Graph Algorithms with Sparse Linear Algebra

Tim Baer

C3.ai Platform Engineering
University of Illinois at Urbana-Champaign

July 19, 2021

Overview

TODO:

- Introduce notation better/add notation slide?
- add pauses?
- Fix footnote numbers

Motivation

Adjacency matrices[3] connect graph algorithms with linear algebra.

For graph algorithms?

- Leverage fast matrix multiplication algorithms.
- Associated matrices have surprising spectral properties.

For linear algebra algorithms? (*Not discussed today*)

- Leverage graph techniques[3].

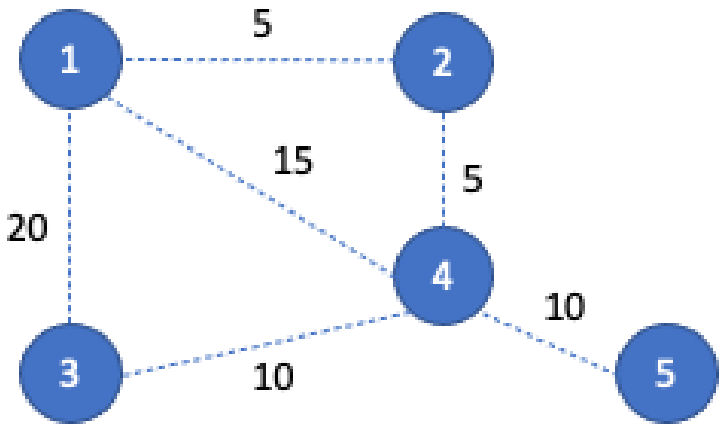
[3] And Laplacian matrices!

[3] See *Nearly Linear Time Algorithms for Preconditioning and Solving Symmetric, Diagonally Dominant Linear Systems* by Spielman and Teng.

Adjacency Matrices

Given a graph $G = (V, E)$ with weights $w : E \to \mathbb{R}$, construct the adjacency matrix[1] $\boldsymbol{A} \in \mathbb{R}^{n \times n}$ where

$$a_{ij} \leftarrow \begin{cases} w(i,j) & \text{if } (i,j) \in E \\ 0 & \text{otherwise} \end{cases}$$



$$\boldsymbol{A} = \begin{bmatrix} 0 & 5 & 20 & 15 & 0 \\ 5 & 0 & 0 & 5 & 0 \\ 20 & 0 & 0 & 10 & 0 \\ 15 & 5 & 10 & 0 & 10 \\ 0 & 0 & 0 & 10 & 0 \end{bmatrix}$$

[1] If G is undirected, store only upper-triangular part of $\boldsymbol{A}$.

Algebraic Structures

Monoids $(\oplus)$ generalize *addition*.

$$(\text{“C3”}) + (\text{“.ai”}) \rightarrow (\text{“C3.ai”})$$

Semirings $(\oplus, \otimes)$ generalize *addition* **and** *multiplication*.

$$x_1 \cdot y_1 + \cdots + x_n \cdot y_n \rightarrow \min\{(x_1 + y_1), \ldots, (x_n + y_n)\}$$

Monoids and semirings have several nice properties[2]:

- Identity elements
- Associativity
- ...

[2] However, neither structure enforces the existence of inverses.

Matrix Multiplication

Matrix-vector multiplication:

$$y \leftarrow \boldsymbol{A}x$$

Writing as an elementwise expression:

$$y_i \leftarrow \sum_k a_{ik} x_k$$

Generalizing to $(\oplus, \otimes)$:

$$y_i \leftarrow \bigoplus_k a_{ik} \otimes x_k$$

Matrix-matrix multiplication:

$$\boldsymbol{C} \leftarrow \boldsymbol{AB}$$

Writing as an elementwise expression:

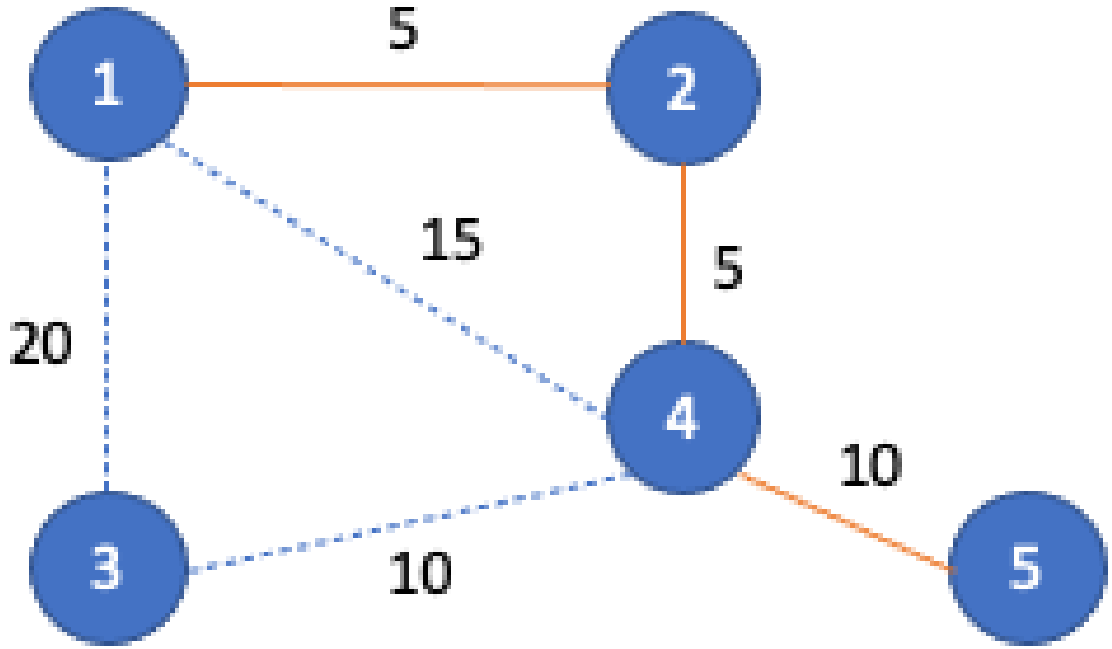
$$c_{ij} \leftarrow \sum_k a_{ik} b_{kj}$$

Generalizing to $(\oplus, \otimes)$:

$$c_{ij} \leftarrow \bigoplus_k a_{ik} \otimes b_{kj}$$

All Pairs Shortest Paths (APSP)

For each pair of vertices $i, j \in V$, compute the length of a shortest path from i to j.

