Commun. Comput. Phys.
doi: 10.4208/cicp.OA-2023-0072

Vol. **36**, No. 1, pp. 200-220
July 2024

AutoAMG(θ): An Auto-tuned AMG Method Based on Deep Learning for Strong Threshold

Haifeng Zou[1,2], Xiaowen Xu[3,*], Chen-Song Zhang[4] and Zeyao Mo[3]

[1] *Graduate School of China Academy of Engineering Physics, China Academy of Engineering Physics, P.R. China.*
[2] *Shenzhen International Center for Industrial and Applied Mathematics, Shenzhen Research Institute of Big Data, P.R. China.*
[3] *Laboratory of Computational Physics, Institute of Applied Physics and Computational Mathematics, P.R. China.*
[4] *Academy of Mathematics and Systems Science, Chinese Academy of Sciences, P.R. China.*

Received 8 March 2023; Accepted (in revised version) 30 November 2023

Abstract. Algebraic Multigrid (AMG) is one of the most widely used iterative algorithms for solving large sparse linear equations $Ax=b$. In AMG, the coarse grid is a key component that affects the efficiency of the algorithm, the construction of which relies on the strong threshold parameter θ. This parameter is generally chosen empirically, with a default value in many current AMG solvers of 0.25 for 2D problems and 0.5 for 3D problems. However, for many practical problems, the quality of the coarse grid and the efficiency of the AMG algorithm are sensitive to θ; the default value is rarely optimal, and sometimes is far from it. Therefore, how to choose a better θ is an important question. In this paper, we propose a deep learning based auto-tuning method, AutoAMG(θ) for multiscale sparse linear equations, which are common in practical problems. The method uses Graph Neural Network (GNN) to extract matrix features, and a Multilayer Perceptron (MLP) to build the mapping between matrix features and the optimal θ, which can adaptively predict θ values for different matrices. Numerical experiments show that AutoAMG(θ) can achieve significant speedup compared to the default θ value.

AMS subject classifications: 65F08, 65F10, 65N55, 68T05
Key words: AMG, strong threshold, graph neural network, auto-tuning, multiscale matrix.

1 Introduction

Solving sparse linear equations $Ax=b$ is ubiquitous in numerical simulations, and is a major bottleneck affecting computational efficiency. Owing to its good generality and

[*] Corresponding author. *Email addresses:* zou_haifeng@foxmail.com (H. Zou), xwxu@iapcm.ac.cn (X. Xu), zhangcs@lsec.cc.ac.cn (C.-S. Zhang), zeyao_mo@iapcm.ac.cn (Z. Mo)

optimal computational complexity, the AMG algorithm [1–3] is one of the most widely used algorithms for large-scale sparse linear equations, which uses only information from the matrix to construct components, including coarsening, interpolation, and restriction operators. During the coarsening procedure, a subset of points from the adjacency matrix A is selected as points in the coarse grid, which is the basis for constructing a coarse grid matrix A_c. Different coarsening strategies will result in different coarse matrices A_c. In the classical AMG algorithm, points in the subset are selected based on the strong threshold θ and the strength of the connectivity between points, which is calculated by the value of the matrix entries. Hence the value of θ directly affects the grid coarsening results, and is a key factor affecting the algorithm's efficiency.

In the classical AMG algorithm, most coarsening algorithms are based on heuristic strategies for coarse grid construction. A basic principle is to perform coarsening along the direction of strong connectivity to accommodate the property that algebraic errors are smoothed or relaxed along the same direction. If the strong threshold θ is large, then the number of points in the corresponding coarse grid is large, which means the AMG algorithm has high complexity. If θ is small, although the number of points in the coarse grid is smaller, the residuals may decrease more slowly, requiring more iterations to converge. Since there is no strict theoretical guarantee on the size of the optimal coarse grid, the current value of θ can only be chosen empirically. For example, in the HYPRE AMG solver [4], depending on the physical dimension of the sparse matrix, θ equals 0.25 for 2D problems and 0.5 for 3D problems. However Vakili [5] and Nikola [6] utilize the incompressible Navier Stokes equation and linear poroelasticity equation, respectively, as the test cases, both of which show the increase of θ along with the monotone decrease of time. Here, we take the diffusion problem as the example, and find that the number of iterations changes irregularly with the increase of θ. If the diffusion coefficients are isotropic, the default values of θ can achieve the desired convergence rate. If the diffusion coefficients are anisotropic, which means there are significant differences in the strength of connectivity between points, then the default values of θ maybe far from the optimal. Notably, small changes in θ may have a large impact on the construction of the coarse grid, thus affecting the convergence rate and efficiency of AMG. In particular, we focus on the so-called multiscale sparse matrices [7]. In some typical test cases, the number of iterations of the default θ is 10 times larger than the minimum number of iterations obtained by grid search (see Section 2.3 for detail).

The above problem can be summarized as follows: how to choose an appropriate θ for any given sparse matrix. Considering that the properties of the input matrix may vary dynamically, the automatic selection of a suitable θ for different linear systems is a crucial and challenging task, since there is no theoretical guarantee yet. Machine learning and deep learning algorithms provide a feasible approach. Paola F [8] used a Convolutional Neural Network (CNN) to extract matrix features, and built a regression model with those features. The inputs of the regression model are matrix features, strong threshold θ, and $-\log_2 h$ (h is the edge length in the mesh), and the output y is an approximated convergence factor. After training, the regression model is used to optimize θ. There are other ways to enhance the robustness of iterative methods with machine learning and deep learning. For example, a variety of classification algorithms are used to select optimal iterative methods based on the input matrix features [9–12]; deep learning algorithms are utilized to optimize the prolongation matrix P, restriction matrix R, and smoother S in AMG [13–16].

Our target is optimizing θ adaptively according to the input matrices, and our contributions are as follows:

- Classical graph convolution networks such as GCN [17], GIN [18] are used to extract matrix features, but they didn't work well. Therefore, a new variant of graph convolutional network is proposed in this paper as the feature extractor (see Section 3.3 for detail).
- We utilize MLP to directly build the mapping between matrix features and the optimal θ, avoiding optimizing the regression model.

The strong threshold θ auto-tuning method is called AutoAMG(θ), and its effectiveness is verified by matrices from the diffusion equations, radiation diffusion equations [7, 19], and time-harmonic Maxwell's equations [20–22]. Numerical experiments show that AutoAMG(θ) can achieve acceleration by a factor of 4.47 compared to the default θ in diffusion equations, a factor of 11.63 compared to the default θ in radiation diffusion equations, and a factor of 1.69 compared to the default θ in time-harmonic Maxwell's equations.

The rest of this paper is organized as follows. Section 2 briefly introduces the rationale behind AMG and shows how θ affects iteration. Section 3 explains the details of AutoAMG(θ). Section 4 presents numerical experiments and results of AutoAMG(θ). Section 5 summarizes our work.

