Graph Algorithms with Sparse Linear Algebra

Tim Baer

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

July 19, 2021

Overview

Motivation

Adjacency matrices[1] connect graph algorithms with linear algebra.

Designing graph algorithms?

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

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

- Leverage graph techniques[2].

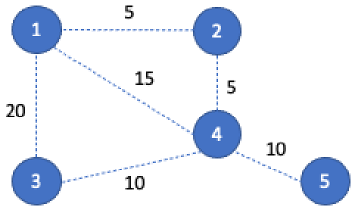
[1] And Laplacian matrices!

[2] 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[3] $\boldsymbol{A} \in \mathbb{R}^{n \times n}$ where

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



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

[3] 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[4]:

- Identity elements
- Associativity
- ...

[4]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:

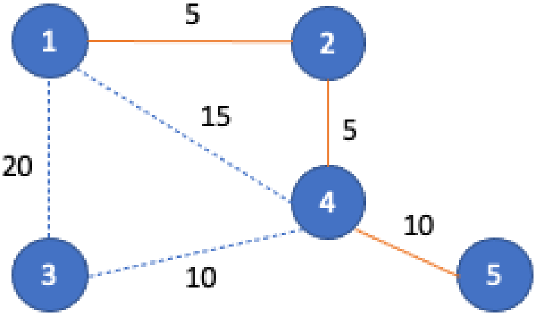
$$c_{ij} \leftarrow \sum_k a_{ik} b_{kj}$$

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.

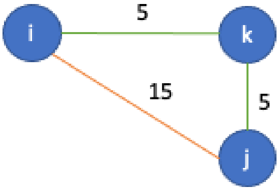


Sample application: driving directions, where road junctions are vertices and road distances are edge weights.

Bellman-Ford Algorithm

Fix a vertex i. Let $\mathrm{d}_{ij}^{(\ell)}$ 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:

$$d_{ij}^{(\ell+1)} \leftarrow d_{ij}^{(\ell)} \oplus \Big(\bigoplus_k a_{ik} \otimes d_{kj}^{(\ell)}\Big)$$

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

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

Here, $\boldsymbol{A}$ is the adjacency matrix so $a_{ik} = w(i,k)$.
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[5]
- Parallel and distributed
- With sparse data

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

Strassen's Algorithm

$$\begin{bmatrix} \boldsymbol{C}_{11} & \boldsymbol{C}_{12} \\ \boldsymbol{C}_{21} & \boldsymbol{C}_{22} \end{bmatrix} \leftarrow \begin{bmatrix} \boldsymbol{A}_{11} & \boldsymbol{A}_{12} \\ \boldsymbol{A}_{21} & \boldsymbol{A}_{22} \end{bmatrix} \begin{bmatrix} \boldsymbol{B}_{11} & \boldsymbol{B}_{12} \\ \boldsymbol{B}_{21} & \boldsymbol{B}_{22} \end{bmatrix}$$

$$\begin{aligned}
\boldsymbol{M}_1 &\leftarrow (\boldsymbol{A}_{11} + \boldsymbol{A}_{22}) \cdot (\boldsymbol{B}_{11} + \boldsymbol{B}_{22}) \\
\boldsymbol{M}_2 &\leftarrow (\boldsymbol{A}_{21} + \boldsymbol{A}_{22}) \cdot \boldsymbol{B}_{11} \\
\boldsymbol{M}_3 &\leftarrow \boldsymbol{A}_{11} \cdot (\boldsymbol{B}_{12} - \boldsymbol{B}_{22}) \\
\boldsymbol{M}_4 &\leftarrow \boldsymbol{A}_{22} \cdot (\boldsymbol{B}_{21} - \boldsymbol{B}_{11}) \\
\boldsymbol{M}_5 &\leftarrow (\boldsymbol{A}_{11} + \boldsymbol{A}_{12}) \cdot \boldsymbol{B}_{22} \\
\boldsymbol{M}_6 &\leftarrow (\boldsymbol{A}_{21} - \boldsymbol{A}_{11}) \cdot (\boldsymbol{B}_{11} + \boldsymbol{B}_{12}) \\
\boldsymbol{M}_7 &\leftarrow (\boldsymbol{A}_{12} - \boldsymbol{A}_{22}) \cdot (\boldsymbol{B}_{21} + \boldsymbol{B}_{22})
\end{aligned}$$

$$\begin{aligned}
\boldsymbol{C}_{11} &\leftarrow \boldsymbol{M}_1 + \boldsymbol{M}_4 - \boldsymbol{M}_5 + \boldsymbol{M}_7 \\
\boldsymbol{C}_{12} &\leftarrow \boldsymbol{M}_3 + \boldsymbol{M}_5 \\
\boldsymbol{C}_{21} &\leftarrow \boldsymbol{M}_2 + \boldsymbol{M}_4 \\
\boldsymbol{C}_{22} &\leftarrow \boldsymbol{M}_1 - \boldsymbol{M}_2 + \boldsymbol{M}_3 + \boldsymbol{M}_6
\end{aligned}$$

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[6].

