Learning Algebraic Multigrid using Graph Neural Networks

Ilay Luz, Meirav Galun, Haggai Maron,
Ronen Basri, Irad Yavneh

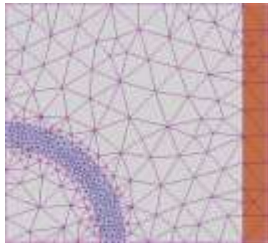
מכון ויצמן למדע
WEIZMANN INSTITUTE OF SCIENCE

NVIDIA

TECHNION
Israel Institute
of Technology

Goal: Large scale linear systems

- Solve $Ax = b$
- A is huge, need $O(n)$ solution!
- Some applications:
  - Discretization of PDEs



$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = f(x, y)$$

  - Sparse graph analysis

Efficient linear solvers

- Decades of research on efficient iterative solvers for large-scale systems

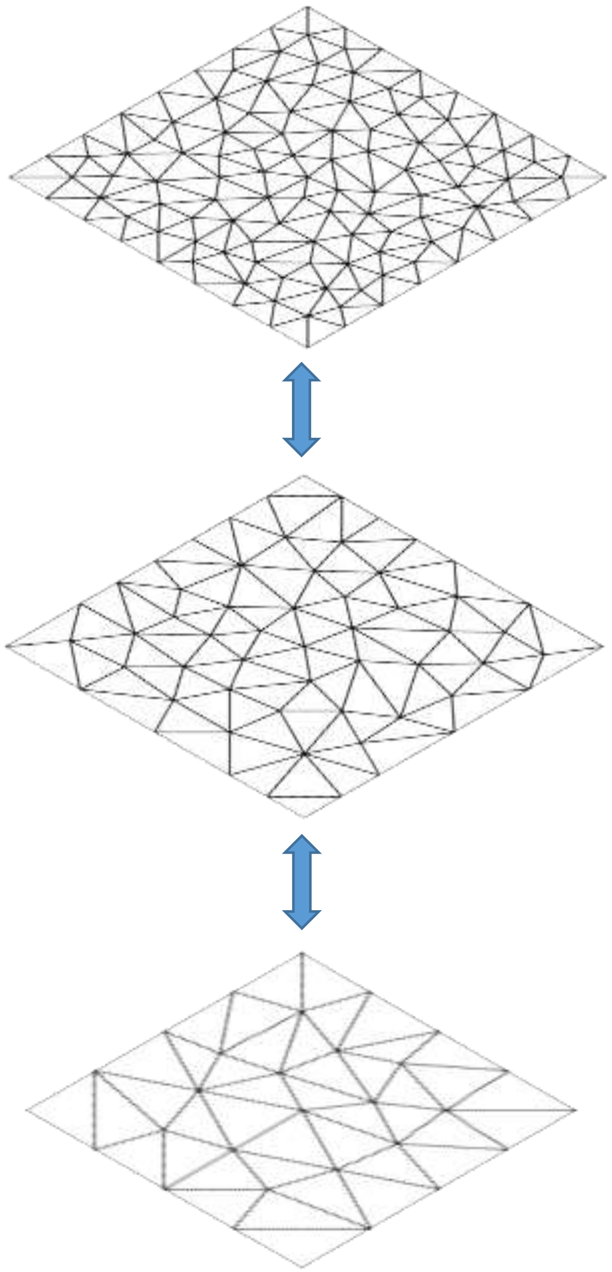
- We focus on Algebraic Multigrid (AMG) solvers

- Can we use machine learning to improve AMG solvers?

- Follow-up to Greenfeld et al. (2019) on *Geometric* Multigrid

What AMG does

- AMG works by successively **coarsening** the system of equations, and solving on multiple scales
- **Prolongation** operator P that creates the hierarchy
- We want to learn a mapping $P_\theta(A)$ with **fast convergence**



Learning P

- Unsupervised loss function over distribution $\mathcal{D}$:

$$\min_{\theta} \mathbb{E}_{A\sim\mathcal{D}}\, \rho\left(M\left(A, P_\theta(A)\right)\right)$$