2 Sensitivity of strong threshold

2.1 AMG algorithm

The AMG algorithm can be divided into two phases: SETUP and SOLVE, as described in Algorithms 1 and 2, respectively. Considering the complexity of AMG, we introduce its simplified version, the Two-Grid (TG) algorithm.

In the SETUP phase, the TG algorithm constructs a coarse-level grid, an interpolation matrix P, and a restriction matrix R based on the matrix A. In the SOLVE phase, it performs a standard multigrid cycle based on the matrices generated in SETUP phase, including pre-smoothing, restricting residuals to the coarse grid, solving residual equations in the coarse grid, interpolating the error back to the fine-level grid for correction, and post-smoothing. In particular, if the TG algorithm is called recursively to solve linear equations in the coarse-level grid (line 6, Algorithm 2), it becomes a multigrid algorithm.

Algorithm 1: SETUP phase

```
Coarsening: Construct the fine-level grid based on the matrix $A$ and let $\Omega$ be the
  set containing all fine-level variables. Split the set $\Omega$ into set $C$ containing all
  coarse-level variables and set $F$ containing the remaining fine-level variables,
  according to the strong threshold $\theta$. In addition, $F\cap C=\emptyset$, $F\cup C=\Omega$.
Computing $A_c$: Based on the coarse variable set $C$, compute the interpolation
  matrix $P$ and restriction matrix $R$. Then compute the coarse-level matrix $A_c$ by
  $A_c=RAP$.
```

Algorithm 2: SOLVE phase

```
Pre-smoothing: smoothing $\mu_1$ times on $Ax=b$, get the approximate solution $x_f$
if deepest level then
    Solve $Ax=b$ directly
else
    Restricting residuals into coarse grid: $b_c=R(b-Ax_f)$
    Solving the coarse grid equation: $A_cx_c=b_c$
    Interpolating and correcting: $x_f=x_f+Px_c$
end
Post-smoothing: smoothing $\mu_2$ times on $Ax=b$, update $x_f$
```

In the SETUP phase of the classical AMG, the algorithm will split all variables into a coarse variable set C and fine variable set F (C/F splitting), which is the first step in Algorithm 1. More specifically, let $N_i=\{j \mid a_{ij}\neq 0,\ j\neq i\}$ be the dependency set of variable i, i.e., i strongly depends on j (or j strongly influences i). If

$$|a_{ij}|\geq\theta\max_{k\in N_i,k\neq i}|a_{ik}|, \tag{2.1}$$

where $0<\theta\leq 1$ is the strong threshold, then we can define the strong dependency set S_i and strong influence set S_i^T of variable i,

$$\begin{aligned} S_i&=\left\{j\ \middle|\ |a_{ij}|\geq\theta\max_{k\in N_i,k\neq i}|a_{ik}|,\ j\in N_i\right\},\\ S_i^T&=\left\{j\ \middle|\ i\in S_j,\ j\in N_i\right\}. \end{aligned}$$

According to the definitions, a basic principle of coarsening is that the larger $|S_i^T|$ is, the more important the variable i is, and the more likely it is to be selected as a coarse variable. Following this principle, the result of grid coarsening is closely related to the strength of connectivity between variables, i.e., it relies on the strong threshold θ in Eq. (2.1).

2.2 Multiscale matrix

Multiscale matrices are common in practical problems. Factors such as multimedia (e.g., anisotropy, discontinuity, oscillating coefficients), large deformations, strong nonlinearities, and multiphysics coupling all lead to the multiscale property of matrices obtained by discretization. Define the matrix $A\in\mathbb{R}^{n\times n}$, and let $\Omega=\{0,1,2,\cdots,n\}$ be the set containing all row indices of the matrix. Given a multiscale threshold $\delta\geq 0$, define the multiscale set

$$\Omega_{MS}=\left\{i\;\middle|\; i\in\Omega,\ \log_{10}\left(\frac{\max_{k\in N_i,k\neq i}|a_{ik}|}{\min_{k\in N_i,k\neq i}|a_{ik}|}\right)\geq\delta\right\}. \tag{2.2}$$

If $\Omega_{MS}\neq\emptyset$, then A is defined as a multiscale matrix (under the threshold δ). If $\Omega_{MS}=\emptyset$, then A is a single-scale matrix.

A detailed definition of the multiscale matrix and how the multiscale property affects the AMG algorithm can be found in [7]. From Eq. (2.2), the multiscale property reflects the strength of the numerical difference between the maximum and minimum absolute values of the nondiagonal elements in the same row of the matrix.

2.3 Impact of θ

The effect of θ on the efficiency of the AMG algorithm is illustrated by the diffusion equation below,

$$\begin{aligned} -\nabla\cdot(\kappa\nabla u)&=f_1, \quad x\in\Omega,\\ u&=f_2, \quad x\in\partial\Omega, \end{aligned} \tag{2.3}$$

where κ is the diffusion coefficient. In a two-dimensional (2D) diffusion problem, we define the diffusion coefficient as

$$\kappa=\begin{bmatrix}10^{4\varepsilon} & 0\\ 0 & 1\end{bmatrix},$$

where $0\leq\varepsilon\leq 1$ is a random number.

We theoretically verify the effect of θ based on a specific small matrix whose inverse we can compute. The matrix comes from diffusion equation (2.3), with a random diffusion coefficient κ and a mesh size of 12×12. We gradually drop the element with the minimum absolute value in the matrix to obtain a ”boundary” matrix that is one step from a single-scale matrix. Then the matrix is solved by the TG algorithm, with results as shown in Fig. 1, where the x-axis is θ, in the interval $(0,1)$, with a common difference of 0.01, and the y-axis is the number of iterations (upper limit 500). This shows that there is a critical value $\theta^*=0.26$ in the matrix, where the number of iterations is 8 when $\theta\leq\theta^*$, and 74 when $\theta>\theta^*$.

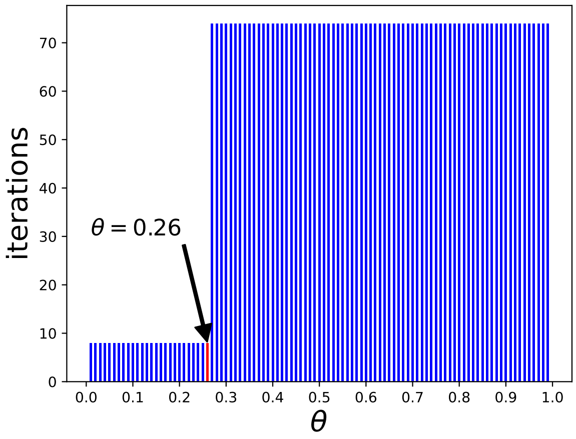

Figure 1: "Boundary" matrix with 144 rows, 218 nonzeros.

Table 1: Theoretical and computed convergence factors.