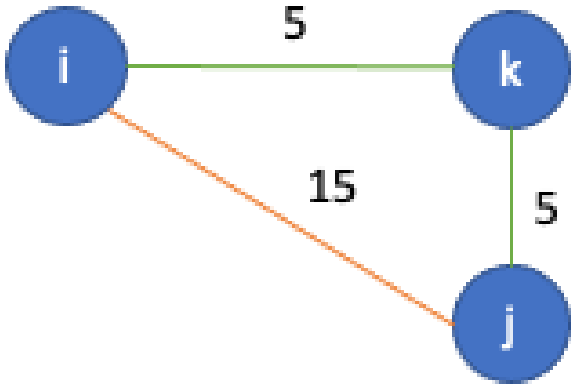


Sample application: driving directions, where road junctions are vertices and road distances are edge weights.

Bellman-Ford Algorithm

Fix a vertex i. Let $d_{ij}^{(\ell)}$ denote the length of a shortest path from i to another vertex j consisting of at most ℓ edges.

An edge (i, j) is tense if $d_{ij}^{(\ell)} > w(i, k) + d_{kj}^{(\ell)}$.



For $\ell \leftarrow 1$ to $n - 1$, relax all tense edges:

$$d_{ij}^{(\ell+1)} \leftarrow \min_k \{w(i, k) + d_{kj}^{(\ell)}\}$$

A Linear Algebraic Algorithm

We store tentative shortest path distances in $\boldsymbol{D}^{(\ell)} \in \mathbb{R}^{n \times n}$ where

$$\boldsymbol{D}^{(\ell+1)} \leftarrow \boldsymbol{D}^{(\ell)} \oplus \boldsymbol{A}\boldsymbol{D}^{(\ell)} \quad \text{on } (\min, +)$$

Writing as an elementwise expression:

$$d_{ij}^{(\ell+1)} \leftarrow d_{ij}^{(\ell)} \oplus \left(\bigoplus_{k} a_{ik} \otimes d_{kj}^{(\ell)}\right)$$

Replacing the generic $(\oplus, \otimes)$ with the tropical semiring $(\min, +)$:

$$d_{ij}^{(\ell+1)} \leftarrow \min\{d_{ij}^{(\ell)}, \min_{k}\{w(i,k) + d_{kj}^{(\ell)}\}\}$$

We may accelerate via matrix squaring $\boldsymbol{D}^{(\ell)} \leftarrow \boldsymbol{D}^{(\ell/2)} \oplus \boldsymbol{D}^{(\ell/2)}\boldsymbol{D}^{(\ell/2)}$.

Matrix Multiplication is Fast

Matrix multiplication has been studied from a variety of algorithmic lenses:

- Time complexity[3]
- Parallel and distributed
- With sparse data

[3] The current best time complexity is $O(n^{2.3728596})$ by Alman and Vassilevska via the laser method.

Strassen's Algorithm

$$\begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} \leftarrow \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$$

$$M_1 \leftarrow (A_{11} + A_{22}) \cdot (B_{11} + B_{22})$$
$$M_2 \leftarrow (A_{21} + A_{22}) \cdot B_{11}$$
$$M_3 \leftarrow A_{11} \cdot (B_{12} - B_{22})$$
$$M_4 \leftarrow A_{22} \cdot (B_{21} - B_{11})$$
$$M_5 \leftarrow (A_{11} + A_{12}) \cdot B_{22}$$
$$M_6 \leftarrow (A_{21} - A_{11}) \cdot (B_{11} + B_{12})$$
$$M_7 \leftarrow (A_{12} - A_{22}) \cdot (B_{21} + B_{22})$$

$$C_{11} \leftarrow M_1 + M_4 - M_5 + M_7$$
$$C_{12} \leftarrow M_3 + M_5$$
$$C_{21} \leftarrow M_2 + M_4$$
$$C_{22} \leftarrow M_1 - M_2 + M_3 + M_6$$

The time complexity recurrence is given by

$$T(n) = 7T(n/2) + O(n^2) = O(n^{\log_2 7}) \approx O(n^{2.807})$$

Requires the existence of an *additive inverse*.

Many Real-World Graphs are Large

Real-world graphs are too large to process on a single machine.

Graph	n	m
2002 crawl of the .uk domain	18.5M	261.8M
Friendster	65.6M	1.8B
Full USA road network	23.9M	28.9M
Orkut social network	3.1M	117M

Table: Publicly available large graphs

We may distribute computations across multiple machines[3].

[3] TODO: It is cheaper to use multiple commodity machines than building specialized machines

Parallel Matrix Multiplication

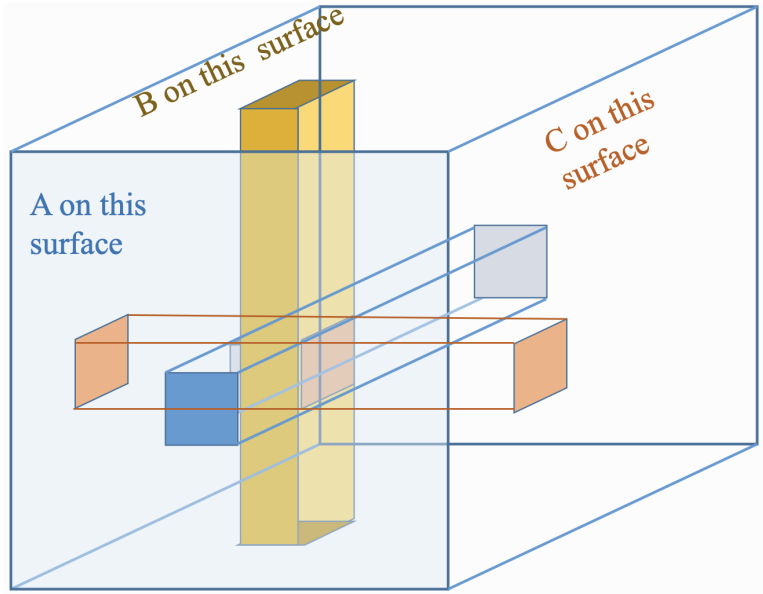
TODO
3D Matrix Multiplication was poorly introduced... introduce data layout in another slide (alex what does process owns ... mean?) or introduce parallel/distribute matrix algorithms? Would be nice to show trivial matrix-matrix distribution and why it's bad for communication

3D Matrix Multiplication

TODO, also might create new ideas for multilinear distribution

Organize processes into a 3D cube.

```
for i in range(n):
  for j in range(n):
    for k in range(n):
      c[i,j] += a[i,k] * b[k,j]
```



Process (r, s, t) owns $\boldsymbol{A}^{(r,t)}$ and $\boldsymbol{B}^{(t,s)}$ and contributes $\boldsymbol{C}^{(r,s)}$ where $\boldsymbol{C}^{(r,s)} \leftarrow \boldsymbol{A}^{(r,t)}\boldsymbol{B}^{(t,s)}$.

