1 Background

Here, we are attempting to train an ML agent to output both aggregates and interpolation for an AMG solver given the matrix $\boldsymbol{A}$ for a discretized PDE. We will be primarily be looking at the finite element discretization of the isotropic and anisotropic diffusion problems, i.e. PDEs of the form

$$-\nabla \cdot (\boldsymbol{D}\nabla u) = 0, \tag{1}$$

$$\boldsymbol{D} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} 1 & \\ & \varepsilon \end{bmatrix} \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}^T, \tag{2}$$

for some arbitrary θ and ε. In the isotropic case, $\theta = 0$ and $\varepsilon = 1$.

In both the isotropic and anisotropic cases, unstructured datasets of the following are generated:

1. A training set of 1000 unstructured grids

2. A testing set of 300 unstructured grids

Each of these grid meshes are fairly small, overall containing between around 15 and 400 points. To create a mesh, random points are generated in $[0,1]^2$, a convex hull constructed, then meshed with gmsh[3]. The training and testing sets are intentionally limited to a small number of grids to reduce training complexity.

For the anisotropic problems, the rotation, θ, uniformly varies between $(0, 2\pi)$. The scaling of the $y-$axis is log-uniformally distributed between 10^{-5} and 10^5.

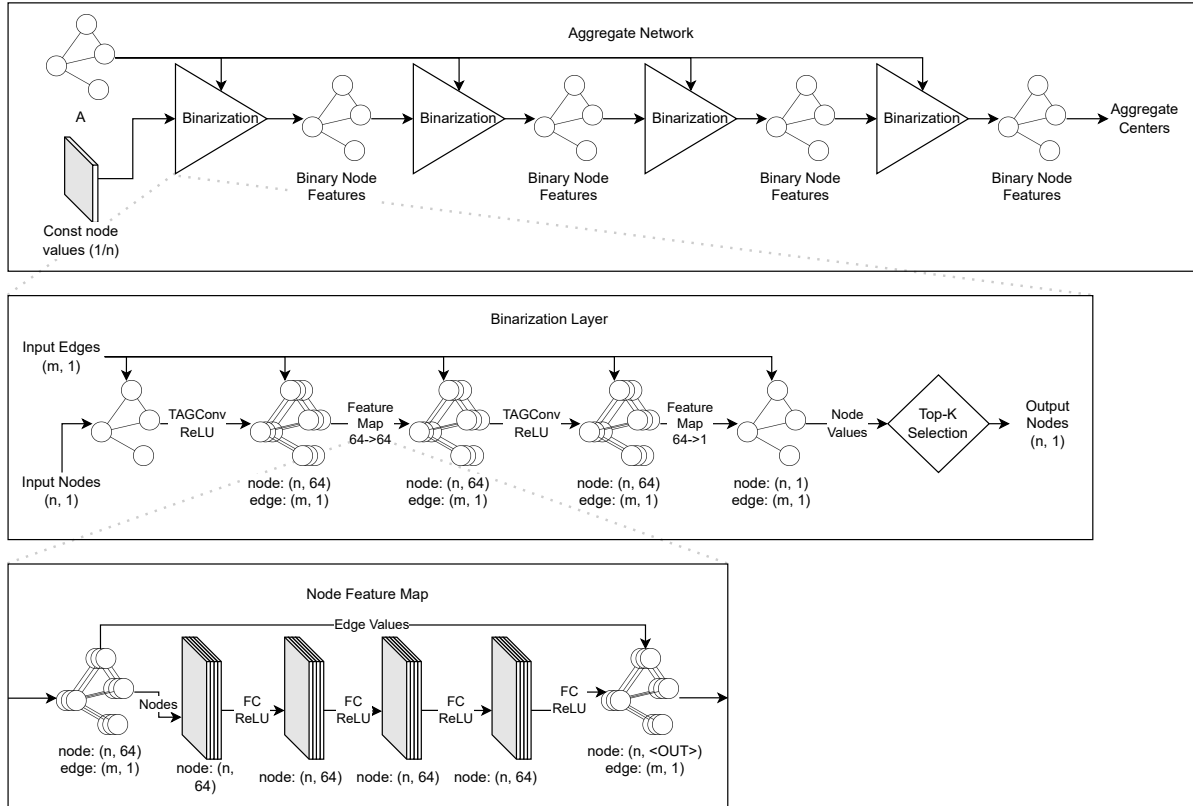# 2 Generating Aggregates and Interpolation

Figure 1: Architecture for the aggregate-picking network. This consists of 4 binarization layers, each having two node convolution passes.

To compute the aggregates and final interpolation operator, we concurrently train three graph networks to give different outputs:

1. Θ_{agg}, outputting aggregate centers

2. Θ_{soc}, outputting a strength-of-connection matrix for Bellman-Ford

3. Θ_P, outputting a smoothing operator

The algorithm for computing the interpolation operator is roughly sketched below:

1. Let $\boldsymbol{A} \in \mathbb{R}^{n \times n}$ be the system we are trying to solve and $\alpha \in (0, 1]$ be some parameter that determines the ratio of aggregates to vertices, i.e. we will be roughly coarsening the graph by $1/\alpha$. Define $k := \lceil \alpha n \rceil$, the number of aggregates we will be outputting.

2. Convolve the graph of $\boldsymbol{A}$ with Θ_{soc} to obtain $\boldsymbol{C}$, a matrix of weights that will be used for Bellman-Ford.

3. Convolve the graph of $\boldsymbol{A}$ concatenated with $\boldsymbol{C}$ using Θ_{agg} to obtain a new set of node values, restarting multiple times and inserting the tenative cluster centers as a node feature. We will use the output node values as a *scoring*. Define the aggregate centers as the indices of the largest k node scores.

4. Run Bellman-Ford on the graph with these aggregate centers to obtain a tentative aggregation operator, Agg.

5. Now, again convolve the graph of $\boldsymbol{A}$ but with Θ_{Interp} (with aggregate information) to obtain the aggregate smoother $\hat{\boldsymbol{P}}$. Form $\boldsymbol{P} := \hat{\boldsymbol{P}}\text{Agg}$.

These steps are formalized in Alg 1.

