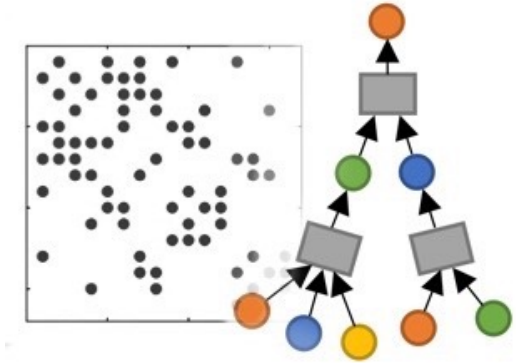
Preconditioning 2024

International Conference On Preconditioning Techniques
For Scientific and Industrial Applications

GNNS FOR SELECTION OF PRECONDITIONERS AND KRYLOV SOLVERS



ZIYUAN TANG
Department of Computer Science and Engineering
University of Minnesota

HONG ZHANG
Mathematics and Computer Science Division
Argonne National Laboratory

JIE CHEN
MIT-IBM Watson AI Lab
IBM Research

This material was based upon work funded by the U.S. Department of Energy, Office of Science, under contract DE-AC02-06CH11357. This work was supported by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of the U.S. Department of Energy Office of Science and the National Nuclear Security Administration, and by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research under Award Number DE-SC0016140.

- Joint work with Hong Zhang and Jie Chen.

- Accepted by New Frontiers in Graph Learning Workshop @ NeurIPS 2022.

TABLE OF CONTENTS

BACKGROUND

BACKGROUND

Motivation

- Solving linear systems of the form:

$$Ax = b.$$

- For large and sparse problems, we usually apply **iterative solvers** as well as **preconditioners.**

- An optimal choice of solver and preconditioner calls for a solid background and knowledge of the hardware.

BACKGROUND

Solvers and preconditioners in PETSc[1,2]

Capability	Algorithm
Preconditioners	Jacobi point block Jacobi block Jacobi additive Schwarz
Incomplete factorizations	ILU dt
Matrix-free	infrastructure
Multigrid	infrastructure geometric (DMDA for structured grid) geometric/algebraic structured geometric classical algebraic (BoomerAMG/hypre) classical algebraic (ML/Trilinos) unstructured geometric and smoothed aggregation
Physics-based splitting	relaxation and Schur-complement least squares commutator
Approximate inverses	approximate inverses
Substructuring	balancing Neumann-Neumann BDDC
Krylov methods	Richardson, Chebyshev, conjugate gradients, GMRES, Bi-CG-stab, transpose-free QMR, conjugate residuals, conjugate gradient squared, bi-conjugate gradient , MINRES, flexible GMRES, LSQR, SYMMLQ, LGMRES, GCR, conjugate gradient on the normal equations

[1] S. Balay, et al. “PETSc Web Page (2022).” `https://petsc.org/`.

[2] 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 (2016): S750-S771.

DATA PREPARATION

DATA PREPARATION

Subset of solvers and preconditioners

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 Parasails approximate inverse from Hypre Block Jacobi + GMRES
Krylov iterative solvers	CG GMRES(30) BiCGStab LSQR Flexible GMRES (inner GMRES)

DATA PREPARATION

Data sample

- Matrices are collected from the SuiteSparse Matrix Collection[1]. Only square matrices with a size $1{,}000 \leq m \leq 10{,}000$ and $nnz \leq 200{,}000$ are selected. The dataset contained 614 matrices.

- RHS is randomly generated. Each linear solve is run in parallel using 64 MPI processes on the KNL partition on Theta[2].

- Each sample contains: matrix id, solver id, preconditioner id, running time (or error code), absolute residual, relative residual.

[1] T. A. Davis and Y. Hu. "The university of florida sparse matrix collection." ACM Transactions on Mathematical Software (TOMS), 38(1):1, 2011.
[2] "Theta at argonne." https://www.alcf.anl.gov/theta

DATA PREPARATION

Data label

- **Multi-label classification:** the label of each matrix is a binary vector $y_A \in \{0,1\}^{33}$.

- **Scoring function** of running time and residuals:

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

  where w_1 and w_2 are user-defined weights.

- **Label**: we mark the top 10% scores with a 1, and the others with a 0.

DATA PREPARATION

Why GNNs?

- **Objective**: extract/summarize matrix features into an embedding vector for classification.

