1 Background

Currently, we are attempting to learn an interpolation operator $\boldsymbol{P}$ given a pre-existing aggregation of some system $\boldsymbol{A}$. The graphnet being used is a combination of *node convolution* and *edge convolution* layers. The main distinction between the two is that the node convolution accepts some nodal values $\boldsymbol{v}$, and outputs a new set of nodal values $\boldsymbol{v}'$, while the edge convolution accepts edge attributes $\boldsymbol{e}$ and outputs $\boldsymbol{e}'$. We can think of this as modifying the values (but not the structure) of the input graph at each layer.

Because of this, the graphnet takes as input some graph $\boldsymbol{G} = (\boldsymbol{v}, \boldsymbol{e})$ and outputs a new graph $\boldsymbol{G}' = (\boldsymbol{v}', \boldsymbol{e}')$. The graphs $\boldsymbol{G}$ and $\boldsymbol{G}'$ have the same underlying structure and node connectivity, but may contain differing nodal and edge weights (including a different number of attributes per node/edge).

We can extend the above to work analogously with sparse matrices. If we have some sparse matrix $\boldsymbol{A}$, we can create the graph $\boldsymbol{G_A}$ by outputting one node per row/column and assigning edges based on nonzero entries of $\boldsymbol{A}_{ij}$. Therefore, if we were to generate the graph $\boldsymbol{G_A}$ based on $\boldsymbol{A}$, we will get some output graph $\boldsymbol{G}'_{\boldsymbol{A}}$ that can be transformed back into a sparse matrix $\boldsymbol{A}'$. Because the graph structure is unchanged, we must have that $\boldsymbol{A}'$ contains the same sparsity as $\boldsymbol{A}$, though the nonzero entries may be different.

This detail prevents us from inputting some matrix $\boldsymbol{A}$ to the graphnet and directly obtaining a $\boldsymbol{P}$ back; we are constrained by the sparsity of the original $\boldsymbol{A}$. If we convolve on some matrix containing the sparsity of $\boldsymbol{P}$ instead, we may lose information that was present in $\boldsymbol{A}$. In section 2, we will describe the network, inputs, and outputs in more detail. In section 3, we will explore how given some $\boldsymbol{A}'$ from the output of the graphnet, we can transform it into a suitable $\boldsymbol{P}$ that can be used in an AMG method.

2 Network Details

The matrix $\boldsymbol{A}$ is first turned into a graph $\boldsymbol{G_A}$ by the description above; each row/column is converted into a node n_i and non-zero entries of $|\boldsymbol{A}_{ij}|$ are converted into edges between nodes i and j. We then add an additional binary edge attribute θ_{ij}, given by

$$\theta_{ij} = \begin{cases} 0 & \text{nodes } i, j \text{ in same aggregate} \\ 1 & \text{otherwise} \end{cases}. \tag{1}$$

This means that each edge has two attributes: $|\boldsymbol{A}_{ij}|$ and the binary feature from above. The feature θ_{ij} is how the graphnet is "aware" of the aggregate information.

The network is defined as three message-passing node-convolution layers, followed by one edge-convolution layer. This takes as input a graph $\boldsymbol{G} = (\boldsymbol{v}, \boldsymbol{e})$ and gives a new graph $\boldsymbol{G}' = (\boldsymbol{v}', \boldsymbol{e}')$ with the same connectivity information. Of course, we are only interested in the edge values; the nodal values are a by-product used in intermediate convolutions. So, given a sparse matrix $\boldsymbol{A}$, the graphnet will output a new sparse matrix $\boldsymbol{A}'$ with the same sparsity pattern.

3 Aggregation-Based Average Pooling

This simple approach works by "collapsing" columns of nodes in similar aggregates, forming a $\boldsymbol{P}$ that contains a reasonable sparsity pattern for aggregation-based AMG. This section will be illustrated by a small sample graph, as shown in figure 1.

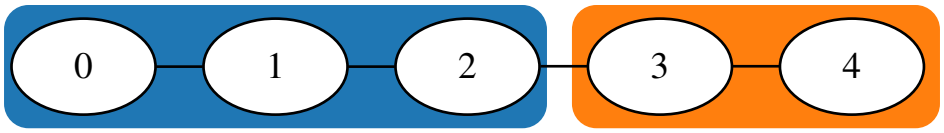

Figure 1: Example graph. The graph is partitioned into two aggregates.

Denote Agg the aggregation operator without smoothing, for example,

$$Agg = \begin{bmatrix} 1 & \\ 1 & \\ 1 & \\ & 1 \\ & 1 \end{bmatrix}, \tag{2}$$

would correspond to the above 5×5 system with two aggregates, the first containing nodes $0 - 2$ and the second containing nodes 3 and 4. Let $\boldsymbol{A}$ be the system being solved and $\boldsymbol{P}$ the interpolation operator, i.e. that obtained by running some relaxation scheme on Agg.

