Graph Neural Networks for Selection of Preconditioners and Krylov Solvers

Anonymous Author(s)
Affiliation
Address
email

Abstract

Solving large sparse linear systems is ubiquitous in science and engineering, generally requiring iterative solvers and preconditioners, as many problems cannot be solved efficiently by using direct solvers. However, the practical performance of solvers and preconditioners is sometimes beyond theoretical analysis and an optimal choice calls for intuitions from domain experts, knowledge of the hardware, as well as trial and error. In this work, we propose a new method for optimal solver-preconditioner selection using Graph Neural Networks (GNNs), as a complementary solution to laborious expert efforts. The method is based upon the graph representation of the problem and the idea of integrating node features with graph features via graph convolutions. We show that our models outperform traditional machine learning models investigated by the prior literature by a margin of 25% under the NDCG evaluation metrics.

1 Introduction

Solving linear systems in the form:

$$A\mathbf{x} = \mathbf{b} \tag{1}$$

where A is a large and sparse matrix, is a core computing task in numerous scientific and engineering problems, such as piezoelectric crystal vibration simulation [1], superconductor modeling [2], and airborne electromagnetic problems [3]. These problems typically scale poorly for direct solvers due to the constraints of memory and computational resources. Therefore, iterative methods, as well as preconditioners, are continuously being developed to improve effectiveness and accelerate the computations [4]. However, because of the wide variety of algorithms, data structures, and hardware architectures, selecting the optimal combination of a solver and a preconditioner to a given system is difficult for application developers and researchers and may require extensive background and expertise in numerical analysis and high-performance computing. For example, the scientific computing library PETSc [5, 6, 7] already includes 17 iterative methods, 17 preconditioners, and 6 matrix formats (not counting support for third-party packages). The growing pool of new techniques becomes a barrier to productivity. Therefore, it has been a long-standing aim to automate the process and ease the reliance on numerical analysis and hardware knowledge. It becomes more crucial when the problem size is enormous and its solution requires parallel computation resources.

However, this is a challenging task because the decision cannot be made based on the algorithms analysis alone and the problem characteristics must be taken into account. While most existing machine learning methods rely on finding features that are relevant for the performance prediction and easy to compute, we propose a machine learning method that uses graph neural networks (GNNs) for fast selection of preconditioners and sparse iterative solvers adaptive to matrix properties. Matrices of sparse linear systems entail a graph interpretation. Hence, GNNs are a promising and feasible

Submitted to 36th Conference on Neural Information Processing Systems (NeurIPS 2022). Do not distribute.

framework for learning and encapsulating the domain knowledge required for efficient solution of sparse linear systems.

For a proof of concept, only selected built-in Krylov solvers and preconditioners in PETSc are considered here for the performance benchmark. Users accessible to other software libraries can follow a similar procedure and create a suitable training dataset to obtain specific predictions for solver and preconditioner selection in their computing environment.

Our main contributions are listed below.

- We introduce a general graph representation of large sparse matrices so that the local structure of the matrix is exploited by the graph convolution process.
- We formulate a scored-based metric that account for the budget of running time and accuracy requirement of the target problem, thus allowing for a trade-off between these two factors.
- We analyze four known graph convolution models through the proposed GNN architecture that combines node features with graph features.
- We introduce a two-level pooling strategy to simulate the mechanism by which human experts select iterative solvers and preconditioners.

The remaining contents of this paper are organized as follows. Section 2 reviews the literature. Section 3 introduces the main procedure of the proposed model, from building dataset to generating classification. Section 4 shows the numerical results and comparisons. Section 5 describes conclusions and future work.

2 Related work

Over the years, the literature has studied automatic methodologies for selecting preconditioned linear solvers for sparse linear systems. In 1996, Barrett et al. [8] proposed a poly-iterative approach that executed several Krylov methods simultaneously to the same problem to predict the best solver. Bhowmick et al. [9, 10] present ideas of applying alternating decision tree to choose linear solvers adaptively. In the follow-up work of Bhowmick et al. [11], other machine learning (ML) methods, including k-nearest neighbor, naive Bayes, and support vector machines, are investigated. Holloway and Chen [12] extend the methodology to neural networks, while Kuefler and Chen [13] evaluate the effectiveness of reinforcement learning. Other similar research includes [14], in which Eller et al. introduce a dynamic model, especially for adaptive hydraulics (ADH) problems during transient simulations. Kotthoff et al. [15] compare ML techniques to address the more general problem of algorithm selection.

