COMPUTER METHODS IN MATERIALS SCIENCE

2026, vol. 26, no. 1, 43–47
https://doi.org/10.7494/cmms.2026.1.1039

# Artificial intelligence-enhanced algebraic multigrid for 3D finite element simulations

Damian Goik, Krzysztof Banaś*

AGH University of Krakow, al. Mickiewicza 30, 30-059 Krakow, Poland.

## Abstract

The paper presents preliminary investigations into a strategy for solving linear systems resulting from 3D finite element simulations, based on the algebraic multigrid (AMG) method, enhanced using artificial intelligence techniques. In particular, we adapt to 3D problems the algorithm presented in Luz et al. (2020) for using a graph neural network to create the prolongation and restriction operators in a way that will improve convergence. The process of training the network proceeds on the basis of a set of system matrices obtained for tasks much smaller in scale than the target problem to be solved. Learning is aimed at decreasing the spectral radius of the error propagation matrix after applying modified prolongation and restriction. We describe some implementation details of the solver developed based on the presented strategy and show several numerical examples of its application for medium-sized problems.

**Keywords:** graph neural networks, algebraic multigrid, finite element method

## 1. Introduction

In the past few years, the progress in developing new applications of artificial intelligence in the process of numerical solutions of partial differential equations has been particularly rapid. Methods have been designed to optimise various stages of the process, from grid preparation, through approximation techniques, to solvers of linear equations. The paper by Lagaris et al. (1998) presents a backpropagation algorithm for a neural network designed to predict the solution of a given equation outside the computational domain. Other authors use artificial intelligence to improve well-known methods. In the article by Hsieh et al. (2019), a successful approach is described for adding to each iteration of an iterative solver a component derived from a convolutional network, trained to minimise the errors of the iterative method used for given grids. The method described by Mishra (2018) applies artificial intelligence to search for the best parameters of known time integration schemes, thus obtaining an error reduction of 2 or even 3 orders of magnitude. In the context of solving systems of linear equations, the paper by Katrutsa et al. (2017) presents a method for improving the selection of parameters for the geometric multigrid method in order to minimise the spectral radius of the iteration matrix of the two-level method.

Since their inception in the report by Gori et al. (2005), graph neural networks have gained multiple applications in different domains, including numerical solvers. This approach was later extended to Message Passing Neural Networks in Gilmer et al. (2017).

In the current paper, we present the further development of the solver for ill-conditioned linear systems, described previously by Goik & Banaś (2020). We concentrate on the solution of systems for scalar unknowns (previously, we considered the pressure part of the

* Corresponding author
Authors' e-mails: dagoik@gmail.com, kbanas@agh.edu.pl
ORCID ID: 0000-0002-4045-1530 (K. Banaś)
Received: 19.03.2026, accepted: 14.04.2026, published: 30.04.2026

system obtained by stabilised FEM discretisation of the incompressible Navier–Stokes equations). We enhance the AMG algorithm used for this purpose using a graph neural network. To this end, we adapt the method presented in Luz et al. (2020). We describe the algorithm, its implementation details, and an example of using it for 3D ill-conditioned diffusion problems with large jumps in coefficients.

## 2. Algebraic multigrid solution process

In order to solve the linear systems $\boldsymbol{A}\,\boldsymbol{x} = \boldsymbol{b}$ resulting from 3D scalar finite element simulations, the current paper employs a solver that uses a sequence of standalone multigrid V-cycles. Each V-cycle recursively applies the two-grid algorithm:

- perform $N_{pre}$ pre-smoothing steps for the initial vector $\boldsymbol{x}_i$,
- calculate the residual on the fine grid and project it onto the coarse grid: $\boldsymbol{r} := \boldsymbol{P}^T(\boldsymbol{b} - \boldsymbol{A}\,\boldsymbol{x}_i)$,
- solve the coarse-grid correction problem: $\boldsymbol{z} := (\boldsymbol{P}^T\boldsymbol{A}\,\boldsymbol{P})^{-1}\,\boldsymbol{r}$,
- correct the current solution: $\boldsymbol{x}_i := \boldsymbol{x}_i + \boldsymbol{P}\,\boldsymbol{z}$,
- perform $N_{post}$ post-smoothing steps.

