DISC 2019

D Ellis Hershkowitz

October 21, 2019

These are the notes I took while at the International Symosium on Distributed Computing (DISC) 2019. These notes were written while trying to keep up with the talks and so are not free from errors. I mostly attended the message-passing type talks and did not take notes on most of the talks on the third day. Cheers!

Contents

1 October 15, 2019

1.1 Dan Alistarh on Distributed and Concurrent Optimization for Machine Learning (Keynote Talk)

Backdrop: explosive recent progress in ML; ML surpasses humans in

- image classification and segmentation
- speech recognition / translation
- strategic games

Even if progress stalls, ML is here to stay

Three Factors

1. Great ideas
2. High quality data
3. Efficient computation

Distributed/parallel computing is key enabler of computational speedups

Distribution is Key

For e.g. training a deep net, distributed training is necessary

- Large datasets
- Large models

Conflict between distributing as much as possible and keeping many parameters synchronized

Can ask if efficient distributed ML is a solved problem

The Scalability Problem

Gave training task

- Single node: 19 days
- 1024 nodes: 24 minutes (in theory)

In reality don't really get linear scalability with number of GPU nodes; time goes up instead of down after 32 nodes

If you split time into communication and computation the communication goes up and computation goes down as number of GPU nodes increases

Thus, efficient distribution is a non-trivial challenge

The Algorithm: Parallel Stochastic Gradient Descent

Synchronous message-passing system with n fully connected nodes; each node has a fraction of the data and in each round they

1. Do some computation; do model update
2. Average their updates
3. Update the model

When update model the compute time increases but so does the averaging time; as you scale the cost of the averaging starts to pass the compute time

Today's Talk

About overcoming this phenomena

Communication-efficient algorithms for scalable machine learning based on

1. quantization

2. sparsification
3. efficient aggregation

Trade-offs: compression vs convergence vs parametrization

ScaleML: open-source framework implementing these techniques

Overview and open problems

General Setting

n nodes doing synchronous message-passing on a fully-connected topology

Dataset D: node p_i assigned dataset partition D_i

Loss function $Loss(x,e)$= how "good" prediction of model x is on example e

Trying to solve optimization problem which is minimize expected loss of example e over dataset D_i

Want model x minimizing the sum of their losses; so nodes have to communicate; abstract question is what is the **communication complexity** of this task

The Algorithm: Data-Parallel SGD

Each node maintains an (identical) copy of x

Each node selects e_i uniformly at random from D_i and computes gradient update; nodes then average their gradient updates and update their respective models using this average gradient

Because each node has to send its gradient it has to send $32d$ bits each round

Distributed mean estimation: Imagine loss function is just average l_2 distance from examples

A Bit of Theory

Suppose want to minimize differentiable $f : \mathbb{R}^D \to \mathbb{R}$

Do SGD

Have variance bound on the gradient

Given a convex and smooth f need $O(R^2 \frac{\sigma^2}{\epsilon^2})$ to get ϵ-optimal where R is initial distance from optimal

If averaging over P gradient estimators then gain a $1/P$ term which is the advantage of averaging

Communication-Efficient Algorithms

Method 1: gradient quantization; reduce amount of information in vector in a structured way which still allows one to converge

E.g. suppose gradient has 5 components; could try to reduce each component 1 to 1 bit by taking the sign bits and sending the average over positive and negative values

Compression rate is ~32; but does not always converge

Stochastic Quantization

A different version which always converges

First extract a scaling factor, say 2 norm; also take all the signs as before; split interval into 4 evenly spaced quantiles (encoded by 0,1,2,3); will randomly quantize each component in a way that preserves its expectation

Compression factor loses number of buckets but provable converges

Informal claim: QSGD converges similarly to the original but sends 10x fewer bits; this compression-variance trade-off is tight

Does It Actually Work

Showed graph where training time goes down by pushing down communication time

Method 2: Structured Sparsification

Basic idea: go over gradient vectors and isntead of reducing the regularity, start throwing away stuff

Find an integer parameter k, only send top $k$from each gradient vector, in order of absolute values

1. How much compression? Potentially huge
2. Still guarantees convergence
3. Doesn't need extra hyperparameter tuning

Summary so far

Can reduce communication

But no software support

So implemented a communication framework that supports this sort of quantization and sparsity

Also gave example where worked at scale

Open Questions

Interaction between convergenec and communication topology

General set of consistency conditions for distributed machine learning

Can we model distributed learning in social groups

1.2 Ruben Becker on Distributed Algorithms for Low Stretch Spanning Trees

Motivation

Recurring theme in graph algorithms: trees are easy, general graphs are hard

Common approach

1. transform G into tree T
2. solve problem on T
3. project T back to G

Here

- distance-based problems
- trees are spanning trees
- measure quality of solution by stretch

Low-Stretch Spanning Trees

Given graph G with edge length function l

Defined distances induced by spanning tree T and stretch w.r.t. tree

People interested in trees with good average stretch which is equivalent to finding a randomized algorithm with good expected stretch

Many applications

State of the Art

Centralized setting

- Alon $2^{O(\sqrt{\log n \log\log n})}$ stretch
- Best: $O(\log n \log\log n)$ stretch by Abraham et al.

In CONGEST previously Alon-type result

Contribution: reduce spanning tree of expected stretch $O(\log^3 n)$ to a single SSSP computation in CONGEST

Main Ingredient: Star Decomposition

(δ, ϵ)-star decomposition: partition node set s.t. graph induced by parts are connected and bridge edges connecting V_0 to V_i s.t.