Figure from CS 484: Parallel Programming.

See *Communication-Optimal Parallel 2.5D Matrix Multiplication and LU Factorization Algorithms* by Solomonik and Demmel (2011).

Many Real-World Graphs are Very Sparse

Many real-world graphs have relatively few edges.

Graph	n	m	sp
2002 crawl of the .uk domain	18.5M	261.8M	1.5e-6
Friendster	65.6M	1.8B	8.4e-9
Full USA road network	23.9M	28.9M	1.0e-7
Orkut social network	3.1M	117M	2.4e-5

Table: Publicly available large graphs

However, storing an adjacency matrix requires $O(n^2)$ entries.

Compressed Sparse Matrix Formats

We may store only the *nonzero* entries for sparse matrices.

Compressed sparse row[4] (CSR) stores 3 lists:

- Values (length nnz)
- Column indices (length nnz): column of value
- Row offsets (length $n+1$): number of values above this row

$$\begin{bmatrix} 0 & 1 & 0 & 2 \\ 0 & 0 & 3 & 0 \\ 4 & 0 & 0 & 0 \\ 5 & 6 & 0 & 0 \end{bmatrix}$$

Values: $1, 2, 3, 4, 5, 6$

Column indices: $1, 3, 2, 0, 0, 1$

Row offset: $0, 2, 3, 4, 6$

[4] See also coordinate list and compressed sparse column (CSC) formats.

Compressed Sparse Matrix Algorithms

Sparse matrix-vector multiplication (SpMV):

$$y_i \leftarrow \sum_k a_{ik} x_k$$

```
for i in range(n):
  for t in range(row_off[i], row_off[i+1]):
    y[i] += val[t] * x[col_idx[t]]
```

Sparse matrix-matrix multiplication (SpMSpM):

- Communication-optimal algorithm is equivalent to partitioning a hypergraph[5].

[5] *Hypergraph Partitioning for Sparse Matrix-Matrix Multiplication* by Ballard et al (2016).

Cyclops Tensor Framework

Matrix multiplications can be generalized to tensors[6]:

$$u^{(1)}_{i_1 r} \leftarrow \sum_{i_2 \ldots i_d} t_{i_1 \ldots i_d} u^{(2)}_{i_2 r} \cdots u^{(d)}_{i_d r}$$

The Cyclops Tensor Framework[7] automatically parallelizes custom operations on tensors:

- Distributed memory
- Sparse tensors and operations
- Arbitrary semirings

Tensor algebra is a *framework* for expressing graph algorithms.

---

[6] For example, MTTKRP is the main kernel for tensor decomposition.

[7] *A Massively Parallel Tensor Contraction Framework for Coupled-Cluster Computations* by Solomonik et. al (2014).

Hybrid Programming (OpenMP + MPI)

The Cyclops Tensor Framework uses hybrid programming to leverage parallelism at the *thread* and *process* level.

OpenMP

- Shared memory
- Fork, exec, wait model

```
#pragma omp parallel for
for (i = 0; i < n; i++) {
    // do something
}
```

MPI[8]

- Distributed memory
- Point-to-point and collectives
- Standard for HPC

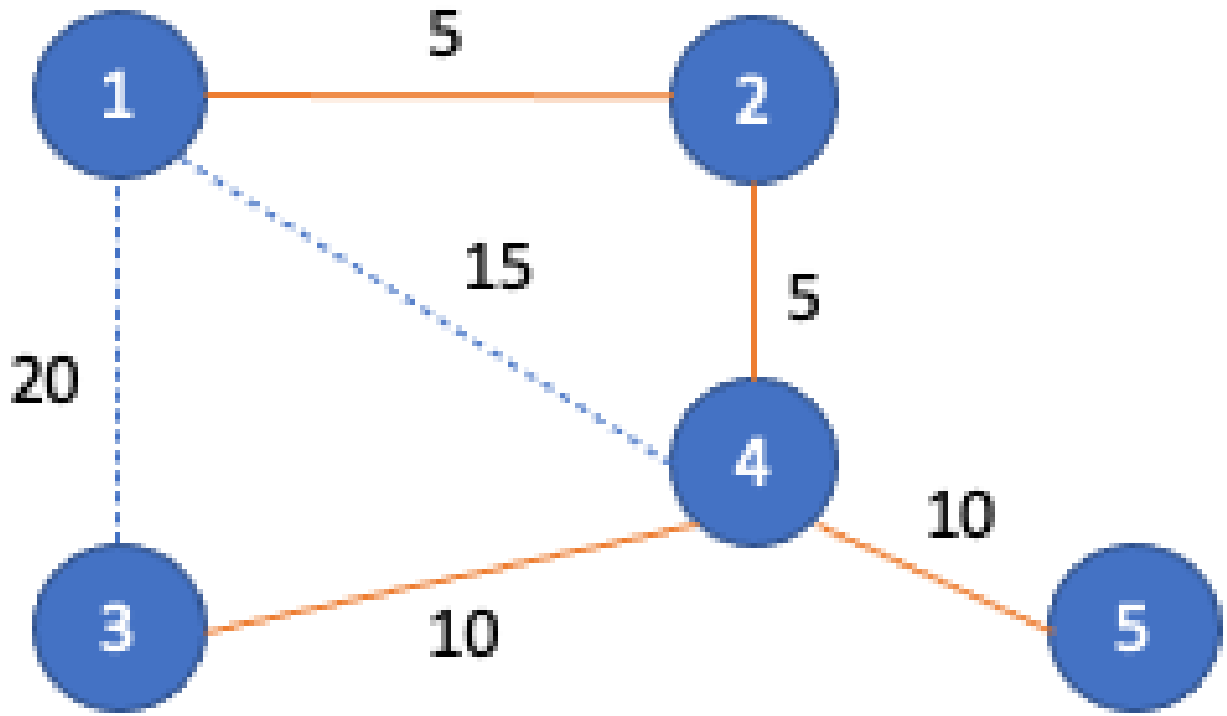
```
// first node
MPI_Send(&send_data, ...)

// second node
MPI_Recv(&recv_data, ...)
```

[8] In distributed deep learning, the *AllReduce* model often communicates via NCCL, which is modeled after MPI.