- Traditional machine learning methods can only process **matrix-level features.**
  e.g., condition number, number of nonzeros, rank estimation, etc.

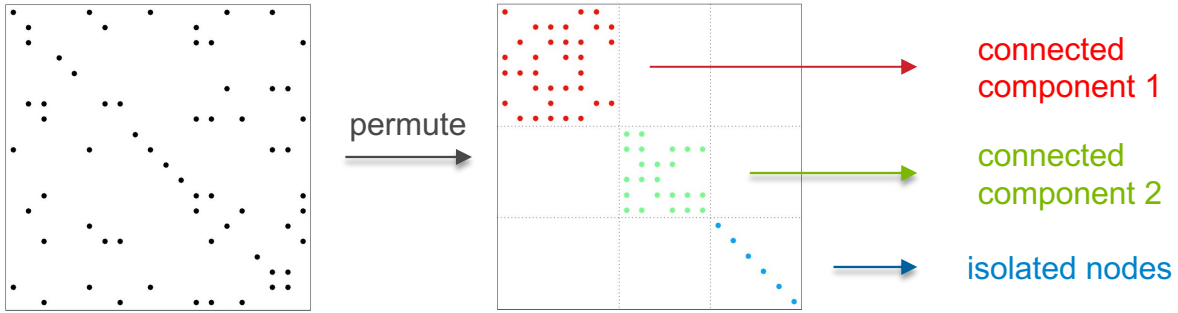
- GNNs can process **node, edge,** and **graph features** simultaneously w.r.t. graph structure.
  → the matrix pattern and entry values are processed simultaneously.

DATA PREPARATION

Matrix-to-graph object

- **Data object:** torch_geometric.data.Data from PyTorch Geometric (PyG[1]).
  - **x (node features):** shape [num_nodes, num_node_features].
  - **edge_index:** COO matrix for graph connectivity, shape [2, num_edges]
  - **other attributes:** edge features, y (label), batch id, connected component id, etc.

- **Transform functions:**
  e.g., RemoveIsolatedNode()



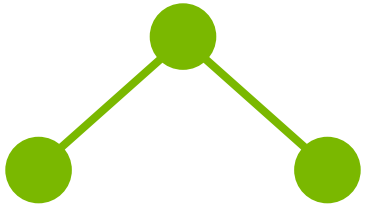
[1] Fey, Matthias, and Jan Eric Lenssen. "Fast graph representation learning with PyTorch Geometric." arXiv preprint arXiv:1903.02428 (2019).

DATA PREPARATION

Matrix-to-graph object

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

- Generate a graph based on the adjacency of A.
  edge_index = [i, j].transpose() where i, j are row and column index of the entry in A.

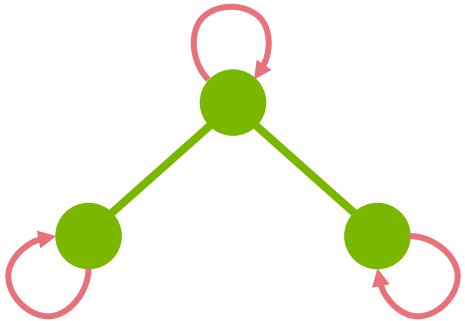


DATA PREPARATION

Matrix-to-graph object

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

- Add a self-loop to each node (isolated nodes are removed).
  Built-in transform function AddSelfLoops() or AddRemainingSelfLoops()

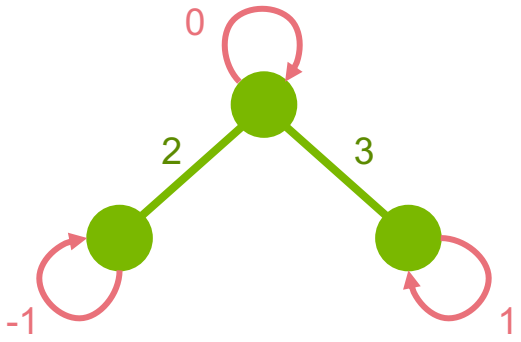


DATA PREPARATION

Matrix-to-graph object

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

- Set **edge features** as the entry values of A.