Software and system for tuning and recommending numerical solvers have also been developed. In particular, PYTHIA [16] is a knowledge-based recommender system for elliptic partial differential equations (PDEs). In 2010, Eijkhout and Fuentes [17] develop a self-adapting large-scale solver architecture (SALSA) for multi-stage solver selection, with the help of their matrix feature extraction software named AnaMod [18]. Later, a comprehensive taxonomy and framework are given in project Lighthouse [19, 20, 21], which targets on preconditioned solvers in PETSc and Trilinos libraries [22].

3 Method

Figure 1 shows an overview of the proposed GNN architecture. An input matrix of the COO format is processed through two paths. In the first path, a graph with well-defined node features is first generated. Then, by applying graph convolutional layers and pooling layers, we obtain a graph embedding. In the second path, matrix features are extracted and treated as graph features. A feature vector is processed by a neural network, i.e., a multi-layer perceptron (MLP). Thus, the second path also generates a graph embedding. Two separate embeddings are concatenated and integrated into a new graph embedding through another MLP. We may regard this output embedding as an input of a classifier or a prediction, after applying a scaling function, e.g. Sigmoid. Note that, these two paths are independent until the mixing step, and we can remove either path or add new paths of similar structure.

The remaining content of this section describes the implementation details of this work.

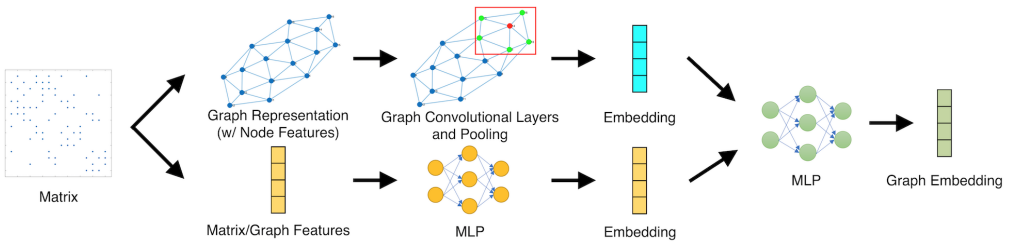


Figure 1: An overview of the proposed GNN architecture.

3.1 Problem formulation

The objective of the selection problem is defined as a *multi-label classification*, where each class represents a combination of a solver and a preconditioner. Table 1 shows the Krylov iterative solvers and preconditioners used in our settings. In total, 33 classes are considered, excluding combinations where solvers and preconditioners are incompatible. Therefore, for any input matrix, the output of our GNN model is a binary vector $\hat{y} \in \{0,1\}^{33}$, with a value of 1 representing recommendation, and 0 otherwise.

Table 1: Available preconditioners and Krylov iterative solvers

Capability	Algorithm
Preconditioners	Block Jacobi + ILU(0), QMD reordering
	Block Jacobi + ILU(1), QMD reordering
	Block Jacobi + LU
	ASM(1)
	ASM(2)
	Hypre/BoomerAMG
	Hypre Euclid (ILUT is not supported)
	Parasails approximate inverse from Hypre
	Block Jacobi + GMRES
Krylov iterative solvers	CG
	GMRES(30)
	BiCGStab
	LSQR
	Flexible GMRES (inner GMRES)

The performance of solvers and preconditioners on a given matrix A arises from solving a linear system $A\mathbf{x} = \mathbf{b}$ where all elements in $\mathbf{b}$ are 1's. By applying various combinations, we record the running time t and the residual $r = \|\mathbf{b} - A\hat{\mathbf{x}}\|_2$ for each choice, where $\hat{\mathbf{x}}$ is the computed solution. Then, we use the following metrics to compute a score:

$$\text{score}(t, r) = \log(1 + w_1/t)\log(1 + w_2/r) \tag{2}$$

where w_1, w_2 are user-defined weights (default 1) based on the budgets of running time and tolerance of residual error. For the cases that run out of time or fail to solve the system, the score will be 0. Eq. 2 is designed to distinguish between the classes with exponential differences in residual or runtime cost.

We define an empirical threshold equal to 90% of the highest score. Any class that has a score higher than the threshold will be labeled as 1, and others will be labeled as 0.

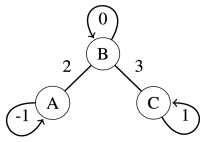
3.2 Graph representation and preprocessing

For any input matrix A, a graph is first generated based on the adjacency matrix of A. Then a self-loop is added to each node. The off-diagonal nonzero elements of A are attached to corresponding edges as edge features and diagonal elements are attached to self-loops respectively. As a rule of thumb, diagonal elements are deemed more important compared with off-diagonal elements. Our graph

representation emphasizes diagonal elements, as self-loops are visited more frequently throughout the message passing process. Naturally, diagonal elements with larger magnitudes corresponds to larger edge values on self-loops and thus have a larger impact on the computation graph.