θ	Theoretical	Computed
0.26	0.2500	0.2498
0.27	0.9477	0.9477

Based on the analysis of the convergence factor in the TG algorithm [23], we compare the theoretically estimated and computed convergence factors in Table 1. The theoretical results remain consistent with the computed results, which indicates that the phenomenon of an oscillating number of iterations is caused by the algorithm itself, and is an essential feature of the algorithm. Such results further illustrate the necessity of optimizing θ.

Furthermore, the numbers of iterations based on two random seeds are depicted in Fig. 2, where the mesh size is 1024×1024, with 1048576 degrees of freedom (DoF). The iterative method is GMRES, with AMG as the precondition and PMIS [24] as the coarsening algorithm. Fig. 2 shows that first, for both random coefficients, the number of iterations changes irregularly with θ, and second, different random seeds have different behaviors.

Table 2 shows the maximum and minimum number of iterations for these cases, as well as the number of iterations corresponding to the default θ. The maximum and minimum number of iterations for both cases are 500 and 7, which means there is a large gap between the maximum and minimum. Moreover, the values of θ corresponding to the maximum are not the same (0.68 and 0.94). Concerning the default value $\theta = 0.25$, the number of iterations is 35 and 95 in two cases, which are 5 and 13 times larger than the corresponding minimum. These results also imply that for random diffusion coefficients,

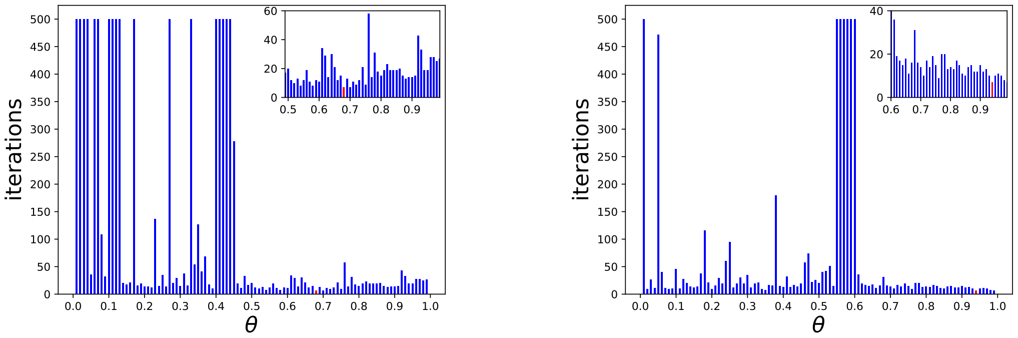

Figure 2: Two random seeds with the same DoF$=1048576$.

Table 2: Min, Max, Default iterations and corresponding θ

	Min/θ	Max/θ	Default ($\theta=0.25$)
Left	7 / 0.68	500 / 0.01	35
Right	7 / 0.94	500 / 0.01	95

the number of iterations is sensitive to the value of θ, and the value of θ corresponding to the minimum is different for different matrices.

3 AutoAMG(θ): Auto-tune θ for multiscale matrices

3.1 AutoAMG(θ) procedure

The comprehensive AutoAMG(θ) procedure is depicted in Fig. 3. The input of AutoAMG(θ) is the matrix, which is treated as the adjacent graph. GNN based on message passing is utilized to extract graph features. Subsequently, AutoAMG(θ) establishes a mapping between these extracted features and the optimal value of θ_{opt}. Note that θ_{opt} pertains to the θ value yielding the fewest iterations during grid search.

The key step in AutoAMG(θ) is feature extraction. Considering matrices discretized from the same equation, their sparsity patterns exhibit a degree of similarity, differing in the number of rows and element values. Notably, the comparison depicted in Fig. 2 illustrates that conventional structural and numerical matrix features (e.g., dimensions, sparsity patterns) fall short of adequately capturing the intricate influence of θ on the iterations across diverse matrices. Besides, the calculation of spectral attributes (e.g., condition number, eigenvalue distribution) is time-consuming, sometimes even surpassing the time required for solving the linear equation. In AutoAMG(θ), GNN is utilized to extract node features in graphs, then graph features are derived based on the extracted node features.

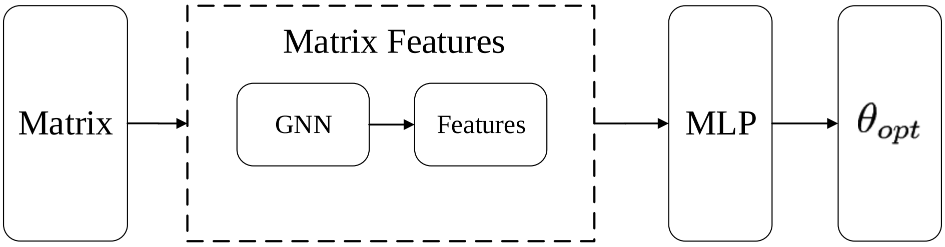

Figure 3: AutoAMG(θ) procedure.

3.2 GNN

GNN is one of the deep learning algorithms specifically designed for the analysis of graph data structure. Nowadays, GNNs are utilized in diverse domains such as social recommendation, traffic prediction, and molecular structure prediction, et al. [25]. A graph G is represented as $G=(V,E)$, where V is the set of nodes and E is the set of edges in the graph. The number of nodes is $|V|$ and the number of edges is $|E|$. Let $v_i \in V$ denote the i-th node and $e_{ij}=(v_i,v_j)\in E$ denote the directed edge from node v_i to node v_j. Let $N(v)$ denote all neighbor nodes of the node v. Since every node and edge may have features, let $X_v \in \mathbb{R}^{|V|\times d}$ denote feature matrix of all nodes and $X_e \in \mathbb{R}^{|E|\times c}$ denote feature matrix of all edges, where $X_{v_i} \in \mathbb{R}^d$ is the feature vector of the i-th node and $X_{e_{ij}} \in \mathbb{R}^c$ is the feature vector of edge e_{ij}. Let X_g denote the feature vector of the graph.

The standard operation of a GNN involves the following process: commencing with the initial node feature vector $X_v^{(0)}$ and edge feature vector $X_e^{(0)}$, diverse GNN variants employ distinct strategies to iteratively update these feature vectors for nodes and edges. This evolution is often visualized as a mechanism of message passing that transpires among the nodes within the graph, whose formula is[†]

$$X_{v_i}^{(k)}=\gamma^{(k)}\left(X_{v_i}^{(k-1)},Aggr_{j\in N(i)}^{(k)}\,\phi^{(k)}(X_{v_i}^{(k-1)},X_{v_j}^{(k-1)},X_{e_{ji}})\right), \tag{3.1}$$

where $\phi^{(k)}$, $Aggr$ and $\gamma^{(k)}$ are three kernel functions of the GNN algorithm:

- $\phi^{(k)}$ is the message function that dictates the content of messages propagated by the neighboring nodes and edges of node v_i;
- $Aggr^{(k)}$ is the aggregation function that defines the approach taken to process the sent messages;
- $\gamma^{(k)}$ is the update function that specifies how the node feature vector $X_{v_i}^{(k-1)}$ and the aggregated messages are combined to derive the updated node feature vector $X_{v_i}^{(k)}$.