Algorithm 1 Outputting Aggregates and Interpolation

---

procedure FULLAGGNET($\boldsymbol{A} \in \mathbb{R}^{n \times n}, \alpha$)
    $k := \lceil \alpha n \rceil$ ▷ Number of aggregates to choose
    $\boldsymbol{C} := \Theta_{\text{soc}}(\boldsymbol{A})$ ▷ Obtain strength-of-connection matrix

    $\boldsymbol{s} := \Theta_{\text{agg}}(\boldsymbol{A}, \boldsymbol{C}, k)$ ▷ Get aggregate scores for each node
    $\boldsymbol{r} := \text{TOP-K}(\boldsymbol{s}, k)$ ▷ Get indices of k largest nodes to be centers
    $\text{Agg} := \text{BELLMAN-FORD}(\boldsymbol{C}, \boldsymbol{r})$ ▷ Get tentative aggregate assignment matrix

    $\hat{\boldsymbol{P}} := \Theta_{\text{Interp}}(\boldsymbol{A}, \text{Agg})$ ▷ Aggregate smoother
    return $\hat{\boldsymbol{P}}\text{Agg}$ ▷ Smooth aggregate assignment
end procedure

---

3   Genetic Training

Training of both networks is done at the same time with a custom-written genetic algorithm that takes advantage of the easily parallelizable fitness calculation. A genetic algorithm is used because it does not require any gradient information[4], which benefits us because the algorithm used to output the aggregates is not easily differentiable.

A basic overview of the training algorithm is that the method is seeded with some number of randomly generated networks. A subset of the best performing (most fit) networks are selected and *bred* with one another (crossing weights/traits) and *mutations* inserted (random perturbations to weights) to create another population of networks. This is then repeated for many *generations* until a set of hopefully trained networks is obtained, from which we can pick the best fit as our final network.

3.1 Folded Crossover

So-called "folded crossover", as explored in [2], is a genetic crossover technique that groups together consistent regions of the population's chromosomes. When training neural networks, it is entirely possible that specific modules (layers) can contain several hundred or even several thousand weight entries, meaning traditional crossover that naïvely slice arbitrary points of the weight space can partially replace weight values for modules and produce suboptimal offspring.

By "folding" regions of the weight space, we ensure that whole modules are copied to offspring. We further modify the crossover procedure by allowing the two children of a pair of parents to randomly pick which parent they receive each module from. This is an additional level of flexibility compared to, e.g., single-point or two-point crossover which divide the weight-space into two or three regions, respectively.

Initial testing of the folded crossover provides subjective speedup to the rate at which networks learn, but I haven't really timed it or anything.

3.2 Stochastic Batching

As mentioned previously, the most expensive part of training the networks is the computation of the fitness values, as we must compute an average convergence rate over every mesh in our training set, for every network in our population. Reducing the number of meshes used at each generation can drastically speed-up the training routine, at perhaps a reduction in training effectivity. This also eliminates any notion of caching fitness values for regions of the population that do not change between generations, as we have a different batch each generation and must totally re-compute fitness to have any sort of reasonable comparison.

In practice, this gives a healthy speed-up for minibatch sizes of only 16. I had to decrease the maximum perturbation during mutation to $|m| = 0.1$ to get it to do anything sensible, but perhaps it was too high in the first place?

3.3 Greedy Selection

I've experimentally found that using steady-state selection will train the network up to some point, then completely fail to make any more progress. At this point in the training, I've switched to what I call a greedy selection, where the population at generation $i + 1$ will totally consist of the one fittest individual from i. Then, these will be mutated as usual.

I kind of think of this as a sort of greedy approximation to gradient descent: we have one network and at every step we attempt to take the most optimal step towards a better network.

4 Lloyd Aggregation

The baseline method for computing some multigrid smoothing operator is given by *Lloyd aggregation*, the implementation of which was taken from PyAMG[6]. This algorithm is *seeded* via a random set of aggregate centers which are then iteratively refined by an implementation of Lloyd's method (also known as K-Means) reformulated for graph inputs[1]. One may conject that

To allow the method to output something sensible for anisotropic and generally more difficult problems, the distance between nodes is defined by a heuristic *strength-of-connection* measure. Here, we use a combination of an evolution-based measure[5] and the inverse of the magnitude of the entries of the system. Formally, let the strength measure being used be defined as

$$\boldsymbol{C} := \boldsymbol{S} + \frac{1}{|\boldsymbol{A}|}, \tag{3}$$

where $\boldsymbol{S}$ is the evolution measure, $\boldsymbol{A}$ is the original system, and $\frac{1}{|\cdot|}$ is applied entrywise to $\boldsymbol{A}$.

Once a set of aggregate centers is found, each remaining node is assigned to an aggregate by finding its nearest (by means of $\boldsymbol{C}$) center node. This is done by a modified Bellman-Ford algorithm[1], whose implementation is given in Alg. 2. After each nearest center node is found, a tentative aggregate operator is returned by allotting each aggregate a column in Agg $\in \mathbb{R}^{n \times c}$, where $\text{Agg}_{ij} = 1$ iff node i belongs in aggregate j.

Note that Agg is not suitable to use directly as a multigrid interpolation/restriction operator as there is no direct information crossover between aggregates. Therefore, we opt to *smooth* the columns of the aggregate matrix by means of a Jacobi smoother,

$$S := I - \omega D^{-1} A, \tag{4}$$

$$\omega := \frac{4}{3\lambda_{\max}(A)}, \tag{5}$$

where D is the diagonal portion of A. The final interpolation is formed by

$$P = S\text{Agg}. \tag{6}$$

5   Results

The ML method is compared against a baseline Lloyd aggregation with Jacobi smoothing, for a coarsening ratio of $\alpha = 0.1$. Performance is measured by *average rate of convergence* for some set of problems; detailed information is given in table 1. Each ML agent is trained for that specific class of problem, i.e. *isotropic* or *anisotropic*.

Problem Type	Data Set	Lloyd, SA Conv.	ML Conv.
Isotropic	Train	0.4208	0.4109
Isotropic	Test	0.4177	0.4075
Anisotropic	Train	0.7705	0.7451
Anisotropic	Test	0.7978	0.7697

Table 1: Convergence of ML vs baseline Lloyd+SA on various sets of problems, individual networks trained on problem type.

Additionally, an agent was trained on datasets of both isotropic and anisotropic problems (the union of the problems in table 1. The results of this new network are given in 2.

Problem Type	Data Set	Lloyd, SA Conv.	ML Conv.
Isotropic	Train	0.4208	0.3889
Isotropic	Test	0.4177	0.3828
Anisotropic	Train	0.7705	0.7421
Anisotropic	Test	0.7978	0.7688

Table 2: Convergence of ML vs baseline Lloyd+SA on various sets of problems, one agent trained on both isotropic and anisotropic problems.