Here we demonstrate this idea with an example of a 3×3 matrix.

$$A = \begin{bmatrix} -1 & 2 & 0 \\ 2 & 0 & 3 \\ 0 & 3 & 1 \end{bmatrix} \quad \rightarrow$$



Recalling that the essence of our GNN model is to predict the performance of solvers and preconditioners in solving linear systems, any modification is reasonable as long as it does not affect the selection. Therefore, all columns and rows that contain only diagonal nonzero elements can be removed. In the context of graph interpretation, this refers to the operation of deleting isolated nodes. Diagonal matrices are ignored from the dataset for the same reason. Furthermore, we add a small constant value to every self-loop in the graph because diagonal shifting is commonly applied to singular matrices in iterative linear solvers.

3.3 Node and graph features

As mentioned in the beginning of Section 3, the graph embedding is generated by processing two types of features. *Node features* attached to graph nodes are processed through graph convolutional layers while *graph features* are processed via other neural network layers, e.g., MLP. Then a mixing layer, e.g. another MLP, is performed to render a final graph embedding.

We introduce a node feature to describe diagonal dominance of row i. Assume that α_i denotes the ratio between magnitudes of diagonal and off-diagonal elements for each row,

$$\alpha_i = \begin{cases} \frac{|A_{ii}|}{\sum_{j \neq i} |A_{ij}|}, & \sum_{j \neq i} |A_{ij}| > 0 \; ; \\ \infty, & \sum_{j \neq i} |A_{ij}| = 0 \; . \end{cases} \tag{3}$$

Then, the node feature $\mathbf{x}_i$ for node i is defined as

$$\mathbf{x}_i = \frac{\alpha_i}{\alpha_i + 1}, \tag{4}$$

such that $\mathbf{x}_i \in [0, 1]$ and $\mathbf{x}_i = 0.5$ is the critical point of whether row i is diagonally dominant. Similarly, the diagonal decay of row i can be defined by replacing $\sum_{j \neq i} |A_{ij}|$ in Eq. 3 with $\max_{j \neq i} |A_{ij}|$. Both diagonal dominance and decay can also be computed for each column. Moreover, the local degree profile [23] is also considered.

For graph features, readers can refer to Table 2 for a complete list of features used in this paper. To reduce the cost of extracting features that have no significant impact on the classification accuracy, we use the reduced feature set provided by [20, 21]. Additionally, it is essential to perform normalization or standardization when concatenating various features.

3.4 Graph convolutional layers

We analyze four conventional graph convolution models:

- *Graph Attention Networks* (GAT). GAT and its successor [24, 25] support taking edge features as input, where edge features are multiplied with transformation matrices and then concatenated with node features.
- *The edge version of Graph Isomorphism Networks* (GINE) [26, 27]. GINE aggregates edge features along with node features.
- *Graph Convolutional Networks* (GCN) [28]. GCN is rather different from above models. GCN can only take non-negative edge weights, so a Sigmoid or softmax function on edge features is required.

Table 2: List of node and graph features.

Capacity	Features
Node	Diagonal dominance (col & row)
	Diagonal decaying (col & row)
	Local degree profile
Graph	Average distance of non-zero to diagonal
	Number of non-zero elements
	1-norm
	∞-norm
	Column variability
	Minimum number of column non-zero elements
	Row variability
	Minimum number of row non-zero elements
	Number of diagonals that have non-zero elements
	Estimated condition number

- *A modified version of GraphSAGE* [29]. The aggregator is represented by

$$\mathbf{x}_i \leftarrow W_1 \mathbf{x}_i + W_2 \cdot \text{mean}_{j \in \mathcal{N}(i)} \mathbf{x}_j e_{ji}, \tag{5}$$

  where $\mathbf{x}_i$ is the node embedding of node i, $\mathcal{N}(i)$ is the neighbor of node i, W_1, W_2 are training weights, and e_{ji} is the edge feature from node j to node i.

We make slight modifications to GCN and graphSAGE in order to utilize both node and edge features. Edge features, in other words, nonzero elements of the matrix, can be arbitrary real values. This requires that the desired convolutional network should be able to distinguish between A and its absolute counterpart $|A|$ if there are both positive and negative elements in A. It also should produce the same results for A and kA for any scalar $k \neq 0$.