[†] https://pytorch-geometric.readthedocs.io/en/latest/tutorial/create_gnn.html

These three functions can either be differentiable functions or MLPs. Each message passing step corresponds to a GNN layer, and these functions may vary across different layers. After K steps, the resultant node feature vector $X_{v_i}^{(K)}$ is used for downstream tasks, such as node classification. It's worth noting that Eq. (3.1) focuses on the nodes within the graph, while there exist GNNs that involve the updating of the edge feature vector $X_{e_{ji}}$ [26]. Utilizing $X_{v_i}^{(K)}$, the computation of the graph feature vector X_g is facilitated via a Readout function. A variety of Readout functions are available for selection, such as the SUM function

$$X_g = \sum_{i=1}^{|V|} X_{v_i}^{(K)}, \tag{3.2}$$

which is the sum of all nodes features; or MEAN function

$$X_g = \frac{1}{|V|}\sum_{i=1}^{|V|} X_{v_i}^{(K)}, \tag{3.3}$$

which is the average of all nodes features, et al.

At first, we tried to use GCN [17] and GIN [18] to extract graph features. According to Eq. (3.1), a single GCN layer is defined as

$$X_{v_i}^{(k)} = MLP^{(k)}\left(\sum_{v_j\in N(v_i)\cup v_i} \frac{w_{ji}}{\sqrt{\hat{D}_i\hat{D}_j}} X_{v_j}^{(k-1)}\right), \tag{3.4}$$

where w_{ji} is the weight of edge e_{ji}, if the graph is unweighted, then $w_{ji}=1$; $\hat{D}$ is the diagonal degree matrix and $\hat{D}_i$ is the degree of node v_i in the graph. In GCN, $\phi^{(k)}$ is the feature vector of neighbor nodes, $Aggr^{(k)}$ is weighted average, and $\gamma^{(k)} = MLP^{(k)}$. MEAN function (Eq. (3.3)) commonly serves as the Readout function of GCN. Besides, the corresponding matrix form of one layer of GCN is

$$X_v^{(k)} = \hat{D}^{-\frac{1}{2}} A \hat{D}^{-\frac{1}{2}} X_v^{(k-1)} \Theta^{(k)}, \tag{3.5}$$

where matrix $\Theta^{(k)}$ is the weight matrix of the $MLP^{(k)}$ that needed to be optimized during training. If the number of node features is $d^{(k-1)}$ in the $(k-1)$-th layer and $d^{(k)}$ in the k-th layer, then $X_v^{(k)} \in \mathbb{R}^{|V|\times d^{(k)}}$, $A\in\mathbb{R}^{|V|\times|V|}$, $\hat{D}\in\mathbb{R}^{|V|\times|V|}$, $X_v^{(k-1)}\in\mathbb{R}^{|V|\times d^{(k-1)}}$, $\Theta^{(k)}\in\mathbb{R}^{d^{(k-1)}\times d^{(k)}}$. Based on Eq. (3.5), it is evident that the weight matrix $\Theta^{(k)}$ is related to the number of features per node and remains independent of the number of nodes $|V|$ within the graph. Therefore, GCN can handle graphs with different sizes.

A single GIN layer is defined as

$$X_{v_i}^{(k)} = MLP^{(k)}\left(w_{ii}(1+\epsilon^{(k)})\cdot X_{v_i}^{(k-1)} + \sum_{v_j\in N(v_i)} w_{ji} X_{v_j}^{(k-1)}\right), \tag{3.6}$$

where w_{ii} is the weight of node v_i's self loop, w_{ji} is the weight of edge e_{ji}, and $\epsilon^{(k)}$ can be a trainable parameter or a fixed constant number. Compared to Eq. (3.4), the aggregation function $Aggr^{(k)}$ is summation. Similarly, The matrix form of one layer of GIN is

$$X_v^{(k)} = \left[A + (1+\epsilon^{(k)}) \right] X_v^{(k-1)} \Theta^{(k)}. \tag{3.7}$$

The authors of GIN demonstrated that in specific scenarios, the MEAN and MAX functions would impair the expressiveness of the GNN. Consequently, both the aggregation and Readout functions in GIN are summation rather than average. The recommended Readout function for GIN is

$$\begin{aligned} &X_g = \text{CONCAT}\left(\text{SUM}\left(X_v^{(k)} \right) \middle| k = 0,1,\cdots K \right), \\ &\text{SUM}\left(X_v^{(k)} \right) = \sum_{i=1}^{|V|} X_{v_i}^{(k)}, \end{aligned} \tag{3.8}$$

where CONCAT is the concatenation function that concatenate several vectors into a long vector.

3.3 GCIN

We choose GCN and GIN from the existing GNNs for matrix feature extraction due to their low computational complexity ($\mathcal{O}(N)$). Moreover, each layer can be implemented using Sparse Matrix-Vector Multiplication (SpMV) operations, facilitating integration of these GNNs into existing iterative software frameworks.

However, our experimental results revealed that GCN and GIN did not yield satisfactory outcomes. The issue with GCN was the occurrence of NAN (Not A Number) errors during the training phase. Upon conducting a thorough debugging process, we identified the source of these NAN errors to be the degree matrix $\hat{D}$ in Eq. (3.4). These matrices in the data set originate from the PDE discretization. Therefore, some row-sums are equal to 0, indicating that certain $\hat{D}_i = 0$, which results in $\hat{D}_i^{-1/2}$ being "INF" and "NAN" in the following computation. When $|a_{ij}|$ was utilized as the edge weight and matrices from 2D diffusion equations (Section 4.1.1) were used as data set, the training process of GCN finished without issues. However, in the test set, the computational efficiency of the θ predicted by GCN was inferior to that predicted by GCIN.

The problem encountered with GIN pertained to the absence of a reduction in the loss value during training, as shown in Fig. 4. This phenomenon is plausible given the nature of this problem, where the absence of normalization in GIN (refer to Eq. (3.6) and Eq. (3.8)) allows values to accumulate, consequently impeding the convergence process.