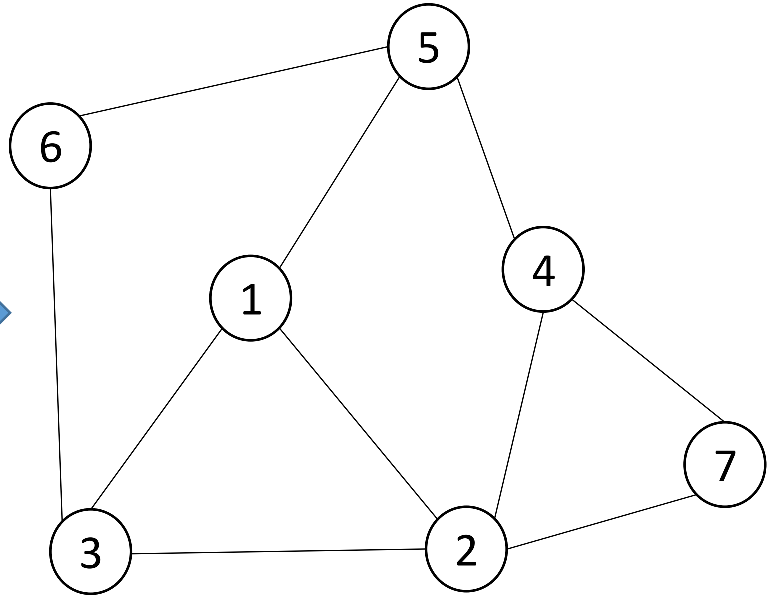
- $\rho\left(M\left(A, P_\theta(A)\right)\right)$ measures the convergence factor of the solver

- $P_\theta(A)$ is a NN mapping system A to prolongation operator P

Graph neural network

- Sparse matrices can be represented as graphs – we use a **Graph Neural Network** as the mapping $P_\theta(A)$

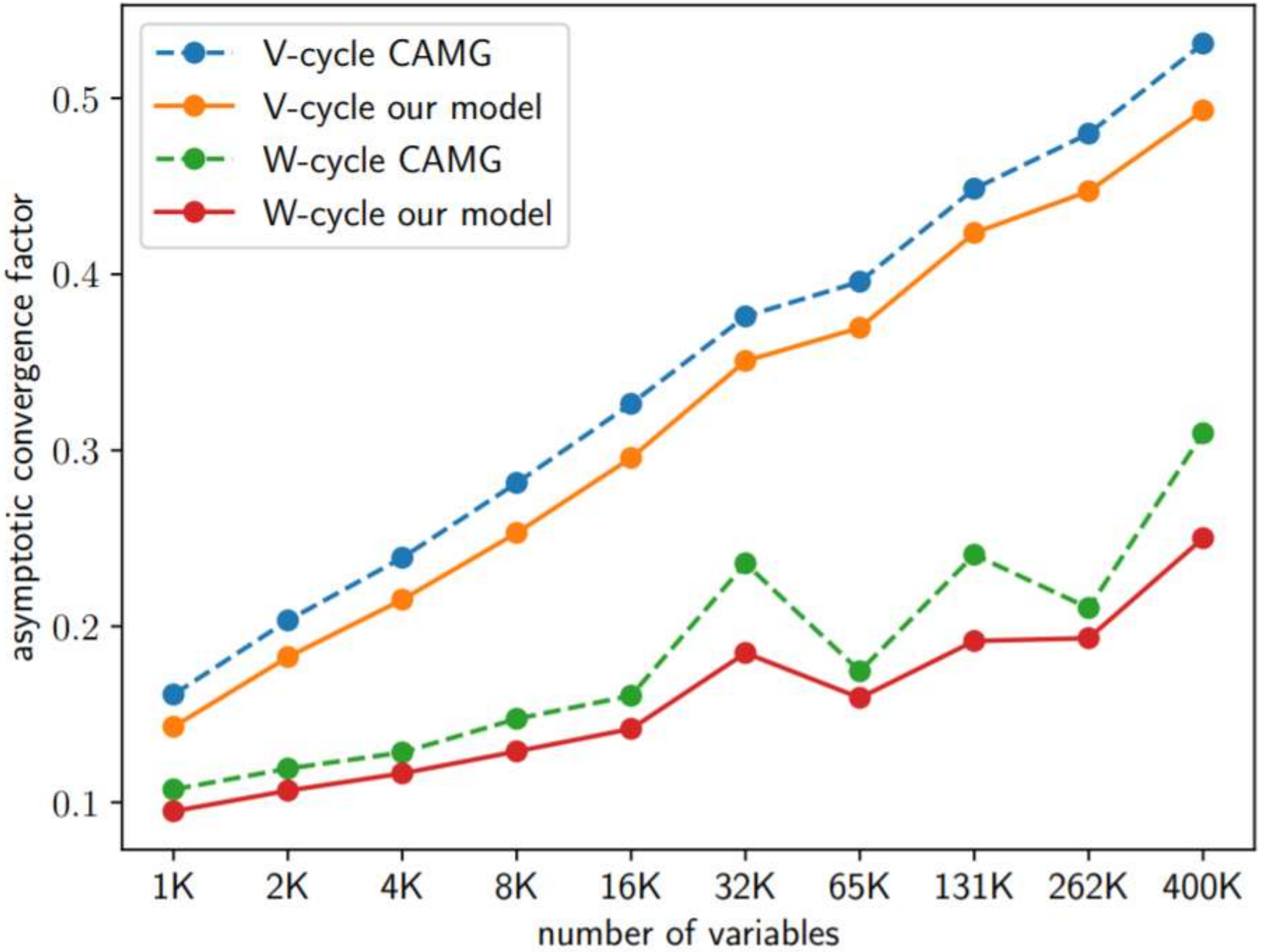
$$\begin{pmatrix} 2.7 & -0.5 & -0.5 & 0 & -1.7 & 0 & 0 \\ -0.5 & 7.7 & -4.9 & -0.6 & 0 & 0 & -1.7 \\ -0.5 & -4.9 & 6.2 & 0 & 0 & -0.8 & 0 \\ 0 & -0.6 & 0 & 2.9 & -0.6 & 0 & -1.7 \\ -1.7 & 0 & 0 & -0.6 & 13.1 & -10.8 & 0 \\ 0 & 0 & -0.8 & 0 & -10.8 & 11.6 & 0 \\ 0 & -1.7 & 0 & -1.7 & 0 & 0 & 3.4 \end{pmatrix}$$



