testers that don't depend on the size of the graph

Lower bounds: none

Main Ingredient: Disjoint Copies

If graph is ϵ-far from H-free then graph must contain at least $\epsilon m/|E(H)|$ edge-disjoint copies of H

Thus, if pick a random edge, a good chance of hitting a copy of H, namely ϵ

Main Ingredient: Color Coding

To find C_k could just do a BFS but problem is if a non simple cycle then still will hear back yes and would like to prevent that

Solution: every node picks a random element from the cycle (i.e. in $[k]$); now will only allow for cycles where your color is your "role" in the cycle

Still do BFS but respect the colors; only go from color i to $i+1$

Algorithm 1

1. color coding
2. select random edge colored in $(0, 1)$
3. color-coded BFS

If receive back the BFS token then reject

To pick a random edge in CONGEST everybody starts BFS at once each with a priority where higher priority BFSs kill lower priority BFSs

Algorithm 2

T-freeness

1. color coding
2. convergecast; initially
3. state = "closed" if color = leaf of T
4. state = "open" otherwise

in each round: send (state, color); if received "closed" for each child set state to "closed"

Any copy of a tree has constant probability of being detected

Combination

Combine these two algorithms to get above $1/\epsilon$ result; removing $\{u, v\}$ gives a forest then use algorithm 2 for trees and algorithm 1 for cycle connecting trees

Natural Questions

K_5

Lower bounds: don't know anything

Natural suspicion: $\Omega(1/\epsilon)$ for any H (but not always tree, e.g. triangles if $\epsilon \geq c \cdot \min(m^{-1/3}, n/m)$)

Triangle-Freeness

In parallel at each $v \in V$

1. Randomly color neighbors with $O(\deg(v))$ colors
2. In parallel: search for triangles in each color class

Probability of a node in a triangle finding a particular triangle is at least $1/\deg(u) > 1/n$ and a lot of triangles by ϵ assumptions

So Why Don't Lower Bound Techniques Work?

Lower bound for C_5-detection

Reduction from set disjointness

Claim: R-round algorithm for C_5 detection gives a protocol for set disjointness on n^2 bits in $\tilde{O}(Rn)$ so R