Observe Locally, Classify Globally: Using GNNs to Identify Sparse Matrix Structure

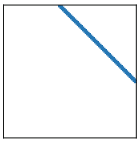
Khaled Abdelaal(✉) and Richard Veras

University of Oklahoma, Norman, OK 73019, USA
{khaled.abdelaal,richard.m.veras}@ou.edu

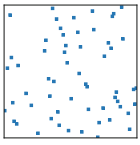
Abstract. The performance of sparse matrix computation highly depends on the matching of the matrix format with the underlying structure of the data being computed on. Different sparse matrix formats are suitable for different structures of data. Therefore, the first challenge is identifying the matrix structure before the computation to match it with an appropriate data format. The second challenge is to avoid reading the entire dataset before classifying it. This can be done by identifying the matrix structure through samples and their features. Yet, it is possible that global features cannot be determined from a sampling set and must instead be inferred from local features. To address these challenges, we develop a framework that generates sparse matrix structure classifiers using graph convolutional networks. The framework can also be extended to other matrix structures using user-provided generators. The approach achieves 97% classification accuracy on a set of representative sparse matrix shapes.

Keywords: Sparse Matrix · Graph Neural Networks · Classification

1 Introduction



(a)



(b)

	Diagonal	Random	Rand+Diag	Kronecker
Diagonal	**500**	0	0	0
Random	0	**490**	10	0
Rand+Diag	0	40	**460**	0
Kronecker	0	20	0	**480**

(c)

Fig. 1. Efficacy of our classifier framework at determining structure when the data is permuted. (a) is an off-diagonal matrix, (b) is a re-labelled variant of (a), and (c) is the confusion matrix for the classifier framework on re-labelled matrices similar to (b). By using GCNs our approach is invariant to node labelling and achieves around 97% accuracy.

I. Rojas et al. (Eds.): IWANN 2023, LNCS 14134, pp. 149–161, 2023.
https://doi.org/10.1007/978-3-031-43085-5_12

Sparse matrices represent a fundamental building block used throughout the field of scientific computing in applications, such as graph analytics, machine learning, fluid mechanics, and finite element analysis [6,13]. Such matrices appear as operands in numerous fundamental computational kernels such as sparse matrix-vector multiplication (SpMV), Cholesky factorization, LU factorization, sparse matrix-dense matrix multiplication, and matricized tensor times Khatri-Rao product (MTTKRP) among others. Building efficient algorithms for this class of kernels mainly depends on the storage format used for the sparse matrix as observed in different studies [1,4]. A variety of such formats are proposed in literature [9,10]. Hence, it is crucial to identify the structure of the matrix to choose the ideal sparse format, and eventually tailor the algorithm to that format to optimize the workload performance. However, identifying the structure of the matrix is not always trivial. Figure 1a shows a spy plot of an off-diagonal sparse matrix, and Fig. 1b shows the same matrix, with some of the original row indices and column indices re-labelled. It is less obvious for the latter figure to provide an insight of the original structure of the non-zeros within the matrix. Additionally, in case of huge sparse matrices, we might only have access to samples of the matrix. This could be because of computational or storage restrictions, or missing data. In these two cases (re-labelling and sub-sampling), we need efficient techniques to recognize the shape of the input matrix.

To tackle mentioned issues, we propose a framework to identify sparse matrices structures, using graph neural networks. Figure 1c shows the confusion matrix for the proposed framework using four sample classes on re-labelled variants. The framework design is modular, allowing users to easily augment it with new structures generators or feature sets. The main contributions of this paper are as follows:

- Proposing a novel, modular Graph Neural Network framework to accurately predict the shapes of sparse matrices, including partial samples, and re-labelled variants of original matrices.
- Presenting a new balanced synthetic dataset for structured sparse matrices.
- Providing a performance analysis of graph-level classification on sparse matrices, using different feature sets.
- Introducing two new compact and efficient feature sets for matrices as graphs, namely: Linear and Exponential Binned One-Hot Degree Encoding.

The rest of this paper is organized as follows: Sect. 2 introduces the necessary background, Sect. 3 details the design of the proposed framework, Sect. 4 discusses the evaluation and results of the framework, while Sect. 5 describes related work. Finally, Sect. 6 summarizes the findings of the paper.