3.5 Two-level pooling

By adapting the idea of *Multiset Pooling* technique [30], we designed a two-level pooling based on the fact that computations between distinct connected components are independent. In our approach, the graph embedding is generated in the following steps: (1) the node embeddings are first aggregated within the connected components to which they belong; (2) then the connected component embeddings are summarized to yield the graph embedding.

The message passing mechanism can be viewed as a voting procedure that the central node collects messages from its neighbor nodes and votes for the classification of the graph from its perspective, i.e., via node embedding. However, since each node can only reach the connected component to which it belongs, we aggregate the node embeddings within each connected component. Common embedding techniques include mean-pooling, histogram of voting, and virtual node embedding [31].

Without treating each connected component independently, the selection of solvers and preconditioners for the whole matrix heavily depends on the most difficult connected components to solve, e.g., the most ill-conditioned one. Therefore, we apply a min-pooling technique to connected component embeddings to generate the graph embedding, which indicates the solver and preconditioner that are suitable and efficient for all connected components.

To illustrate the second level of pooling, we use a simple example where we choose from two candidate preconditioned solvers for a block diagonal matrix that consists of two square matrices, i.e.,

$$C = \begin{bmatrix} A & \\ & B \end{bmatrix}. \tag{6}$$

Here A and B are two distinct connected components of C. If both solvers can be applied to A and only the first solver is applicable to B, it gives rise to multi-labels $y_A, y_B \in \{0,1\}^2$ where $y_A = [1,1]$ and $y_B = [1,0]$. Then, the multi-label for C is $y_C = [1,0] = \text{min-pooling}(y_A, y_B)$.

4 Numerical results

The experiments for the GNN models were performed on a workstation with an NVIDIA RTX 3090 GPU and an Intel i7-11700KF CPU. Scikit-learn [32] is used to implement traditional ML methods. GNN frameworks are mainly based on PyTorch [33] and PyTorch-Geometric [34], and evaluation metrics are provided by TorchMetrics [35].

4.1 Benchmark dataset

To assesses the performance of the GNN models, we built a benchmark using matrices from the SuiteSparse Matrix Collection [36]. Only square matrices $A \in \mathbb{R}^{n \times n}$ with a number of rows $1000 \leq m \leq 10000$ and a number of nonzero elements $\text{nnz}(A) \leq 200000$ are selected. The dataset contains 614 matrices and is split into training and testing datasets using 5-fold cross-validation. Each matrix results in 33 entries because of the 33 combinations of solvers and preconditioners. Each entry in the dataset consists of the matrix ID, the solver choice, the preconditioner choice, the solution time (with negative values indicating failure reasons such as using an incompatible preconditioner, time out and divergence) and the ℓ_2 residual norm. Each linear solve was run in parallel using 64 MPI processes on the KNL partition on Theta [37], on Theta, a Cray XC40 supercomputer at Argonne National Laboratory.

4.2 Evaluation metrics

The evaluation metrics for multi-label classification in this paper includes *Label Ranking Average Precision* (LRAP) [38] and *Normalized Discounted Cumulative Gain* (NDCG) [39, 40, 41]. Both methods enable taking prediction probability as input and return scores in $[0, 1]$, where higher scores result in better classification accuracy. Here, we do not use *true positive rate* (TPR) because it is meaningless for multi-label classification tasks. TPR can be arbitrarily high if one sets the decision threshold sufficiently low.

We denote the true labels by y and the prediction by $\hat{y}$. The definition of LRAP is as follows:

$$LRAP(y, \hat{y}) = \frac{1}{n} \sum_{i=0}^{n-1} \frac{1}{\|y_i\|_0} \sum_{j:y_{ij}=1} \frac{|\mathcal{L}_{ij}|}{\text{rank}_{ij}} , \tag{7}$$

where n is the number of samples, $\mathcal{L}_{ij} = \{k : y_{ik} = 1, \hat{y}_{ik} > \hat{y}_{ij}\}$, $\text{rank}_{ij} = |\{k : \hat{y}_{ik} > \hat{y}_{ij}\}|$, and $\|\cdot\|_0$ computes the l_0-norm which equals to the number of nonzeros.

The DCG score is defined as:

$$DCG(y, \hat{y}) = \sum_{r=1}^{m} \frac{y_{f(r)}}{\log(1 + r)} , \tag{8}$$

where m is the number of classes, f is a ranking function induced from y and $\hat{y}$. The NDCG score is the DCG score divided by the DCG score of y, which is considered as the ideal DCG score.

