Second Workshop on Local Algorithms (WoLA)

D Ellis Hershkowitz

June 18, 2018

I happened to be around MIT for some the talks at WoLA 2018. I mostly attended the talks that dealt with the LOCAL model of computation. These notes were written while trying to keep up with the talks and so are not free from errors and typos. For more and the full schedule see here. Cheers!

Contents

1 June 14, 2018

1.1 Christian Sohler on Property Testing and Random Order Streams

Motivation: understanding large social networks

Graph Stream Models

- Streaming model: n known, stream of edges, $poly \log n$ space (or e.g. sublinear space);

  Problem: almost impossible to follow a path in the graph (e.g. gradually give every other edge, then every every other etc.)

- Semi-streaming

  Criticism of practitioners: most graphs people are interested in are sparse

From theory perspective what can we do

Weaken stream: edges come in random orders

Weaken input: paramaterize the input (e.g. small vertex cover)

Weaken assumption on space: i.e. just give more space like $o(n)$ instead of $poly \log n$

Topic of this talk

Connection between property testing and streaming algorithms (for bounded degree graphs)

Bounded degree graph model in property testing

Graph properties are closed under isomorphism

ϵ-far from having a property definition (have to modify more than ϵDn) edges to get a graph with the property

Property tester can query if an edge exists; must accept with prob $\geq 2/3$ if G has property; reject with prob at least $\geq 2/3$ if graph is ϵ-far from the property

Performance measured by query complexity and running time

Canonical Tester

Sample uniformly q vertices (q is constant)

Do BFS up to q

Simulate property tester T on this sample set

Observation These q disks won't intersect for large enough n

Implies an approximation of the distribution of q-disks suffices to simulate tester

Approximating q-disk distribution

Difficulty: in a stream doing a BFS if only have a subset of edges might not give actual BFS tree; problem is miss edges if consider edges only in the order of the stream

Solution for maximal q-disks uses the fact that we have a maximum degree (maximal q-disk every vertex has degree d); any maximal q-disk must really appear in the graph; from this observation can estimate maximal q-disks in the graph

So a constant probability of observing a maximal q-disk if it is really there (just need the edges to show up in the right order)

Can use this to maximal q-disks missing only 1 edge. Can then generalize this to missing any number of edges (i.e. any q-disk)

Main Theorem

Every constant query testable property of a bounded degree graph is constant space testable in random order streams

If no degree bound

Connectivity testing

MST weight

MIS in planar graphs

Open Question 1

Random order streams

Other problems

Connectivity in $(c/\epsilon)!$ space? if

1. connected graph
2. bound on connected components

Open Question 2

Graph traversal streams: i.e. can we do something better if the edges come from a graph traversal ("I don't know if this helps for connectivity")

Open Question 3

Are random order streams richer than bounded degree graphs?

1.2 Sofya Raskhodnikova on Local Model for Differentially Private Data Analysis

"Different meaning of a LOCAL model"

Differential Privacy

A bunch of individuals with data that data analysts query. An intermediate curator analyzes data and scrubs it to preserve privacy of individuals.

E.g. medical studies is an application

Conflicting goals: (1) protect privacy; (2) accurate answers

Datasets x and x' are **neighbors** if they differ in one persons data.

A (randomized) algorithm is ϵ- **differentially private** if for all pairs of neigboring datasets and for all sets of answers $\Pr(A(x) \in S) \leq e^{\epsilon} \Pr[A(x') \in S]$

Properties

If A_1, A_2 are ϵ private then A_1(x), $A_2(x)$ is $2\,\epsilon$ DP

Local version of DP

Before defined centralized model. But could have a local model where each person holds their own data which they randomize via local algorithms. (noninteractive)

Could also define a local interactive version where user asks questions and gets replies

The noninteractive local model is a special case of interactive local is a special case of centralized algorithm

Advantages of the local model:

Private data never leaves a person's hands; no single point of failure; distributed

Disadvantages: data-thirsty; exponentially more data for some problems (e.g. learning parity)

When companies use DP they do it in the local model

Formal definition in local model: same as last definition but now its over different pairs of values x_i and x_i' that an individual can have

Canonical Simple Algorithm [Warner 65]