2 Background

2.1 Graph Neural Networks

Graph neural networks (GNNs) [17] are a class of deep learning models that operate on graphs or networks. Unlike traditional neural networks that operate

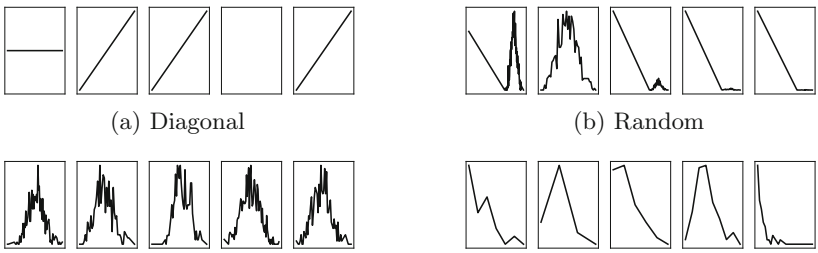
on structured data such as images or sequences, GNNs can handle arbitrary graph structures with varying node and edge attributes, enabling them to learn powerful representations of graph-structured data. The key idea behind GNNs is to iteratively update node embeddings by aggregating information from the embeddings of their neighbors through the "graph convolution" operation. By stacking multiple layers of graph convolution and non-linear activation functions, GNNs can learn hierarchical representations of the graph that capture both local and global information.

2.2 Structured Matrices

Several common structures are observed in sparse matrices, such as:
Diagonal all non-zeros are located on the main or a secondary diagonal. This structure represents a 1D mesh and commonly appears in various scientific and engineering applications.
Random the non-zero elements are randomly distributed across the matrix, with variable density. Such matrices have no specific identifiable structure.
Kronecker Graphs [11] are a class of synthetic graphs that have been widely used to model real-world networks, and are generated by recursively applying the Kronecker product of a small base graph with itself. Let A and B be two matrices. Then, their Kronecker product $A \otimes B$ is given by

$$A \otimes B = \begin{pmatrix} a_{11}B & \cdots & a_{1n}B \\ \vdots & \ddots & \vdots \\ a_{m1}B & \cdots & a_{mn}B \end{pmatrix} \tag{1}$$

where a_{ij} are the entries of A. We use these three classes of structures, and a combination of them, as a representative set that can be combined to form more complex relationships [16,20]. Our framework is not limited to only these structures, and they serve as an example to evaluate its performance.



(a) Diagonal (b) Random

(c) Random+Diagonal (d) Kronecker Graph

Fig. 2. Global Degree Distribution for samples in each matrix (graph) class studied in this paper. In our approach we classify the shape based on local views from sampled data.

Degree as a Representative Node Feature. Figure 2 illustrates that one can accurately distinguish between the different classes based on the degree distribution of the representative graph. For example, for Diagonal matrices (Fig. 2a), the degree for all nodes is low, and is either constant or linear across all nodes. Kronecker graphs follow a power-law degree distribution, with only a few nodes having many connections (high degree) and most of the nodes having relatively few connections (low degree). However, only the local per-node degree view may be immediately available, and not the global graph view. An example of such a case is only having a sample of the graph and not the entire graph due to storage or computational limitations. The power of GNNs can be leveraged to carry out the required task: the prediction of the sample matrix structure.

3 Framework Description

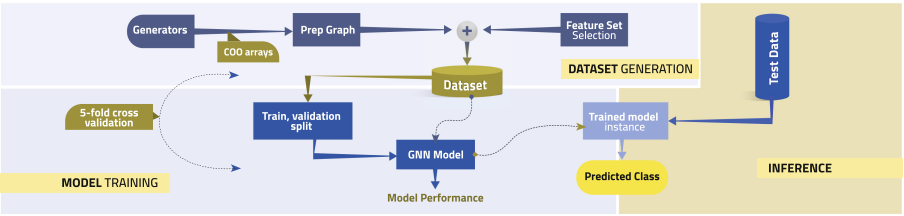


Fig. 3. High-Level overview of the framework. It consists of three main phases: **dataset generation**, where the synthetic sparse matrices are generated, prepared as graphs, and have feature set attached. Then, the GNN **model training** using 5-fold cross validation to capture the model performance, and then generate a trained model instances, that is used later in the **inference** phase.