[6] Using a cluster of commodity machines is much cheaper than building a large, specialized machine.

Parallel and Distributed Matrix Multiplication

Partition each matrix into blocks.

$$\begin{bmatrix} \boldsymbol{C}_{11} & \boldsymbol{C}_{12} & \boldsymbol{C}_{13} \\ \boldsymbol{C}_{21} & \boldsymbol{C}_{22} & \boldsymbol{C}_{23} \\ \boldsymbol{C}_{31} & \boldsymbol{C}_{32} & \boldsymbol{C}_{33} \end{bmatrix} \leftarrow \begin{bmatrix} \boldsymbol{A}_{11} & \boldsymbol{A}_{12} & \boldsymbol{A}_{13} \\ \boldsymbol{A}_{21} & \boldsymbol{A}_{22} & \boldsymbol{A}_{23} \\ \boldsymbol{A}_{31} & \boldsymbol{A}_{32} & \boldsymbol{A}_{33} \end{bmatrix} \begin{bmatrix} \boldsymbol{B}_{11} & \boldsymbol{B}_{12} & \boldsymbol{B}_{13} \\ \boldsymbol{B}_{21} & \boldsymbol{B}_{22} & \boldsymbol{B}_{23} \\ \boldsymbol{B}_{31} & \boldsymbol{B}_{32} & \boldsymbol{B}_{33} \end{bmatrix}$$

Organize processes into a 2D grid.

Process (r, s) owns $\boldsymbol{A}^{(r,s)}$ and $\boldsymbol{B}^{(r,s)}$ and contributes to all $\boldsymbol{C}^{(r,*)}$ and $\boldsymbol{C}^{(*,s)}$.

Process $(1, 1)$ must contribute to $\boldsymbol{C}^{11}, \underbrace{\boldsymbol{C}^{12}, \boldsymbol{C}^{13}, \boldsymbol{C}^{21}, \boldsymbol{C}^{31}}_{\text{on different processes}}$.

Can we *minimize communication*?

- What if we replicate certain blocks across certain processes?

3D Matrix Multiplication

Organize processes into a 3D cube.

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

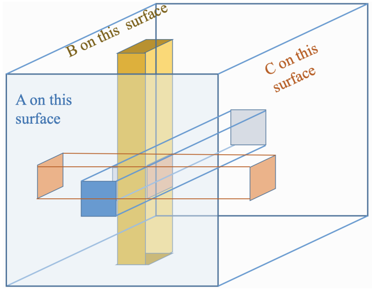


Figure from CS 484: Parallel Programming.

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[7] (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:	$0, 2, 3, 4, 6$

[7] 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[8].

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

Cyclops Tensor Framework

Matrix multiplications can be generalized to tensors[9]:

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

The Cyclops Tensor Framework[10] automatically parallelizes custom operations on tensors:

- Distributed memory
- Sparse tensors and operations
- Arbitrary semirings

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

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

[10] *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[11]