- Radius of parts is a constant smaller than radius of whole graph
- Going from center of decomposition to another part over a bridge edge is bounded by constant times shortest path length

Tree construction in previous result just recursively computed these star decompositions

How to construct: V_0 is just standard ball growing; V_i is "cone growing" which seems sequential

Their contribution: a distributed version

Distributed Star Decomposition

For V_0 grow ball

For rest of graph compute an LDD (partition s.t. diameter bounded by $O(\log n/\beta)$ and probability an edge e is cut is $\beta \cdot l_e$)

Every node on the ball shell of V_0 takes an exponentially distributed value and grows balls

Algorithmically you can do this by one SSSP computation by introducing a super source node for V_0

Easy to show get a star decomposition; but must show probability an edge is cut is small

- If cut by V_0: handled by random choice of V$_0$\$ radius
- If cut by LDD: use fact that probability of smallest and second smallest values of a bunhc of β-exponential RVs within c of eachother is at most $O(\beta c)$

Tree Stretch Analysis

Probability of getting cut on one level of recursion is $\beta \cdot l_e$; an edge getting in the tree preserves distance but if cut on level k then distance is bounded by twice the radius at this level of the recursion

Conclusion

Contributions

- maybe (?) simplest argument to compute low-stretch spanning tree
- First algorithm in distributed

To remember: efficient reduction of poly-og stretch spanning tree to a SSSP

1.3 Naoki Kitamura on Low-Congestion Shortcut and Graph Parameters

Motivation: low-congestion shortcuts are way of getting efficient algorithms in CONGEST for special graph classes

They investigate how: clique width, chordality, diameter affect quality of low congestion shortcuts

Defined CONGEST model

Defined MST

Partwise Aggregation (PA)

Each node has a value and a partition of graph into connected "parts"; goal is to simultaneously compute function on all parts

Was shown that solving PA implies solving MST up to logs

Naive solution for PA is to only compute within parts; but might take $\Omega(n)$ time if part diameter is very large

To speed up PA can use edges outside of parts

But using many might incur a lot of new congestion

Low congestion shortcuts formalize how to use edges outside of parts efficiently

(d, c)-Shortcut

Induced dilation of each part with its shortcut edges of is at most d

Induced congestion is at most c

$d + c$ is "quality" of shortcut and implies solution for PA of same quality

Main question is what graphs induce good shortcuts

Results

Give quality of

Clique-width-6: $\tilde{\Omega}(\sqrt{n + D})$

k-Chordal: $O(kD)$

$D = 3$: $\tilde{O}(n^{1/4})$

$D = 4$: $\tilde{O}(n^{1/3})$

High Level Idea

Parts of small diameter ($\leq n^{1/3}$) take no shortcut edges

Parts of large diameter are split into connected subgraphs (called "groups"); connect groups with length 4 paths between groups in the same part

Outline of Algorithm

- 1-hop extension
- Finding length 2 low-congestion paths connecting two extended groups

Open Problems

Good shortcuts for $D \geq 5$

Find a problem that can uses benefit of bounded clique-width

1.4 Shiri Chechik on Reachability and Shortest Paths in the Broadcast CONGEST Model

Defined LOCAL and CONGEST

Reachability and Shortest Paths

Goal: compute short paths/reachability from s in braodcast CONGEST

This problem not fully understood in PRAM and CONGEST

Always assume communication network is bi-directional (regardless of direction of underlying directed graph)

Must pay D

In undirected graphs BFS can be computed in $O(D)$

In directed graphs, problem is more complicated; path making a node reachable might be much larger than diameter of undirected version of graph

Related Work

Lower bound of $\Omega(\sqrt{n} + D)$ rounds if diameter is $\Omega(\log n)$; better for $D = 3, 4$

Upper bounds getting close; bounds match if D is poly $\log$

So consider small diameter graphs in broadcast CONGEST in this paper

Results

$D \geq 2$ show require roughly $\Omega(\sqrt{n})$ single source eachability

$D = 1$ all pairs reachability in one round; almost 3-APSP in two rounds

Will focus on $D = 1$ result; if a node learns in and out degree of all nodes it suffices to learn reachability relationships

Key lemma 1: given graph with $D = 1$ for any subset of vertices A the sum of the difference of in and out degree is $|A \times \bar{A}|$ only if there are no edges from A to $\bar{A}$ Did out the proof; key idea is to count how each edge affects the above sum

Key lemma 2: consider vertices in non-decreasing order of out degree; for index i there must be some index k after i such that the reachable vertices from v_i is $v_1 \ldots, v_k$. Also this k is minimal index with a particular counting property

Did out proof; key idea is to sort into set A reachable from v_i and $\bar{A}$ and then do similar counting argument; then apply lemma 1 since no edges from A to $\bar{A}$

Now local algorithm that all nodes do: each v_i computes the smallest k which satisfies the stated condition which tells vertices what their reachability set is $v_1, \ldots, v_k$

Conclusions and Open Problems

Stronger lower bounds in CONGEST model for small diameter

Optimal algorithms for large diameter

1.5 Kristian Hinnenthal on Fast Distributed Algorithms for LP-Type Problems of Low Dimension

Smallest Enclosing Circle Problem

Points in the plane; want to find smallest radius enclosing circle of the points

Let set of points be H

Look at function $f(G)$ which gives min radius of circle containing all points G

Monotonicity: f is **monotone**

Locality: if a set is violated by a point then so too is a set which contains it and is of equal f value

