Learning Aggregates and Interpolation for Algebraic Multigrid

Nicolas Nytko[1] Matthew West[1] Luke Olson[1] Scott MacLachlan[2]

[1]University of Illinois Urbana-Champaign

[2]Memorial University of Newfoundland

April 6th, 2022

Introduction

- AMG methods can be fast, robust methods for solving sparse linear systems
- Convergence is dependent on choice of interpolation/restriction, $\mathbf{P}$ and $\mathbf{R}$
    - For example: smoothed aggregation depends on aggregation, candidate vectors, interpolation smoothing
- For class of problems, **can we learn this choice** of aggregation and smoothing operator?
- How? Create an ML *agent* composed of several graph neural networks that outputs aggregation and smoothing to form final interpolation.
- Focus on Isotropic and anisotropic diffusion problems in 2D

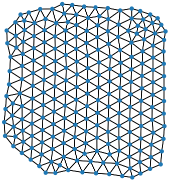
Diffusion Problem

- Consider 2D diffusion with homogeneous Dirichlet conditions,

$$-\nabla \cdot (\mathbf{D}\nabla \mathbf{u}) = f, \quad \mathbf{u}(\partial\Omega) = 0,$$

$$\mathbf{D} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} 1 & \\ & \varepsilon \end{bmatrix} \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}^T.$$

- P_1 finite elements; fixed rotation θ and fixed anisotropy ε in y-direction
- Obtain mesh through refining random convex hull



Dataset Generation

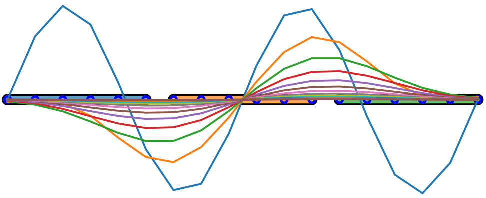
- Create two datasets of anisotropic and isotropic-only problems with testing and training split:
  - 1000 training problems
  - 250 testing problems
- Problems have between 50 and 500 degrees of freedom in resulting system.
- For isotropic, $\theta = 0$; $\varepsilon = 1$.
- For anisotropic, $\theta \sim \mathcal{U}\left(0, 2\pi\right)$; $\log \varepsilon \sim \mathcal{U}\left(-5, 5\right)$.

Training Loss

- Start from random guess and solve $\mathbf{A}\mathbf{x} = \mathbf{0}$ to some tolerance with 2-level multigrid solver.
- Approximate asymptotic convergence factor from error history of last several iterations.

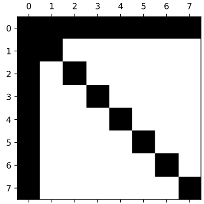
$$\mu := \left(\left\| \mathbf{e}^{(k)} \right\| - \left\| \mathbf{e}^{(k-n)} \right\| \right)^{1/(n-1)}$$

Take average μ over set of data — this creates unsupervised loss of *2-level V-cycle multigrid solver*

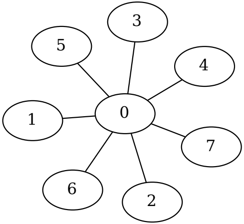


Graphnets in a Nutshell I

- Very helpful to think of sparse matrices as graphs.
- Let each column of matrix $\mathbf{A}$ define a node.
- Edge $j \to i$ exists if $\mathbf{A}_{ij} \neq 0$.