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

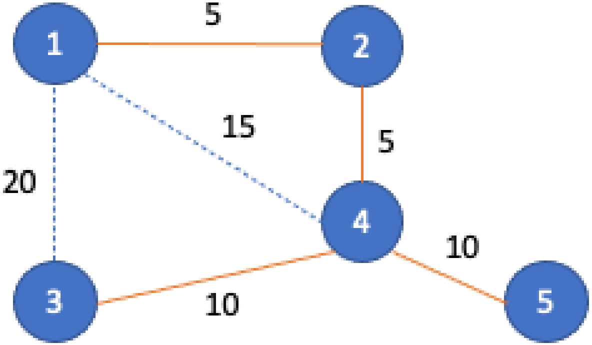
```
// first node
MPI_Send(&send_data, ...)

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

[11] 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.

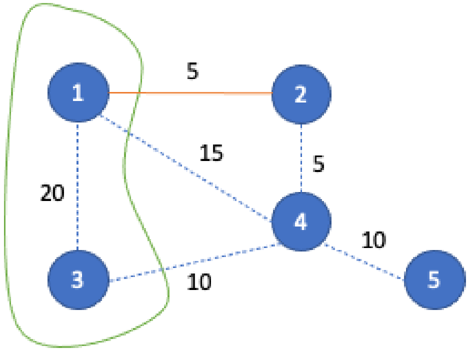


Sample application: a phone company laying cable in a neighborhood, where houses are vertices and cable lengths are edge weights.

Minimum Spanning Tree Algorithms

Most[12] 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.



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

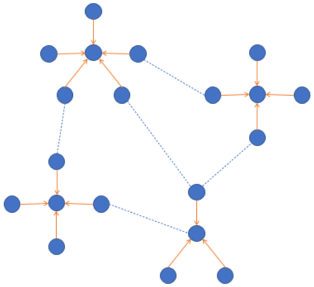
Awerbuch-Shiloach Parallel MST Algorithm

Maintain a forest of stars[13], where each vertex has a parent.

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

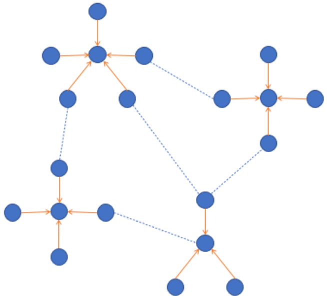
Repeat the following steps until convergence of the forest:

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



[13] A star is a directed tree of height at most 1 and is intuitively a part of the minimum spanning tree. A star root is a vertex that is its own parent.

Forest of Stars



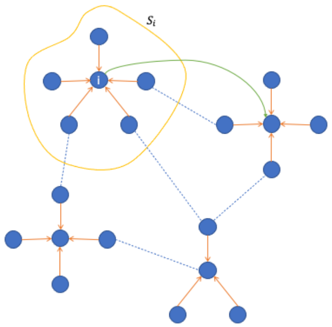
The parent vector $p \in \mathbb{N}^n$ stores the parent of each vertex.

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

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

Hooking

Each star root attempts[14] to hook with its *smallest cut edge*.



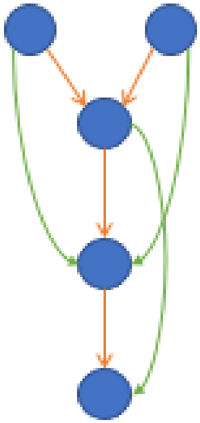
Hooking: joins two stars together.

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

[14]Need to break ties to prevent cycles.

Shortcut

Hooking may generate trees of arbitrary height.



Shortcutting[15]: reduces the height of each tree in the forest by a factor of nearly two. $p_i \leftarrow p_{p_i}$

[15] 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.

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

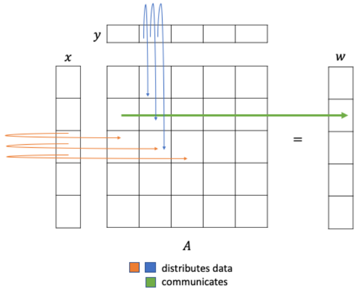
Here, $\boldsymbol{A}$ is the adjacency matrix so a_{ij}'s are edge weights.

Data Distribution

We update vertices by simultaneously using data from both adjacent edges and vertices. More generally,

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

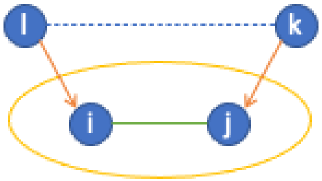
Process (r, s) owns $\boldsymbol{A}^{(r,s)}$ and needs $x^{(r)}$ and $y^{(s)}$. Process (r, s) contributes $w^{(r)}$ where $w^{(r)} \leftarrow f(x^{(r)}, \boldsymbol{A}^{(r,s)}, y^{(s)})$.