The goal of the proposed framework is to predict the structure of the input sparse matrix through its classification into one of the configured target classes. We use diagonal, random, diagonal+random, and Kronecker graph as examples of these classes to evaluate the performance of the framework. New structures can be seamlessly integrated. Figure 3 shows a high-level overview of the proposed framework. It consists of three main stages: Dataset generation, Model Training, and Inference. A synthetic dataset is generated using different generators for different shapes of matrices, which are then represented as graphs. In the training phase, we use GNN with 5-fold cross validation to evaluate the model performance. Finally, the trained model instance is used for later inference.

3.1 Dataset Generation

We generate a balanced dataset of 40K synthetic sparse matrices, covering the four sample classes through individual generators. Each of the generators returns a Coordinate (COO) representation for the matrix, excluding the actual non-zero values. The COO representation is then used as the adjacency list to build the graph representation.

3.2 Feature Set Selection

A per-node feature vector is necessary for the graph neural network to classify matrices. Node degree can be calculated for rows/columns in input matrices from their graph representation.

One-Hot Degree Encoding uses a number of features equal to the maximum degree + 1. A limitation of this encoding is that it requires the knowledge of the maximum degree in the entire dataset before training. Also, the required storage is proportional to the maximum degree recorded in the dataset, which increases memory requirements and reduces maximum possible batch size during training. Moreover, it poses complications during inference if the input matrix has a degree greater than the maximum degree in the training set. In our dataset, the maximum training set degree is 7710, so the length of one-hot encoding feature vector per node is 7711, limiting the maximum batch size on GPU to only 1 graph.

Local Degree Profile (LDP) [3] captures the local structural information of nodes in their immediate neighborhood. LDP is calculated for each node as a five-feature vector: the node degree, the minimum degree of its neighbors, the maximum degree of its neighbors, the mean degree for its neighbors, and the standard deviation of the degrees of each neighbors. LDP features are easy to compute for any given graph. Additionally, the number of features per node is fixed, regardless of the used training data. LDP incurs low storage overhead.

Linear Binned One-Hot Degree Encoding. (LBOH) We implement a modified version of one-hot encoding, to address its limitations. LBOH works by having a fixed number of buckets for representing one-hot degrees. Buckets ranges are designed as follows: a set individual sequential buckets from 0 t α (inclusive) where α is a small integer (less than 10). Then, we add a set of buckets with fixed step β from α: $(\alpha + \beta)$, $(\alpha + 2\beta)$, ..., $(\alpha + k\beta)$ where $(\alpha + k\beta)$ is the maximum degree threshold. Any degree greater than $(\alpha + k\beta)$ is mapped to the final bucket.

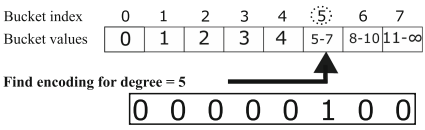


Fig. 4. An example of finding the linear binned one-hot degree encoding for a node with degree = 5, where the parameters for the encoding scheme are $\alpha = 5$, $\beta = 3$, and $k = 2$. Degree 5 is mapped to its associated bucket (5 to 7), then the bucket index (5) is represented using one-hot encoding (1 at the position where the value 5 exists, 0 otherwise).

Figure 4 shows an example of LBOH encoding. As opposed to One-Hot Encoding, LBOH provides a fixed number of features regardless of the maximum degree in the training dataset. At the inference stage, only the values of α, β, and k are needed.

Exponential Binned One-Hot Degree Encoding (EBOH). The main difference between EBOH and LBOH is the kind of step between buckets ranges. Instead of a linear step in LBOH, EBOH uses an exponential step to cover more degree values with a small number of features. First, the value of α is chosen such that $1 \leq \alpha \leq 3$. Then for the buckets, a sequential one-to-one mapping is performed for values 0 through 2^{α}. For the following buckets, the upper bound (inclusive) is $2^{\alpha+i}$ where $i \in [1, k]$ and $k \in \mathbb{N}^*$.

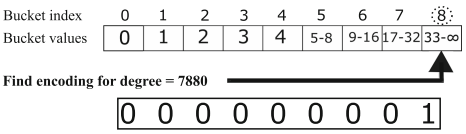


Fig. 5. An example of finding the exponential binned one-hot degree encoding for a node with degree = 7880, where the parameters for the encoding scheme are $\alpha = 2$ and $k = 3$. Degree 7880 is mapped to its associated bucket (33 to ∞), then the bucket index (8) is represented using one-hot encoding (1 at the position where the value 8 exists, 0 otherwise).