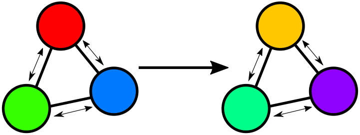
$$\Longleftrightarrow$$



Graphnets in a Nutshell II

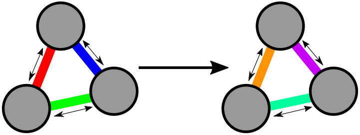
Define two useful graphnet operations:

Node convolution



$$(\mathbf{A}, \mathbf{x}) \to \mathbf{x}'$$

Edge convolution



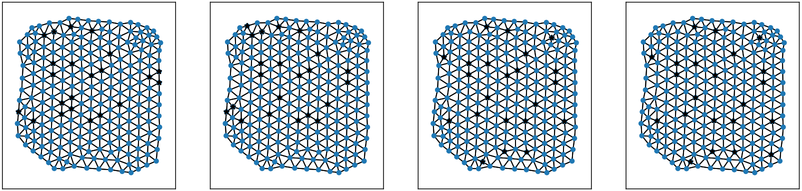
$$(\mathbf{A}, \mathbf{x}) \to \mathbf{A}' \quad ^{(*)}$$

(*) the sparsity pattern of $\mathbf{A}$ is preserved.

Will be using TAGConv, MPNN for node convolutions.

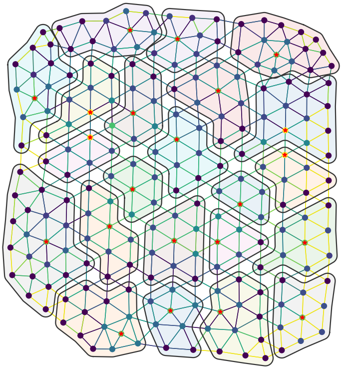
Learning Aggregation I

- Want to form coarse-grid through *aggregation*
- Define two networks, Θ_{agg}, Θ_{soc}, for outputting *aggregate roots* and *strength of connection*.
- Let $k := \lceil \alpha n \rceil$ be number of nodes on coarse grid, for coarsening factor α.
- For Θ_{agg}, run node convolutions then replace k largest value nodes with 1, rest 0. Repeat.
- Output of Θ_{agg} are roots.



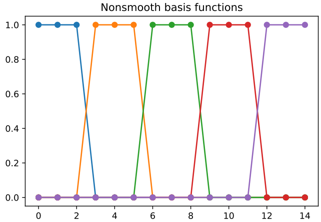
Learning Aggregation II

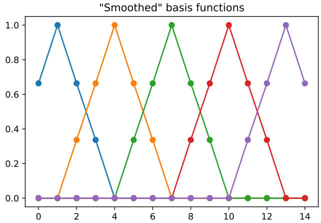
- Let Θ_{soc} be a series of edge convolutions that map $\mathbf{A}$ to some strength matrix $\mathbf{C}$:
    - Connection matrix $\mathbf{C}$ has same edge connections as $\mathbf{A}$.
- Run Bellman-Ford with roots and $\mathbf{C}$ to assign each node to *nearest* root – every node now uniquely assigned to aggregate.
- Obtain the binary aggregate assignment matrix, Agg $\in \mathbb{R}^{n \times k}$; $\text{Agg}_{ij} = 1$ means node i belongs to aggregate j.



How do we learn the interpolator $\mathbf{P}$ given Agg?

- Add additional binary edge feature for edges that connect nodes between different aggregates.
- Let Θ_S be network that maps $\mathbf{A}$ and inter-aggregate feature to smoother, $\hat{\mathbf{S}}$.
- Form $\mathbf{P} = \hat{\mathbf{S}}\text{Agg}$ — *smoothes* columns of Agg. (Or, use Agg to take linear combinations of $\hat{\mathbf{S}}$.)





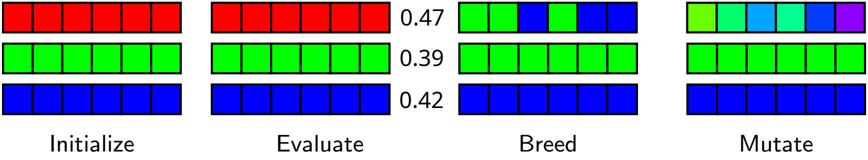
Full ML Agent Overview

- Overall, *Agent* composed of three networks: $\Theta_{\text{agg}}, \Theta_{\text{soc}}, \Theta_S$.
- Use Θ_{agg} to get *aggregate roots*, Θ_{soc} to get $\mathbf{C}$.
- Run graph traversal on roots and $\mathbf{C}$ to get assignment matrix, $\text{Agg} \in \mathbb{R}^{n \times k}$.
- Use Θ_{S} to output *smoother*, $\hat{\mathbf{S}}$.
- Finally, define interpolation operator as $\mathbf{P} := \hat{\mathbf{S}}\text{Agg}$.
- ML agent is some function of $\mathbf{A}$ and coarsening ratio α, outputs $\mathbf{P}$.