LP-Type Problems

H is a set of constraints

f maps subsets to T for which monotonicity and locality hold

LPs can be stated as these sorts of problems

Basis B is a subset of constraints such that $f(B') < f(B)$ for $B' \subset B$

Basis B is **optimal** if $f(B) = f(H)$

Combinatorial dimension is the maximum cardinality of any basis (e.g. = 3 for above smallest enclosing circle problem; for LPs it's the number of variables)

Clarkson's Algorithm

Clarkson's algorithm solves LP-type problems

Chooses a subset of constraints $R \subset H$ of size $6d^2$ where d is the combinatorial dimension

Compute $f(R)$ and set of violators V which are set of violators (i.e. points that adding to

$$Rincreasesthevalueof$$

f)

If V is small, double multiplicity of violators; makes them more likely to be samples so next time chances are these won't be violators

Only need $O(d \log |H|)$ until no violators anymore

This work: is there a distributed algorithm with $O(d \log |H|)$ rounds?

Related

Simulation of Clarkson in a hypercube: $O(d \log^2 n)$

Some PRAM solvers for linear programs

Contribution

$O(d \log n)$ algorithms in the **gossip model**

- H is randomly distributed among n nodes
- Random **push** and **pull** operations in synchronous rounds
- Message size is $O(\log n)$
- $d = O(\log(|H|))$

Low load case: $H = n \log n$ and work $O(d^2 + \log n)$

High load case: $|H| = \text{poly}n$ and work is $O(d \log n)$

Gave pseudocode for low-load Clarkson's Algorithm: all nodes in parallel choose a random subset of constraints by pulling from random subset of nodes; if the sampling succeeds then nodes locally looks at the constraints which violate the solution and then nodes increase the multiplicity of these constraints; also do some filtering where only keep constraints with some probability; push and validate potential bases

Pseudocode for high-load Clarkson: no compute basis of all gathered constraints so far

1.6 Hsin-Hao Su on Distributed Data Summarization in Well-Connected Networks

Data Summarization Problem

Node has input x_i

f_x is number of occurrences of x

Goal is to compute $F_p = \sum_x f_x^p$

Examples

1. F_0: number of different inputs
2. F_2: skewness of data

Consider problem in CONGEST; defined CONGEST

Need $\Omega(D)$ time

What about when D is small?

Can simulate streaming algorithms for a $(1+\epsilon)$ approximation via BFS tree

Relateded work studies computing the mode

A lower bound for F_0 from Gap-Hamming-Distance and F_P from multiparty disjoint set and $\tilde{\Omega}(D+n)$ via ?

All these reductions always have a thin cut; question then is can we do better if the graph is well-connected

Results

Let $\tau(G)$ be the mixing time of G

1. exact algorithm in time $O(\tau(G)) \cdot 2^{O(\sqrt{\log n})}$
2. GOSSIP algorithm in time $\tilde{O}(n^{1-k/p}/\epsilon^2)$ where k is a parameter that controls thet number of different values and non-empty nodes in the input
3. General simulation for gossip in CONGEST: one round of gossiip in $\tilde{O}(\tau(G))$ rounds of CONGEST

First Result

Main tool is a permutation routing algorithm; in **permutation routing** a bunch of source, dest pairs where each node is the source/dest in at most one pair

Step 1: sort the values; impose a sorting network where each level corresponds to a permutation routing; need $\log n$ levels

Step 2: count the f_x; each node can figure out if its a head or tail; then remove all values that are not heads or tails and then collapse down

Question then is can we do better.

Could try and improve the permutation routing running time but that seems hard

GOSSIP

Another approach: simulate GOSSIP algorithms; in GOSSIP model each node can pull or push to uniformly chosen neighbor

A simple algorithm for computing F_2 in GOSSIP: each node samples a target node and tests if their values are equal; output n times the sum of indicators are equal

Easy to see that this is an unbiased estimator

Can then generalize this to higher p

But then question is how to simulate GOSSIP in CONGEST

Simulating GOSSIP in CONGEST

Each node needs to sample a node $t(v)$ uniformly at random

A natural approach is to do this via a random walk

Previously showed T steps of independent random walk can be done in $\tilde{O}(T)$ time; would work for regular graphs but not non-regular graphs

A natural approach would be to add self-loops to make the graph regular (but then running time could increase)

So divide each node u into $\deg(u)$ compartments; when a node enters a node it will go through one of its compartments uniformly at random

3 steps

1. Distribute **destination tokens** over compartments so that each compartment has only one token and only a constant fraction of compartments are empty
2. Each node sends out a source token and has it walk for the mixing time; if it ends in a compartment with a destination token then it enters the place of the token
3.