Figure 5 shows an example of EBOH encoding. EBOH encoding still provides the benefit of having the number of features independent of the maximum degree in the training set.

3.3 The Graph Neural Network Architecture

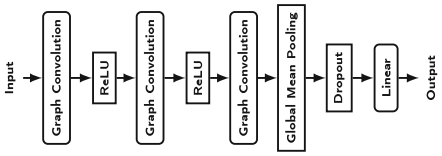


Fig. 6. Graph Neural Network architecture.

To identify the structure of the input matrix, the matrix is viewed as the adjacency list of a graph, enabling the use of machine learning methods designed for graph data. GNNs provide additional benefits such as allowing the use of matrices (graphs) of arbitrary sizes as input, Also, GNNs are agnostic to node ordering. This powerful property enables re-labelling or permuting nodes in a graph representing a sparse matrix, while maintaining accurate predictions. The machine learning task of interest is graph-level prediction since a single label (class) is needed for an entire graph (matrix). The GNN architecture is shown in Fig. 6. The hidden layers are three graph convolution layers and one linear (output) layer. Graph convolution is an operation where node embeddings are iteratively

generated as the aggregations from the node neighborhoods. This operation is used to capture complex features of the graph. The first convolution layer aggregates information from the local neighborhood of each node. This operation is repeated in subsequent convolution layers in order to propagate information to increasingly larger neighborhoods. By the end of three convolution layers, the model has learned a hierarchical representation of the graph, where the features at each layer capture increasingly complex structural patterns. The learned representation so far is "node embeddings". Then, learned node embeddings are reduced into a single graph embedding using a global mean pooling operation (called readout layer). Samples are randomly dropped out to reduce overfitting. Finally, a linear classifier is applied to the graph embedding.

```
def generateDiagRandom(size, threshold=2):
    """ A function to generate a Diag+Random square matrix """
    tuples = [(x,y) for x in range(size) for y in range(size) if (random
        .randint(0,10) <= threshold or x == y)]
    # seperate tuples into two lists: the row array and the column array
    coo_rep = list(map(list, zip(*tuples)))
    return coo_rep[0], coo_rep[1], [size, size]
```

```
def process(self):
    catMap = [...., {
            # Number of instances to generate for this class
            'num_iter':10000,
            # Name of the generator function
            'generator':generateDiagRandom,
            # A string list of required generator function param
            'gen_params':['random.randint(MIN_DIM_SIZE,MAX_DIM_SIZE)'] }]
```

Fig. 7. The two steps needed to add a new class to the classifier framework. First (top), create a new generator function in the generators file, and second (bottom), add a dictionary entry to `catMap` list in the `process` method of the dataset class.

Modularity. New shapes of matrices can be easily integrated in our framework. To achieve this, two steps are needed as shown in Fig. 7: (1) write a generator for that new shape, and (2) add an entry to the categories (shapes) map in the dataset class for this shape, containing the number of dataset instances to generate, the name of the generator function, and the different required parameters. The generator is required to return the COO representation excluding values, and the matrix dimensions. After generating the new dataset instances for this class, one does not need to re-train the entire model again. Transfer learning [21] can be used to replace the last layer of the trained model with a new layer that has the appropriate number of outputs, after introducing the new shape(s). Then, the weights of all previous layers are frozen and only the new layer is trained. Another aspect of modularity in our framework is the ability to seamlessly attach different feature sets. Feature sets are only computed when the graph is queried. To implement a new feature set, a modification to the `get` method of the dataset is needed. This method first reads in the graph file from disk, calculates the new feature set, and attaches it to the graph.

4 Analysis

We run a set of experiments to evaluate the accuracy of our approach in detecting structures, using the different feature representation discussed in Sect. 3.2.

4.1 Evaluation

Experimental Setup. Table 1 shows the experimental setup and learning parameters used in the experiments. We use PyTorch Geometric [8] for the GNN.

Evaluation Metrics. To evaluate the prediction accuracy of the framework, four derived metrics are used: accuracy, precision, recall, and F1-score. We report per-class and overall accuracy and F1 score numbers, since the latter is the harmonic mean of precision and recall. Using both accuracy and F1-score helps provide a more comprehensive evaluation of the framework's performance. Accuracy gives an overall view of how well the classifier is performing, while the F1-score provides insights into its ability to correctly classify positive instances.

Table 1. Experimental setup and Training parameters used in the experiments. * Batch size used for traditional one-hot encoding is 1.