Training

- Gradient information is difficult to pass through full agent. (Networks are deep, aggregate selection is inherently *non-differentiable*.)
- Turn to *genetic evolution strategies* for training.

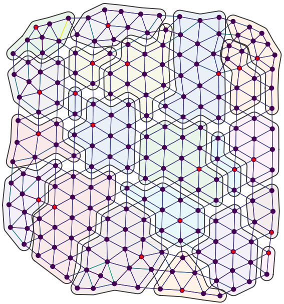
Use following steps to train agent:



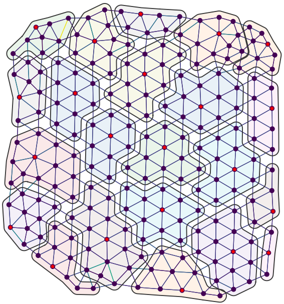
Repeating steps (2)-(4) converges population towards an optimally trained agent without any gradient information.

Baseline, Lloyd Aggregation

- Compare against baseline of randomly selected aggregate roots and aggregates refined using *Lloyd aggregation*.
- Lloyd aggregation attempts to uniformly space aggregates according to some strength measure; iteratively re-centers seeds.
- Run Bellman-Ford afterwards as in ML to assign nodes to aggregates. Jacobi smoother on columns of aggregate matrix.



Random Seeds



Refinement with Lloyd Aggregation

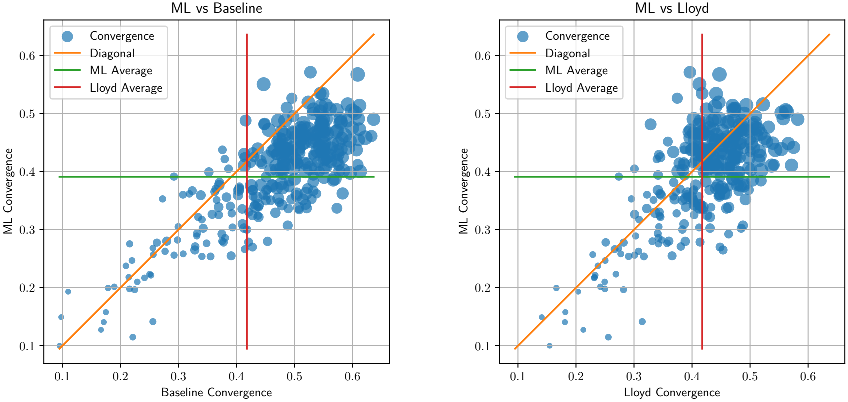
Overall Results

Trained one agent on *both* anisotropic, isotropic problems; $\alpha = 0.1$.

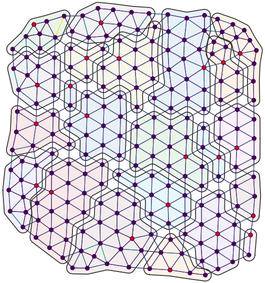
Convergence factors (lower better) of ML vs baseline, Lloyd on both sets of problems. ML is competitive with Lloyd on reducing convergence compared to baseline.