All following numerical results were generated from the model trained on both sets of problems.
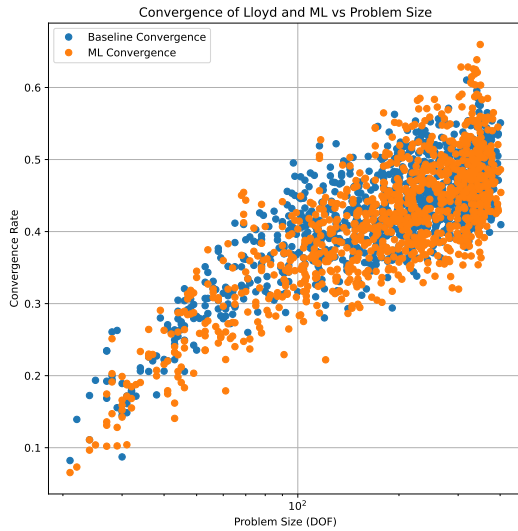
5.1   Isotropic Results
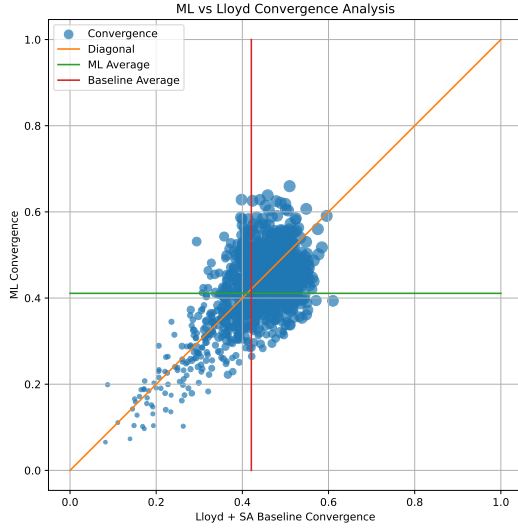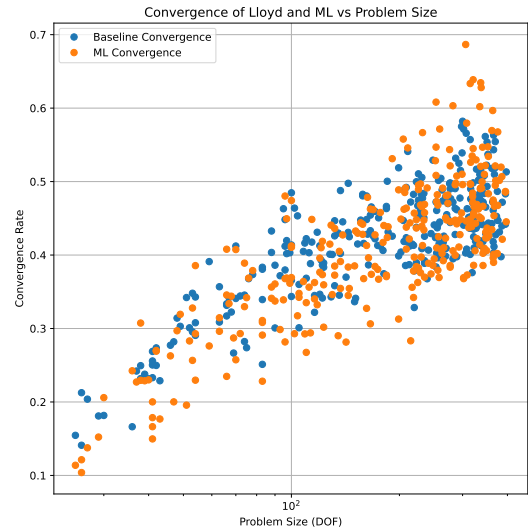
(a) Training convergence
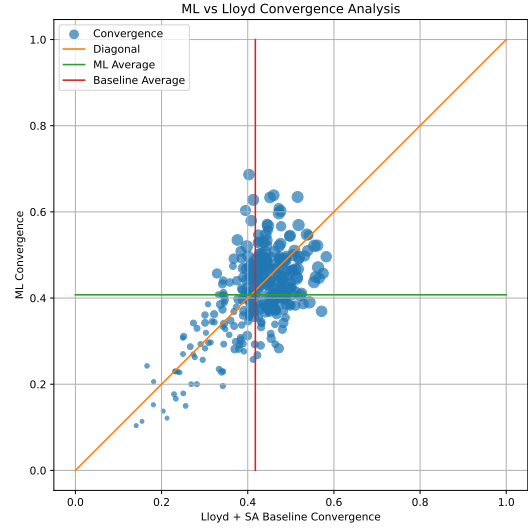
(b) Testing convergence

Figure 2: Convergence data for the ML AMG method vs a baseline Lloyd and Jacobi SA method. Values below the diagonal indicate a better convergence for the ML. Markers are scaled by problem size.
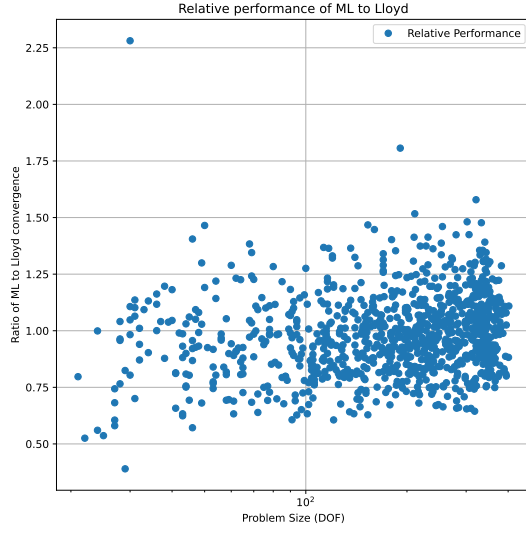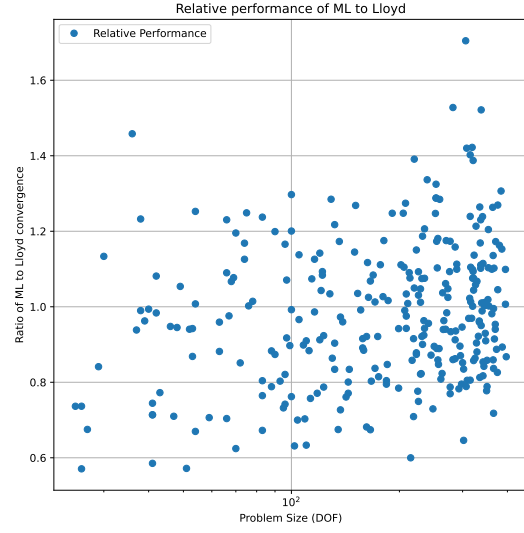

(a) Training convergence

(b) Testing convergence

Figure 3: Convergence data for the two methods plotted against problem size (DOF).