Minimum Spanning Tree (MST)

Find a spanning tree whose total weight is as small as possible.

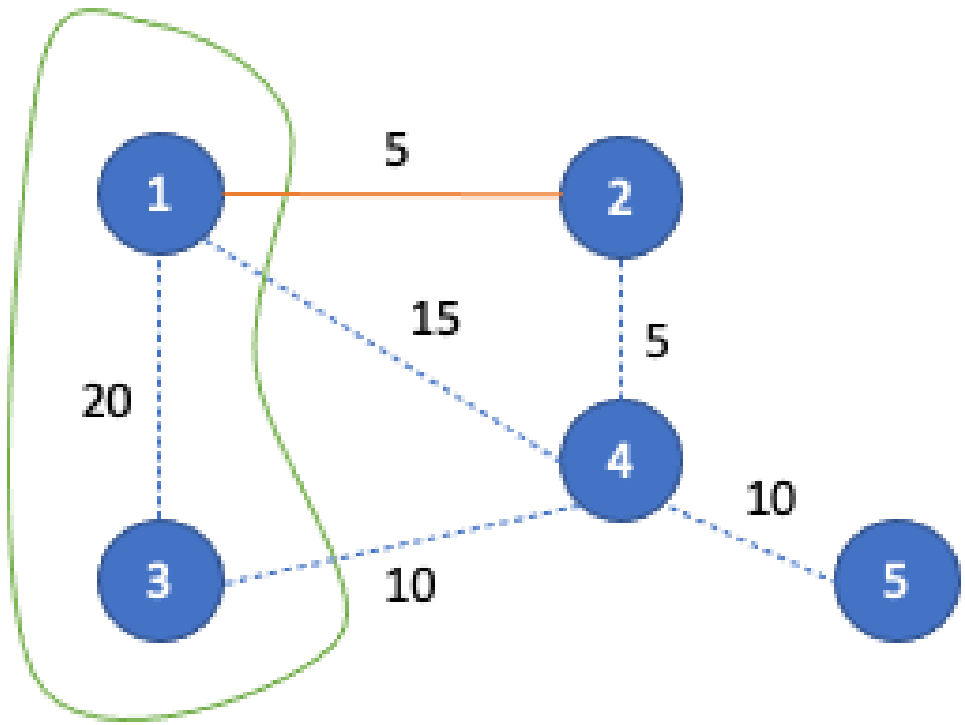


Sample application: a phone company laying cable in a neighborhood, where houses are vertices and cable lengths are edge weights.

Minimum Spanning Tree Algorithms

Most[9] algorithms rely only on the cut property of MSTs.

> Given a cut, the edge with the smallest weight that crosses the cut belongs to the mst.



[9] Karger et. al's $O(m)$ expected time algorithm also uses the cycle property.

Awerbuch-Shiloach Parallel MST Algorithm

TODO: improve?
Maintain a forest of stars[10], where each vertex has a parent.

At the first iteration, the forest consists of n vertices, each with a self-loop.

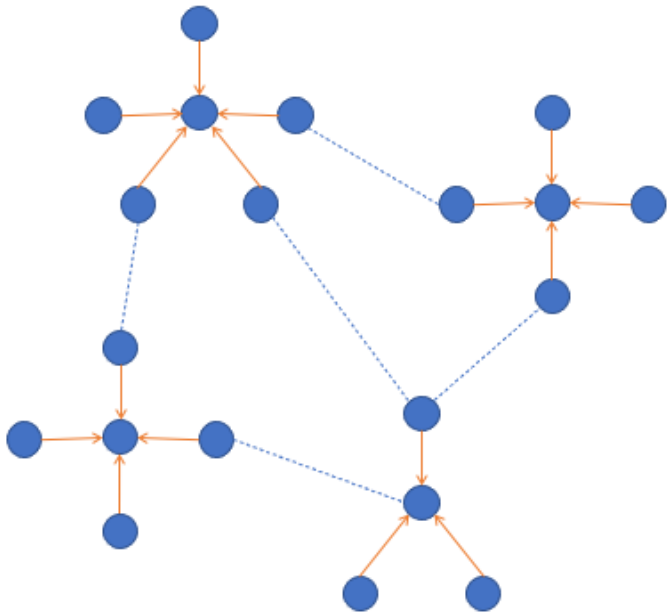
Repeats the following steps until convergence of the forest:

- Hooking: grows the MST by joining two stars together.
- Shortcutting: compresses trees into stars.

Can we rewrite this algorithm with sparse linear algebra?

[10] A star is a directed tree of height at most 1 and is intuitively a part of the minimum spanning tree.

Forest of Stars



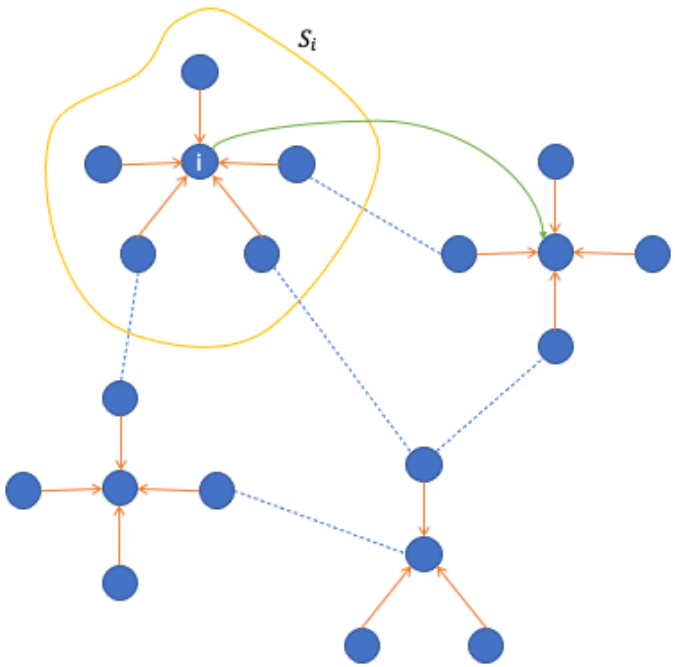
The parent vector $p \in \mathbb{R}^n$ stores the parent of each vertex.