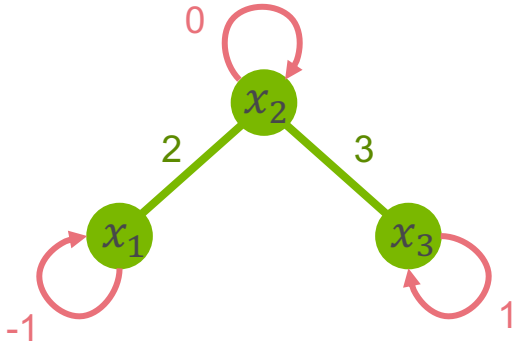
DATA PREPARATION

Matrix-to-graph object

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

- Attach **node features** to each node.
  Attach **graph features** to the graph object.

→ Flow-graph[1-3]



[1] Coates, C. "Flow-graph solutions of linear algebraic equations." IRE Transactions on circuit theory 6.2 (1959): 170-187.
[2] Grementieri, Luca, and Paolo Galeone. "Towards neural sparse linear solvers." arXiv preprint arXiv:2203.06944 (2022).
[3] Häusner, Paul, et al. "Neural incomplete factorization: learning preconditioners for the conjugate gradient method." arXiv preprint arXiv:2305.16368 (2023).

DATA PREPARATION

Node feature selection

- **Diagonal dominance**: assume that α_i denotes the ratio between the magnitudes of diagonal and off-diagonal elements for row i, then:

$$\alpha_i = \begin{cases} \dfrac{|A_{ii}|}{\sum_{j \neq i} |A_{ij}|}, & \sum_{j \neq i} |A_{ij}| > 0 \\ +\infty, & \sum_{j \neq i} |A_{ij}| = 0 \end{cases}, \qquad x_i = \frac{\alpha_i}{\alpha_i + 1}.$$

  The diagonal dominance of node i is given by x_i.

- **Diagonal decay:** ratio between $|A_{ii}|$ and $\max_{j \neq i} |A_{ij}|$.

- **Local degree profile**[1]:

$$x_i = \left[\deg(i), \min\left(DN(i)\right), \max\left(DN(i)\right), \text{mean}\left(DN(i)\right), \text{std}\left(DN(i)\right)\right].$$

[1] C. Cai and Y. Wang. "A simple yet effective baseline for non-attributed graph classification." arXiv preprint arXiv:1811.03508, 2018.

DATA PREPARATION

Graph feature selection

- **Estimated condition number, number of nonzeros, etc.**

- More features can be found in AnaMod[1], Lighthouse[2].

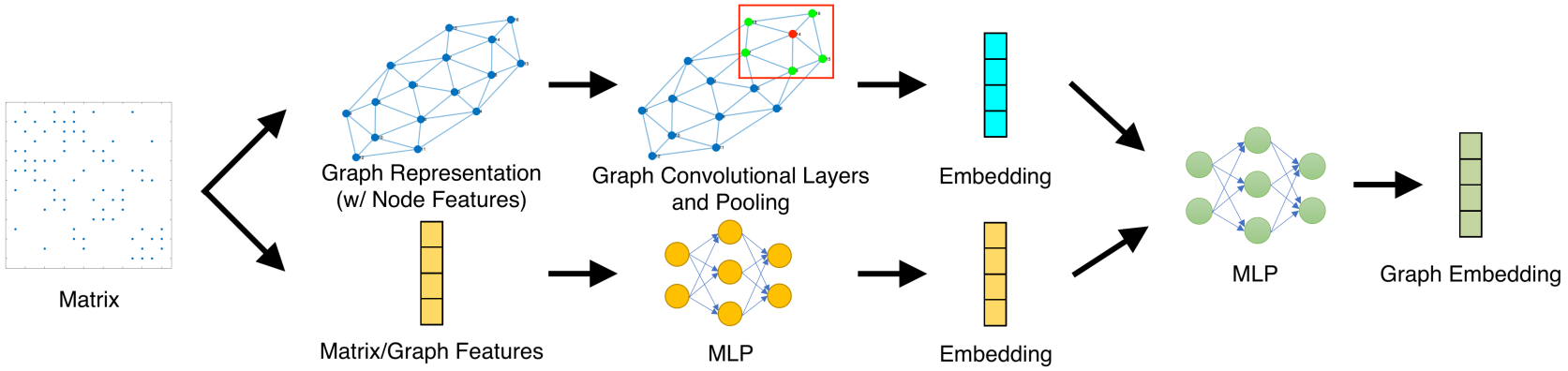
- Some graph features can be converted to node features.
  e.g., **left/right bandwidth**.