4.3 Comaparison with traditional ML methods

The implementations of the proposed GNN models have been compared with traditional ML methods supporting multi-label classification, such as Random Forest (RF) [42], k-nearest neighbor (kNN) [43], Multi-layer Perceptron (MLP) [44, 45], and Ridge Classifier.

GNN models in our experiments follow the same architecture and differ solely from the convolutional layers. Each graph convolution model consists of 5 layers. The MLP for processing graph features has 2 layers, and the MLP for combining node and graph embeddings has 2 layers. More GNN layers can cause over-smoothing, and effective training strategies [46] need to be used for such situations. Table 3 gives 5-fold average LRAP and NDCG scores and standard errors of considered methods on the test dataset. It shows that GNN models are comparable to traditional ML-based models on the LRAP metric, with an exception that RF noticeably beats the rest. However, GNN models significantly outperform traditional models on the NDCG metric. Specifically, the best-performing GNN model, GAT, outperfms the best-performing traditional method, RF, by a margin of more than 25%.

Table 3: Classification scores of considered methods on test dataset.

Method	LRAP	NDCG
RF	**0.7778 ± 0.0262**	**0.5618 ± 0.0299**
MLP (1 layer)	0.7231 ± 0.0378	0.5181 ± 0.0366
MLP (2 layers)	0.7570 ± 0.0530	0.5424 ± 0.0475
k-nearest neighbors	0.6465 ± 0.0405	0.4902 ± 0.0307
Ridge Classifier	0.3245 ± 0.0706	0.2473 ± 0.0239
GINE	0.7480 ± 0.0357	0.8126 ± 0.0285
GAT	**0.7770 ± 0.0267**	**0.8246 ± 0.0325**
GraphSAGE	0.7492 ± 0.0068	0.8043 ± 0.0388
GCN	0.7664 ± 0.0338	0.8222 ± 0.0216

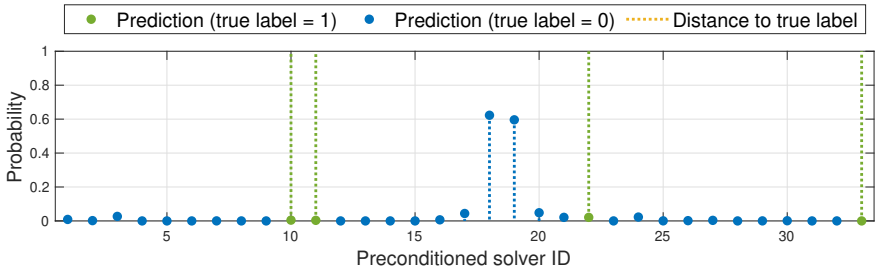


Figure 2: Prediction for matrix *rbd3200l*. Dots are prediction of GAT model, where blue ones have true label equals to 0 and green ones have true label equals to 1. Dotted line indicates the distance to the corresponding true label.

4.4 Case studies

To interpret our experimental results and explain the large discrepancy between LRAP and NDCG scores, we perform a case study with the matrix *rbd3200l* selected from the test dataset. Its evaluation scores are $LRAP = 0.1809$ and $NDCG = 0.4216$. Figure 2 shows the prediction results made by the GAT model. We notice that the LRAP metric focuses on the "good" classes, which are the classes associated with labels equal to 1. The metric penalizes heavily the cases when "good" classes are predicted to be "bad" classes while false prediction for "bad" classes are ignored. Therefore, there are four "good" classes (10, 11, 22, and 33) in *rbd3200l* as shown in Figure 2, and none of them is correctly classified, leading to a very low LRAP score. In contrast, the NDCG metric computes a score according to the ranking induced by the prediction. For all six misclassified classes, only the middle two classes (18 and 19) have large prediction values (or higher ranks), thus leading to a low NDCG score. Nevertheless, the other four classes (10, 11, 22, and 33) do not have significant impact on NDCG scores for the same reason.

In practice, "good" classes (the classes with label 1) attract more attention. The properties of LRAP and NDCG suggest that our GNN model tends to provide safe recommendations, that is, avoid false

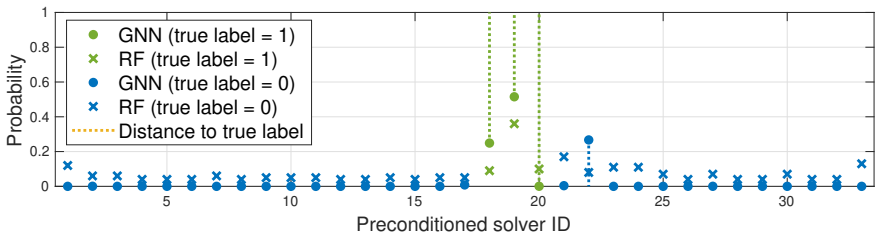


Figure 3: Prediction for matrix *Trefethen_2000*. Crosses denote predictions of RF. Here, GAT scores $LRAP = 0.7222, NDCG = 0.8711$, and RF scores $LRAP = 0.5536, NDCG = 0.7737$.