Analyst trying to figure out an average

Each person either answers their question or flips the answer at random: i.e. say $+y$ w.p $\frac{e^\epsilon}{e^\epsilon+1}$ and $-y$ otherwise where y is truthful answer

Ratio is exactly e^ϵ so it's DP

Now consider expected value of output: in epxectation for one person it is $y \cdot \frac{e^\epsilon-1}{e^\epsilon+1}$ so if we rescale by $\frac{e^\epsilon-1}{e^\epsilon+1}$ get y in expectation Noisy for one person but averaged over many people has good small expected deviation

Can also generalize to functions with codomain $[0, 1]$

Power of Local Models

Everything solvable with local interactive is SQ and non-interactive is nonadaptive SQ.

Also containment is strict

SQ

A statistical query (SQ) algorithm can do its computation via an SQ oracle; algorithm gives a function from individuals to $[0, 1]$ and accuracy τ and gets back an expected value of this function applied to everybody's data with additive error τ (for distribution P)

A non-adaptive algorithm has to specify all its queries in advance

[Kearns 93] shows many learning/optimization algorithms are SQ

Gave proof that everything solvable with local interactive is SQ and non-interactive is nonadaptive SQ for rest of talk.

1.3 Yufei Zhao on Efficient property testing of induced bipartite patterns in groups

Goal: determine if an n-vertex graph is triangle-free or ϵ-far from triangle free (as above)

Algorithm: sample $O(\epsilon)$ triples at random and just say it's free of triangles if you never see a triangle. This succeeds with probability $> 2/3$.

Proof: by Szemeredi's regularity lemma

Determine if $A \subset G$ (abelian group) determine if no solutions to $x + y = z$ (sum-free).

Algorithm: sample $(x, y, x + y)$ and do same thing.

Proof [Green 2005]: by some form of the regularity lemma

For testing "bipartite patterns" you only need $poly(1/\epsilon)$ samples

Szemeredi's regularity lemma

For every ϵ there exists a partition of graph into $M(\epsilon)$ so that every graph can be partitioned into $\leq M$ parts so that all but $< fraction$ of pairs are ϵ regular

Graph removal lemma

Contrapositive of theorem about graph property testing lemma from before

When $poly(1/\epsilon)$ bounds?

For a graph with bounded VC dimension can always do this

VC Dimension

Gave definition for sets

For graphs its the VC dimension of the collection of vertex neighborhoods

Bounded VC dimension equivalent to forbidding a specific bi-induced subgraph (can partition vertices and then only care about edges between the two parts)

Regularity lemma for graphs of bounded VC dimension

If G is bi-induced-Hfree then can partition G to be regular a la Szemeredi

VC Dimension of Abelian Group

Look at translates of A and look at the translates of the corresponding sets.

Definition of a bi-induced copy of a graph w.r.t to a Cayley graph of a group

Arithmetic regularity results analogous to graph regularity results

Applications to removal lemma and property testing

Open Questions Given

Property testing for bi-induced arithmetic patterns in a general group

Property testing for induced arithmetic patterns

1.4 Elena Grigorescu on Relaxed Locally Correctable Codes in Computationally Bounded Channels

Locally Decodable/Correctable Codes

Encode given a noisy channel

Decode s.t. given that channel hasn't introduced too much error then can decode

LDC: want for every index i can output the bit of the decoded message given only $o(n)$ queries

LCC: same as LDC but output the codeword bit (not the decoded codeword bit)

Relaxed LDCs/lCCs

Decoder can say "I don't know"

Properties to satisfy:

1. If no error then correct
2. Don't say IDK too often
3. Output correct bit with probablity at least 2/3

(1) and (2) imply (3) for constant query codes and constant error rate

Results

Get constant rate for $poly \log n$ query complexity for "crypto" version of definitions

Codes for Computationally Bounded Channels

Hamming channel: corrupts any pattern

Shannon channel: independent errors

Lipton channel: computationally bounded - the channel is a PPT adversary (this talk)

Computational Rekaxed KCCs (CRLCC)

LCC as above but with lipton channel

Their results: weak and strong CRLCC for binary alphabetm constant info assuming collision-resistant hash functions

Key idea is Local Expander Graphs

(A, B) contains a δ-expander if for all subsets of fractional size δ there is an edge between these subsets