The algorithm starts with an initial guess $x_i$, performs several pre-smoothing steps, applies coarse-grid correction, and finally performs several post-smoothing steps. In the investigations described in the paper, we always use a single Gauss–Seidel iteration for both pre- and post-smoothing. The presented solver algorithm can be easily extended to an AMG preconditioner for Krylov subspace methods.

The most specific for multigrid and the most important for the convergence properties of the method are the steps involving interpolation operators $\boldsymbol{P}$, from the coarse grid to the fine grid.

In the AMG context, the operators are constructed based on purely algebraic considerations, with the particular form of the system matrix **A** as the sole input to the process. The coarse and the fine grids are just represented by sets of indices of degrees of freedom (DOFs).

The creation of the operator $\boldsymbol{P}$ consists of two steps (Stüben, 2001):

- selection of indices retained at the coarser level (interpolatory nodes) and the indices removed from the system (non-interpolatory nodes), i.e., the C–F (coarse-fine) division,
- creation of the interpolation operator transforming values associated with the set of coarse-level nodes to values associated with all nodes forming the fine-level set.

In order to partition the DOFs, we follow the algorithm described by Stüben (2001) for the classical AMG, using the notions of dependence and influence of nodes. The influence is a measure of how many other rows are affected by the DOF associated with a given row. The relation opposite to the influence is called the dependence.

We adopt the following formula for defining the set $S_i$ of DOFs influencing a given DOF from the article by Henson & Yang (2002) (with entries of the system matrix $\boldsymbol{A}$ denoted by $a_{ij}$):

$$S_i = \{j \neq i : -a_{ij} \geq \alpha\, max_{k\neq i}(-a_{ik})\} \tag{1}$$

with the parameter $\alpha \in (0,1)$ specifying the threshold for the strength of the influence, which determines the inclusion of a DOF into the set $S_i$ (further, we call α the strength threshold). An important observation is that only a single row of a matrix itself defines what influences a particular DOF.

The DOFs are ordered according to the number of DOFs they influence. After selecting the first DOF, all DOFs that are influenced by this DOF are moved to the set of non-interpolatory DOFs and the algorithm continues.

The key parameter affecting the process of coarse-level formation is the strength threshold, α. Increasing the coefficient leads to levels with higher matrix density (the percentage of non-zero entries) which should lead to better convergence at the cost of a longer iteration time and higher memory requirements. For the Gauss–Seidel method, the iteration time depends linearly on the number of non-zeros in the matrix, so when comparing two AMG methods that give the same convergence, the better one will be the one that has the lower-density matrix.

After the partition of the DOFs, an interpolation matrix $\boldsymbol{P}$ is created with each DOF having its row, and each interpolatory DOF having its corresponding column. The rows associated with the interpolatory DOFs should have just a single entry equal to one in a column related to this DOF. The rest of the rows should be filled with the entries whose sum in each row is equal to one (Stüben, 2001). The interpolation matrix is used for the restriction and prolongation operations, and for the creation of the coarse system matrix using the Galerkin projection: $\boldsymbol{P}^T\boldsymbol{A}\,\boldsymbol{P}$.

In the strategy presented in the current paper, instead of using heuristic arguments for designing formulae to calculate the entries of $\boldsymbol{P}$, we employ a graph neural network, following the approach described in the report by Luz et al. (2020).

The system matrix $\boldsymbol{A}$ is used as the input to the network, together with the non-zero structure of the in-

terpolation operator $\boldsymbol{P}$, obtained by the classical AMG algorithm and the C–F division.

The values of the entries of the interpolation operator $P$ are obtained as the output of the algorithm.