The experiments of GIN reveal that normalization is essential for our problem. Nonetheless, improper normalization can lead to NAN errors during training. After testing and analyzing, we introduce the Graph Convolutional Isomorphism Network

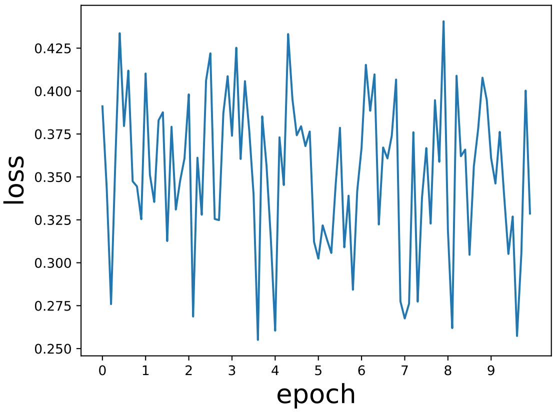

Figure 4: The training process of GIN.

(GCIN), which amalgamates the attributes of both GCN and GIN. A single layer of GCIN is defined as

$$X_{v_i}^{(k)} = MLP^{(k)}\left(\sum_{v_j \in N(v_i) \cup v_i} w_{ji} X_{v_j}^{(k-1)}\right), \tag{3.9}$$

and the matrix form is

$$X_v^{(k)} = AX_v^{(k-1)}\Theta^{(k)}. \tag{3.10}$$

The Readout function is

$$X_g = \sum_{k=1}^{K}\left(\frac{1}{|V|}\sum_{i=1}^{|V|} X_{v_i}^{(k)}\right). \tag{3.11}$$

Notably, normalization is integrated within the Readout function rather than being incorporated into the message passing process.

3.4 Optimizing strong threshold θ

Following the extraction of matrix features, the subsequent phase involves the optimization of the strong threshold θ. A conventional approach encompasses training a regression model, where matrix features and θ are inputs, and the performance metric (such as computation time, iteration count, or convergence factor) serves as the output. Then the optimization of θ relies on this regression model. Here, let the graph feature vector X_g denote the matrix features, y denote the performance metric, and f denote the regression function. Consequently, the regression model is expressed as follow

$$y = f(X_g, \theta). \tag{3.12}$$

Upon completion of the training phase, the regression function f is established. Given any matrix, the optimization problem can be written as

$$\max_{\theta\in(0,1)} y=f(X_g,\theta),$$

which is a black-box optimization problem. To circumvent the need for solving this problem, we forego the creation of a regression model like Eq. (3.12), opting to establish a direct mapping between matrix features and the optimal θ:

$$\theta_{opt}=g(X_g),$$

where g is the mapping constructed through MLP. Let θ_{auto} denote the predicted value of θ by AutoAMG(θ), and θ_{opt} denote the optimal value of θ. We use MSE (Mean Squared Error) [27] function as the loss function, then the loss is defined as

$$\begin{aligned} Loss &= MSE(\theta_{opt},\theta_{auto}) \\ &= \frac{1}{M}\sum_{i=1}^{M}(\theta_{opt,i}-\theta_{auto,i})^2, \end{aligned} \tag{3.13}$$

where M is the batch size, $\theta_{opt,i}$ is the optimal θ value of the i-th matrix in the batch and $\theta_{auto,i}$ is the predicted θ value of the i-th matrix in the batch. The program of GCIN and optimization are implemented by PyTorch Geometric [28].

4 Numerical experiments

We validated the effectiveness of AutoAMG(θ) based on three types of problems: the diffusion equations (in Section 4.1), the 3D radiation diffusion equations from inertial confinement fusion (in Section 4.2), and the 3D time-harmonic Maxwell's equations (in Section 4.3). The matrix generation programs are available on github[‡].

The optimal θ for each matrix is determined through grid search. We calculate the number of iterations by considering values of θ in increments of 0.01 within the range of [0.01,0.99]. The optimal θ is chosen as the one that results in the minimum number of iterations. The linear equations are solved using the JXPAMG software [29], utilizing the GMRES algorithm with the AMG preconditioner. The coarsening algorithm in AMG is PMIS. We set an upper limit of 500 iterations, and the stopping criterion is that the relative residual is less than 10^{-8}.

In the experiments, the GCIN model comprises three layers. The initial node features have a dimension of 1, representing the node degrees. Then the input feature dimension is equal to 1 in the first layer of GCIN, and the output feature dimension is set to 32. The input and output feature dimensions for the second and third layers are also both set to

[‡] https://github.com/zhf-0/autoamg-matrix

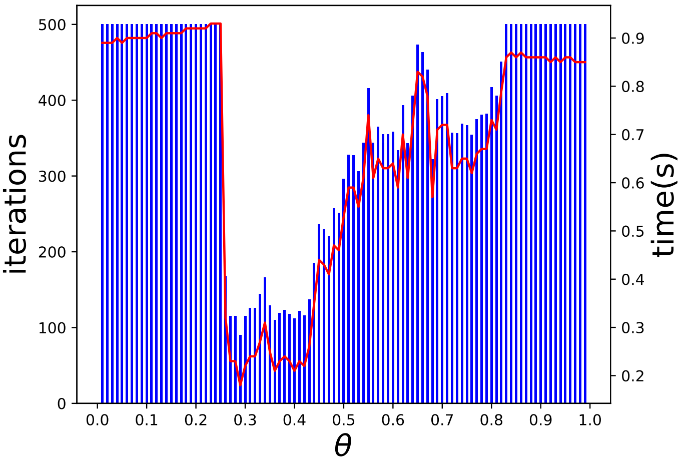

Figure 5: The matrix is from 2D diffusion equation with 9409 rows. Histogram is the number of iterations with left y-axis, and red line is the elapsed time with right y-axis.

32. Consequently, the matrix features extracted by GCIN have a dimensionality of 32. Furthermore, the MLP used to approximate the mapping between matrix features and the optimal θ consists of only one hidden layer. It has an input dimension of 32 and an output dimension of 1.

Remark 4.1. The number of iterations is selected as the performance metric. While considering the operator complexity of AMG is closely related to the value of θ, the elapsed time may seem like a preferable alternative. However, after plotting the number of iterations and time in the same picture (Fig. 5), it is evident that their trends are quite similar. Furthermore, given the matrix sizes in our experiments, some elapsed times are too short for precise measurement and are susceptible to the runtime environment. In contrast, the number of iterations remains unaffected by the environment. Hence, we have decided to utilize the number of iterations as our primary metric.

The meaning of notations in the following tables are similar. Take Table 3 as an example, the first column "nrow" is the average number of rows of matrices in the test set; "iter" is the average number of iterations; "time" is the average time used to solve linear equations in the test set. In the column of "AutoAMG(θ)", the "iter" and "time" correspond to the average number of iterations and average time based on the θ predicted by AutoAMG(θ). The column "speedup" is the average time of default θ ($\theta=0.25$ in 2D, $\theta=0.5$ in 3D) divided by the average time of AutoAMG(θ).