δ-local expander is a DAG s.t. for any radius r and vertex v a certain partition (not clear on what partition) contains a δ-expander

Nice property: explicitly constructible (with small degree)

1.5 Guy Even on Local Computation Algorithms (a tale of two computation models)

Labeled Graph

Graph with n nodes where every node has an ID

Edges have port numbers in $[\Delta]$

A graph that you can probe nodes on; get a list of neighbors when node is probed

Focus on maximal independent set problem (MIS)

Local Computation Alg (for MIS); work with a fixed MIS (don't care which)

Local Computation Alg

External world asks if a vertex is in a fixed MIS

LCA generates a seq of probes and always must answer correctly; minimize the number of probes (hopefully sublinear)

LCA is stateless; doesn't remember previous queries, probes, neighbors, answers, MIS etc

One natural approach

Always answer yes; but then if you say yes for v you need to say no for a neighbor of v; but can't store previous answers b/c then not stateless

LCA Application

Multiple servers acceding graph and answering queries but they don't interact with eachother

LCA Variants

1. Randomized (use a random seed)
2. With states (to generate random graph)

LCA vs Distributed LOCAL

Output of v determined by ball of radius r around v

Equivalent r rounds of communication with unbounded message lengths

Running time is unbounded

Minimize the number of rounds

Easy reductions between the two:

If an LCA with queries confined to ball of radius r then can do this in LOCAL with r rounds

If LOCAL algorithm with r rounds then query the at most Δ^r probes around v; ok if $\Delta = 2$

Classic Parallel Challenges

MIS, Vertex cover (2-apx), maximal matching, $\Delta + 1$ coloring

All greedy sequential algorithms: order vertices; add v_i to solution if you can.

Q: simulate by LCA, LOCAL, PRAM

LCA simulation of Seq Greedy

Compute acyclic orientation of G

Simulate greedy by DFS; follow neighbors until you're stuck; when a leaf reached, safe to add it to the MIS

Theorem: DFS version agrees with greedy w.r.t. topological ordering of vertices as dictated by the orientation

Main challenge is to find orientation s.t. the DFS tree is small

Acyclic Orientation

1. Use IDs to orient edges: problem: may induce long paths, so can't use IDs
2. Take a random permutation: every node draws from $U[0, 1]$; then orient edges towards larger draws; in expectation the reachable set from a vertex is $e^{\Delta}[NO - 08]$; also concentration by [RV 16] show that the probability that a vertex max reachability exceeds $2^{\Delta} \cdot \log n$ with probability at most $1/n^2$

Would need state and probes $\geq \Omega(\log n)$ if you did this

1. Do a vertex coloring; orient by colorings; the reachability is bounded by the number of colors, in particular $\delta^{\text{num colors}}$. Lots of distributed vertex coloring algorithms (What they do in this work)

Will compute colors on the fly as they do the DFS. Thus probe radius is bounded by num colors + num rounds to simulate distributed LOCAL coloring. And num probes is $\Delta^{\text{probe radius}}$

LCA Simulation of Sequential Greedy

Theorem upper bounding num probes and prob radius (but $\log^*$ in wrong place (exponent)) here

(Slightly Better LCA Vertex Coloring)

1. Partition edges into Δ^2 disjoin paths/cycles [Kuhn 09]
2. 3-color every $E_{i,j}$, since upper bound on degree you can do this in $\log^* n$ rounds by [PR]
3. Linial gives way of reducing down to Δ^2 total colors in 3 rounds

So $\log^* n$ rounds plus 3

Now simulate this coloring in LCA

$(1 - \epsilon)$-apx max match in distributed LOCAL

Previous algorithms uses a sequence of MIS calls; so get this for free from last result

More LCAs etc

1. Random graph generation
2. $2 + \epsilon$-apx max matching [Fischer 17]
3. SLOCAL: generalize greedy seq to balls of radius r [GKM 16]

1.6 Merav Parter on Local Computation Algorithms for Spanners

Defined a spanner. Gives fact that every graph has a $(2k-1)$ spanner with edges $O(n^{1+1/k})$. Tight by Girth Conjecture.

Originally in distributed setting; since then used in non-distributed setting.