Component	Specification	Parameter	Value
GPU	NVIDIA RTX A6000	Optimizer	Adam
GPU Memory	48 GB GDDR6	Learning Rate	0.01
CUDA Version	11.8	Error Criterion	Cross Entropy
Main Memory	64 GB DDR4	Batch Size	256*
Operating System	Ubuntu 22.04	Cross Validation Folds	5

4.2 Results

Table 2. Performance of the classifier for different degree representations

	Degree Representation							
	One-Hot Encoding		LDP		LBOH		EBOH	
	Accuracy	F1 Score	Accuracy	F1 Score	Accuracy	F1 Score	Accuracy	F1 Score
Diagonal	1.0	0.90	1.0	0.97	1.0	1.0	1.0	1.0
Random	0.90	0.91	0.64	0.76	0.92	0.95	0.95	0.96
Random+Diagonal	0.86	0.99	0.98	0.83	0.96	0.94	0.97	0.96
Kronecker	0.90	0.94	0.90	0.93	0.98	0.97	0.96	0.98
Overall	0.90	0.90	0.88	0.88	0.97	0.97	0.97	0.98

Classification Performance. Table 2 shows the accuracy and F1 score for the classifier using the different degree representations discussed in Sect. 3.2.

Performance results show that both LBOH, and EBOH provide high prediction accuracy of around 97% and a F1 score of around 98%. On the other hand, traditional one-hot encoding exhibits a lower accuracy of around 90%. One-Hot Encoding requires a significantly large number of features per node (7711), limiting the training batch size on the A6000 GPU to only one graph. This forces the optimizer to adjust the neural network weights very frequently, hence hurting the overall accuracy. Using LDP as a feature set exhibits variant model performance across folds depending on the validation set being used. In some folds, LDP provides high accuracy of around 97% to 98% similar to EBOH. In other folds, LDP fails to converge to an acceptable loss value, and ends up with an accuracy of around 74% on the last few epochs. This performance variance across folds deems LDP unfit for the purposes of our application. It significantly fails in two classes: Random and Kronecker. It predicts Random matrices as Random+Diagonal for more than 32.5% of the instances. This is likely due to the prevalence of the local degree neighbor summary features (the last four LDP features) instead of focusing on the node degree. This eventually results in failing to discover the global hierarchical structures in the matrix. LDP still shows perfect accuracy in case of diagonal matrices since almost all nodes in the matrix's graph have the same degree. LDP prediction quality for Kronecker graphs is also lower than other evaluated feature sets (around 81% in some folds) for the same reasons.

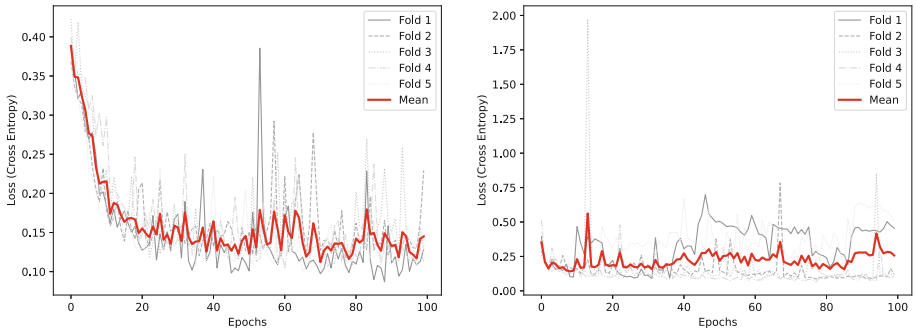


Fig. 8. Cross Entropy Loss across different folds in 5-fold cross validation training using (a) EBOH, and (b) LDP feature set.

Fig. 8a demonstrates the validation loss across the 5 different folds for EBOH. It shows almost no variance in the loss across the different folds, indicating the stability of the model's performance across folds. On the other hand, Fig. 8b shows the validation loss for LDP and illustrates that the loss does not converge in 2 out of 5 folds.

Classifying Sub-samples and Re-labelled Subgraphs. To test the efficacy of GNN on both aspects, we generate 200 new matrices: 50 for each of the four classes. For each of them, we generate 10 subgraphs and 10 re-labelled variants. To generate the subgraphs, we use uniform random node sampling (URNS) [12]: nodes are randomly selected with uniform probability, as well as the edges connecting the selected nodes. Re-labelling of a graph G simply renames the nodes V of the graph, and produces a new graph G' with the same size and degree distribution of the original graph G. Figure 9 shows an example of both URNS and random re-labelling.