4.1 Diffusion equations

Diffusion equations (Eq. (2.3)) include the 2D and 3D cases. The domain is $[0,1]^d (d=2,3)$, and the diffusion coefficients are

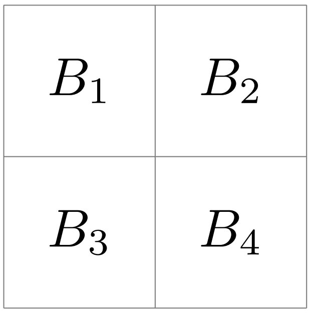

Figure 6: An example of $bx=by=2$ blocks (B_i, $i=1,2,3,4$) with equal size. Diffusion coefficient κ is the same in each block, when $B_i \neq B_j$, $\kappa_i \neq \kappa_j$.

$$\kappa=\begin{bmatrix} 10^{Mr_0} & 0 \\ 0 & 10^{Mr_1} \end{bmatrix}, \quad \kappa=\begin{bmatrix} 10^{Mr_0} & 0 & 0 \\ 0 & 10^{Mr_1} & 0 \\ 0 & 0 & 10^{Mr_2} \end{bmatrix}, \tag{4.1}$$

where r_0, r_1, and r_2 are random numbers in the interval $(0,1)$, and $M \in \mathbb{N}_+$ is the parameter that influences the multiscale property of the matrix. A larger value of M generally leads to a more pronounced multiscale property within the generated matrix. The computational domain is uniformly divided into blocks or subdomains with equal size, as shown in Fig. 6. While the diffusion coefficient κ remains consistent within each block, it differs between different blocks. Therefore, even with identical mesh size and block count, different random seeds can generate distinct matrices.

When discretizing Eq. (2.3), matrices with varied properties and sizes can be generated by selecting different random number seeds *Seed*, mesh sizes nx,ny,nz in each axis direction, block counts bx,by,bz in each axis direction, and the parameter M. More specifically, the matrix data used in experiments are obtained from the following two cases:

- 2D diffusion equations: $nx=ny \in (50,100)$, $bx=by \in (10,20)$, $M=5$, and random seed *Seed* is equal to the index of the matrix.
- 3D diffusion equations: $nx=ny=nz \in (30,40)$, $bx=by=bz \in (10,20)$, $M=5$, and random seed *Seed* is equal to the index of the matrix.

4.1.1 2D diffusion equations

The training and test sets consist of 80 and 20 matrices respectively. The mesh size $nx=ny \in (50,100)$ and the number of blocks $bx=by \in (10,20)$ are both random values. The test results are shown in Table 3.

Our objective is to assess the solving efficiency of the θ predicted by AutoAMG(θ) in comparison to the default θ. Given that the default value of θ for 2D problems is 0.25,

Table 3: Test results of 2D diffusion equations.

nrow	optimal θ		$\theta=0.25$		AutoAMG(θ)		speedup
	iter	time(s)	iter	time(s)	iter	time(s)	
5659	185.25	0.15	496.20	0.38	257.30	0.21	1.81

Table 3 presents the number of iterations and time corresponding to $\theta=0.25$. Despite the improved solving efficiency achieved by AutoAMG(θ), a noticeable gap remains between the attained performance and the optimal one.

4.1.2 3D diffusion equations

The training and test sets consist of 80 and 20 matrices respectively. The mesh size $nx=ny=nz\in(30,40)$ and number of blocks $bx=by=bz\in(10,20)$ are random values. The results of the test set are shown in Table 4, and the notations used are similar to those in Table 3. In 3D equations, The number of iterations and time tuned by AutoAMG(θ) are close to the optimal ones, which is a significant improvement over the default value $\theta=0.5$.

Table 4: Test results of 3D diffusion equations.

nrow	optimal θ		$\theta=0.5$		AutoAMG(θ)		speedup
	iter	time(s)	iter	time(s)	iter	time(s)	
40515	34.00	0.29	233.20	1.52	42.75	0.34	4.47

4.1.3 Mixed 2D and 3D diffusion equations

A more common scenario arises when the origin of a matrix is unknown, making it challenging to determine whether it was discretized from a 2D or 3D problem. In such cases, AutoAMG(θ) is required to process the input matrix without additional information. Matrices from 2D and 3D diffusion equations are combined to make up the training and test sets, comprising 160 and 40 matrices respectively. To ensure a balanced distribution of matrix data, half of the data originates from 2D problems and the remaining half from 3D problems, both in the training and test sets. Since the dimension is unknown, we calculate the average number of iterations and computation time for all matrices in the test set at $\theta=0.25$ and $\theta=0.5$, as displayed in Table 5.

From Table 5, it is evident that the predicted θ by AutoAMG yields higher solving efficiency compared to default values of $\theta=0.25$ and $\theta=0.5$. However, training with mixed matrices results in a less robust model. The speedup over $\theta=0.25$ and $\theta=0.5$ is 1.34 and 3.10, whereas the speedup in Table 3 and 4 are 1.81 and 4.47. Consequently, it is advisable to train the model using matrices from the same dimension.

Table 5: Test results of the mixed problems.

optimal θ		$\theta=0.25$		$\theta=0.5$		AutoAMG(θ)		speedup	
iter	time(s)	iter	time(s)	iter	time(s)	iter	time(s)	0.25	0.5
109.63	0.22	273.83	0.39	291.00	0.90	179.88	0.29	1.34	3.10

4.2 3D radiation diffusion equations

The matrices employed in the previous sections originate from diffusion equations, containing fewer than 5×10^4 rows. To ascertain the generalizability of AutoAMG(θ), we employ all matrices from Section 4.1.2 for training and 10 matrices discretized from 3D radiation diffusion equations (with approximately 6.29×10^6 rows) for testing. Experimental results confirm that AutoAMG(θ) can be trained on smaller matrices and subsequently applied to larger matrices.

The formulas of 3D radiation diffusion equations [7,30] are

$$\begin{aligned} c_{vr}\frac{\partial T_r}{\partial t}-\frac{1}{\rho}\nabla\cdot(K_r\nabla T_r)&=\omega_{er}(T_e-T_r),\\ c_{ve}\frac{\partial T_e}{\partial t}-\frac{1}{\rho}\nabla\cdot(K_e\nabla T_e)&=\omega_{ei}(T_i-T_e)+\omega_{er}(T_r-T_e),\\ c_{vi}\frac{\partial T_i}{\partial t}-\frac{1}{\rho}\nabla\cdot(K_i\nabla T_i)&=\omega_{ei}(T_e-T_i), \end{aligned} \tag{4.2}$$

where ρ is the density; T_r,T_e,T_i are the temperatures of photons, electrons, and ions, respectively; c_{vr},c_{ve},c_{vi} are the specific heat at constant volume of photons, electrons, and ions, respectively; $K_r=f_r(\rho,T_r)$, $K_e=f_e(\rho,T_e)$ and $K_i=f_i(\rho,T_i)$ (f_r,f_e,f_i are functions) are diffusion coefficients; and ω_{ei} and ω_{er} are the respective energy exchange coefficients between electrons and ions, and electrons and photons. Eq. (4.2) is a nonlinear partial differential equation. It is discretized in time by the backward Euler method, then the nonlinear problem is transformed into a linear problem by the coagulation coefficient method, and the linear problem is discretized by the finite volume method. The sparse pattern of the discretized matrix is

$$A=\begin{bmatrix} A_R & D_{RE} & 0\\ D_{ER} & A_E & D_{EI}\\ 0 & D_{IE} & A_I \end{bmatrix}. \tag{4.3}$$

The block matrices A_R, A_E, A_I in Eq. (4.3) have the same sparse pattern, and the block matrices D_{RE}, D_{EI} are diagonal matrices.