Today study spanners in an LCA setting

LCA Setting

LCA must decide locally (make primitive probes of neighbors) if a given edge e is in the spanner without computing the entire spanner (consistent: illusion of having already computed a spanner)

The model [Alon, Rubinfield, Vardi, Tami '12]: input graph represented by adjacency list. LCA implements oracle access to the output by probing neighborhood of vertices. (Another kind of probe like property testing: are u and v neighbors; if yes learn index of neighbor; usually not considered because usually in LCA assume bounded degree graph)

In this setting get $poly \log n$ bit random string

LCA Motivation

"Swarms of LCAS" can just run in parallel (provided public randomness)

Complexity will be number of probes along with quality of spanner

Previous Work

Provide fast random access to a sparse connected subgraph. If wanted to give a tree in this setting need $\Omega(n)$ probes. So if $(1+\epsilon)n$ edges LB of $\Omega(\sqrt{n})$ for bounded-degree graphs

Their Results

3-spanner with $\tilde{O}(n^{3/2})$ edges and $\tilde{O}(n^{3/4})$ probes

Extend to 5-spanners

Extend to k^2 spanners

First Attempt: LCA through Distributed Algorithm

Fact: Distributed construction of $(2k-1)$ spanners (with optimal number of edges) in k rounds [Baswana-Sen; Elkin-Neiman]

But if apply to this case get superlinear probes so not helpful; goal then is some sublinear probe complexity

Simple 3-Spanner Algorithm

(`http://cse.iitd.ernet.in/~ssen/journals/randstruc.pdf`)

Will tweak it for an LCA algorithm

$\tilde{O}(n^{3/2})$ edges

Vertex has high degree if more than $\sqrt{n}$ incident edges. Add all low degree vertex edges. Sample centers $O(\sqrt{n} \cdot \log n)$ so that every high degree vertex adjacent to a center and connect high degree nodes to adjacent centers.

Adapting to LCA

(Idea: always pick the first selection)

1. Picking the first sampled neighbor as a center: look at neighbor list of v until find a sampled center

2. Add the first edge in your neighbor list

Remains to handle edges between high-degree vertices: Δ^2 queries

First Multiple Center Approach

Just look at first block of $\sqrt{n}$ vertices; Chernoff bound shows this is sufficient; instead of joining one cluster you join $\log n$ clusters

Advantage: no longer need to read the entire neighborhood list: if index j is less than $\sqrt{n}$ you're done.

To get a sublinear complexity need a bit more. Need new approach with vertices with very high degree: break neighborhood lists into bocks of size $n^{3/4}$; now it's sufficient to sample even fewer centers for these vertices since they have such high degree.

Sketch of k^2 stretch

Sample $\tilde{O}(n^{2/3})$ centers.

Divide into **sparse** vertices and **dense** vertices where sparse intersect no centers (and so must be low degree). Easy for sparse vertices. Harder for dense. Divide subgraph into small depth Voronoi cells and connect them somehow.

1.7 Krzysztof Onak on Round Compression for Parallel Matching Algorithms

"Mandatory big data slide"

Model of Computation

MPC model: M machines, S space per machine, N items; initially each machine receives N/M items; in each round machines do local computation and send messages to eachother (receiving at most $O(S)$ data)

Inspired by MapReduce; Sleak abstraction that hides details of MapReduce

Required near quadratic total space orginally; then relaxed to linear total space

Goals: small number of rounds; small space per machine; fast local computation

This talk: linear total space

Matching Algorithms

Study matchings because

1. Non-trivial packing problem
2. Testbed for new algorithmic ideas
3. Helpful to understand the power of the model
4. Practical applications

Maximum matching: find maximum disjoint set of edges

If space is $n^{1+\Omega(1)}$ can get a $1+\epsilon$ approximation in $O(1)$ rounds

If space $n^{\Omega(1)}$ 2-approximation in $O(\log n)$ rounds with classical algorithms from the '80s

Their result

For $O(n)$ space, improve $O(1)$-approximation to $poly(\log\log n)$ rounds

Highlights of Approach

Start from $O(1)$-approximation distributed algorithm; simulating it takes $\Theta(\log n)$ rounds; but use round compression to compress a superconstant number of rounds to a constant number (also use vertex sampling; previous algorithms use edge sampling)

Review of Distributed Algorithms

Distributed $O(\log n)$-approximation for vertex cover

Pick all vertices of degree at least $n/2$, $n/4$, $n/8$ etc. The union is a vertex cover of size $O(\log n) \cdot OPT$

$O(1)$-approximation for verte cover from 2010: in each round not only high degree vertices that are removed but also those in a matching with high degree vertices

Goal: efficiently emulate this algorithm (the peeling algorithm) in MPC

1. Apprixmate vertex degrees
2. Random neighbor for each high degree vertex

Random Vertex Partitioning

Partition vertices into random $\sqrt{n}$ size groups. Do same thing as above until max degree becomes about $\sqrt{n}$.