Table 3. Accuracy comparison for node sampling, node re-labelling, and original graphs using EBOH feature set.

Class	Node Sampling	Node Re-labelling	Original Graphs
Diagonal	1	1	1
Random	0.83	0.98	0.98
Random+Diagonal	0.92	0.92	0.92
Kronecker	0.94	0.96	0.96
Overall	**0.92**	**0.97**	**0.97**

Table 3 shows the model's performance on subgraphs and re-labelled variants as compared to original full graphs. The table shows that re-labelling node has no impact on the classification accuracy; it shows the same overall accuracy of around 97% which is observed for the original graphs. This is expected because the arrangement of nodes in a graph is irrelevant, since the graph has the same degree distribution. For subgraphs generated using URNS of larger graphs, the overall accuracy drops to around 92%. The reason being that random node sampling can alter the degree distribution of the graph. The random choice of nodes can result in either isolated nodes (no edges) or much lower degree nodes as compared to the original graph. This affects the accuracy specially for complex

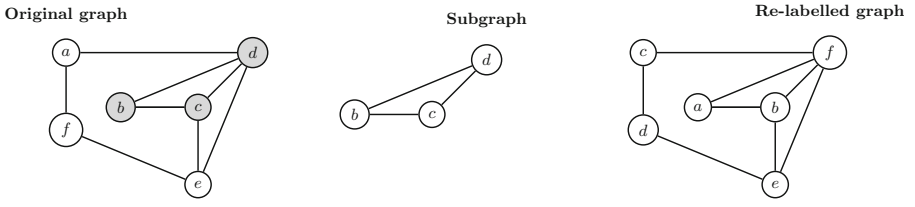


Fig. 9. Example of generating a random sub-sample and a re-labelled variant of a graph. The original graph (left) contains six nodes. Using URNS, a random subgraph (middle) of three nodes is generated. A re-labelled variant (right) is generated using a random 1:1 mapping between the original and new node labels.

shapes such as random and Kronecker graphs. One way to reduce the accuracy loss for samples is to use a more sophisticated graph sampling technique rather than randomly selecting nodes or edges.

5 Related Work

Prediction on Sparse Matrices. Several studies investigated the use of machine learning to predict the optimal sparse format for SpMV on CPU and GPU [2,14,15,18,19,23]. Our framework does not directly predict the best sparse format, instead, we only predict the structure of the input matrix. This allows de-coupling the sparsity pattern from the sparse format, following the argument adopted by AlphaSparse [7] since our framework also allows the seamless integration of new classes. Existing techniques collect a set of features from each matrix such as: the number of diagonals, the ratio of true diagonals to total diagonals, the (maximum) number of non-zeros per row, the variation of the number of nonzeros per row, the ratio of nonzeros in DIA and ELL data structures, and a factor or power-law distribution. We only need to calculate one feature per node: its degree. Also, [23] uses a CNN approach to treat matrices as images, and in order to fix the size of the matrix, they normalize input matrices into fixed size blocks, losing partial matrix information in the process. In contrast, our approach handles arbitrary sizes of matrices, without losing precision, leveraging the power of Graph Neural Networks. We can optionally sample large matrices and maintain high prediction accuracy. An additional benefit to our framework is that it is order in-variant, since matrices are represented as graphs.

Graph Representation for Learning. Representing non-attribute graphs is an open problem [5]. Common approaches employ graph properties such as node degree, more specifically a one-hot encoding of the degree [22]. One-hot encoding suffers from numerous limitations (Sect. 3.2). LDP [3] provides a compact representation for graph using five features per node. Although the computation of such feature vector is efficient, using LDP results in unreliable model performance for our task (Sect. 4.2). Both our representations (LBOH and EBOH) outperform one-hot encoding and LDP while addressing their shortcomings.