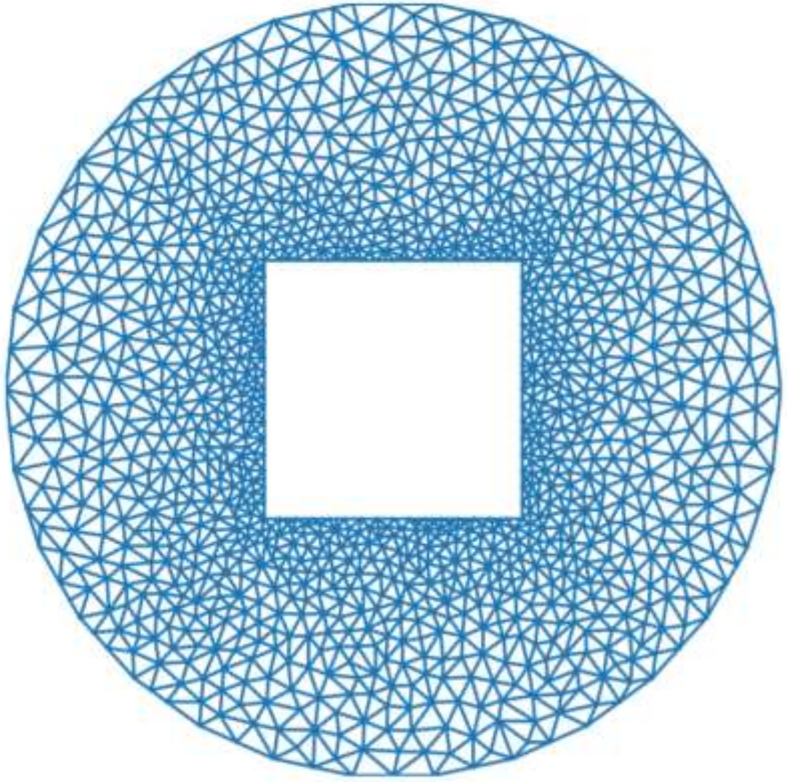
Benefits of our approach

- **Unsupervised training**– rely on algebraic properties
- **Generalization** –learn general *rules* for wide class of problems
- **Efficient training** – Fourier analysis reduces computational burden

Sample result; lower is better, ours is lower!