(a) Relative training performance

(b) Relative testing performance

Figure 4: Relative performance of the ML to the Lloyd method, plotted against problem size. Relative performance is obtained by dividing the ML convergence by the Lloyd convergence for each problem. Values below 1 indicate better ML performance, while values above 1 indicate better baseline performance.
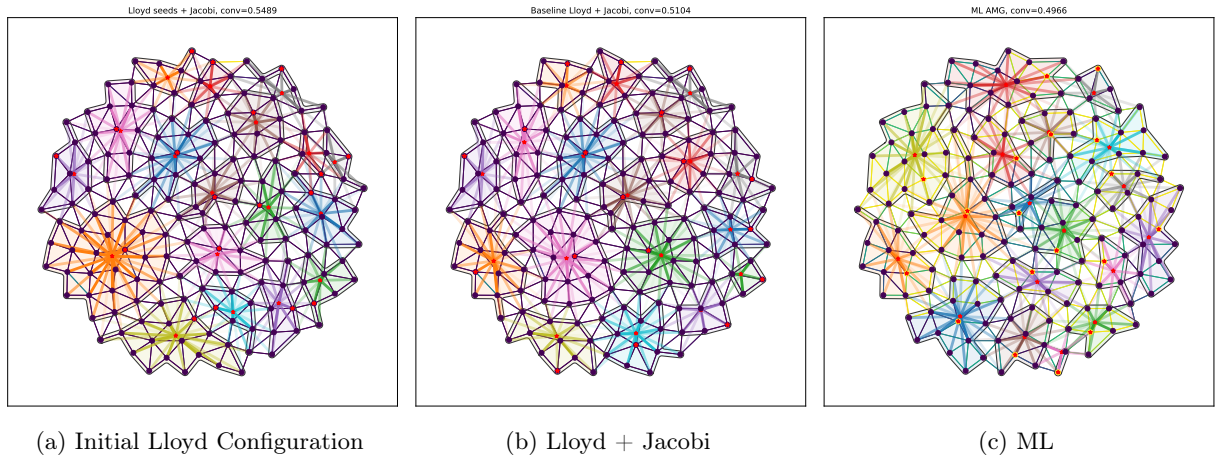
(a) Initial Lloyd Configuration

(b) Lloyd + Jacobi

(c) ML

Figure 5: Aggregate and interpolation data for a circular unstructured mesh. This particular mesh has 173 DoF.
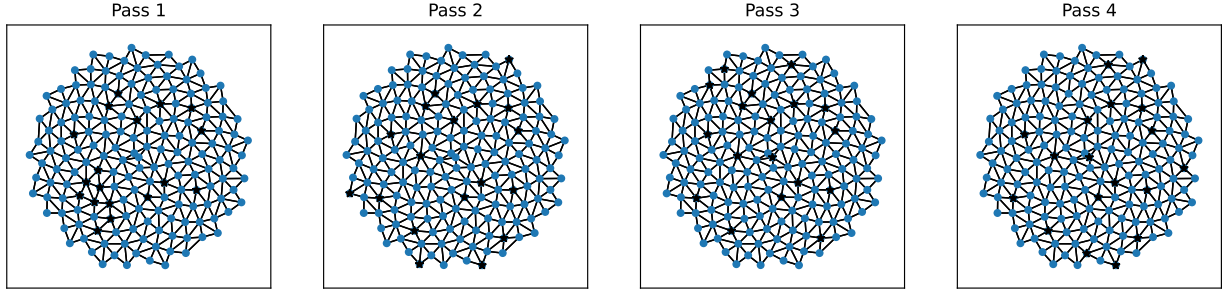
Figure 6: Tentative aggregate centers after each binarization "pass". This is the same mesh as in figure 5.
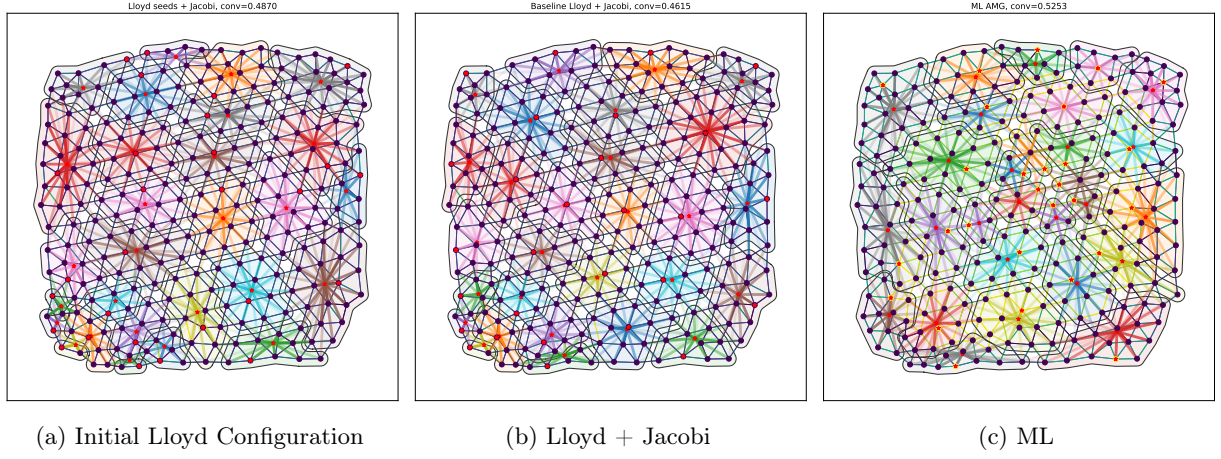