The parent matrix $\boldsymbol{P} \in \mathbb{R}^{n \times n}$ is defined

$$p_{ij} \leftarrow \begin{cases} 1 & \text{if } p_i = j \\ 0 & \text{otherwise} \end{cases}$$

Hooking

Each star attempts[11] to hook with its *smallest cut edge*.



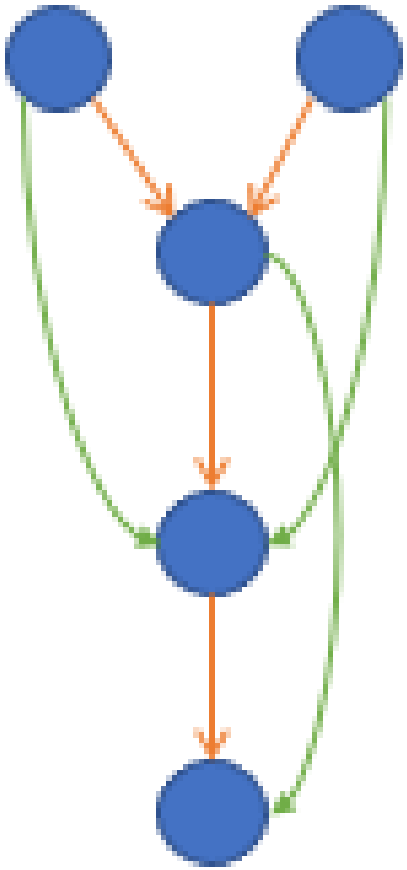
Hooking: joins two stars together.

$$p_i \leftarrow p\Big[\underbrace{\operatorname*{argmin}_{e \in \text{cut}(S_i, V \setminus S_i)}}_{\text{edges that cross cut}} w(e)\Big]$$

[11] Need to break ties to prevent cycles.

Shortcut

Hooking may generate trees of arbitrary height.



Shortcutting[12]: reduces the height of each tree in the forest by a factor of nearly two.

$$p_i \leftarrow p_{p_i}$$

[12] Until completion so that all trees become stars.

Hooking as a Multilinear Operation

f computes for each pair of nodes (i, j), whether they belong to the same star.

$$f(p_i, a_{ij}, p_j) \leftarrow \begin{cases} a_{ij} : p_i \neq p_j \\ \infty : \text{otherwise} \end{cases}$$

Accumulating $f(p_i, a_{ij}, p_j)$ over min computes for each node i, its smallest cut edge.

$$q_i \leftarrow \bigoplus_j f(p_i, a_{ij}, p_j)$$

For each star root i, we contract its children's smallest cut edges into itself.

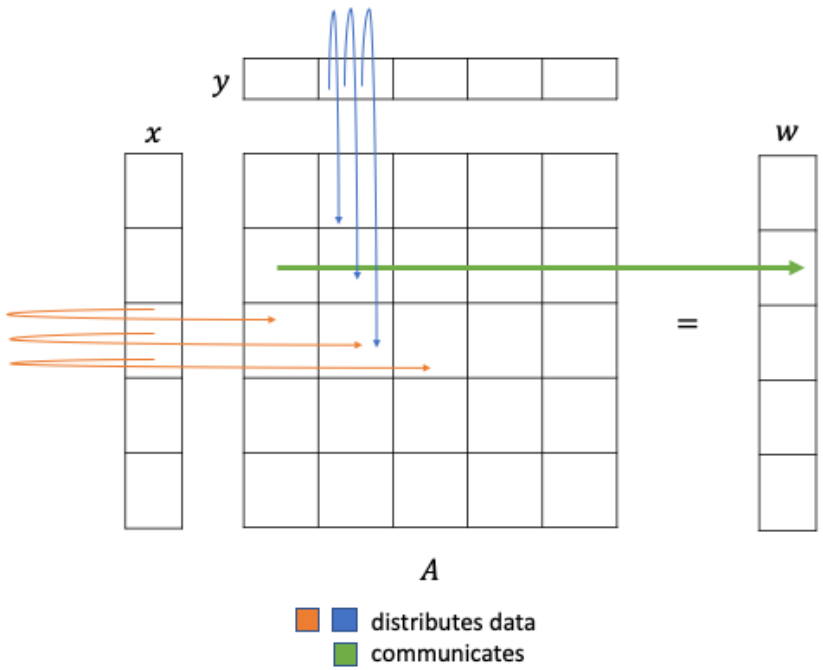
$$r_{p_j} \underset{\text{MinWeight}}{\leftarrow} q_j$$

Data Distribution

How can we leverage the structure of $f(p_i, a_{ij}, p_j)$?

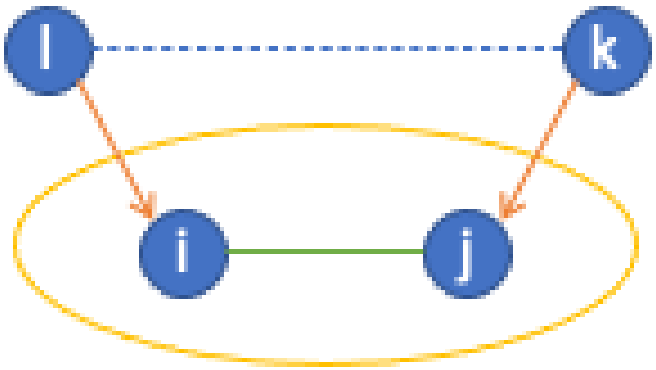
$$w_i \leftarrow \bigoplus_j f(x_i, a_{ij}, y_j)$$

Process (r, s) owns $\boldsymbol{A}^{(r,s)}$ and needs $x^{(r)}$ and $y^{(s)}$. Process (r, s) contributes $w^{(r)}$ where $w^{(r)} \leftarrow f(x^{(r)}, \boldsymbol{A}^{(r,s)}, y^{(s)})$.



Contraction

To reduce the size of the graph in the next iteration, we may merge vertices via contraction.



$\boldsymbol{C} \leftarrow \boldsymbol{P}^T \boldsymbol{A} \boldsymbol{P}$ on the $(\min, \cdot)$ semiring performs a contraction of $\boldsymbol{A}$.