[1] V. Eijkhout and E. Fuentes. “A standard and software for numerical metadata.” ACM Transactions on Mathematical Software, 35(4):1–20, February 2009.
[2] Norris, Boyana, et al. "Lighthouse: A user-centered web service for linear algebra software." arXiv preprint arXiv:1408.1363 (2014).

GNN ARCHITECTURE

GNN ARCHITECTURE

Overview



GNN ARCHITECTURE

Convolutional layers

- Graph attention network[1] (GAT)
- GNN with SAmple and aggreGaEt[2] (GraphSAGE)
- Graph convolutional network[3] (GCN)
- Graph isomorphism network with edge features[4] (GINE).

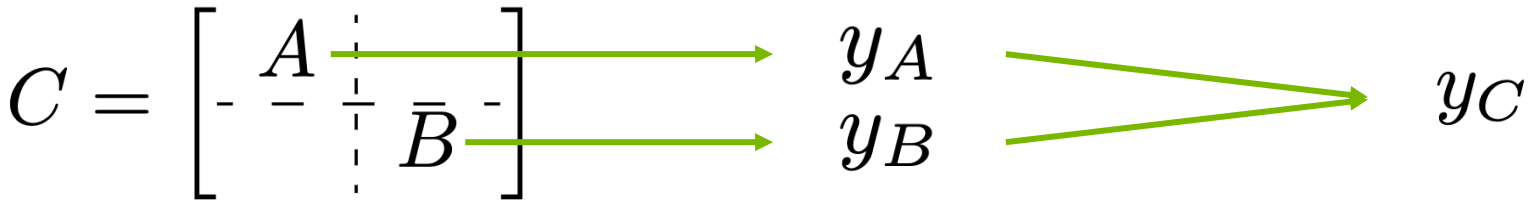
[1] Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. “Graph attention networks.” arXiv preprint arXiv:1710.10903, 2017.
[2] Will Hamilton, Zhitao Ying, and Jure Leskovec. “Inductive representation learning on large graphs.” In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, Advances in Neural Information Processing Systems, volume 30. Curran Associates, Inc., 2017.
[3] Thomas N. Kipf and Max Welling. “Semi-supervised classification with graph convolutional networks.” arXiv preprint arXiv:1609.02907, 2016.
[4] 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.

GNN ARCHITECTURE

Two-level pooling

- Node embeddings are aggregated within their connected components.
  e.g., torch_scatter with respect to connected component id

- Graph embedding is generated from these connected components embeddings.



NUMERICAL TESTS

NUMERICAL TESTS

Configuration and evaluation metrics

- ML tests are conducted on an NVIDIA RTX 3090 GPU and an Intel i7-11700KF CPU.

- Evaluation metrics can be found in TorchMetrics:
  - **Label Ranking Average Precision[1] (LRAP)**
  - **Normalized Discounted Cumulative Gain[2-4] (NDCG)**

[1] Grigorios Tsoumakas, Ioannis Katakis, and Ioannis Vlahavas. “Mining multi-label data.” Data Mining and Knowledge Discovery Handbook, pages 667–685, 2009.
[2] Kalervo Järvelin and Jaana Kekäläinen. “Cumulated gain-based evaluation of IR techniques.” ACM Transactions on Information Systems (TOIS), 20(4):422–446, 2002.
[3] 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.
[4] 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.

NUMERICAL TESTS

Comparison with traditional ML

Table 2: 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

- GNN approaches are comparable to or outperform traditional ML approaches.

NUMERICAL TESTS

Overhead and speedup

Table 3: Building, processing and solving time of selected matrices.

Matrix	size	nnz	t_b	t_p	predicted t_s	worst t_s
1138_bus	1138	4054	0.2084	0.0035	0.5195	5.7607
msc01440	1440	44998	2.6324	0.0035	0.5183	8.3124
cage9	3534	41594	2.4863	0.0035	0.2116	4.2138
cavity13	2356	72034	4.4684	0.0039	0.6144	>600
circuit_1	4875	105339	2.0890	0.0035	0.4243	>600

- The overhead of building a graph can be improved by implementing in parallel and processing on a GPU.

CONCLUSION

CONCLUSION

- Extensible to other software by following a similar routine.
- Translate a matrix related problem into a graph learning problem.
- Some useful **features.**
- **Future work:** modify GNN models to address over-smoothing.

Thank you!

Link to this paper: https://openreview.net/forum?id=tMlBpP1I3Bt