Supports sparse input matrix $\boldsymbol{A}$ and is communication-efficient.

Contraction

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



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

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

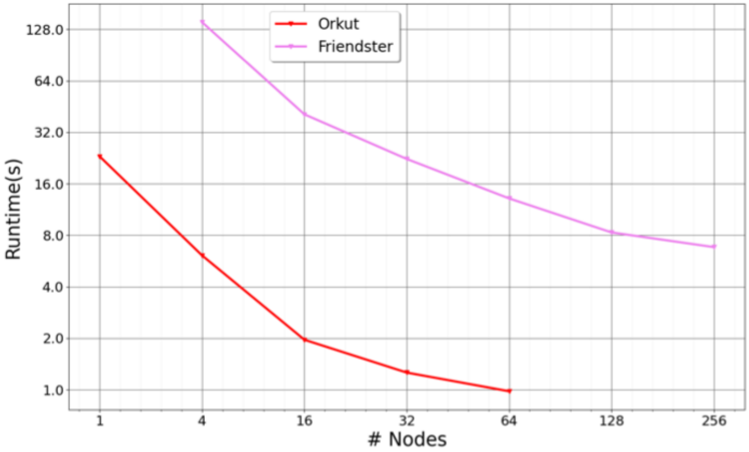
$w(l,k)$ is projected onto $w(i,j)$ if:

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

Here, $\boldsymbol{A}$ is again the adjacency matrix so $a_{lk} = w(l,k)$.

MST Strong Scaling Data

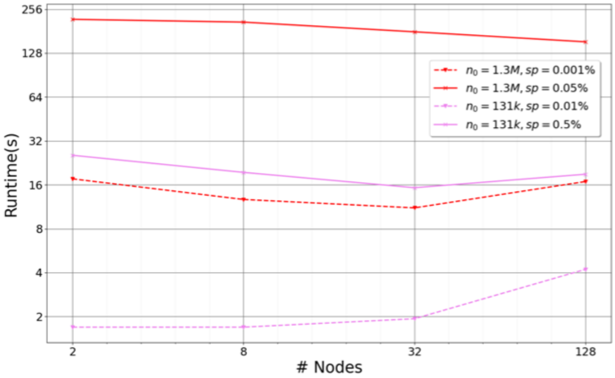
Strong scaling: fix the *total* problem size and vary the # of nodes.



(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.

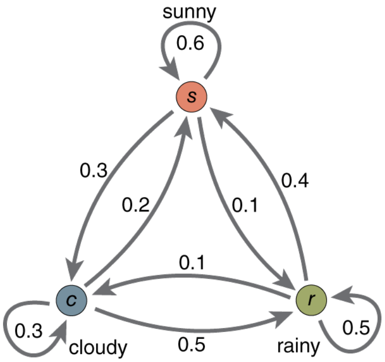


(a) Uniform random graph

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.

$$\begin{aligned} \pi^{(t)} &\leftarrow \pi^{(t-1)} \boldsymbol{P} \\ &\vdots \\ &= \pi^{(0)} \boldsymbol{P}^t \end{aligned}$$

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

The above algorithm is the same as power iteration[16]:

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

[16] Instead of normalization, enforce that $\sum_i \pi_i = 1$.

Expanders

A d-regular graph is an α-expander[17] 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 ($\alpha = \frac{1}{\sqrt{d}}$)

[17] $\alpha < 1$. 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
- ...

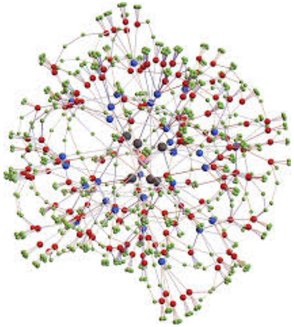


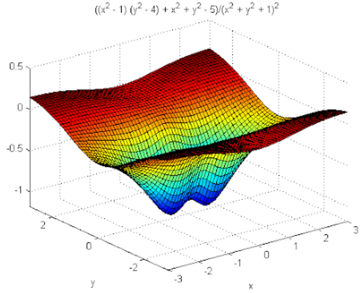
Figure from *Cutoff on All Ramanujan Graphs* by Lubetzky and Peres (2015).

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[18] $P(x_1, \ldots, x_k)$ 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.

[18]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?

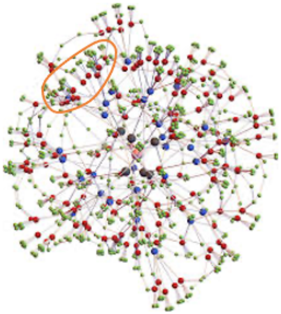
- Expanders!

Derandomization

Construct a d-regular α-expander G with $n = 2^m$ vertices[19].

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

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



[19]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[20].

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.

[20] 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

Fin

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: generalizations of SVD.
- Tensor completion: Netflix Prize.
- Hardware accelerators for matrix multiplications.