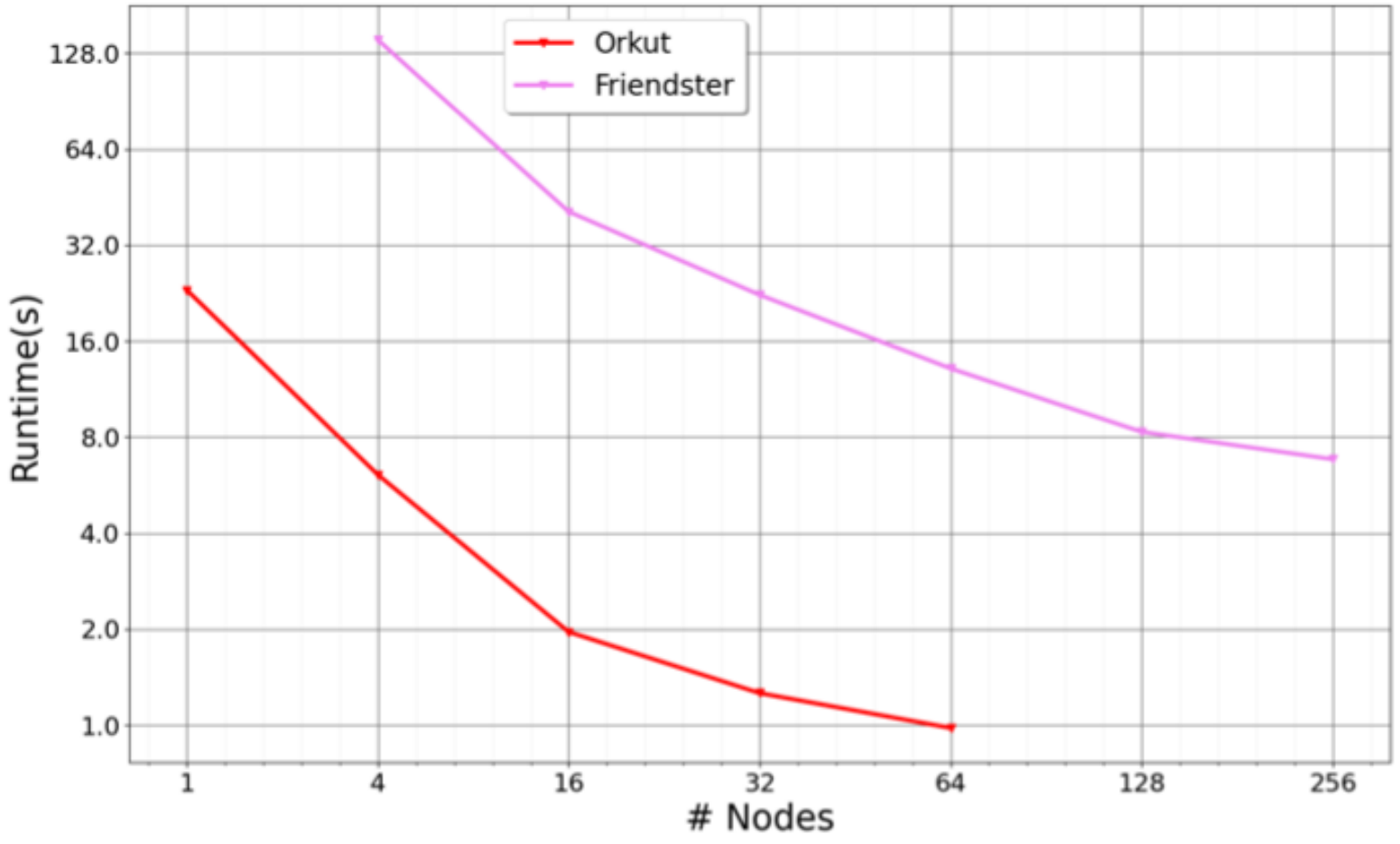
$$
\begin{aligned}
c_{ij} &\leftarrow \sum_l p_{il}^T \left(\sum_k a_{lk} p_{kj} \right) \\
&= \sum_l p_{li} \left(\sum_k a_{lk} p_{kj} \right) \\
&= \sum_{l,k} p_{li} a_{lk} p_{kj}
\end{aligned}
$$

$w(l, k)$ is projected onto $w(p_l, p_k)$ if:

- The edge (l, k) exists
- i is l's parent $(p_{li} = 1)$
- j is k's parent $(p_{kj} = 1)$

MST Strong Scaling Data

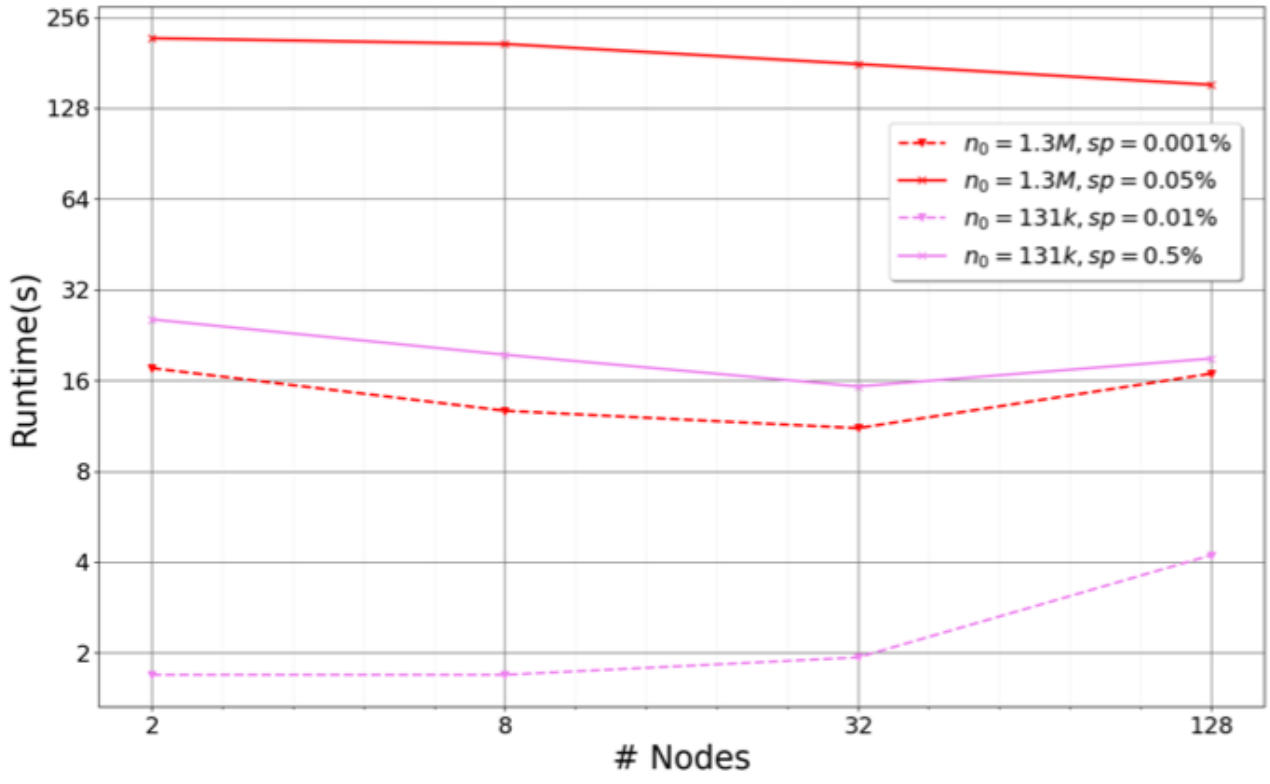
Strong scaling: fix the *total* problem size and vary the # of nodes.



