1 Learning Aggregation-Based Interpolation

The edge-based convolution network we are using currently outputs a matrix that contains the same sparsity (or a subset of) the graph that it was given. This means that it can be difficult to directly output some matrix with a different sparsity pattern without some intermediate transformation.

Denote Agg the tentative aggregation operator without smoothing, for example,

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

would correspond to a 5×5 system with two aggregates, the first containing nodes $1-3$ and the second containing nodes 4 and 5. 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

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

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. This would give

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

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

$$\boldsymbol{P} = \hat{\boldsymbol{P}}\hat{Agg} = \begin{bmatrix} 0.277 & \\ 0.333 & \\ 0.222 & 0.166 \\ 0.111 & 0.416 \\ & 0.416 \end{bmatrix}, \tag{4}$$

which gives a reasonable sparsity pattern for the interpolation, given Agg. Note that if we view this as a graph, we get edge connections between aggregates "for free", which is exactly what is wanted.

2 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.

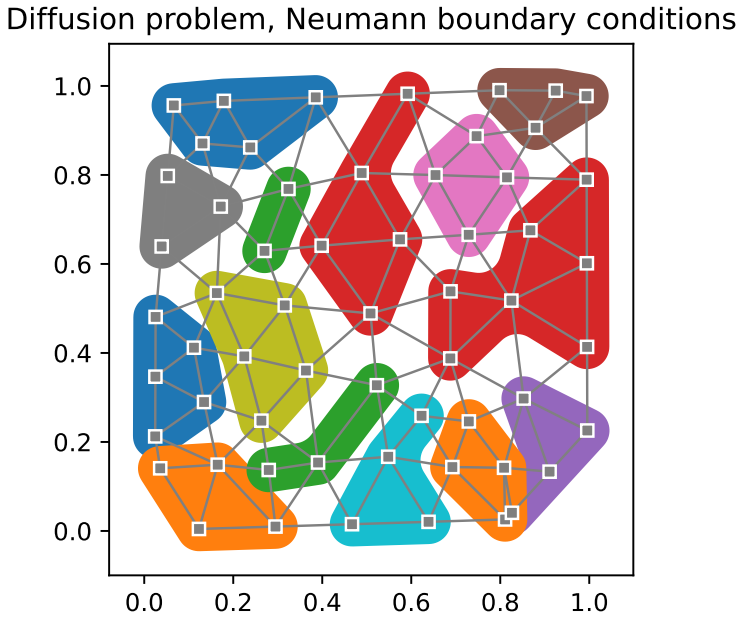

Figure 1: Specific problem being solved, with aggregate information.

3 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.

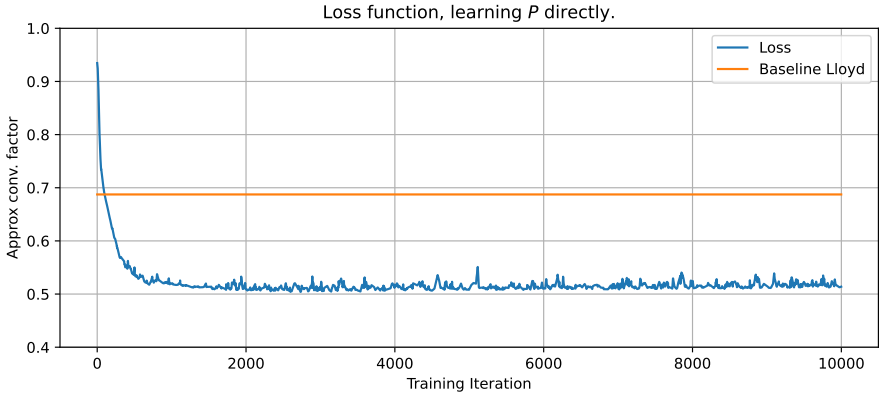

Figure 2: Loss function when directly optimizing the nonzero entries of $\boldsymbol{P}$. Using Adam optimizer with a learning rate of 0.01.

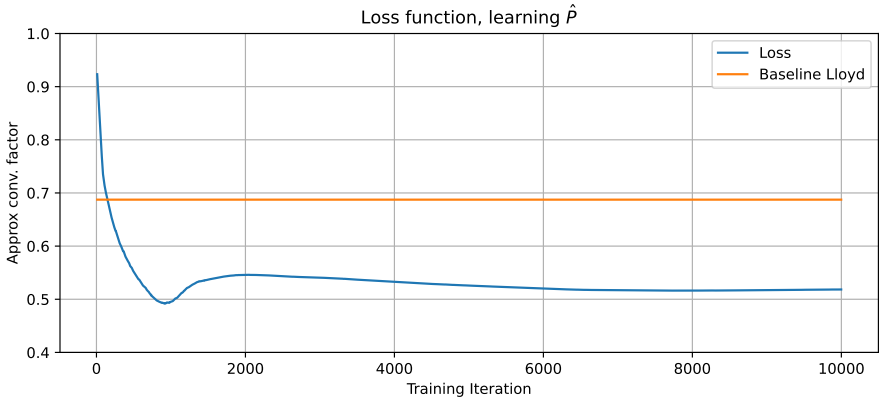

Figure 3: Loss function when optimizing the nonzero entries of $\hat{\boldsymbol{P}}$, then forming $\boldsymbol{P} = \hat{\boldsymbol{P}}\hat{Agg}$. Using Adam optimizer with a learning rate of 0.01.

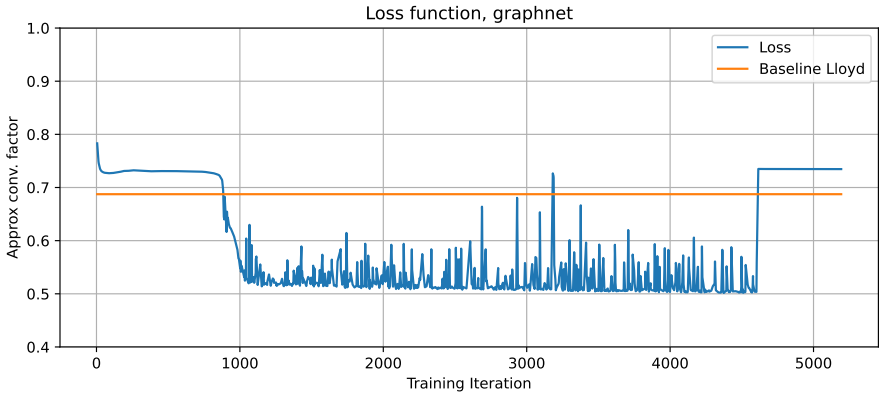

Figure 4: Loss function when using a Graphnet to generate $\hat{\boldsymbol{P}}$. Using Adam optimizer with a learning rate of 10^{-5}.