Finite Element PDE



Outline

- Overview of AMG
- Learning objective
- Graph neural network
- Results

1st ingredient of AMG: Relaxation

- System of equations: $a_{i1}x_1 + a_{i2}x_2 + \cdots a_{in}x_n = b_i$

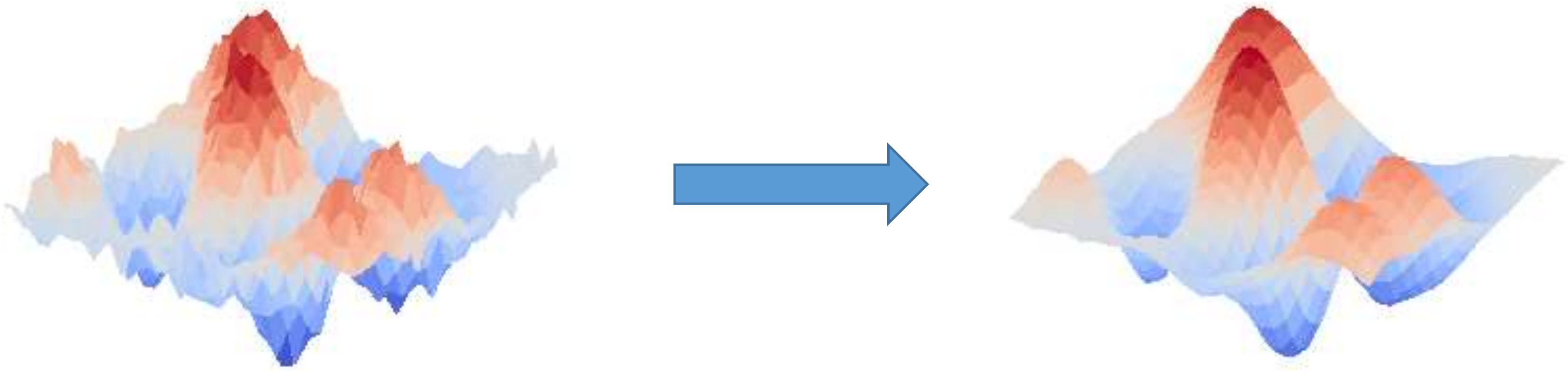
- Rearrange: $x_i = \frac{1}{a_{ii}}\left(b_i - \sum_{j \neq i} a_{ij}x_j\right)$

- Start with an **initial guess** $x_i^{(0)}$

- **Iterate** until convergence: $x_i^{(k+1)} = \frac{1}{a_{ii}}\left(b_i - \sum_{j \neq i} a_{ij}x_j^{(k)}\right)$

Relaxation smooths the error

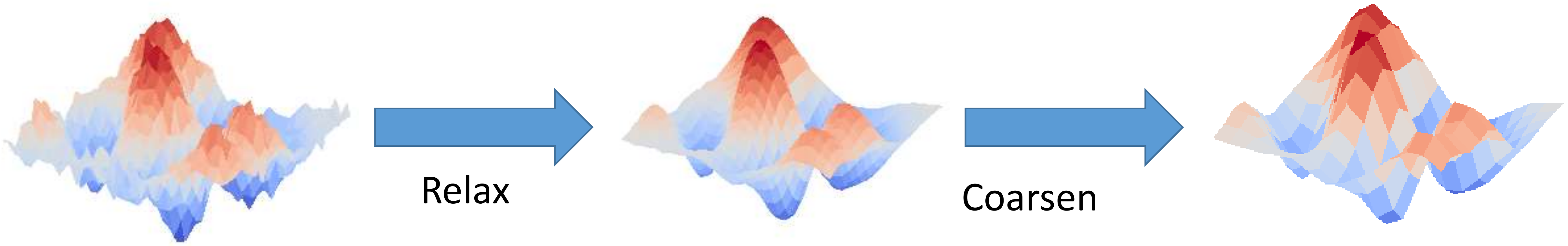
- Since relaxation is a local procedure, its effect is to **smooth out** the error



- How to accelerate relaxation by dealing with low-frequency errors?