To achieve first step: create a destination token with multiplicity: splitting phase: each token splits into two tokens of half value and walks for mixing time, after repeating $\log k$ times all tokens have multiplicity 1 but need to make sure that each compartment only has 1 token; since constant fraction of compartments are empty with constant probability token ends in an empty compartment and so can just repeat logarithmically many times

1.7 Orr Fischer on A Distributed Algorithm for Directed Minimum-Weight Spanning Tree

Motivation: Minimum Spanning trees

Dates back to start of distributed computing

Round complexity: $\tilde{O}(D + \sqrt{n})$

Look at analogue in directed case; goal is to find a minimum weighted directed spanning tree root at r

Previously best known bound was $O(n^2)$

History: Edmonds gave $O(m + n \log n)$ in sequential setting; PRAM in $O(\log n)$ by Lovasz; Humblet gave $O(n^2)$ rounds and $\tilde{O}(n^2)$ messages

This paper gives first sublinear algorithm in CONGEST

CONGEST

Could talk about different models of communication in CONGEST if the graph is directed

- Bidirectional communication (focus of this talk but paper gives solutions for other models)
- Directed communication

- Congested clique

Results

Using a black box SSSP can solve DMST in same time up to poly logs

Plugging in best SSSP algorithm gives a roughly $O(\sqrt{n}D^{1/4})$

Also show that DMST is at least as hard as s-t shortest path (this was known in sequential setting previously)

Main message is DMST is sandwiched between SSSP and st-SP

Talk Outline

Edmonds' algorithm

Meta algorithm

Taste of CONGEST implementation

Reduction from st-SP

Edmonds' Algorithm

1. subtract min incoming value from each vertex
2. add a zero-weight edge to H
3. contract all zero cycles
4. Unpack and delete unnecessary arcs

Worst case: gets rid of one vertex per iteration

Want some accelerated Edmonds variant which e.g. halves the number of components by "skipping ahead"

Lovasz Algorithm

Observation: Contractions that we do in Edmond's is the shortest path from outside the cycle to the cycle

While there are (non-root) active compoents:

- Each new vertex reduces its minimum weight from all its incoming edges
- Each active component finds a node c_i from active component cycles
- Run from c_i an SSSP instance inside its component
- Contract everything within shortest-path radius (and updates weights)

Finding a vertex in active cycle

Just propogate on edges your ID and if you see a higher ID then propagate it if cycle less than $\sqrt{n}$

Otherwise, sample $\sqrt{n}$ centers and forward BFS until reach another center

Lower Bounds

Reduction from DMST to st-SP

Construct some G' s.t. DMST for G' gives the st-SP for G; then simulate DMST on G'

Reduction creates identical "shadow network" where all edges have cost 0 and each shadow node has an edge into its node

The optimal DMST is just the s-t shortest path along with the shadow network from t back to s

2 October 16, 2019

2.1 János Kertész on Network science – a bridge between social and "hard" sciences

Complexity

Hawking: "next century will be century of complexity"

What is complexity? Difficult to define; so what is not complexity?

Watch is complicated but not complex; with a book and enough time one could understand the watch

Brain is complex; a non-trivial hierarchy of interactions from biochemical interactions to thoughts and emotions

Complex systems:

- many components
- feedback
- nonlinearity
- cooperativity
- emergent phenomena

E.g. cell, brain, society, economy

Networks: Scaffold of Complexity

What is the essential feature that makes a system complex?

Is it the number of components? No; there are single cell organisms with larger genomes than humans

What is it then? It's the way these constituients interact with eachother

That leads us to networks; they are the scaffold of these complex systems

Networks Everywhere

- Network theory to catch Saddam
- Brain network
- Geography of Facebook friendships
- The internet
- The economy
- Political networks
- Management (e.g. searching email traffic)
- History (e.g. marriage links in medieval Florentine families)

Conditions for Network Science

Why has network science gotten so important?

1. Data: high capacity computers and efficient algorithms
2. Interdisciplinarity
    - Math

- Social sciences
- Computer science
- Physics

Time got ripe for this stuff around 2000

1. Small world (1998) in Nature
2. Scale free networks (1999) in Science

Both papers got many many citations

Universal Features of Complex Networks

Properties true of networks from many different places:

- Small worldness: small distance in large networks; e.g. six degrees of separation (Milgram's experiment) and Kevin Bacon game and Erdos number
- High clustering: many triangles; in social networks this is just friends of friends becoming friends
- Scale freeness: broad degree-distribution; degree distribution is well-described by a power law distribution
- Modular structure: modules/communities in the structure which are internally well-wired than to the rest of the network

Network Science

Goals: understand how these universal features come about and the laws of the evolution of the networks

Network science tries to

1. search for general laws
2. apply theory to specific problems

Tools: data mining, graph theory, modeling, computer simulations

Dynamics of and on Netowrks

Networks are not static, they evolve

Spreading on Networks

- Physics: nucleation (e.g. tin pest)
- Biology: epidemics
- Social sciences: information, rumor, behavioral, etc.

Similarities and Differences

Physics: spreading takes place on a lattice, always governed by immediate contact, an external field

Biology: spreading takes place on social network, again by contact, no external influence

Social contagion: network is social but transmission is contact *and* social pressure, an external influence of media (a complex contagion process

Basic Models of Epidemic Spreading

States of nodes:

- S: susceptible

- I: infected
- R: recovered/immune

Can have different models based on these

- SI
- SIR
- SIS (infected but susceptivle after recovering)
- SIRS (etc.)