Maybe do not repartition each time? No clear reason why this would work.

Actual solution: required making the algorithm more complicated to get this to work.

This is a fast moving line of work: gave many other followups

1.8 Grigory Yaroslavtsev on Badger Rampage: Multi-dimensional Balanced Partitioning of Facebook-scale Graphs

Three schools of thought in algos and complexity

1. Boston (MIT + Harvard): youthful and innovative attacks on probabls; relentless optimism
2. NY and Chicago: abstract and skeptical theory building; care about poly time, worst case etc
3. Bay Area: no time for philosophy; ML/ AI/fairness

This talk combines Boston and Bay Area schools of thought

Balanced Graph Partitioning

Divide graph into k parts of roughly the same size with maximum number of edges inside parts

Goal: make it work for real FB graph

Hard in Theory

Minimizing the cut : no constant-factor approximation for $\epsilon = 0$ unless P = NP. Best approximation: poly log

Maxing the number of edges: .64 apx via SDP

Important in Practice

Companies actually do this in practice

FB doesn't really want balanced graph partitioning; want "multidimensional" balanced graph partitioning (balance according to multiple weight functions); generalizes balanced graph partitioning

Note: can be impossible if weights are unrelated

Existing approaches are combinatorial; break for multidimensional; their approach is gradient descent based

QIP for problem

x_i for each vertex that is 1 if x is in V_i and -1 otherwise.

Then have term in objective for each edge.

Easy to impose necessary constraints.

Relax to non-convex relaxation and apply randomized projected gradient descent; use first order method (plain vanilla projected gradient descent). Add randomization to escape from saddle points.

Badger Rampage

Do gradient descent, add noise to escape saddle before projecting.

Formally, what can we say about converges: if all constraints are equalities then converges to a local minimum

2 June 15, 2018

2.1 Fabian Kuhn on the Role of Randomization in Local Distributed Graph Algorithms

Distributed Graph Algorithms

Network is modeled as a graph with n nodes with unique IDs

Synchronous rounds: each node does computation; send a message to neighbors; receive a message from neighbors

This is the LOCAL model: unbounded internal computation and message sizes (introduced by Linial; FOCS '87)

Interested in the round/time complexity of an algorithm

Objective: solve graph problem on the actual network graph

At start: each node knows its ID and nothing else

At the end: each node knows its part of the output (e.g. its color)

Local checkability: they consider graph problems where solution is locally checkable (e.g. coloring); sort of like NP in the distributed setting

Four Classic (Locally Checkable) Problems

1. $\Delta + 1$-vertex coloring; $\Delta + 1$ colors is what a simple sequential greedy algorithm achieves
2. Maximal Independent Set (MIS): non-extendible disjoint nodes
3. $2\Delta - 1$-edge coloring: what you get with the greedy algorithm also
4. Maximal Matching

Some relations between them: MM is MIS in the line graph; $2\delta - 1$ is just vertex coloring in the line graph; so simple reductions; also a reduction from vertex color to MIS; thus all reduces to MIS

Early algorithms for the Four Problems

Randomized algorithm $O(\log n)$-time MIS: [Luby 1985], [Alon, Babai, Itai 1986], implies $O(\log n)$-time randomized for all foour problems by above reductions

Deterministic $2^{O(\sqrt{\log n})}$ based on network decomposition: [Awerbuch, Goldberg, Luby, Plotckin; Focs '89]; best known det. alg for many problems

General picture: significant gap between randomized and deterministic algorithms: we wnat $poly \log n$-time deterministic algorithms

One exception: maximal matching can be solved in $O(\log^4 n)$ deterministic time by [HKP SODA '98, PODC '99]