When calculating the values of $P$, the network tries to minimise the Frobenius norm of the error propagation matrix $\boldsymbol{M}$ of the two-level AMG method, defined as:

$$\boldsymbol{M} = \boldsymbol{S}\,\boldsymbol{C}\,\boldsymbol{S} \tag{2}$$

where $\boldsymbol{S}$ is the error propagation matrix of a single Gauss–Seidel iteration (since we use a single Gauss–Seidel iteration for pre- and post-smoothing in all our tests) and $C$ is the error propagation matrix of the coarse-grid correction:

$$\boldsymbol{C} = \boldsymbol{I} - \boldsymbol{P}\,(\boldsymbol{P}^T\boldsymbol{A}\,\boldsymbol{P})^{-1}\,\boldsymbol{P}^T\boldsymbol{A} \tag{3}$$

where $\boldsymbol{I}$ denotes the identity matrix.

The Frobenius norm is chosen since it corresponds to the spectral radius of the error propagation matrix that determines the convergence of the iterative method, but is easier to calculate (Luz et al., 2020).

The neural network is trained only for the two-grid method, but it is assumed that the network will be able to effectively create the prolongation operators at all levels of the AMG, improving its overall convergence.

Following the work by Luz et al. (2020), we use a message passing neural network (as defined by Battaglia et al. (2018) and Gilmer et al. (2017)) to represent a graph neural network. In this approach, a directed multigraph consisting of vertices and edges is created, in which vertices, edges, and the graph have sets of attributes. An iterative computation is run on the structure thus prepared, in which edge attributes are sequentially updated based on connected vertices, vertices based on connected edges, and global attributes based on all vertices and edges.

This general model is simplified for the purpose of calculating $\boldsymbol{P}$ using a message passing neural network. The system matrix $\boldsymbol{A}$ for which the values of the prolongation matrix $\boldsymbol{P}$ are computed is represented as a graph with DOFs as vertices and the edges corresponding to the non-zero entries of matrix $\boldsymbol{A}$. There are no global attributes of the graph, the vertex attributes correspond to the splitting into interpolatory and non-interpolatory DOFs, and the matrix $\boldsymbol{A}$ values are used as edge weights.

We use the same variant of the encode–process–decode network proposed by Battaglia et al. (2018). That is, the vertices are encoded as sequences of [1, 0] (for nodes belonging to the set C) and [0, 1] (for nodes not belonging to the set C), and the edges as sequences of [value, 1, 0] or [value, 0, 1], depending whether the edges have representation in the coarse set or not, in a similar way as for vertices. An encoder layer is used first for an input prepared in this way, changing the input to configurable-width vectors. The features are then passed through multilayer perceptron (MLP) blocks, after which they are decoded to a single value corresponding to the edge weight.

During network training, matrices $\boldsymbol{A}$ corresponding to many small problems, of the same kind as the final problem to be solved, are employed. The whole dataset is divided into subsets composed of elements related to individual matrices and represented by graphs.

Each subset forms a single input for the network, which leads to the prolongation matrix obtained after processing. Having the prolongation matrix, we can calculate the error propagation matrix for it, its Frobenius norm and, as a result, pass it as an input to the neural network learning backpropagation algorithm. The most costly part of this algorithm is the inversion of the matrix $\boldsymbol{A}$, which is needed to calculate the error propagation matrix.

The learning process is configured using the following network parameters:

- number of rounds of message passing communication,
- number of neurons within one multilayer perceptron (MLP) layer,
- number of MLP layers,
- learning speed.

Increasing or decreasing the number of rounds of communication affects the learning and computation time but does not increase the memory requirements, unlike modifying the MLP layers.

## 3. Implementation of the algorithm

The goal at this stage of development was to provide a proof of concept for employing a graph neural network to improve the algorithm for solving ill-conditioned linear systems resulting from 3D finite element approximations of pure diffusion problems (like, e.g., a heat equation). To this end, we used the implementation of the algorithm presented by Luz et al. (2020), provided by the authors. The code is written in Python using the tensorflow framework.