(a) Initial Lloyd Configuration   (b) Lloyd + Jacobi   (c) ML
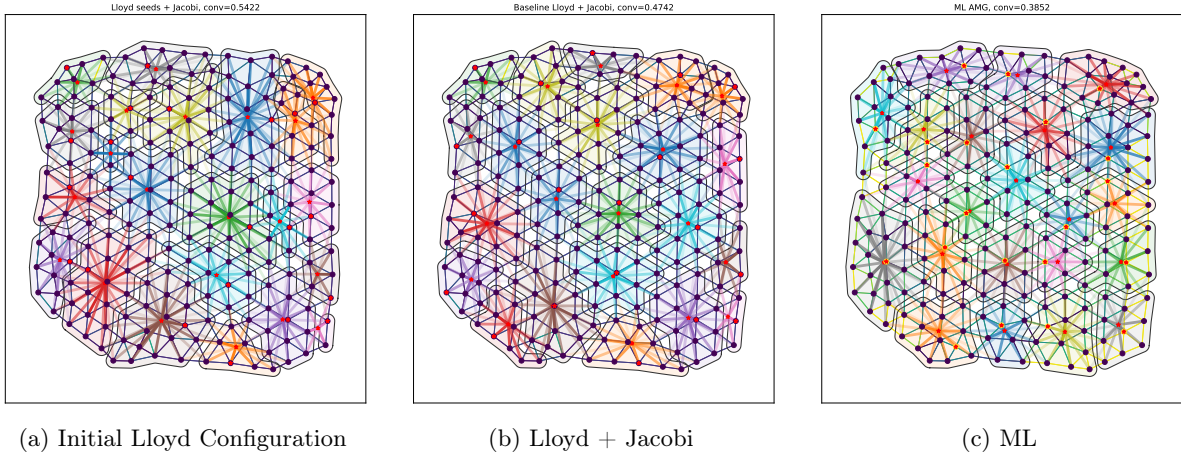
Figure 7: Aggregate and interpolation data for a random unstructured mesh. This particular mesh has 220 DoF.
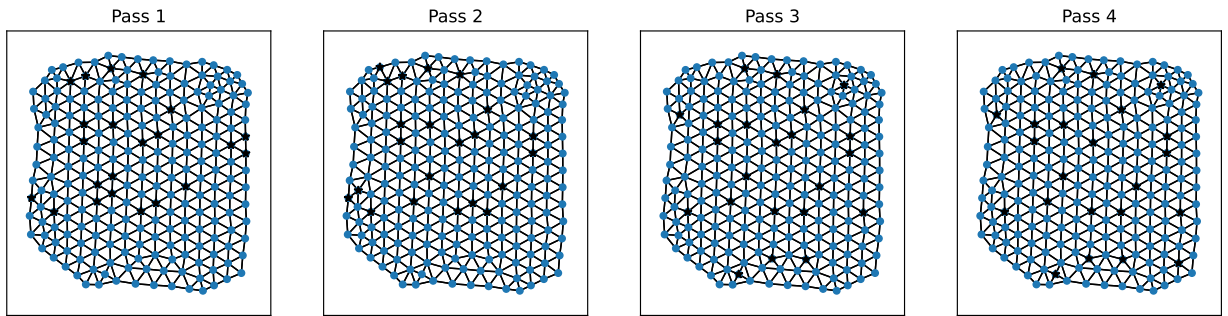
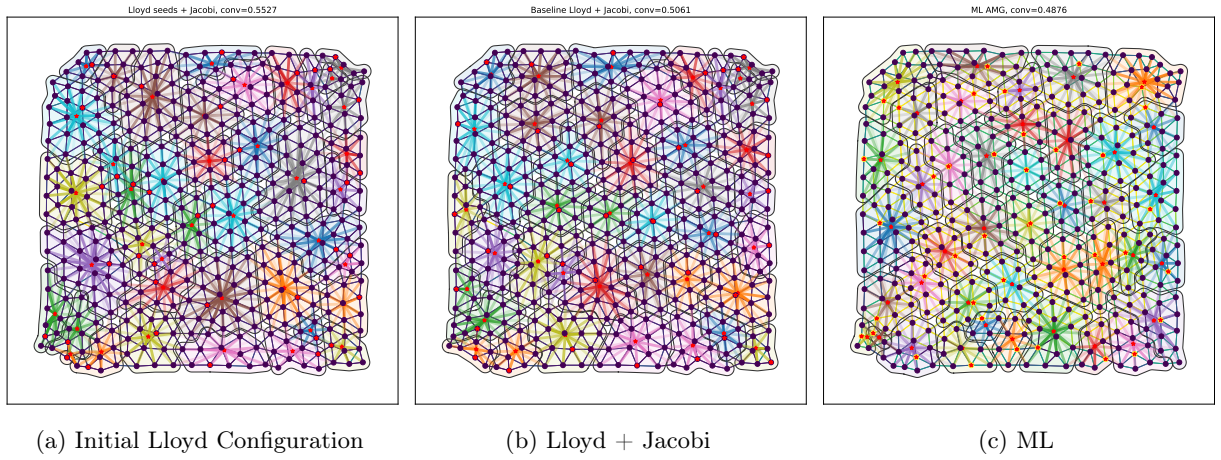Figure 8: Tentative aggregate centers after each binarization "pass". This is the same mesh as in figure 7.

(a) Initial Lloyd Configuration   (b) Lloyd + Jacobi   (c) ML

Figure 9: Aggregate and interpolation data for a random unstructured mesh. This particular mesh has 273 DoF.
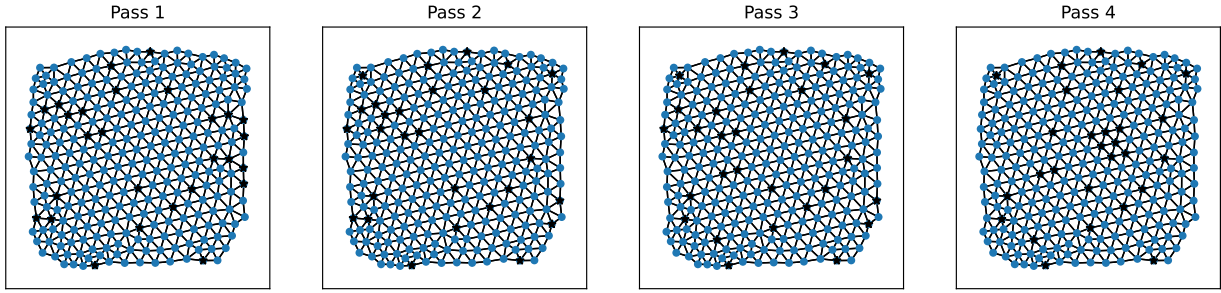
Figure 10: Tentative aggregate centers after each binarization "pass". This is the same mesh as in figure 9.

(a) Initial Lloyd Configuration   (b) Lloyd + Jacobi   (c) ML

Figure 11: Aggregate and interpolation data for the largest mesh in the *training set*. This particular mesh has 410 DoF.
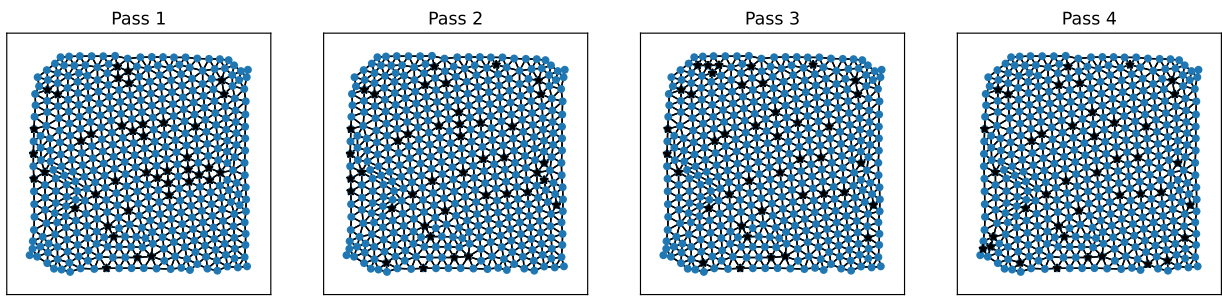
Figure 12: Tentative aggregate centers after each binarization "pass". This is the same mesh as in figure 11.