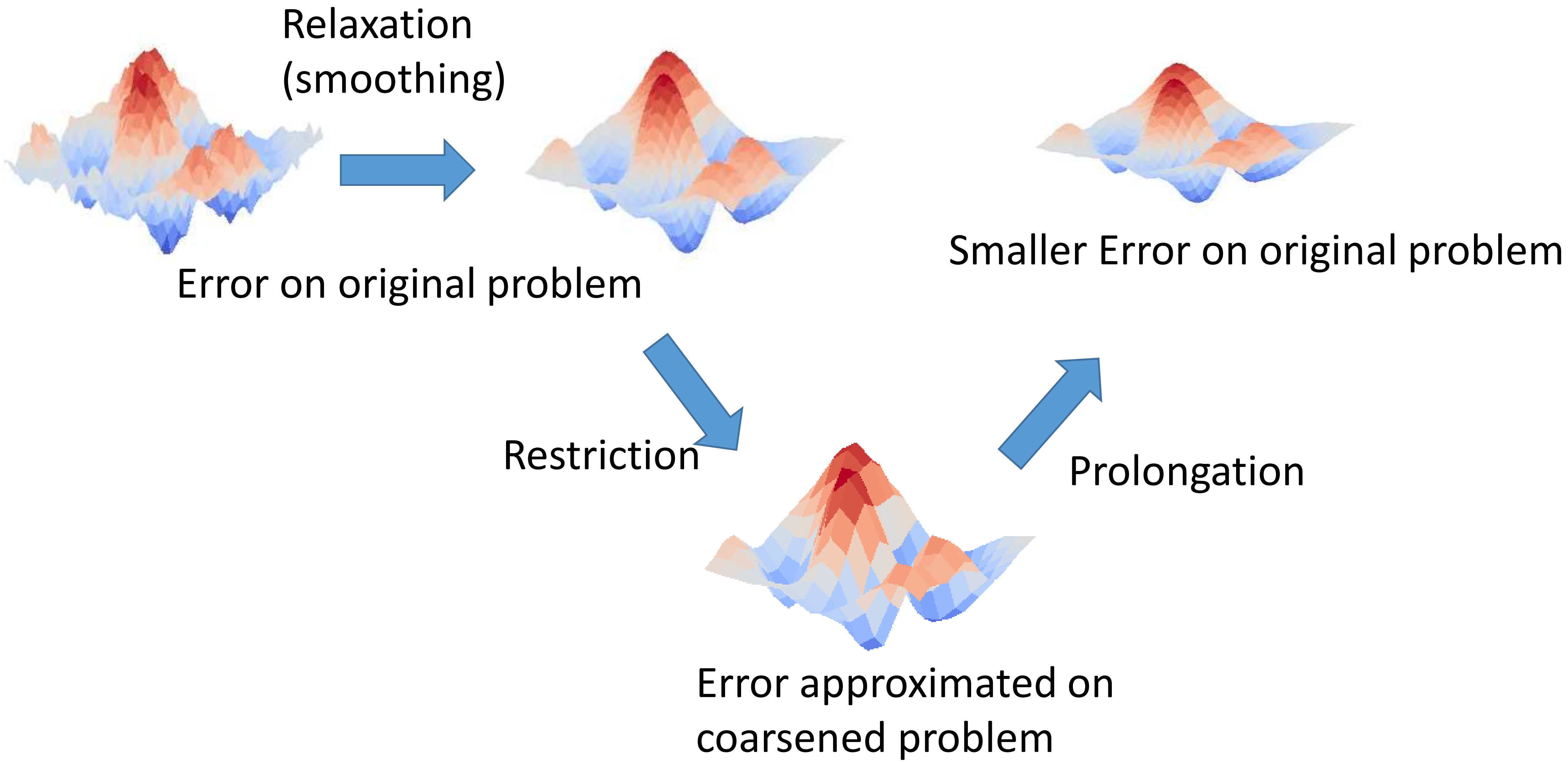
2nd ingredient of AMG: Coarsening

- Smooth error, and then **coarsen**



- Error is no longer smooth on coarse grid; relaxation is fast again!