The training set consists of 100 matrices from 3D diffusion equations (Section 4.1.2), while the test set includes 10 matrices from Eq. (4.2). The results are shown in Table 6. The number of iterations and computation time based on the θ predicted by AutoAMG(θ) are close to optimal ones, which is a substantial improvement compared to the default $\theta=0.5$.

Table 6: Test results of 3D radiation diffusion equations.

nrow	optimal θ		$\theta=0.5$		AutoAMG(θ)		speedup
	iter	time(s)	iter	time(s)	iter	time(s)	
6291456	31.50	31.52	484.20	399.00	35.40	34.27	11.63

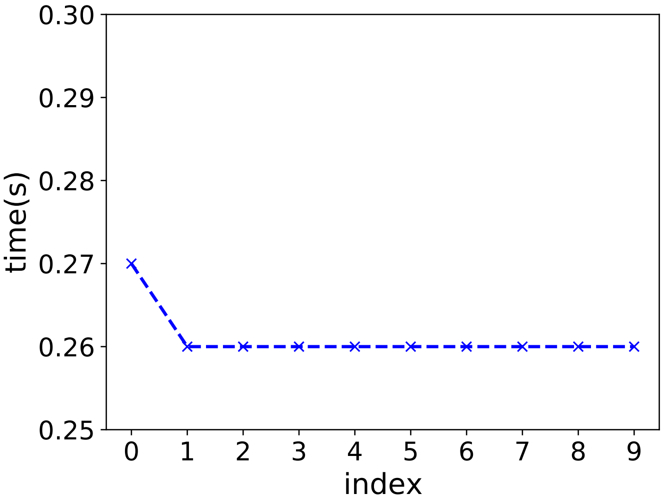

Figure 7: The inference time of each matrix in the test set.

Moreover, in contrast with the speedup shown in Table 4, AutoAMG(θ) demonstrates the capability to achieve even greater speedup. Such results illustrate the benefit of tuning θ in practical problems.

In terms of the overhead induced by AutoAMG(θ), we measure the inference time of each matrix in the test set, and the results are shown in Fig. 7. The x axis in the figure is the index of the matrix, and the y axis is the inference time. Note that the average inference time is 0.26 s, which is negligible compared to the average solving time of 34.27 s in Table 6. In fact, according to Eq. (3.9), the message-passing process of GCIN can be effectively realized through the SpMV operation, hence it's conceivable that the overhead of GCIN would be inconsequential.

4.3 3D time-harmonic Maxwell's equations

The time-harmonic Maxwell's equations are derived from the original Maxwell's equations, which describe the behavior of electromagnetic fields in the time domain. The time-harmonic Maxwell's equations are particularly useful when dealing with electromagnetic waves and phenomena at a single frequency, such as those found in radio waves,

Table 7: Test results of 3D time-harmonic Maxwell's equations.

nrow	optimal θ		$\theta=0.5$		AutoAMG(θ)		speedup
	iter	time(s)	iter	time(s)	iter	time(s)	
60508	234.45	7.70	535.95	13.54	245.50	8.02	1.69

microwaves, and optical frequencies. Solving $Ax=b$ from the discretization of time-harmonic Maxwell's equations can be significant challenges due to the ill-conditioning associated with high wave numbers.

The formulas of the 3D time-harmonic Maxwell's equations [20–22] are:

$$\begin{aligned} \nabla\times(\nabla\times E)-k^2E&=f, \quad E\in\Omega,\\ n\times E&=g, \quad E\in\partial\Omega, \end{aligned} \tag{4.4}$$

where $E\in\mathbb{R}^3$ is the electric field, k^2 is the wave number, n is the outer normal vector, f is the source item and g is the Dirichlet boundary condition. Eq. (4.4) are discretized using the Nédélec finite element method [31] on a tetrahedral mesh. The mesh size is defined as $nx=ny=nz\in(10,40)$, then the number of rows in the matrices varies between 7930 and 462520. The wave number k^2 is within the range of $(1,15)$.

Solving linear algebraic equations derived from time-harmonic Maxwell's equations is more challenging than those stemming from diffusion equations. As a result, in this experiment, we set the upper limit of iterations to 1000 and the relative residual to be less than 10^{-7}. The experiment employed training and test sets comprising 80 and 20 matrices, respectively. Table 7 displays the results of the test set. While the speedup achieved by AutoAMG(θ) may not be exceptionally significant, its efficiency closely approaches the optimal performance.

5 Summary

In this paper, we propose AutoAMG(θ), an auto-tuning method designed to adaptively adjust the strong threshold θ in the AMG algorithm for matrices from different problems. The effectiveness of this method is verified through a variety of numerical experiments.

An innovative contribution of this paper is the introduction of the GCIN algorithm for extracting matrix features. In diffusion problems, when compared to default θ, the AutoAMG(θ) method based on GCIN demonstrates a speedup by a factor of 1.81 in 2D diffusion problems and 4.47 in 3D diffusion problems. Furthermore, AutoAMG(θ) displays versatility by effectively handling matrices from both 2D and 3D problems. Although it shows superior efficiency compared to default values, the speedup is only 1.34 in 2D problems and 3.10 in 3D problems.

Notably, in 3D radiation diffusion problems, AutoAMG(θ) effectively tunes the number of iterations and time that are close to the optimal results, achieving an impressive acceleration by a factor of 11.63 over the default $\theta = 0.5$. The experiments reveal that AutoAMG(θ) generalizes well to new large matrices after training on small matrices.

AutoAMG(θ) can also be used to accelerate computation in 3D time-harmonic Maxwell's equations, achieving an acceleration by a factor of 1.69 over the default $\theta = 0.5$. Additionally, the number of iterations of the predicted θ closely approximates the optimal result, indicating that AutoAMG(θ) has the potential to attain optimality in different problems.

AutoAMG(θ) serves as a proof of concept, which illustrates the applicability of GNN-based feature extraction methods for optimizing parameters of iterative methods. This strategy can be applied to different equations and different iterative methods.

Our future research will continue to focus on AMG algorithm optimization, using GNN to optimize the smoothing, interpolation, restriction, and other operators in AMG.

Acknowledgments

This work is financially supported by the National Natural Science Foundation of China (62032023).