6 Summary

In this paper, we proposed a GNN based framework to classify structured sparse matrices. We introduced two novel non-attribute graph representations based on node degrees: LBOH, and EBOH. We evaluated the efficacy of our framework on a synthetic, balanced dataset of matrices that we generated containing random matrices from four sample classes: diagonal, random, random+diagonal, and Kronecker graphs. Performance results demonstrate a high classification accuracy of 97% for the framework when using our feature sets: LBOH and EBOH. They also show high accuracy of 92% and 97% on random node subsamples and re-labelled variants respectively. Our framework is modular, allowing the

inclusion of additional classes with minimal user effort. Future endeavors target the automatic generation of the optimal sparse data format and algorithm for sparse matrix kernels, using the obtained prediction results from the current framework.

References

1. Bell, N., Garland, M.: Implementing sparse matrix-vector multiplication on throughput-oriented processors. In: Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis, pp. 1–11 (2009)
2. Benatia, A., Ji, W., Wang, Y., Shi, F.: Sparse matrix format selection with multiclass SVM for SPMV on GPU. In: 2016 45th International Conference on Parallel Processing (ICPP), pp. 496–505 (2016)
3. Cai, C., Wang, Y.: A simple yet effective baseline for non-attribute graph classification. arXiv preprint arXiv:1811.03508 (2018)
4. Choi, J.W., Singh, A., Vuduc, R.W.: Model-driven autotuning of sparse matrix-vector multiply on GPUs. Association for Computing Machinery, New York (2010)
5. Cui, H., Lu, Z., Li, P., Yang, C.: On positional and structural node features for graph neural networks on non-attributed graphs. Association for Computing Machinery, New York (2022)
6. Davis, T.A., Hu, Y.: The university of Florida sparse matrix collection, vol. 38, no. 1 (2011)
7. Du, Z., Li, J., Wang, Y., Li, X., Tan, G., Sun, N.: Alphasparse: generating high performance SPMV codes directly from sparse matrices. In: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis, SC 2022. IEEE Press (2022)
8. Fey, M., Lenssen, J.E.: Fast graph representation learning with PyTorch Geometric. In: ICLR Workshop on Representation Learning on Graphs and Manifolds (2019)
9. Filippone, S., Cardellini, V., Barbieri, D., Fanfarillo, A.: Sparse matrix-vector multiplication on GPGPUs, vol. 43, no. 4 (2017)
10. Langr, D., Tvrdík, P.: Evaluation criteria for sparse matrix storage formats. IEEE Trans. Parallel Distrib. Syst. **27**(2), 428–440 (2016)
11. Leskovec, J., Chakrabarti, D., Kleinberg, J., Faloutsos, C., Ghahramani, Z.: Kronecker graphs: an approach to modeling networks. J. Mach. Learn. Res. **11**(33), 985–1042 (2010)
12. Leskovec, J., Faloutsos, C.: Sampling from large graphs. Association for Computing Machinery, New York (2006)
13. Leskovec, J., Krevl, A.: SNAP Datasets: Stanford large network dataset collection (2014). https://snap.stanford.edu/data
14. Li, J., Tan, G., Chen, M., Sun, N.: SMAT: an input adaptive auto-tuner for sparse matrix-vector multiplication, vol. 48, no. 6 (2013)
15. Li, K., Yang, W., Li, K.: Performance analysis and optimization for SPMV on GPU using probabilistic modeling. IEEE Trans. Parallel Distrib. Syst. **26**(1), 196–205 (2015)
16. Puschel, M., et al.: Spiral: code generation for DSP transforms. Proc. IEEE **93**(2), 232–275 (2005)
17. Scarselli, F., Gori, M., Tsoi, A.C., Hagenbuchner, M., Monfardini, G.: The graph neural network model. IEEE Trans. Neural Networks **20**(1), 61–80 (2009)

18. Su, B.Y., Keutzer, K.: clSpMV: a cross-platform OpenCL SpMV framework on GPUs. Association for Computing Machinery, New York (2012)
19. Tan, G., Liu, J., Li, J.: Design and implementation of adaptive SPMV library for multicore and many-core architecture, vol. 44, no. 4 (2018)
20. Van Loan, C.: Computational frameworks for the fast Fourier transform. SIAM (1992)
21. Weiss, K., Khoshgoftaar, T.M., Wang, D.: A survey of transfer learning. J. Big Data **3**(1), 1–40 (2016)
22. Xu, K., Hu, W., Leskovec, J., Jegelka, S.: How powerful are graph neural networks? In: International Conference on Learning Representations (2019)
23. Zhao, Y., Li, J., Liao, C., Shen, X.: Bridging the gap between deep learning and sparse matrix format selection. Association for Computing Machinery, New York (2018)