Theories: Mean field theories, simulations, compartmental mean field

Social Contagion

Main focus of the talk

Important differences from disease spreading:

- Social pressure (state of neighbors influence transmission probability)
- Flow of external influence

Diffusion of Innovations

Innovation: not enough to have a brilliant idea; success is needed, e.g. typing keyboard as a counterexample

Spreading (diffusion) of innovations: a mostly anecdotal theory

Cascading Phenomena

Want to make a more quantitative theory

Examples:

- Rumor (e.g. false breakdown in nuclear power plant)
- Political movements (e.g. Arab Spring)
- Innovation (e.g. Twitter)

Explosions of growth because of cascades

Threshold Model

A formal model to describe these phenomena which was then put into mathematical terms (the Watts Model)

A random network with degree distribution P_k; average degree of z; every node has a threshold value ϕ; this threshold value represents the **critical ratio** of adopting neighbors needed to make the node adopt

There are **vulnerable** nodes which get infected if they have one adopting neighbor

All other nodes are **stable**

Phase diagram can be calculated where higher z and lower ϕ gives global cascades and the reverse gives no global cascades

However, this model doesn't tell us about the speed of the process

Generalized Watts Model

They extended it in two ways

1. Some **nodes are blocked**; some people reluctant to adopt; nodes are blocked with probability r

2. There are **spontaneous innovators** appearing which act as external influencers

Can be solved like original model with the generating function method; uses fact that these networks have locally tree-like structures which enables recursions between generating functions

Could also add a rate of spontaneous adoption p; different behavior depending on the ratio of blocked nodes

Instead of Anecdotes: Big Data

Studied spreading of Skype services

Structure of data is underlying social network with service of Skype and paid service network ontop of that; we know all but the underlying social network

Corroborated assumption of the theory that it is "the ratio" that matters

Cascadic Collapse

Networks can shrink and die

Happened to "Hungarian Facebook" iWiW

Two origins of people leaving the network

- External info: rise of Facebook
- Peer pressure: not enough friends in the network

A similar model to the previous one can be put forward just with people leaving the network instead of joining

Can also look how cascades of leaving evolve

2.2 Takeharu Shiraga on Phase Transitions of Best-of-Two and Best-of-Three on Stochastic Block Models

Synchronized Randomized Voting Processes

Given undirected and connected graph where each vertex has "red" or "green" opinion

In each time step each vertex updates its opinion according to a rule

In **pull voting** each node adops the opinion of a randomly picked neighbor

Best-of-Two and Best-of$_{\text{Three}}$

In these voting processes, each node updates its opinion to match the majority of 2 randomly picked neighbors and itself or the majority opinion of 3 randomly piced neighbors

Showed a simulation on complete graph

Known Results

Pull voting takes $O(n)$ but Bo2 or Bo3 takes $O(\log n)$ to stabilize on K_n

On non complete graphs pull takes $O(n^3 \log n)$ but seems hard to analyze Bo2 or Bo3 on general graphs

On expanders (e.g. $G(n,p)$) Bo2/3 takes $O(\log n)$

But on some graph classes Bo2 takes $\Omega(n)$ steps

This Work

Studies how graph structure affects Bo2/3.

Study process on **stochastic block model**: two $G(n,p)$ connected with random edges with density q

If $q/p = 1$ it is $G(2n,p)$ and so takes $O(\log n)$

Gave an example where if q/p is small no conensus but if large there is consensus; so question is where is the threshold

Show a phase transition for Bo2 at $p/q = \sqrt{5} - 2 \approx .236$; $O(\log n)$ time when below but $2^{\Omega(n)}$ when above threshold

For Bo3 threshold is $p/q = 1/7 \approx .142$ so Bo3 is more likely to reach consensus

Outlined proof / example

2.3 Seth Gilbert on On Bioelectric Algorithms

Talking about the planarian; a flatworm; when cut it regrows; process of regeneration is controlled by electric currents; driven by the "bioelectric code"; question of whether you can reprogram the electric currents to change how they grow; scientists still trying to understand how to control this process

Why study biological algorithms?

1. Neat algorithmic problems
2. Learn from biology
3. Understand limits of computation
4. Understand better how biology works

State of the Art

Michael Levin at Tufts built a simulator for bioelectric cells

Primary technique: **matrix-based diff eqs**

Analogy: simulate an OS by solving the diff eqs of current flows

Big Picture Goal

Can we raise the level of abstraction?

Cellular Bioelectric Model

- Distributed network of cells
- Each cell has an electric potential
- Cells communicate by exchanging ligands

Looked at typical problems in this world; e.g. symmetry breaking (leader election, MIS).

Why MIS? Supposdely observed in brain cells of fruit flies

Then moved at information processing questions; threshold detection, majority detection, bioelectric cells are Turing complete

Conclusion: cells can compute fairly powerful functions but only with some error / noise

Cell Definition

- Electric potential q
- Equilibirum potential σ
- Gradient (rate) λ

- Minimum potential

Biolectric Interactions

- Small collection of different ligands (i.e. messages)
- Ligands exchanged via bioelectric events
- Probability function $f : q \to [0, 1]$
- Offset δ: on triggering event, increase potential by δ
- Membrane function g maps sets of ligands to changes in potential

Simulating Cells

Cells start with initial potential

Events triggered depending on potential

Each event causes a potential offset

Each cell processes incoming ligands

etc.

Constraints

Small number of different events

Trigger functions are monotonic

Membrane functions reach saturation: cannot differentiate between more than a small number of ligands

Observation

A cell is strictly weaker than a state machine: e.g. cannot output ABCABCABCABC because monotonic firing functions but OTOH a state machine cannot store a real number so these are incomparable

Also not just a neural network; notably don't get to design the network

Example: Leader Election

Trying to elect a leader

Initial potential $q = 0$, equilibirum potential $\sigma = 2$ at rate of $1/2$

One bioelectric event: trigger event via 0, 1/2, 1 step function; if you receive the ligand it knocks down your potential and you become a leader when your potential is 2

Did out an example run

Open Questions

What other sorts of things can you do?