References

[1] John W Ruge and Klaus Stüben. Algebraic Multigrid. In *Multigrid methods*, pages 73–130. SIAM, 1987.

[2] K. Stüben. A Review of Algebraic Multigrid. *J. Comput. Appl. Math.*, 128(1):281–309, 2001. Numerical Analysis 2000. Vol. VII: Partial Differential Equations.

[3] Jinchao Xu and Ludmil Zikatanov. Algebraic Multigrid Methods. *Acta Numer.*, 26:591–721, 2017.

[4] Robert D Falgout and Ulrike Meier Yang. HYPRE: A Library of High Performance Preconditioners. In *International Conference on Computational Science*, pages 632–641. Springer, 2002.

[5] S Vakili and M Darbandi. Recommendations on Enhancing The Efficiency of Algebraic Multigrid Preconditioned GMRES in Solving Coupled Fluid Flow Equations. *Numer. Heat Transf. Part B Fundam.*, 55(3):232–256, 2009.

[6] Nikola Kosturski, Svetozar Margenov, Peter Popov, Nikola Simeonov, and Yavor Vutov. Performance Analysis of Block AMG Preconditioning of Poroelasticity Equations. In *Large-Scale Scientific Computing: 10th International Conference, LSSC 2015, Sozopol, Bulgaria, June 8-12, 2015. Revised Selected Papers 10*, pages 377–384. Springer, 2015.

[7] Xiaowen Xu and Zeyao Mo. Algebraic Interface-Based Coarsening AMG Preconditioner for Multiscale Sparse Matrices with Applications to Radiation Hydrodynamics Computation. *Numer. Linear Algebra Appl.*, 24(2):e2078, 2017.

[8] Paola F Antonietti, Matteo Caldana, and Luca Dede. Accelerating Algebraic Multigrid Methods via Artificial Neural Networks. *Vietnam Journal of Mathematics*, pages 1–36, 2023.

[9] America Holloway and Tzu-Yi Chen. Neural Networks for Predicting The Behavior of Preconditioned Iterative Solvers. In *International Conference on Computational Science*, pages 302–309. Springer, 2007.

[10] Sanjukta Bhowmick, Victor Eijkhout, Yoav Freund, Erika Fuentes, and David Keyes. Application of Machine Learning to The Selection of Sparse Linear Solvers. *Int. J. High Perform. Comput. Appl.*, 2006.

[11] Paul R. Eller, Jing Ru C. Cheng, and Robert S. Maier. Dynamic Linear Solver Selection for Transient Simulations Using Multi-Label Classifiers. In *Procedia Computer Science*, volume 9, pages 1523–1532. Elsevier B.V., 2012.

[12] Pate Motter, Kanika Sood, Elizabeth Jessup, and Boyana Norris. Lighthouse: An Automated Solver Selection Tool. In *Proceedings of the 3rd International Workshop on Software Engineering for High Performance Computing in Computational Science and Engineering*, pages 16–24, 2015.

[13] Alexandr Katrutsa, Talgat Daulbaev, and Ivan Oseledets. Deep Multigrid: Learning Prolongation And Restriction Matrices. *arXiv preprint arXiv:1711.03825*, 2017.

[14] Daniel Greenfeld, Meirav Galun, Ron Kimmel, Irad Yavneh, and Ronen Basri. Learning to Optimize Multigrid PDE Solvers. *arXiv preprint arXiv:1902.10248*, feb 2019.

[15] Ilay Luz, Meirav Galun, Haggai Maron, Ronen Basri, and Irad Yavneh. Learning Algebraic Multigrid Using Graph Neural Networks. *arXiv preprint arXiv:2003.05744*, mar 2020.

[16] Yuyan Chen, Bin Dong, and Jinchao Xu. Meta-Mgnet: Meta Multigrid Networks for Solving Parameterized Partial Differential Equations. *J. Comput. Phys.*, 455:110996, 2022.

[17] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations*, 2017.

[18] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How Powerful Are Graph Neural Networks? *arXiv preprint arXiv:1810.00826*, 2018.

[19] Xu Xiaowen, Mo Zeyao, and An Hengbin. Algebraic Two-Level Iterative Method for 2-D 3-T Radiation Diffusion Equations. *Chinese J. Comput. Phys.*, 26(1):1, 2009.

[20] M El Bouajaji, Victorita Dolean, Martin J Gander, and Stephane Lanteri. Optimized Schwarz methods for the time-harmonic Maxwell equations with damping. *SIAM Journal on Scientific Computing*, 34(4):A2048–A2071, 2012.

[21] Ana Alonso and Alberto Valli. An optimal domain decomposition preconditioner for low-frequency time-harmonic Maxwell equations. *Mathematics of Computation*, 68(226):607–631, 1999.

[22] Chen Greif and Dominik Schötzau. Preconditioners for the discretized time-harmonic Maxwell equations in mixed form. *Numerical Linear Algebra with Applications*, 14(4):281–297, 2007.

[23] Robert D Falgout, Panayot S Vassilevski, and Ludmil T Zikatanov. On Two-Grid Convergence Estimates. *Numer. Linear Algebra Appl.*, 12(5-6):471–494, 2005.

[24] Michael Luby. A Simple Parallel Algorithm for The Maximal Independent Set Problem. In *Proceedings of the seventeenth annual ACM symposium on Theory of computing*, pages 1–10, 1985.

[25] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S Yu Philip. A Comprehensive Survey on Graph Neural Networks. *IEEE Trans. Neural Netw. Learn. Syst.*, 32(1):4–24, 2020.

[26] Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E Sarma, Michael M Bronstein, and Justin M Solomon. Dynamic graph cnn for learning on point clouds. *ACM Transactions on Graphics (TOG)*, 38(5):1–12, 2019.

[27] Peter J Bickel and Kjell A Doksum. *Mathematical Statistics: Basic Ideas and Selected Topics, Volumes I-II Package*. CRC Press, 2015.

[28] Matthias Fey and Jan E. Lenssen. Fast Graph Representation Learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.

[29] Xiaowen Xu, Xiaoqiang Yue, Runzhang Mao, Yuntong Deng, Silu Huang, Haifeng Zou, Xiao

Liu, Shaoliang Hu, Chunsheng Feng, Shi Shu, et al. JXPAMG: A Parallel Algebraic Multigrid Solver for Extreme-Scale Numerical Simulations. *CCF Trans. HPC (2022)*, pages 1–12, 2022.

[30] Silu Huang, Xiaowen Xu, et al. αSetup-PCTL: An Adaptive Setup-Based Two-Level Preconditioner for Sequence of Linear Systems of Three-Temperature Energy Equations. *Commun. Comput. Phys.*, 32(5):1287–1309, 2022.

[31] Jean-Claude Nédélec. Mixed finite elements in $\mathbb{R}^3$. *Numerische Mathematik*, 35:315–341, 1980.