5.2 Anisotropic Results

(a) Training convergence
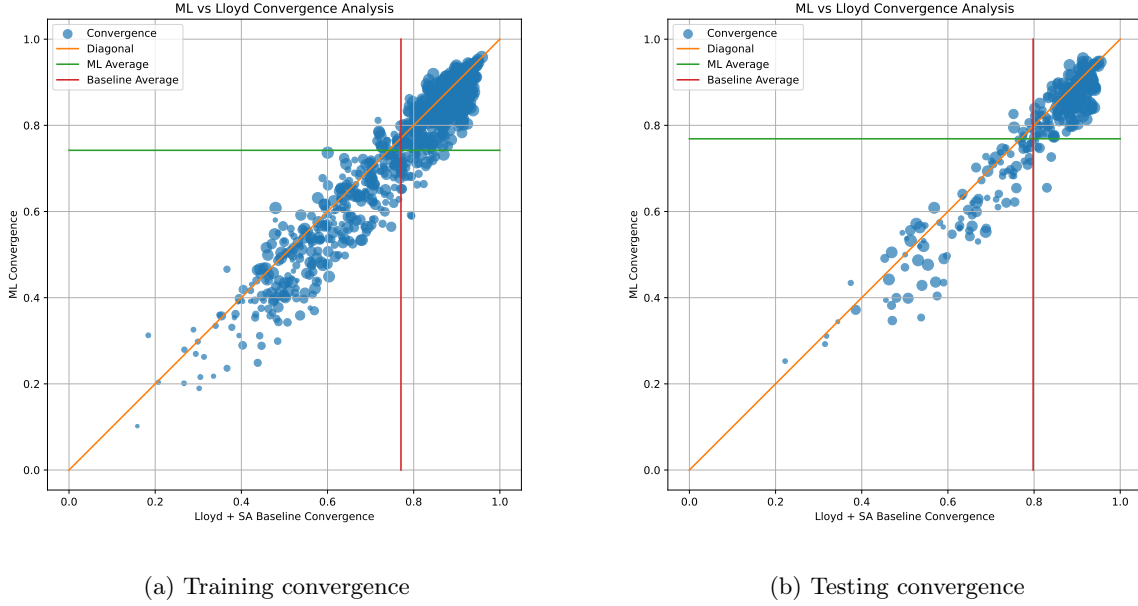
(b) Testing convergence

Figure 13: Convergence data for the ML AMG method vs a baseline Lloyd and Jacobi SA method. Values below the diagonal indicate a better convergence for the ML. Markers are scaled by problem size.

A Graph Algorithms

References

[1] N. BELL, *Algebraic Multigrid for Discrete Differential Forms*, PhD thesis, University of Illinois Urbana-Champaign, Aug. 2008.

[2] P. ESFAHANIAN AND M. AKHAVAN, *GACNN: training deep convolutional neural networks with genetic algorithm*, CoRR, abs/1909.13354 (2019).

[3] GEUZAINE, CHRISTOPHE AND REMACLE, JEAN-FRANCOIS, *Gmsh*.

[4] M. MITCHELL, *An introduction to genetic algorithms*, Complex Adaptive Systems, Bradford Books, Cambridge, MA, Mar. 1998.

[5] L. N. OLSON, J. SCHRODER, AND R. S. TUMINARO, *A new perspective on strength measures in algebraic multigrid*, Numerical Linear Algebra with Applications, 17 (2009), pp. 713–733.

[6] L. N. OLSON AND J. B. SCHRODER, *Pyamg: Algebraic multigrid solvers in python v4.0.* `https://github.com/pyamg/pyamg`, 2018. Release 4.0.

Algorithm 2 Modified Bellman-Ford
---
procedure BELLMANFORD($C \in \mathbb{R}^{n \times n}$, centers $\in \mathbb{R}^n$)
    dist $:= \begin{bmatrix} \infty, \infty, \ldots, \infty \end{bmatrix} \in \mathbb{R}^n$
    nearest $:= \begin{bmatrix} 0, 0, \ldots, 0 \end{bmatrix} \in \mathbb{R}^n$

    for $c \in$ centers **do**
        $\text{dist}_c \leftarrow 0$
        $\text{nearest}_c \leftarrow c$
    end for

    finished $:=$ True
    while ¬finished **do**
        finished $\leftarrow$ True
        for (i, j) such that $C_{ij} \neq 0$ **do**
            if $\text{distance}_i + C_{ij} < \text{distance}_j$ **then**
                $\text{distance}_j \leftarrow \text{distance}_i + C_{ij}$
                $\text{nearest}_j \leftarrow \text{nearest}_i$
                finished $\leftarrow$ False
            end if
        end for
   **end while**

    return distance, nearest
end procedure

Algorithm 3 Assignment of nodes to aggregates
---
procedure ASSIGNAGGREGATES(centers $\in \mathbb{R}^c$, nearest $\in \mathbb{R}^n$)
    center_number $:= \{\}$
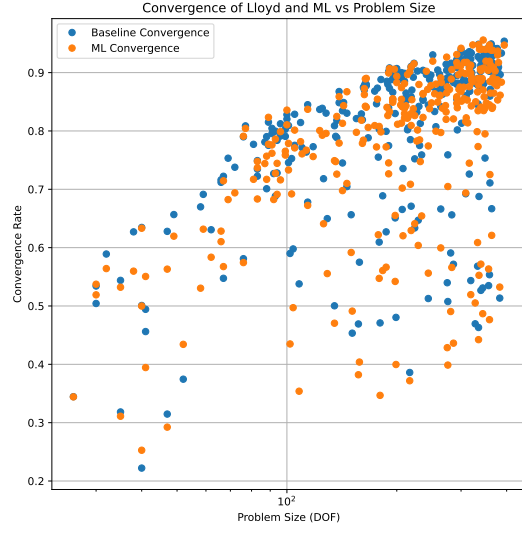   Agg $\in \mathbb{R}^{n \times c}$