(a) Social network graphs

MST Weak Scaling Data

Weak scaling: fix the problem size on *each* node and vary the # of nodes & total problem size accordingly.

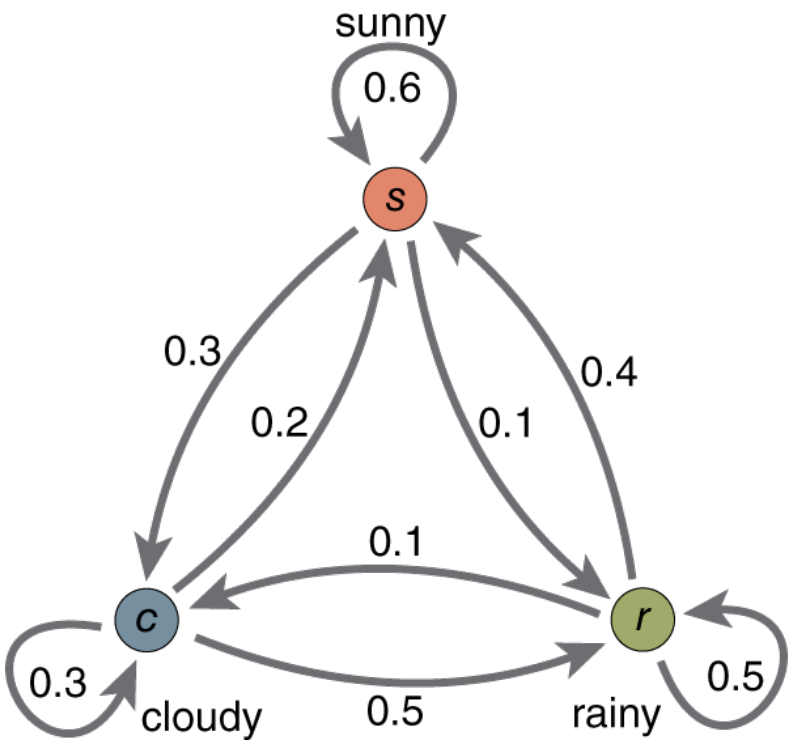


(a) Uniform random graph

Figure 2: Edge weak scaling of uniform random graphs. Constant $n^2/p = n_0^2$ and edge percentage $f = 100 * m/n^2$

Markov Chains

Markov chain: Given a set of states S, if the current state is i then go to state j with probability p_{ij}.



$$\boldsymbol{P} = \begin{bmatrix} 0.6 & 0.3 & 0.1 \\ 0.2 & 0.3 & 0.5 \\ 0.4 & 0.1 & 0.5 \end{bmatrix}$$

Figure from Stackoverflow: r - Creating Three-State Markov Chain Plot

Random Walks

We prove expectations of the below via analyzing random walks:

- Hitting time: number of steps to reach state j from state i?
- Cover time: number of steps to reach all states from state i?
- Stationary distribution: what is the probability of ending in state i in a walk with t steps as $t \to \infty$?
- Mixing time: number of steps to approach the stationary distribution?

A vector $\pi = (\pi)_{i \in S}$ is the stationary distribution if

$$\pi = \pi \boldsymbol{P}$$

The stationary distribution is an eigenvector of $\boldsymbol{P}$ with eigenvalue 1.

Mixing Time

From a starting distribution $\pi^{(0)}$, take a random walk with t steps.

$$\pi^{(t)} \leftarrow \pi^{(t-1)} \boldsymbol{P}$$

$$\vdots$$

$$= \pi^{(0)} \boldsymbol{P}^t$$

Mixing time: what is the expected number of steps to approach the stationary distribution?

The above algorithm is the same as power iteration[13]:

- Converges to the dominant eigenvector of $\boldsymbol{P}$ (namely, π)
- Convergence ratio is given by $|\lambda_1 / \lambda_2|$

[13] TODO: Why is norm 1?

Expanders

A d-regular graph is an α-expander[14] if $|\lambda_2| \leq \cdots \leq |\lambda_n| \leq \alpha$.

Rapid mixing lemma: For an α-expander,

$$(\pi^{(t)} - \pi) \leq \alpha^t$$

Do good expanders ($\alpha = O(\frac{1}{\sqrt{d}})$) exist?

- Random graph (probabilistic method)
- Graph products (zig-zag product)
- Ramanujan graphs ($\alpha = \frac{1}{\sqrt{d}}$)

[14] $\alpha < 1$. Combinatorial definition: A d-regular graph is an α-expander if

$$\frac{\partial S}{|S|} \geq \Omega(d(1-\alpha)) \quad \forall S \subseteq V \ \ (|S| \leq n/2)$$

Applications of Expanders

Expanders are a very powerful tool in modern TCS:

- Derandomization (randomness is a *resource*)
- Error correcting codes
- Sorting networks
- ...