Putting it all together

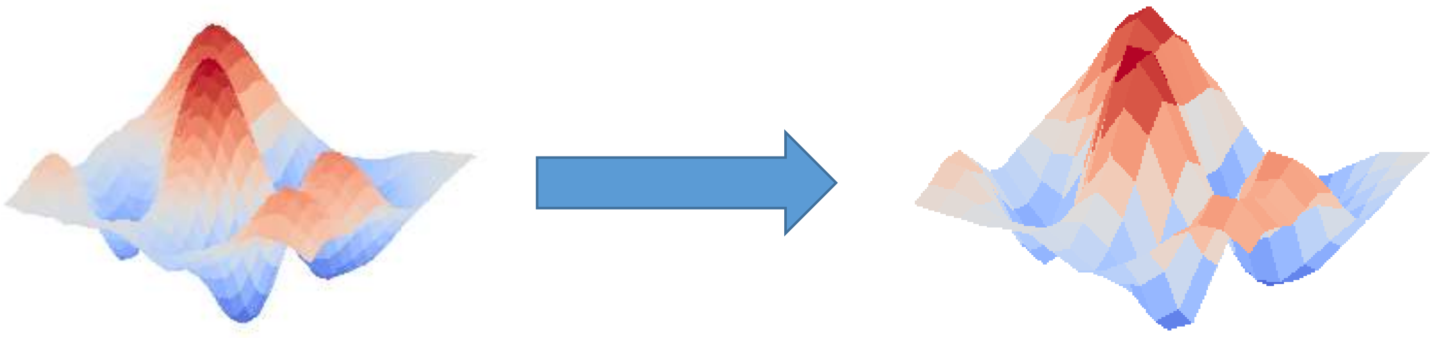


Learning objective

Prolongation operator

- Focus of AMG is prolongation operator P for defining scales and moving between them

- P needs to be sparse for efficiency, but also approximate well smooth errors



Learning P

- Quality can be quantified by estimating by how much the error is reduced each iteration:
  - $e^{(k+1)} = M(A,P)e^{(k)}$
  - $M(A,P) = S(I - P[P^T AP]^{-1}P^T A)S$
  - Asymptotically: $\|e^{(k+1)}\| \approx \rho(M)\|e^{(k)}\|$
  - Spectral radius: $\rho(M) = \max\{|\lambda_1|, \dots, |\lambda_n|\}$

- Our learning objective:

$$\min_\theta \mathbb{E}_{A\sim\mathcal{D}}\, \rho\left(M\left(A, P_\theta(A)\right)\right)$$

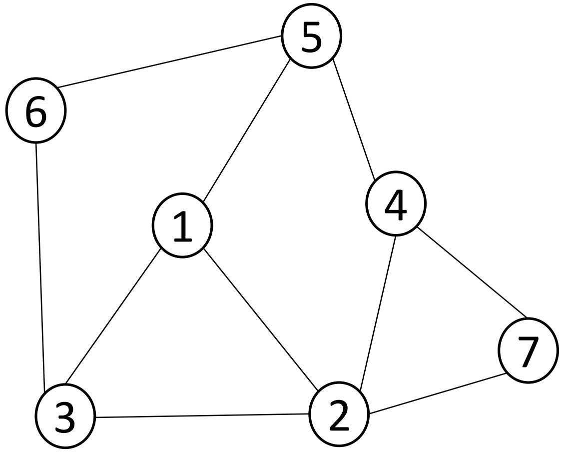
Graph neural network

Representing P_θ

- Sparse matrix $A \in \mathbb{R}^{n \times n}$ to sparse matrix $P \in \mathbb{R}^{n \times n_c}$
- Mapping should be **efficient**
- Matrices can be represented as graphs with edge weights

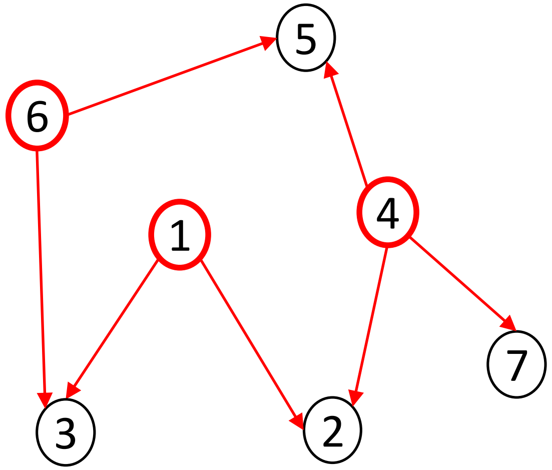
Representing P_θ

$$\begin{pmatrix} 2.7 & -0.5 & -0.5 & 0 & -1.7 & 0 & 0 \\ -0.5 & 7.7 & -4.9 & -0.6 & 0 & 0 & -1.7 \\ -0.5 & -4.9 & 6.2 & 0 & 0 & -0.8 & 0 \\ 0 & -0.6 & 0 & 2.9 & -0.6 & 0 & -1.7 \\ -1.7 & 0 & 0 & -0.6 & 13.1 & -10.8 & 0 \\ 0 & 0 & -0.8 & 0 & -10.8 & 11.6 & 0 \\ 0 & -1.7 & 0 & -1.7 & 0 & 0 & 3.4 \end{pmatrix}$$