positive predictions. In other words, our model may predict fewer "good" classes, but it tends not to mislead users to "bad" classes. Figure 3 also confirms our explanations. For classifying matrix *Trefethen_2000*, GAT accomplishes less error in either "good" classes (class 18-20), therefore, a higher LRAP score. We can also consider top 5 classes with the highest probabilities for GAT (class 19, 22, 18, 17, 21) and RF (class 19, 21, 33, 1, 20). In this context, GAT also has less error comparing with RF, which leads to higher NDCG scores.

5 Conclusions

Graph neural networks show great potential in classification tasks where inputs are sparse matrices of varying sizes. With the help of the message passing mechanism and node features that are properly defined, the structural characteristics of the matrix, e.g., the influence of diagonal elements, and patterns, are exploited inherently and expressed by the output embedding. Our model also exploits graph features together with a two-level pooling, serving as a meaningful extension of standard GNNs.

For future work, as seen in Table 3 that the performances of 4 GNN models are indistinguishable, a refinement is to explore models of a different nature (e.g., transformers) that have the potential to outperform existing GNNs by a noticeable margin. In addition, graph coarsening techniques [47] will be deployed for arbitrary-sized (especially exceedingly large) matrices to improve the robustness and efficiency of model training.

References

[1] TR Canfield, MSH Tang, and JE Foster. Nonlinear geometric and thermal effects in three dimensional galerkin finite-element formulations for piezoelectric crystals. *Argonne National Laboratory, Argonne, IL*, 1991.

[2] Mark T Jones and Paul E Plassmann. Solution of large, sparse systems of linear equations in massively parallel applications. Technical report, Argonne National Lab., IL (United States), 1992.

[3] Esben Auken, Tue Boesen, and Anders V. Christiansen. Chapter two - a review of airborne electromagnetic methods with focus on geotechnical and hydrological applications from 2007 to 2017. volume 58 of *Advances in Geophysics*, pages 47–93. Elsevier, 2017.

[4] Yousef Saad. *Iterative methods for sparse linear systems*. SIAM, 2003.

[5] Satish Balay, Shrirang Abhyankar, Mark F. Adams, Steven Benson, Jed Brown, Peter Brune, Kris Buschelman, Emil M. Constantinescu, Lisandro Dalcin, Alp Dener, Victor Eijkhout, William D. Gropp, Václav Hapla, Tobin Isaac, Pierre Jolivet, Dmitry Karpeev, Dinesh Kaushik, Matthew G. Knepley, Fande Kong, Scott Kruger, Dave A. May, Lois Curfman McInnes, Richard Tran Mills, Lawrence Mitchell, Todd Munson, Jose E. Roman, Karl Rupp, Patrick Sanan, Jason Sarich, Barry F. Smith, Stefano Zampini, Hong Zhang, Hong Zhang, and Junchao Zhang. PETSc Web page. `http://petsc.org/`, 2022.

[6] Satish Balay, Shrirang Abhyankar, Mark F. Adams, Steven Benson, Jed Brown, Peter Brune, Kris Buschelman, Emil Constantinescu, Lisandro Dalcin, Alp Dener, Victor Eijkhout, William D. Gropp, Václav Hapla, Tobin Isaac, Pierre Jolivet, Dmitry Karpeev, Dinesh Kaushik, Matthew G. Knepley, Fande Kong, Scott Kruger, Dave A. May, Lois Curfman McInnes, Richard Tran Mills, Lawrence Mitchell, Todd Munson, Jose E. Roman, Karl Rupp, Patrick Sanan, Jason Sarich, Barry F. Smith, Stefano Zampini, Hong Zhang, Hong Zhang, and Junchao Zhang. PETSc/TAO Users Manual. Technical Report ANL-21/39 - Revision 3.17, Argonne National Laboratory, 2022.

[7] Satish Balay, William D. Gropp, Lois Curfman McInnes, and Barry F. Smith. Efficient Management of Parallelism in Object Oriented Numerical Software Libraries. In E. Arge, A. M. Bruaset, and H. P. Langtangen, editors, *Modern Software Tools in Scientific Computing*, pages 163–202. Birkhäuser Press, 1997.