    for $i \leftarrow 1, c$ **do**
        center_number $[\text{centers}_i] \leftarrow i$ ▷ Assign each center a column in Agg
    end for
    for $i \leftarrow 1, n$ **do**
        $j :=$ center_number $[\text{centers}_i]$ ▷ Find the correct column based on the center
        $\text{Agg}_{ij} = 1$
    end for
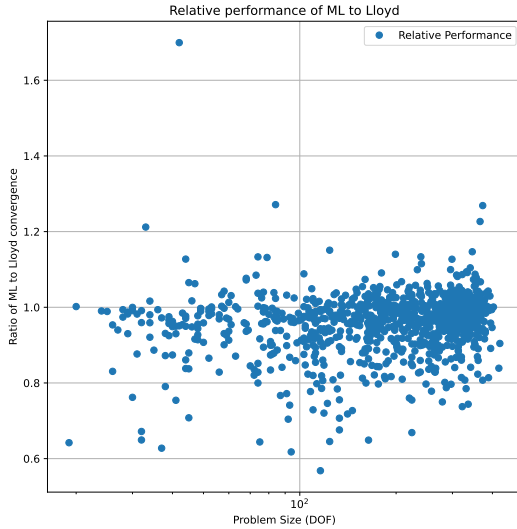   **return** Agg
end procedure
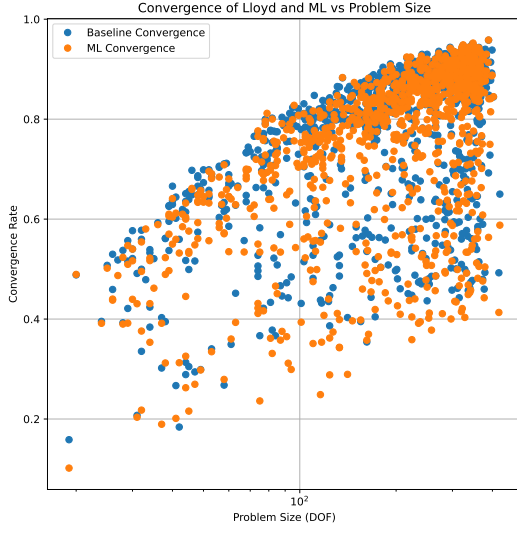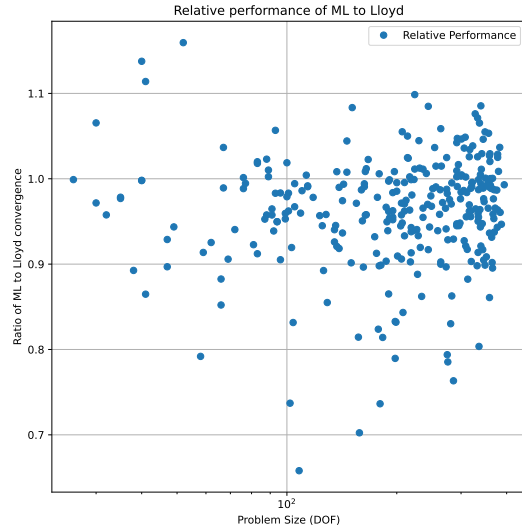

(a) Training convergence

(b) Testing convergence

Figure 14: Convergence data for the two methods plotted against problem size (DOF). The ML method seems biased towards better performance on the smaller problems, and does not do as well on the larger problems.

(a) Relative training performance

(b) Relative testing performance

Figure 15: Relative performance of the ML to the Lloyd method, plotted against problem size. Relative performance is obtained by dividing the ML convergence by the Lloyd convergence for each problem. Values below 1 indicate better ML performance, while values above 1 indicate better baseline performance.

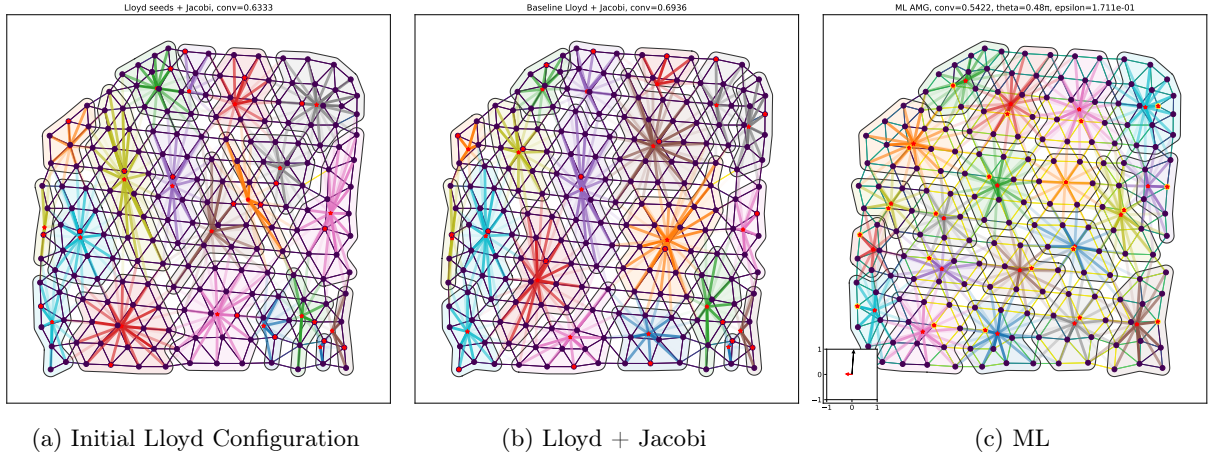(a) Initial Lloyd Configuration   (b) Lloyd + Jacobi   (c) ML

Figure 16: Aggregate and interpolation data for an anisotropic problem with 198 DoF. The problem is stretched by a factor of 0.177 along $\theta = 0.98\pi$.
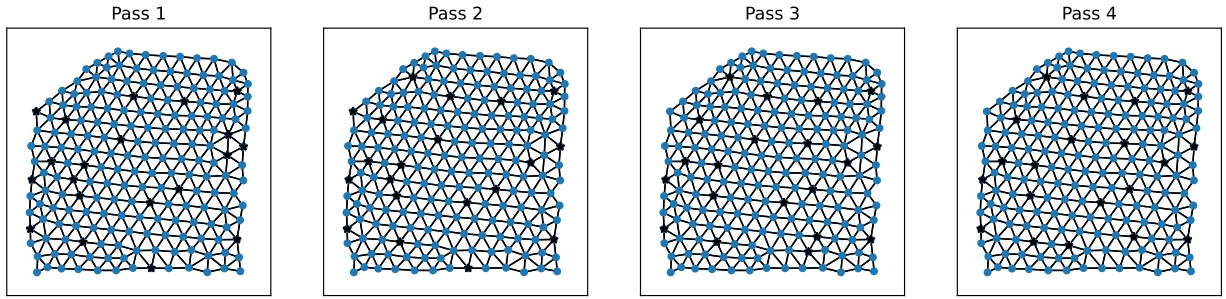