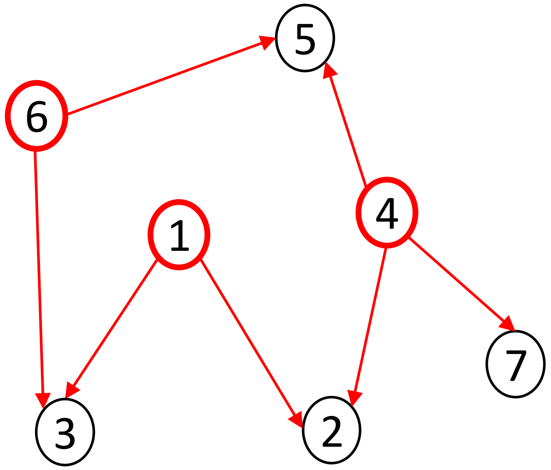
Input A

	1	4	6
1			
2			
3			
4			
5			
6			
7			



Sparsity pattern

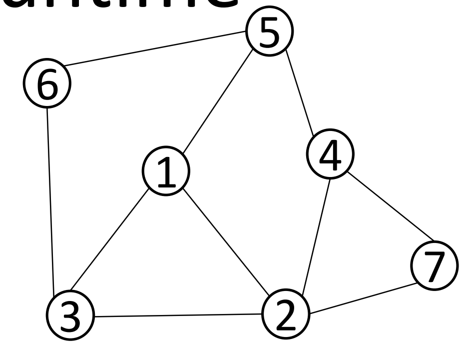
$$\begin{pmatrix} 1 & 0 & 0 \\ 0.3 & 0.7 & 0 \\ 0.9 & 0 & 0.1 \\ 0 & 1 & 0 \\ 0 & 0.2 & 0.8 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix}$$

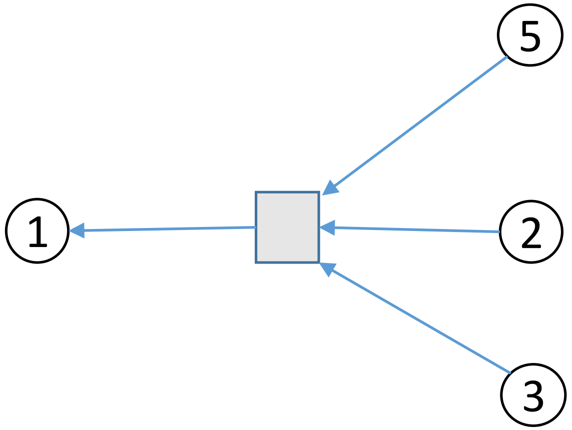


Output P

GNN architecture

- Message Passing architectures can handle any graph, and have $O(n)$ runtime





- Graph Nets framework from Battaglia et al. (2018) generalize many MP variants, handle edge features

Results

Spectral clustering

- Bottleneck is an iterative eigenvector algorithm that uses a linear solver

- Evaluate number of iterations required to reach convergence

- Train network on dataset of small 2D clusters, test on various 2D and 3D distributions

Conclusion

- Algebraic Multigrid is an **effective $\boldsymbol{O(n)}$ solver** for a wide class of linear systems $Ax = b$
- Main challenge in AMG is constructing **prolongation operator $\boldsymbol{P}$**, which controls how information is passed between grids
- We use an $O(n)$, **edge-based GNN** to learn a mapping $P_\theta(A)$, without supervision
- **GNN generalizes** to larger problems, with different distributions of sparsity pattern and elements

Take home messages

- In a well-developed field, might make sense to **apply ML to a part of the algorithm**

- Graph neural networks can be an effective tool for **learning sparse linear systems**