[8] Richard Barrett, Michael Berry, Jack Dongarra, Victor Eijkhout, and Charles Romine. Algorithmic bombardment for the iterative solution of linear systems: A poly-iterative approach. *Journal of Computational and Applied Mathematics*, 74(1-2):91–109, November 1996.

[9] Sanjukta Bhowmick, Victor Eijkhout, Yoav Freund, Erika Fuentes, and David Keyes. Application of machine learning to the selection of sparse linear solvers. *International Journal of High Performance Computing Applications*, 2006. Publisher: Citeseer.

[10] Sanjukta Bhowmick, Victor Eijkhout, Yoav Freund, Erika Fuentes, and David Keyes. Application of Alternating Decision Trees in Selecting Sparse Linear Solvers. In Ken Naono, Keita Teranishi, John Cavazos, and Reiji Suda, editors, *Software Automatic Tuning*, pages 153–173. Springer New York, New York, NY, 2011.

[11] Sanjukta Bhowmick, Brice Toth, and Padma Raghavan. Towards Low-Cost, High-Accuracy Classifiers for Linear Solver Selection. In Gabrielle Allen, Jaroslaw Nabrzyski, Edward Seidel, Geert Dick van Albada, Jack Dongarra, and Peter M. A. Sloot, editors, *Computational Science – ICCS 2009*, volume 5544, pages 463–472. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009. Series Title: Lecture Notes in Computer Science.

[12] America Holloway and Tzu-Yi Chen. Neural Networks for Predicting the Behavior of Preconditioned Iterative Solvers. In David Hutchison, Takeo Kanade, Josef Kittler, Jon M. Kleinberg, Friedemann Mattern, John C. Mitchell, Moni Naor, Oscar Nierstrasz, C. Pandu Rangan, Bernhard Steffen, Madhu Sudan, Demetri Terzopoulos, Doug Tygar, Moshe Y. Vardi, Gerhard Weikum, Yong Shi, Geert Dick van Albada, Jack Dongarra, and Peter M. A. Sloot, editors, *Computational Science – ICCS 2007*, volume 4487, pages 302–309. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007. Series Title: Lecture Notes in Computer Science.

[13] Erik Kuefler and Tzu-Yi Chen. On Using Reinforcement Learning to Solve Sparse Linear Systems. In David Hutchison, Takeo Kanade, Josef Kittler, Jon M. Kleinberg, Friedemann Mattern, John C. Mitchell, Moni Naor, Oscar Nierstrasz, C. Pandu Rangan, Bernhard Steffen, Madhu Sudan, Demetri Terzopoulos, Doug Tygar, Moshe Y. Vardi, Gerhard Weikum, Marian Bubak, Geert Dick van Albada, Jack Dongarra, and Peter M. A. Sloot, editors, *Computational Science – ICCS 2008*, volume 5101, pages 955–964. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008. Series Title: Lecture Notes in Computer Science.

[14] Paul R. Eller, Jing-Ru C. Cheng, and Robert S. Maier. Dynamic Linear Solver Selection for Transient Simulations Using Machine Learning on Distributed Systems. In *2012 IEEE 26th International Parallel and Distributed Processing Symposium Workshops & PhD Forum*, pages 1915–1924, Shanghai, China, May 2012. IEEE.

[15] Lars Kotthoff, Ian P. Gent, and Ian Miguel. An evaluation of machine learning in algorithm selection for search problems. *AI Communications*, 25(3):257–270, 2012.

[16] Sanjiva Weerawarana, Elias N. Houstis, John R. Rice, Anupam Joshi, and Catherine E. Houstis. PYTHIA: a knowledge-based system to select scientific algorithms. *ACM Transactions on Mathematical Software*, 22(4):447–468, December 1996.

[17] Victor Eijkhout and Erika Fuentes. Machine learning for multi-stage selection of numerical methods. *New Advances in Machine Learning. INTECH*, pages 117–136, 2010.

[18] Victor Eijkhout and Erika Fuentes. A Standard and Software for Numerical Metadata. *ACM Transactions on Mathematical Software*, 35(4):1–20, February 2009.

[19] Lighthouse by LighthouseHPC. `http://lighthousehpc.github.io/lighthouse/`.

[20] 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, Austin Texas, November 2015. ACM.

[21] Elizabeth Jessup, Pate Motter, Boyana Norris, and Kanika Sood. Performance-Based Numerical Solver Selection in the Lighthouse Framework. *SIAM Journal on Scientific Computing*, 38(5):S750–S771, January 2016.