E.g. can you make a biolectric network which differentiates the head and tail; do the results correspond to the diff eq simulator

2.4 Mien Brabeeba Wang on Integrating Temporal Information to Spatial Information in a Neural Circuit (Brief Announcement)

Problem: humans very good at recognizing temporal information; question is how does brain do this and how can we model this

Neural coding

1. rate coding (e.g. neuromuscular junction)
2. temporal coding (e.g. visual cortex); timing of the spike matters
3. first-to-spike coding (e.g. retina)

Long Range Temporal Processing

1. single neuron has some limited short term temporal processing
2. but many temporal input comes in at a time scale of minutes

Consider a sychnronized discrete model to see if long range temporal processing is possible

Gave formal model to study counting problems: "first consecutive spikes" and another problem; showed exists a network which can do this

Gave lower bounds which show there is no network which solves these problems with a smaller size

Recap

1. Two toy problems modeling two popular neural coding schemes
2. Networks that optimally solve these
3. Lower bounds that show convergence time is tight

2.5 Julian Portmann on Improved Network Decompositions using Small Messages with Applications on MIS, Neighborhood Covers, and Beyond

Defined LOCAL and CONGEST

Network Decompositions

A $(c(n), d(n))$ ND

- partitions network into clusters of diameter at most $d(n)$
- coloring the cluster graph with $c(n)$ colors

$O(\log n), O(\log n)$ network decompositions exist and are optimal (in the sense that no simultaneous improvements of c and d)

Many problems in LOCAL are easily solved given a network decomposition: work through color classes one-by-one and gather topologies in $O(d)$ rounds and locally compute solutions

These are also helpful in CONGEST in

- derandomization
- amplifying success probabilities

For computing these NDs:

- Deterministic: $2^{\sqrt{\log}}$ LOCAL rounds with $c = d = 2^{O(\sqrt{\log n})}$; similar bounds in CONGEST
- Randomized: some results

k-Separated Network Decompositions

Sometimes problems are less local; e.g. in k-MIS where want no neighbors in the MIS within k hops normal ND doesn't work

But can extend to a k-Separated ND; can compute this by just working on G^k instead of G

Gave previous known results for these

Results

Can compute a k-separated $(2^{\sqrt{\log n}}, k2^{O(\sqrt{\log n})})$ ND in $k2^{O(\log n)}$ rounds of CONGEST

Implies several other results in CONGEST

Algorithm Outline

1. First step
    - Each cluster proposes to all its neighbors
    - Clusters accept up to d proposals
2. Second step; Have low-degree, high-degree and marked clusters
    - Each cluster acknowledges the received proposals
    - Clusters accept up to d^2 acks
    - Clusters are labeled
3. Third step
    - Only consider non-marked clusters (work in a low degree subgraph)
    - Find a maximal 2-independent set among all high-degree clusters
4. Fourth step
    - Create new clusters centered at 2-MIS or marked clusters
5. Fifth step
    - Color remaining low-degree clusters

Challenges in CONGEST

Clusters cannot act as nodes; get around this by only considering low degree subgraphs

Clusters can become disconnected; mainly a "technical issue"

Application to MIS

Use MPX for LDDs in a distributed setting

Open Problems

A $(\text{poly}\log n, \text{poly}\log n)$ in determinstic poly $\log n$; but this was recently solved

Still how to use low diameter more efficiently in CONGEST

2.6 Shimon Bitton on Message Reduction in the LOCAL Model is a Free Lunch

Defined LOCAL and a k-spanner

Peleg-Ulman Transformation

Given distributed task $\mathcal{T}$ and a taks $\mathcal{A}$ an algorithm for solving $\mathcal{T}$ in G can be solved by $\mathcal{A}'$ by

1. Construct a k-spanner H

2. Simulate each communication round of $\mathcal{A}$ by \$k\$communication rounds on $H = (V, S)$

Incurs a multiplicative k in time and $k|S|$ in messages

But what about spanner constrution?

Contribution

A randomized distributed algorithmto compute spanner with $n^{1+\epsilon}$ many edges and in a constant number of rounds using $n^{1+\epsilon}$-many messages

So now can construct a spanner and then consider the entire complexity; get $\mathcal{A}'$ gets same time complexity as $\mathcal{A}$ and lose only a multiplicative $n^{1+\epsilon}$ in the message complexity

Algorithm

Inspired by spanner construction of Baswana and Sen

Uses Hierarchical node sampling

Works in $k + 1$ phases

1. Nodes sample subset edges uniformly at random; adds one edge for each sampled neighbor to spanner; delete all edges to a sampled neighbor
2. Becomes a center with some probability on graph induced by spanner edges
3. If a node is not a center it tries to connect to a neighboring center; if such a neighbor then join this neighbors center
4. Construct global cluster graph and recurse on this graph

Left out details but will talke about 1. more

2.7 Gregory Schwartzman on Parameterized Distributed Algorithms

Described distributed model of communication; assumption of unbounded computational power which is important for this work

Parametrized Algorithms

Way of dealing with NP hard problems; if solution size is $< k$ can e.g. solve a problem in $O(2^k n^{O(1)})$

So why not use size of OPT as well in distributed

Distributed parametrized algorithms

All nodes know k

- For min problems if OPT $\leq k$, compute solution $\leq k$
- If not; all nodes report (i.e. say no solution $\leq k$ in size)

This definition also extends to α-approximations where compute a solution of size at most αk