Problem Type	Data Set	Baseline	Lloyd	ML
Isotropic	Train	0.47	0.42	0.40
Isotropic	Test	0.46	0.42	0.39
Anisotropic	Train	0.77	0.77	0.75
Anisotropic	Test	0.79	0.80	0.77

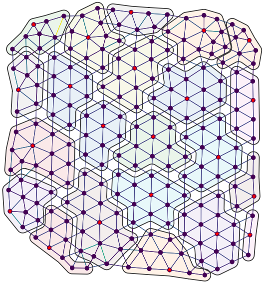
Isotropic Results I



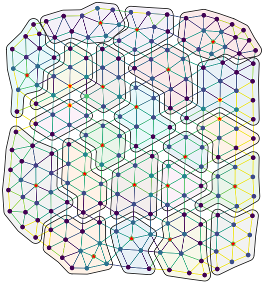
Lloyd Aggregation gives (roughly) evenly-spaced aggregates. ML learns to give larger aggregates at boundary and higher resolution in middle of mesh.



Baseline, conv=0.5420

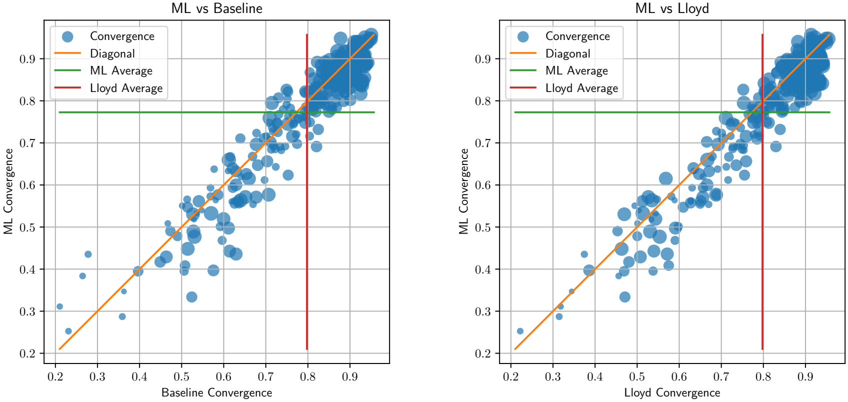


Lloyd, conv=0.4751



ML, conv=0.3709

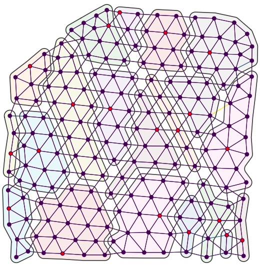
Anisotropic Results I



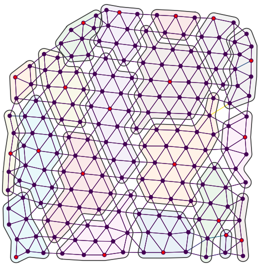
Anisotropic Results II

Lloyd produces clusters that roughly follow direction of strong anisotropy. ML does same, though aggregates become interestingly more "normal" in center of mesh.

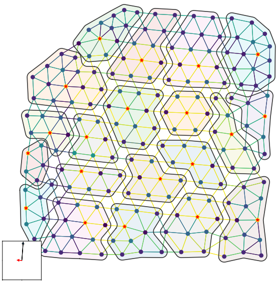
$$\theta = 0.48\pi \quad \varepsilon = 0.1771$$



Baseline, conv$=0.6334$



Lloyd, conv$=0.6936$



ML, conv$=0.5041$

Conclusions

- Graph networks can be used to learn aggregation and interpolation in SA.
- Gradient-free genetic algorithms actually provide good results for training networks.
- Compared to baseline results, ML method offers improvements comparable to (and actually slightly better than) Lloyd.
- Don't need to select strength-of-connection for SA, method can learn something decent.

Future Directions

- Try out some more difficult problems
  - Look at problems at which conventional AMG does not do so well?
- Are we able to reformulate this so that gradient information is available and is trainable with traditional descent methods?
  - (Re-cast as reinforcement learning problem, maybe?)
- More flexibility to ML agent to pick aggregate and final interpolation.
  - Agent is limited to selecting aggregate centers and smoother. By replacing more parts of SA with ML components can we learn deeper or more optimal interpolation?