The implementation was coupled with the ModFEM platform for finite element simulations described in the article by Michalik et al. (2013). The ModFEM code creates linear systems related to the solved diffusion problems and supervises the solution procedure.

ModFEM uses the matrix format provided by the PETSC framework, so for the purposes of this work, an interface was created between the ModFEM code written in C/C++ and the Python code so that it was not necessary to manually copy the matrix data between the Python and C/C++ interfaces. When creating each AMG level matrix, the previous level matrix and the C–F division are passed as parameters to the neural network. The result is an AI-improved restriction matrix from which the next AMG level is created.

## 4. Numerical examples

We tested the implementation of the described strategy for a set of stationary heat transfer problems with heat conduction coefficients randomly distributed over the elements of the initial mesh. The mesh was generated for a simple cube-shaped computational domain and consisted of prismatic elements. The final problems for which we tested the created prolongation operators were defined for meshes obtained from the initial mesh by uniform refinements (divisions of all prismatic elements into eight descendant elements). To make the resulting linear systems ill-conditioned, we used large jumps in coefficient values.

### 4.1. Training the network

As the basis for generating the input and control data, we used a mesh consisting of 10 × 10 × 9 smaller cubes, each of which was divided into 2 prismatic elements. The data was first used to select the learning parameters. After creating dozens of different configurations, we selected a network with 4 MLP layers with 16 neurons each. We also adopted 5 rounds of communication and a learning rate parameter of 2e−7. The parameters chosen in this way ensured a reasonable balance between the time needed to create improved prolongation and restriction operators and the benefits of using them.

Training of the best configuration was carried out on a set of 600,000 matrices. Larger sets did not provide a noticeable benefit to the network's performance. All computations were performed on an 8-core AMD Ryzen 7 3700X CPU, and took 58 h to complete.

### 4.2. Results and discussion

The network performance was evaluated for eight different grid types, where each grid was obtained by two or three uniform refinements of the initial mesh.

Apart from the initial mesh used for the training, we also used initial meshes that differed slightly, resulting from the division of the unit cube into different numbers of small cubes and initial elements.

Tab. 1 presents the results for twice refined meshes for each grid type, averaged over twenty problems with different random assignments of the diffusion coefficient.

We compared the differences in convergence and also the differences in density of original and AI-improved AMG matrix structures using the standard methodology. Our goal was to rule out the possibility of AI generating matrices that result in better convergence but which is not sparse enough to make this trade-off beneficial.

On average, the AI-improved AMG was able to at least slightly outperform the standard AMG in terms of iteration time in each of the examples of twice-refined meshes. For two of the meshes, considerable improvement was achieved.

Unfortunately, that was not the case for three-times refined meshes. This could be due to the training set not being able to prepare the AI for element configurations in which one element is surrounded by multiple siblings with the same material properties.

**Table 1.** Comparison of efficiency properties for the classical and AI-improved AMG projection operators on different meshes for ill-conditioned pure diffusion problems

| Initial mesh | Classical AMG | | AI-improved AMG | |
|---|---|---|---|---|
| | Solution time [s] | Matrix density [%] | Solution time [s] | Matrix density [%] |
| 9 × 9 × 8 | 1.49 | 0.305 | 1.36 | 0.283 |
| 11 × 11 × 10 | 3.17 | 0.174 | 3.14 | 0.176 |
| 10 × 10 × 11 | 3.86 | 0.175 | 3.79 | 0.172 |
| 11 × 11 × 9 | 2.91 | 0.192 | 2.74 | 0.177 |
| 10 × 10 × 9 | 2.10 | 0.216 | 2.10 | 0.213 |

## Conclusions and future work

The presented strategy should be improved in several areas. First, a method should be developed to improve the restriction matrix for parallel MPI calculations, as the current implementation allows multi-threaded processing at best. Another development could be to use the GPU when solving a system of equations, although the key here would be to reduce the memory requirements. It is also necessary to improve the selection of matrices for the training data set, to broaden the class of tasks for which we obtain real improvements in convergence.