State of the Art

MIS: rand $O(\log \Delta) + 2^{O(\sqrt{\log \log n})}$; Det $2^{\sqrt{\log n}}$

Maximal Matching: rand: $O(\log \Delta) + O(\log^3 \log n)$; det $O(\log^3 n)$

Vertex coloring: rand $2^{O(\sqrt{\log \log n})}$; det $2^{O(\sqrt{\log n})}$

Edge coloring: rand $O(\log^6 \log n)$; det $O(\log^6 n)$

Lower bounds: MIS, MM: $\sqrt{\log n / \log \log n}$ish; coloring $\Omega(\log^* n)$

If you compare the deterministic and randomized complexities they look kind of similar: this is not by coincidence: what these algorithms do is use randomized algorithm to solve the problem on most of the graph and then you only have small components which can then be solved by the best determinstic algorihtm

Interestingly Pettie et al '16 shoes Randomized $(n) \geq$ Determinstic Time $(\sqrt{\log n})$

High Level Summary: randomized $O(\log n)$; deterministic $2^{O(\sqrt{\log n})}$. Question is is the gap inherent?

Exponential Separation Below $\log n$

Exponential separation for bounded degree graphs between the two (if we do not ignore log facors) for somputing a sinkless orientation and Δ-coloring trees: randomized is $\Theta(\log \log n)$ and deterministic is $\Theta(\log n)$. Basically same thing for Δ coloring

In this talk ignore logarithmic factors

Challenges in the LOCAL Model Locality: at the beginning of aglorithm node doesn't know anything about the network; after r rounds v only knows about its r-hop neighborhood. In an r-round algorithm each node computes its output as a function of the initial state of its r-neighborhood

Symmetry Breaking / Local Coordination: neighbors u and v; they run in parallel but if the topologies they see are identical but they need to come up with different outputs need to somehow break symmetry between the nodes; here it's clear that randomization should help

So came up with a model where locality is still a challenge but gets rid of the symetry breaking challenge

Sequential LOCAL Model [Ghaffari et al.; STOC '17]

Get rid of parallelism and just go through nodes sequentially in an arbitrary order

The SLOCAL Model (a sequential model): has a locality parameter $r(n)$

Seems that this model is much more powerful than distributed local: e.g. simple greedy algorithm works for $\Delta + 1$-coloring and MIS and locality parameter can just be 1; model can be seen as a generalization of sequential greedy algorithm

Complexity Classes

LOCAL($t(n)$): all graph problems that can be solved determinstically by a $t(n)$ round algorithm in LOCAL

SLOCAL($t(n)$): all graph problems solvable determinstically by an algorithm in SLOCAL with locality $t(n)$

P-LOCAL = LOCAL($poly \log n$) P-SLOCAL = SLOCAL$poly \log n$ Analagous classes for randomized (MC) algorithms

Relations Between Complexity Classes

LOCAL(t) $\subseteq$ SLOCAL(t), P-LOCAL $\subseteq$ P-SLOCAL

Also randomization helps

Also a way to go from sequential to distributed: proofs based on *network decompositions*

1. RSLOCAL(t) $\subseteq$ RLOCAL($O(t \cdot \log^2 n)$)
2. SLOCAL(t) $\subseteq$ LOCAL($t \cdot 2^{O(\sqrt{\log n})}$)

Open Problem: P-LOCAL = P-SLOCAL

Network Decomposition [Awerbuck, Goldberg, Luby, Plotkin; FOCS '89]

Definition: (d, c)-decomposition of G partition of V into clusters of diameter $\leq d$ with a coloring of the cluster graphs with c colors

Claim: given decomp of G can deterministically compute MIS or $\Delta + 1$ coloring in $d \cdot c$ rounds

Idea: iterate through clusters of same cluster; process clusters of same color in parallel; compute partion solution for each cluster in d rounds

Same reduction works for SLOCAL: given decomposition of G^{2t} can run any det SLOCAL algorithm in time $O(t \cdot d \cdot c)$ in the deterministic LOCAL model

Network Decomposition Algorithms

Every graph has a $(O(\log n), O(\log n))$ decomposition

Complexity of computing: det SLOCAL with locality $O(\log^2 n)$ time; Rand LOCAL in $O($ $log^2 n)$ rounds. Implies (1) that P-SLOCAL/P-RSLOCAL $\subseteq P - RLOCAL$; det local is $2^{O(\sqrt{\log n})}$ rounds implies (2) as above

Gave picture of containment of these complexity classes and problems in each one

So again open problem is if P-LOCAL = P-SLOCAL. Unknown.