One way to solve this is to first form $\hat{\boldsymbol{P}}$ as usual using the graphnet's node and edge convolutions. This will give us a matrix that has the same sparsity pattern as $\boldsymbol{A}$. Continuing the example from above, we may obtain some $\hat{\boldsymbol{P}}$ like

$$\hat{\boldsymbol{P}} = \left[\begin{array}{ccc|ccc} 1/2 & 1/3 & & & \\ 1/2 & 1/3 & 1/3 & & \\ & 1/3 & 1/3 & 1/3 & \\ & & 1/3 & 1/3 & 1/2 \\ & & & 1/3 & 1/2 \end{array}\right], \tag{3}$$

with nodes to the left of the bar signifying those belonging to the first aggregate, and those to the right belonging to the second aggregate.

Then, we form $\hat{Agg}$, which is the Agg operator from above except with its columns normalized in the 1-norm.

$$\hat{Agg} = \begin{bmatrix} 1/3 & \\ 1/3 & \\ 1/3 & \\ & 1/2 \\ & 1/2 \end{bmatrix}. \tag{4}$$

Now, we can "collapse" the columns of the nodes corresponding to each aggregate by averaging them together. This can be written concretely as

$$\boldsymbol{P} = \hat{\boldsymbol{P}}\hat{Agg} = \begin{bmatrix} 0.277 & \\ 0.333 & \\ 0.222 & 0.166 \\ 0.111 & 0.416 \\ & 0.416 \end{bmatrix}, \tag{5}$$

which gives a reasonable sparsity pattern for the interpolation, given Agg. Note that if we view this as a graph, we get a 1-ring of edge connections around each aggregate. A somewhat hand-wavy proof is given below.

Proof. Assume that the entries of $\hat{\boldsymbol{P}}$ are strictly positive, which is reasonable because of the activation being used. We can view the nonzero entries $\hat{p}_{ij}$ as edge connections from node j to i.

By averaging the columns $\hat{\boldsymbol{p}}_{i_1}, \hat{\boldsymbol{p}}_{i_2}, \ldots, \hat{\boldsymbol{p}}_{i_n}$ of an aggregate composed of nodes $i_1, i_2, \ldots, i_n$, we are obtaining the union of the neighborhood of each node in the aggregate, because all entries are positive. Thus, an aggregate has connections to each node it is composed of, as well as the union of the neighbors of each node. □

4 Full Algorithm

Now that the average pooling technique has been introduced, the full algorithm for generating an interpolation operator $\boldsymbol{P}$ is defined below.

1. Given some sparse matrix $\boldsymbol{A}$, generate the graph $\boldsymbol{G_A}$.
2. Create aggregates using some established clustering algorithm, such as Lloyd. Store this as Agg and form $\hat{Agg}$ by normalizing the columns.
3. Create the binary inter-aggregate edge features θ_{ij} and append this to the edges of $\boldsymbol{G_A}$.
4. Pass $\boldsymbol{G_A}$ to the graphnet and obtain back $\boldsymbol{G'_A}$, which is a graph containing the same number of nodes and edges, but each having different weights.
5. Convert $\boldsymbol{G'_A}$ back into a matrix, we will denote this as $\hat{\boldsymbol{P}}$.
6. Finally, create $\boldsymbol{P} = \hat{\boldsymbol{P}}\hat{Agg}$. This is our final interpolation operator.

5 Problem Setup

The specific problem being solved is a diffusion problem with Neumann boundary conditions. To test the feasibility of the method, results were fixed for one problem and the networks/methods allowed to *overfit* to see if some reasonable solution is obtained. Aggregates were generated with Lloyd aggregation, using a seed ratio of 0.25. This gives a relatively high number of small aggregates, which may be somewhat unwanted in practice.

6 Numerical Results

Using Lloyd to generate the aggregates and then generating $\boldsymbol{P}$ with a Jacobi prolongation smoother results in an AMG method with a convergence factor of approx. 0.6874, as measured by the loss function. This will be used as a baseline to compare the following results.

All of the following plots are using the same random seeds, meaning both Lloyd aggregates and test vectors for the AMG loss should remain fixed for each training iteration.