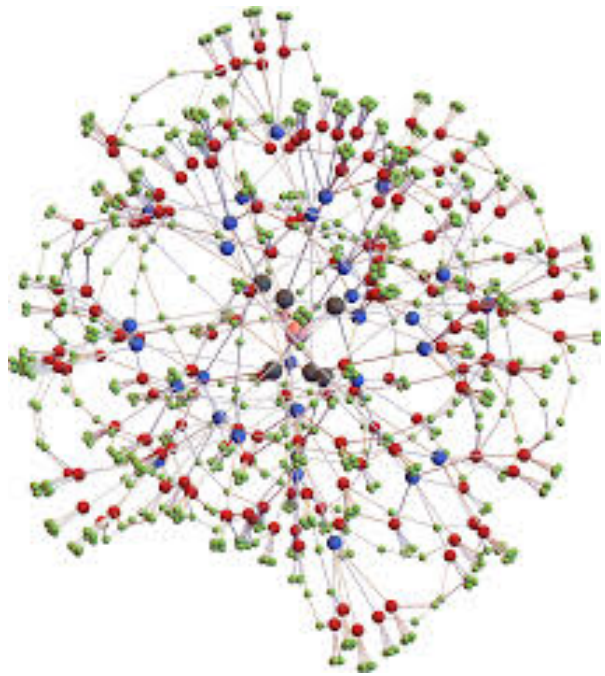


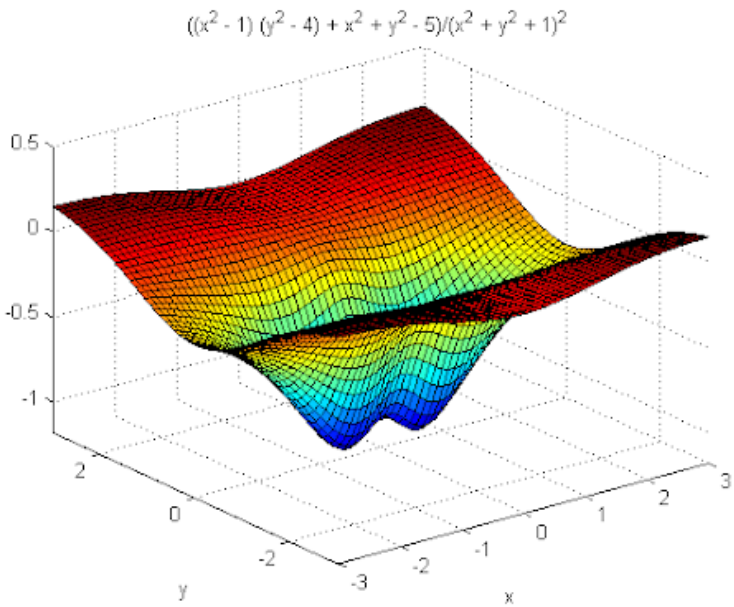
Figure from *Cutoff on All Ramanujan Graphs* by Lubetzky and Peres (2015).

Monte Carlo Algorithms

A Monte Carlo algorithm Π for a decision problem:

- Correctly outputs YES with probability $\geq 1/2$
- Correctly outputs NO with probability 1

Is the polynomial[15] $P(x_1, \ldots, x_k)$ over $\mathbb{Z}_P$ zero everywhere?



P has at most degree d roots:

- Pick a random vector $x_1, \ldots, x_k \in \mathbb{Z}_P$.
- $\Pr[P(x_1, \ldots, x_k) = 0] \leq \frac{d}{p}$.
- If YES, repeat?
- If NO, output NO.

[15] For example, $((x^2-1)(y^2-4)+x^2+y^2-5)/(x^2+y^2+1)^2$.
Figure from *Visualing Functions of Several Variables and Surfaces* by

Probability Amplification

Consider a Monte Carlo algorithm Π:

- Error probability $\leq 1/10$.
- m random bits.

How can we *lower* the error probability?

> Repeat t times $\rightarrow$ error probability $\leq (\frac{1}{10})^t$ and tm random bits.

Can we achieve a similar error probability with fewer random bits?

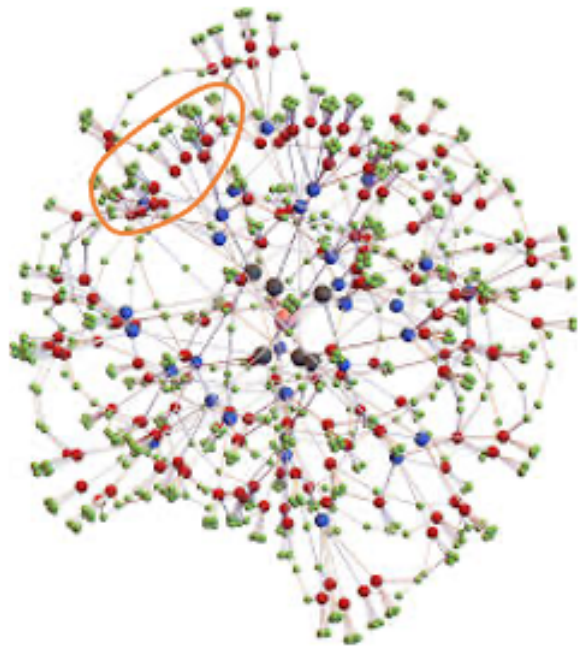
- Expanders!

Derandomization

Construct a d-regular α-expander G with $n = 2^m$ vertices[16].

Fix a subset $B \subseteq V$ with $|B| = \beta n$.
Take a random walk $v_1, \ldots, v_t$ from a random starting vertex.

$$\Pr[v_1 \in B \wedge \cdots \wedge v_t \in B] \leq (\alpha + \beta)^t$$



[16] Efficient representations have been designed to construct G "on the fly".

Derandomization (cont.)

If G is a good expander, we are *unlikely* to get stuck in B[17].

Probability amplification:

- Each vertex corresponds to a m bit string.
- When evaluating Π, we randomly pick a m bit string.

B is the set of m bit strings that output a wrong answer to Π.

$$|B| \leq \frac{1}{10}|n| \implies \beta \leq \frac{1}{10}$$

- Error probability $\leq (\alpha + \frac{1}{10})^t$.
- $m + t \log d$ random bits.

[17] Can also be viewed from the combinatorial definition of expanders.

Summary

Introduction

All Pairs Shortest Paths

Sparse Linear Algebra

Minimum Spanning Trees

Expanders

References

- *Graph Algorithms in the Language of Linear Algebra* by Kepner and Gilbert (2011)
- CS 554: Parallel Numerical Algorithms by Solomonik
- *New Connectivity and MSF Algorithms for Shuffle-Exchange Network and PRAM* by Awerbuch and Shiloach (1987)
- CS 574: Randomized Algorithms by Chan

Thanks. Questions?

Extra Slides

The following are cool related topics that I'd like to talk about at some point.

- Ball: given a vertex, compute its b closest neighbors.
- Probabilistic method: prove existence then derandomize for a construction.
- Dimensionality reduction: does there exist an embedding from ℓ_1 to Hamming space? Take a random projection and use the probabilistic method.
- Convolution as matrix multiplication: use Toepltiz matrices.
- Tensor decomposition/completion.