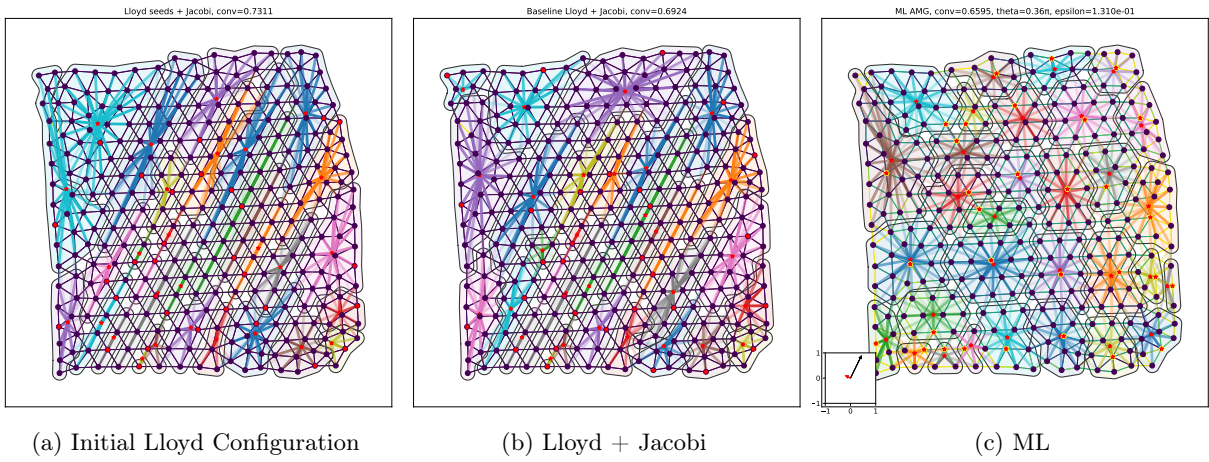Figure 17: Tentative aggregate centers after each binarization "pass". This is the same mesh as in figure 16.

(a) Initial Lloyd Configuration   (b) Lloyd + Jacobi   (c) ML

Figure 18: Aggregate and interpolation data for an anisotropic problem with 309 DoF. The problem is stretched by a factor of 0.131 along $\theta = 0.86\pi$.
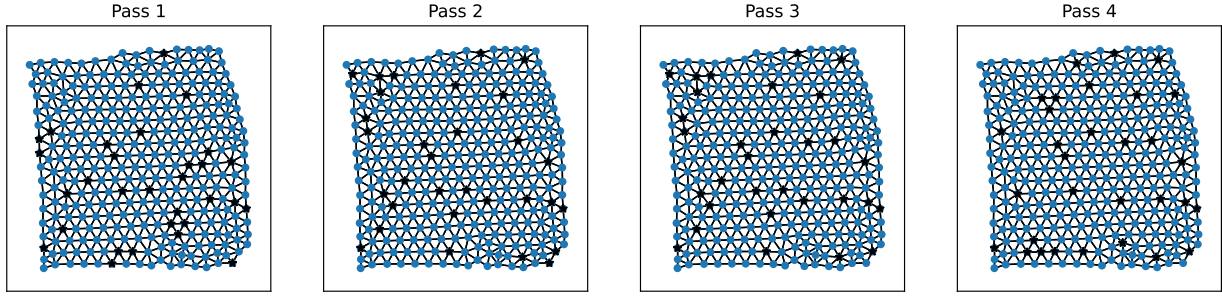
13

Figure 19: Tentative aggregate centers after each binarization "pass". This is the same mesh as in figure 18.

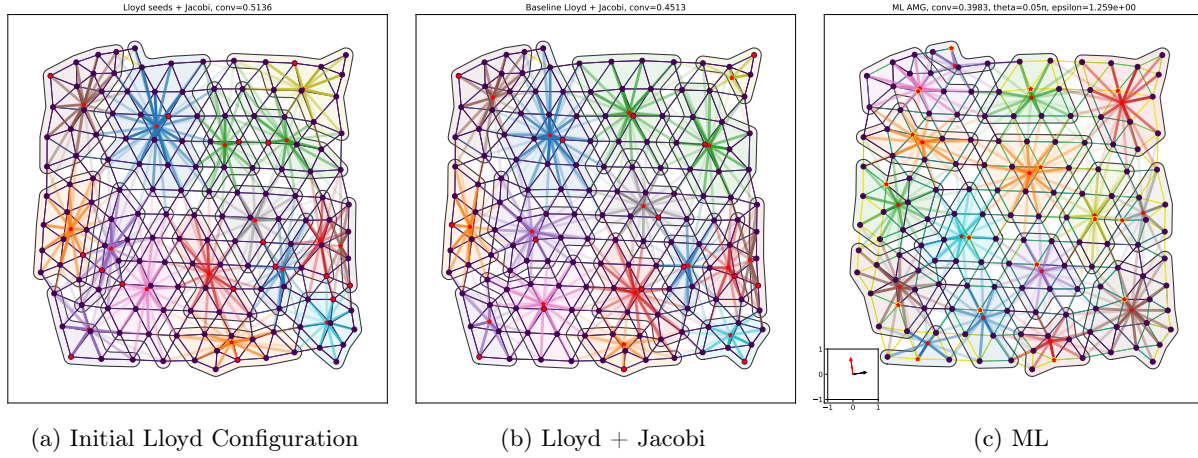(a) Initial Lloyd Configuration   (b) Lloyd + Jacobi   (c) ML

Figure 20: Aggregate and interpolation data for an anisotropic problem with 157 DoF. The problem is stretched by a factor of 1.259 along $\theta = 0.55\pi$.
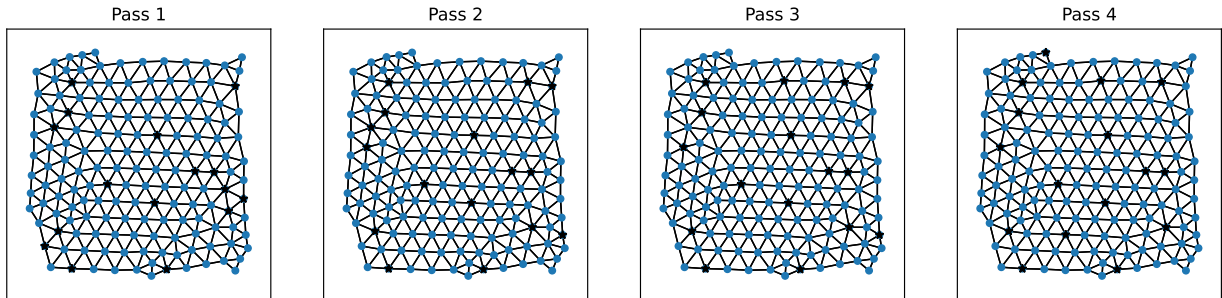

Figure 21: Tentative aggregate centers after each binarization "pass". This is the same mesh as in figure 20.