The more comprehensive, but possibly most important, development would be to design and implement procedures for creating not only the values of interpolation operators but also their non-zero structure.

## Acknowledgements

This work was supported by the AGH University of Krakow under Research Subsidy No. 16.16.110.663.

## References

Battaglia, P. W., Hamrick, J. B., Bapst, V., Sanchez-Gonzalez, A., Zambaldi, V., Malinowski, M., Tacchetti, A., Raposo, D., Santoro, A., Faulkner, R., Gulcehre, C., Song, F., Ballard, A., Gilmer, J., Dahl, G., Vaswani, A., Allen, K., Nash, C., Langston, V., … Pascanu, R. (2018). *Relational inductive biases, deep learning, and graph networks*. arXiv. https://doi.org/10.48550/arXiv.1806.01261

Gilmer, J., Schoenholz. S. S., Riley, P. F., Vinyals, O., & Dahl, G. E. (2017). *Neural message passing for quantum chemistry*. arXiv. https://doi.org/10.48550/arXiv.1704.01212

Goik, D., & Banaś, K. (2020). A block preconditioner for scalable large scale finite element incompressible flow simulations. In V. V. Krzhizhanovskaya, G. Závodszky, M. H. Lees, J. J. Dongarra, P. M. A. Sloot, S. Brissos, & J. Teixeira, (Eds.), *Computational Science – ICCS 2020. 20th International Conference, Amsterdam, The Netherlands, June 3–5, 2020, Proceedings* (part III, pp. 199–211). Springer Cham. https://doi.org/10.1007/978-3-030-50420-5_15

Gori, M., Monfardini, G., & Scarselli, F. (2005). A new model for learning in graph domains. *Proceedings. IEEE International Joint Conference on Neural Networks*. https://doi.org/10.1109/IJCNN.2005.1555942

Henson, V. E., & Yang, U. M. (2002). BoomerAMG: A parallel algebraic multigrid solver and preconditioner. *Applied Numerical Mathematics*, *41*(1), 155–177. https://doi.org/10.1016/S0168-9274(01)00115-5

Hsieh, J.-T., Zhao, S., Eismann, S., Mirabella, L., & Ermon, S. (2019). *Learning neural PDE solvers with convergence guarantees*. arXiv. https://doi.org/10.48550/arXiv.1906.01200

Katrutsa, A., Daulbaev, T., & Oseledets, I. (2017). *Deep multigrid: learning prolongation and restriction matrices*. arXiv. https://doi.org/10.48550/arXiv.1711.03825

Lagaris, I. E., Likas A., & Fotiadis D. I. (1998). Artifial neural networks for solving ordinary and partial differential equations. *IEEE Transactions on Neural Networks*, *9*(5), 987–1000. https://doi.org/10.1109/72.712178

Luz, I., Galun, M., Maron, H., Basri, R., & Yavneh, I. (2020). *Learning algebraic multigrid using graph neural networks*. arXiv. https://doi.org/10.48550/arXiv.2003.05744

Michalik, K., Banaś, K., Płaszewski, P., & Cybułka, P. (2013). ModFEM – a computational framework for parallel adaptive finite element simulations. *Computer Methods in Materials Science*, *13*(1), 3–8. https://doi.org/10.7494/cmms.2013.1.0403

Mishra, S. (2018). *A machine learning framework for data driven acceleration of computations of differential equations*. arXiv. https://doi.org/10.48550/arXiv.1807.09519

Stüben, K. (2001). A review of algebraic multigrid. *Journal of Computational and Applied Mathematics*, *128*(1–2), 281–309. https://doi.org/10.1016/S0377-0427(00)00516-1