Are there any complete problems in P-SLOCAL?

P-SLOCAL Completeness

Give a local reduction: P_1 polylog-reducible to P_2 defined as expected.

P-SLOCAL Completeness is defined as would expect

Actually there is an obvious complete problem: network decomposition

Not so surprising since it was designed as a generic tool for designing distributed algorithms

So the question is are there other complete problems

The Local Degree Splitting Problem

Have a bipartite graph (L, R): color R with red and blue so each node $v \in L$ has roughly the same number of neighbors that are red and blue

Trivial randomized algorithm: just randomly color the right side

Theorem: if all nodes in L have degree $\Omega(\log n)$ then this problem is complete [Ghaffari et al.; Stoc '17]

Note: a trivial 0-round algorithm for both problems

Remark: can be seen as rounding fractional values to integer ones; at the beginning each node is half red and blue then you want to round them to one color; so coursely rounding fractional values to integer values is the only obstacle to efficient deterministic algorithms in LOCAL!

Las Vegas Algorithms

ZLocal(t): all graph problems with LV algorithms in $\leq t$ rounds

Reduction from sequential to sequential works to give LV algorithms

Theorem: ZLocal / RLOCAL $\subseteq$ SLOCAL(O(t))

Theorem: P-RLOCAL = P-ZLOCAL = P-SLOCAL

Also gives a way to turn distributed MC algorithm into a distributed LV algorithm (analogue of what we know for sequential algorithms; here we only have local checkability so you can't just repeat like in sequential setting but use detour through network decomposition)

Open Problems

Main: P-LOCAL = P-SLOCAL? Mohsen thinks yes; Fabian thinks no.

Others given

Answers to Questions

Congested clique kind of converse of SLOCAL

Some improved algorithms for special families of graphs for network decomposition but not many—e.g. unit disc graphs

2.2 Seth Pettie on A Survey of the Distributed Lovasz Local Lemma Problem

Sort of a survey talk

Why do we need LLL?

$O(1)$-time Randomized Experiments

Max-degree Δ and palette size is $(1+\epsilon)\Delta$

Each edge picks a color u.a.r, if no conflict then commit; every conflicting edge tries again in the next round

Want to bound the probability that an edge is colored in a round: about e^{-2}

By tail bounds can show these RVs are relatively close to their expectations

If error $\delta^2\Delta >> \log n$ we're done by a union bound. What if $\delta^2\Delta >> \log\Delta$: only have that union bound holds locally. This is what LLL describes

The LLL

Random variables X

"Bad events": RV deviates too far from its expectation

A dependency graph: events dependent if share a RV

p is max Pr a bad event happens and d is the max degree in dependency graph

Theorem: if $ep(d+1) < 1$ there is some assignment to X that avoids all bad events

The LOCAL Model

Just as in Fabian's talk

$N :=$ num of vertices

$\Delta :=$ max degree

Distributed LLL Problem

G is the dependency graph of the LL instance and the communication network of the LOCAL model

Problem: collectively compute an assignment to X that avoids all bad events

In reality: dependency graph and communication network are not exactly the same; but assuming they're the same is WLOG

Moser-Tardos Resampling

Sequential algorithm is well-known but also a parallel one you can run in LOCAL

Algorithm is as follows

1. Sample a random assignment to the variables according to the distribution
2. Let V' be bad events that happen; if empty you're done
3. If V' is not empty pick a MIS in subgraph induced by V'; resample these variables fresh

Theorem: If $ep(d+1)(1+\epsilon) < 1$ M-T ends after $O(\log_{1+\epsilon} n)$ steps. Time: $O(MIS \cdot \log_{1+\epsilon} n)$ where MIS is the time to compute an MIS

Gives a demo where V' gradually shrinks

Analysis is completely not obvious: you analyze things reverse chronologically: here is event A that I resampled its variables (this could only happen if in previous round somebody screwed up its RVs; so who screwed up these guys RVs). Build a "Witness tree" that shows who screwed up whose random variables. The height of this tree is linear in the number of rounds of computation. The analysis shows that the probability of seeing this tree is $(1+\epsilon)^{-\text{size/height}}$. So if tree is tall enough algorithm must terminate before time h.