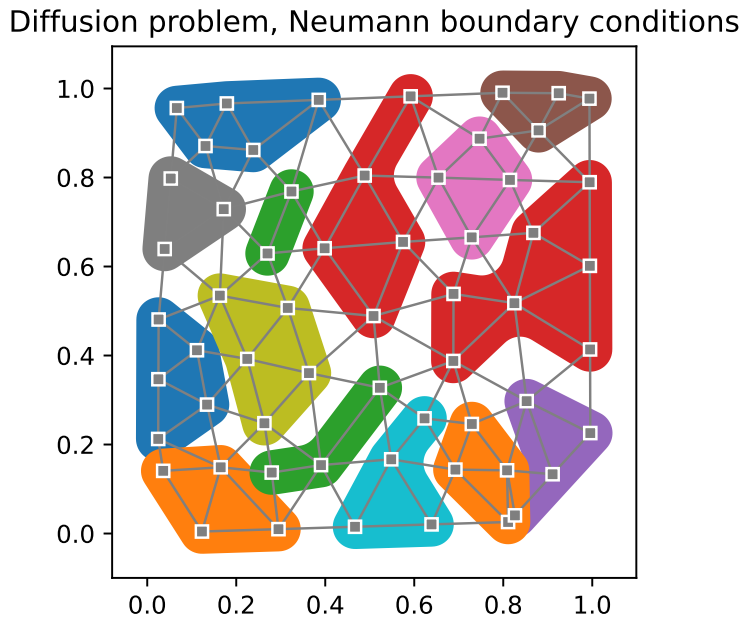

Figure 2: Specific problem being solved, with aggregate information.

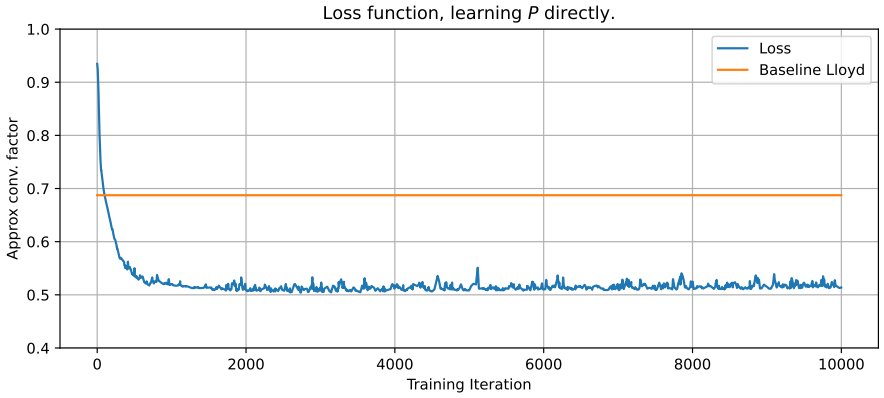

Figure 3: Loss function when directly optimizing the nonzero entries of $\boldsymbol{P}$. Using Adam optimizer with a learning rate of 0.01.

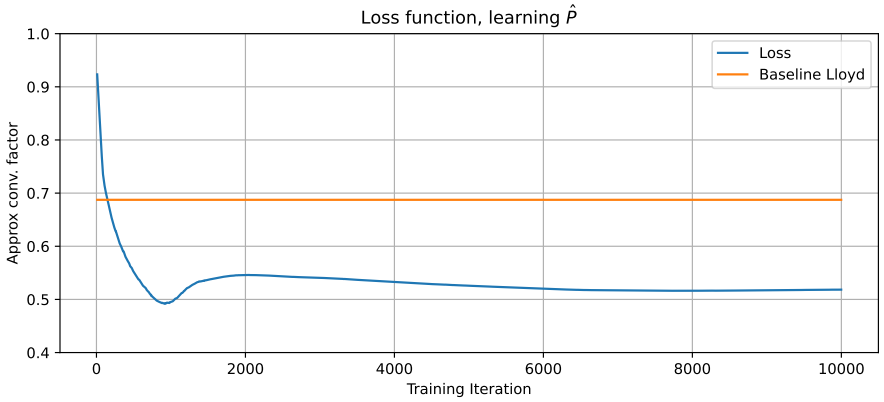

Figure 4: Loss function when optimizing the nonzero entries of $\hat{\boldsymbol{P}}$, then forming $\boldsymbol{P} = \hat{\boldsymbol{P}}\hat{Agg}$. Using Adam optimizer with a learning rate of 0.01.

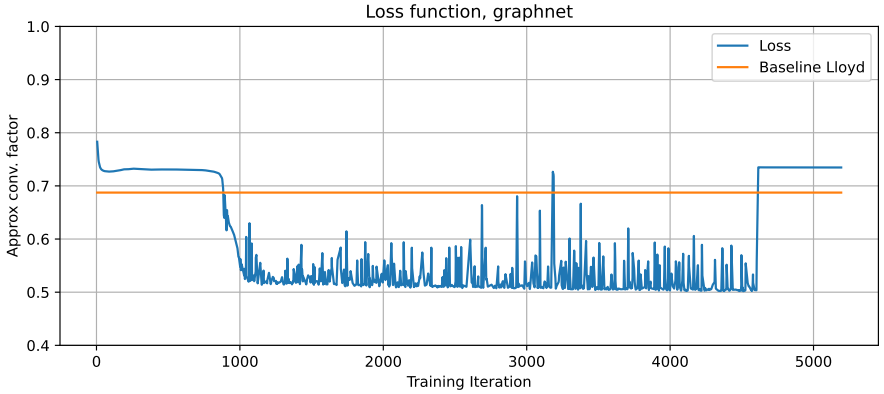

Figure 5: Loss function when using a Graphnet to generate $\hat{\boldsymbol{P}}$. Using Adam optimizer with a learning rate of 10^{-5}.