Previous work

Some work on "detection" problems; this work considers other fundamental problems; e.g. vertex cover with polynomial running time w.r.t. k

Results

Consider variants of many covering / packing problems (some of which are not NP hard in centralized setting like max matching)

Lower / upper bounds in LOCAL / CONGEST

Lower Bounds

Already have LBs for these problems but the size of the solution in these problems is large so they don't extend

But can get parameterized LBs by adding large degenerate graphs like star

Lower bounds of $\Omega()1/\epsilon$ for a $(1+\epsilon)$ approximation for many problems

Upper Bounds

Use subroutine to estimate diameter of graph in $O(k)$ CONGEST rounds

Defined problems DLB where OPT $= \Omega(D)$; e.g. vertex cover; min versions of these problems can be solved in $O(k)$ LOCAL rounds

Gave table of upper bounds with dependence on k

Computer ran out of battery...

2.8 Merav Parter on Small Cuts and Connectivity Certificates: A Fault Tolerant Approach

Revisit classical problems in connectivity in CONGEST + fault tolerant network design

Interested in sparse FT structures that preserve properties of original graph

Two examples

- connectivity certificates
- min cut

Sparse Connectivity Certificate

A subgraph H that keeps important info about small kuts; for parameter k

1. for every pair of nodes s and t and any set of up to $k-1$ edges cut, s and t should stay connected
2. H should have $O(kn)$ edges

Previous Results

Can compute sparse certificates by taking MST and computing min forests on what remains k many times; gives running time of $O(k(D+\sqrt{n}))$

Can get $O(D+k\sqrt{n})$ rounds

Improved to $O(D)$ for $k=2$ and $k=3$; but algorithm was complicated and not clear how to extend to large k

Result

In this work can get in $\tilde{O}(k)$ and $(1-\epsilon)$ in $\tilde{O}(1)$ randomized rounds

Get result by looking at fault tolerant spanners: $O(\log n)$-stretch spanners even after cutting up to $k-1$ edges; these spanners can be efficiently constructed using previous work of Pettie and Chechik

Distributed Min Cut

Weighted: $\Omega(D+\sqrt{n})$

Unweighted: $\Omega(kD)$ where k is the value of the minimum cut

But upper bounds not fully satisfactory

If cut val is 2 then can compute in $O(D)$ rounds

Best known is sublinear rounds for min cut for any cut value

In this talk look at other extreme where the cut is small

Question is whether $o(D + \sqrt{n})$ bound can be beaten by detecting *all* small cuts; will use on poly$D$ rounds

Simple Algorithm for Small Cuts

Assume min-cut is k

Experiment i, fixed source s, repeat (kD^{2k}) repetitions which is poly D:

- G_i by sampling each edge in graph with probability $1 - 1/(KD)$
- Compute BFS tree from s truncated at depth kD
- Each node collects its path from s in the tree

Each node t locally computes s-t cut in the collected subgraph G_t; claim is that at least 1 node computes the min cut

poly$D$ runtime is immediate

Analysis

Based on **observation**: given nodes s and t; if s and t are connected after deleting E' then they must be within kD

To compute a min cut just need to see if E' is a cut

min cut is k -> there is some t such that the s and $t$connectivity is at most $k + 1$ -> so connectivity with just edges collected by t is $\leq k + 1$ -> t outputs F which can prove separates s and t

2.9 Orr Fischer on Sublinear-Time Distributed Algorithms for Detecting Small Cliques and Even Cycles

Subgraph Freeness Problem

Given G and constant-sized H subgraph

In LOCAL trivial

In CONGEST interestin

So illustrates difference between models

Prior Work

$\tilde{O}(n^{1/3})$ triangle enumeration (expander decompositions in CONGEST)

$\tilde{\Omega}(\sqrt{n})$ for K_s-freeness; can be made larger

$\tilde{\Omega}(\sqrt{n})$ for C_{2k}-freeness

etc.

New Subgraph-Freeness Results

Sub-quadratic algorithm for general subgraph freeness

Sublinear algorithms for K_4 and K_5:

Strong lower bounds for even cycles imply circuit lower bounds

Improved sublinear algorithms for even cycles; tools from extremal graph theory

Main tool: **expander decompositions**

Outline

-Intro to expander decompositions

- K_4 result
- One more

Background

Defined CONGEST and congested clique

High conductance expanders is similar to "weak congested cliques"

Expander Decomposition

Intuitively: split graph into clusters ("weak congested cliques") with few edges in between

K_4 - Enumeration Algorithm

Take ED; rest of graph has low arboricity

If K_4 in low arb part do one thing; use fact that a graph of arboricity α can be ordered so that each edges has $O(\alpha)$ outgoing edges; then can broadcast out over these edges and one node will learn K_4-ness

If in a cluster then use expansion; node is ϵ-heavy wrt cluster if n^ϵ neighbors in it then send all edges into cluster; otherwise it is ϵ-light then pool edges in neighbor

- For former case partition neighbors into sets of size n^ϵ and send all edges into cluster; then can simulate n-vertex congested clique in the cluster inside high conductance cluster, even the edges outside the cluster
- For latter case broadcast to all neighbors; then can just query in ith round if nodes are connected to ith neighbor of all neighbors in cluster because ϵ-light

Balancing parameters gives $O(n^{5/6})$ running time

Hardness of Lower Bounds

$\Omega(n^c)$ on C_{2k}-freeness gives circuit complexity lower bounds

Show that previous lower bounds in Congested Clique hold even for high conductance expanders since high conductance expanders can simulate Congested Clique efficiently