[22] The Trilinos Project Website. https://trilinos.github.io, May 2020.

[23] Chen Cai and Yusu Wang. A simple yet effective baseline for non-attributed graph classification. *arXiv preprint arXiv:1811.03508*, 2018.

[24] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

[25] Shaked Brody, Uri Alon, and Eran Yahav. How attentive are graph attention networks? *arXiv preprint arXiv:2105.14491*, 2021.

[26] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826*, 2018.

[27] Weihua Hu, Bowen Liu, Joseph Gomes, Marinka Zitnik, Percy Liang, Vijay Pande, and Jure Leskovec. Strategies for pre-training graph neural networks. *arXiv preprint arXiv:1905.12265*, 2019.

[28] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

[29] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[30] Jinheon Baek, Minki Kang, and Sung Ju Hwang. Accurate learning of graph representations with graph multiset pooling. *arXiv preprint arXiv:2102.11533*, 2021.

[31] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. In *International conference on machine learning*, pages 1263–1272. PMLR, 2017.

[32] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[33] PyTorch. https://www.pytorch.org.

[34] PyTorch-Geometric. https://www.pyg.org/.

[35] PyTorch-Metrics. https://torchmetrics.readthedocs.io/en/stable/.

[36] Timothy A Davis and Yifan Hu. The university of florida sparse matrix collection. *ACM Transactions on Mathematical Software (TOMS)*, 38(1):1–25, 2011.

[37] Theta at argonne. https://www.alcf.anl.gov/theta.

[38] Grigorios Tsoumakas, Ioannis Katakis, and Ioannis Vlahavas. Mining multi-label data. *Data mining and knowledge discovery handbook*, pages 667–685, 2009.

[39] Kalervo Järvelin and Jaana Kekäläinen. Cumulated gain-based evaluation of ir techniques. *ACM Transactions on Information Systems (TOIS)*, 20(4):422–446, 2002.

[40] Yining Wang, Liwei Wang, Yuanzhi Li, Di He, Wei Chen, and Tie-Yan Liu. A theoretical analysis of ndcg ranking measures. In *Proceedings of the 26th annual conference on learning theory (COLT 2013)*, volume 8, page 6, 2013.

[41] Frank McSherry and Marc Najork. Computing information retrieval performance measures efficiently in the presence of tied scores. In *European conference on information retrieval*, pages 414–421. Springer, 2008.

[42] Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[43] David W Aha, Dennis Kibler, and Marc K Albert. Instance-based learning algorithms. *Machine learning*, 6(1):37–66, 1991.

[44] Geoffrey E Hinton. Connectionist learning procedures. In *Machine learning*, pages 555–610. Elsevier, 1990.

[45] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256. JMLR Workshop and Conference Proceedings, 2010.

[46] Kuangqi Zhou, Yanfei Dong, Kaixin Wang, Wee Sun Lee, Bryan Hooi, Huan Xu, and Jiashi Feng. Understanding and Resolving Performance Degradation in Graph Convolutional Networks, September 2021. arXiv:2006.07107 [cs, stat].

[47] Jie Chen, Yousef Saad, and Zechen Zhang. Graph coarsening: from scientific computing to machine learning. *SeMA Journal*, 79(1):187–223, 2022.

Checklist

The checklist follows the references. Please read the checklist guidelines carefully for information on how to answer these questions. For each question, change the default **[TODO]** to [Yes] , [No] , or [N/A] . You are strongly encouraged to include a **justification to your answer**, either by referencing the appropriate section of your paper or providing a brief inline description. For example:

- Did you include the license to the code and datasets? [Yes] See Section **??**.
- Did you include the license to the code and datasets? [No] The code and the data are proprietary.
- Did you include the license to the code and datasets? [N/A]

Please do not modify the questions and only use the provided macros for your answers. Note that the Checklist section does not count towards the page limit. In your paper, please delete this instructions block and only keep the Checklist section heading above along with the questions/answers below.

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes] See Section 1
   (b) Did you describe the limitations of your work? [Yes] See Section 5
   (c) Did you discuss any potential negative societal impacts of your work? [N/A]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [No] The code and data are available upon request.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] See Section 3.2, 3.3 and 4.1
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] See Table 3
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] See Section 4 and 4.1
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes] See Section 3.3, 3.4, and 4

(b) Did you mention the license of the assets? [Yes] See Section 3.3, 3.4, and 4

(c) Did you include any new assets either in the supplemental material or as a URL? [N/A]

(d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]

(e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]

5. If you used crowdsourcing or conducted research with human subjects...

(a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]

(b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]

(c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]