Chung-Pettie-SU

Didn't like that this algo uses MIS because can never get constant time.

Uses same resampling framework. Algorithms as before but if you want to choose an IS just choose the IDs whose values are local minima. An independent set but not necessarily an MIS.

This works beause you can do the same analysis as before with a few tweaks. Either a radius 2 or radius 1 explanation of why a bad event happens at a particular time.

Same as before but time: $O(\log_{1+\epsilon} n)$.

Lower Bounds

[Brandt et al; STOC 2016] simplified proof from SODA this year

Randomized takes $\Omega(\log \log n)$

Deterministic takes $\Omega(\log n)$

Holds for very constrained versions of the problem where the dependency graph is a tree

New problem: sinkless orientation. Orient edges so no vertex is a sink.

Could orient edges randomly; gives an instance of LLL

Simplifying assumptions: put processors on the edges rather than the vetices; graph is bipartite and 2-vertex colored; graph is 2Δ-edge colored; graph is infinite Δ-regular tree; "running time" is a vector $(t_1, t_2, \ldots, t_{2\Delta})$. Edges colored j terminate in t_j rounds

Proof idea: take a randomized algorithm where first i colors terminate in t rounds and remaining terminate in $t-1$ rounds; will change it so color class i terminates in $t-1$ rounds—this is impossible b/c can't unilaterally reduce rounds; show error probability will go up polynomially.

Completeness

LLL is complete in two ways

Theorem: weak splitting is PSLOCAL-complete as in Fabian's talk. Can solve this with a deterministic-LLL algorithm so it's PSLOCAL-hard (and complete according to Fabian).

The LLL is complete for sublogarithmic time: an LLL algorithm that works for some "fairly flexible" criteria; an algorithm A solves some LCL problem; can improve A to match the running time of the LLL algorithm.

Proof idea: imagine running the LCL algorithm but we tell it that n is $n^* < n$ so the running time is smaller. Alg will look at radius t^* ball. If these balls intersect for some vertices then their outputs could be dependent. Set up the dependency graph where variables are random X_v: random bits generated by v. Bad event is local neighborhood being incorrectly labeled. Chose t^* to work for LLL. Run a distributed LLL on this dependency graph so algorithm never fails for any vertex. Gives a constant factor slow down; eventaully LCL algorithm will match the runtime of the LLL.

Modern Distributed LLL Algorithms

All use the **graph shattering method**

1. (randomized) take big dependency graph and want to fix some variables so any even with unset variables breaks into $poly \log n$ components.
2. (deterministic) Then solve the problem on each component.

A new twist uses a 3 [Ghaffari et al. '17]: Derandomize the algorithm from (1) and (2) and plug it back in as the deterministic algorithm for step (2)

To some course approximation this is house the LLL must be solved: to improve the complexity of this problem work on (1) or (2). These are two completely separate tasks.

An idea for improving step 1:

Graph Shattering via Complex Contagions

Consider a **partial** assigment ϕ to the variables X. An event is **dangerous** if $\Pr(v \mid v\phi) \geq \sqrt{p}$.

A natural way to settle almost all the events in the graph and still end up with an LLL instance with small p. Pick random total assigment ϕ. For each dangerous event v, unset all of its variables. Unsetting an event could create a neighboring dangerous event. This is like an infectious process

This process terminates in $O(\log n)$ time but this $\log n$ is too big.

Complex Contagions:

- At time 0 every node ingected with $p_o = d^{-O(1)}$
- Node with $<$ infected neighbors becomes infected
- State is **stable** if it induces no more contagion

Stable State Problem: compute a non-trivial stable state

The Hypochondriac Algorithm: if only a few infected neighbors you believe that you become infected ($u/2$ instead of u). Run this for $O(\log \log n)$ times instead of $O(\log n)$ times and use smaller probability of infection. Then you "shrink": everybody with fewer than 5 actually infected neighbors becomes better then the betterness spreads; this will terminate with a stable state if the graph has good expansion. Open if it works for general graphs

Conjecture: $O(\log \log n)$ is the right answer for randomized LOCAL