Split graph into clusters with only $o(n)$ intercluster edges

To check if an edge is in a cycle basically can do BFS for k rounds; if a cycle then done; otherwise delete the edge; then just left with vertex disjoint clusters; then apply fact that these clusters simulate CC to get lower bounds

3 October 17, 2019

3.1 Seth Gilbert on When is an algorithm robust? (Keynote Talk)

Here as a "prophet of doom": your networks will fail and your algorithms are not robust

Gave a bunch of articles about networks that died; some of these are failure critical; e.g. see self driving cars

Key Trends

Key Trends: everything is getting worse

- Bigger networks (more failures)
- Networks are more connected
- Faster cahnges in networks (more failures)
- More optimized networks (more fragile = more failures)
- Less transparent
- More attacks

Solutions: distributed computing

This community has a long history of solving exactly these problems; many Dijkstra prizes for fault tolerance

A Fundamental Divide?

2010 PODC talk by Pierre Fraigniaud talking about how a political divide in the community

Recharacterization in terms of Candide pragmatists (red team) and Pangloss optimists (blue team)

At the time thought was red team was on the rise

Now shoe is on the other foot

Even at business meeting talked about high acceptance of blue team stuff and lower acceptance of red team stuff

Three Possible Reasons

1. Search for new problems
2. Amazing new results
3. Less applicable to modern systems? Some of the fault tolerance techniques are less and less relevant to modern systems

As an aside: why didn't we invent bitcoin?

In key trends talked about how red/blue team can help with each issue

Toward more robust algorithms

Act 1: living in a noisy world; design algorithms that are robust to noise

Act 2: give me a little slack; design algorithms that expect things to change

World is a noisy place

We should design algorithms that tolerate noise

Normally we assume some powerful adversary and analyze the worst-case outcomes

Two issues

1. sometimes we don't add enough uncertainty
2. sometimes we add too much noise, making many problems fundamentally too hard

Some ideas for designing useful algorithms

Could look at parameterizing and instance-optimality

Focus today is: smoothed analysis and hybrid adversaries

2 aspects of noise

1. Noise can make things harder with e.g. unexpected behavior
2. Noise makes it easier; stochastic behavior is easier to predict

Smoothed analysis

Basic idea is we still have an adversary but we add some random noise to that input

Spielmen and Tang used this to explain why Simplex works well in practice

How to think of this in a distributed setting?

1. consensus
2. dynamic graphs
3. backoff

Shared memory consensus

FLP: deterministic wait-free consensus is impossible; even if randomized its slow

What happens if there is a little noise?

- Adversary: schedules steps for processors
- Noise: add Gaussian noise to time between steps

Aspnes PODC 2000 shows that now consensus is fast; get down to $O(\log n)$

Dynamic Networks

Graph topology changes in every round

In this setting many problems are now hard

What if add a little noise? Imagine that each graph is randomly perturbed a little such that

- perturbed graph is *close* to the original graph
- and satisfies needed properties

E.g. could use edit distance

Adding noise

Defined k-smoothing

Now discussed how this affected stuff on dynamic graphs

- random walks; recover $O(n^3)$ hitting time of static graphs (as opposed to exponential time in dynamic graphs)
- flooding time; bounds improved as before
- aggregation; nodes begin with a token; node have pairwise interactions until all tokens collected in one place; competitive ratio is $\Omega(n)$ always

What about wireless dynamic networks?

Defied radio network model

Could try and solve local broadcast or global broadcast

Defined DECAY algorithm; works in $O(\log^2 n)$ for local and $O(D \log n + \log^2 n)$ for global broadcast

But decay is not robust to dynamic graphs; adversary makes sure you have many neighbors when broadcast probability is large

In fact, can show the same thing for any broadcast algorithm in a network

What if we add a little noise?

But Decay is robust if we have static dynamic networks

Could have adversary that can only changed edges every τ rounds

Can show Decay is also efficient in this case (systems with fast and slow fading)

Now for alternative way of thinking about robustness from an optimization perspective

Scheduling appointments

Have a doctor with a bunch of scheduled appointments; somebody comes in and shifts the whole schedule down annoying many people

Alternative is to have left a bit more space between meetings in the schedule

Shows a tradeoff: a more optimized schedule is more fragile to disruption

Specific Goal

Maintain near optimal schedule with minimal reallocation

Big picture:: balance optimality and robustness

A lot of work in this area

Example: scheduling simple tasks

Given unit-length tasks and need to put each job in a legal time slot; can just run earliest deadline first algorithm

What if this is online with reallocation; can be as expensive as $O(n)$

Moral is: need to leave some slack

Could pretend every job is e.g. twice as big as it really is

Theorem: if have $O(1)$ slack then can maintain a schedule with a reallocation cost of $O(\log^* n)$ per insertion and deletion

Example: minimize disk use

Can insert jobs onto a disk; then stuff gets deleted and disk gets fragmented

If you can't move things this is an $\Omega(\log n)$ competitive-ratio lower bound

But if can reallocate can do better; a tradeoff between optimality and robustness

Cost of reallocation

- could be unit
- could be linear wrt size of job
- could be unknown and just monotone and subadditive

For all three show how to use $(1+\epsilon)$-competitive with OPT with $O(1)$-competitive reallocation

Overview

Noisy systems: robustness to noise vs smoothed analysis

Planning ahead: trade-off of robustness and efficiency