DENSITY-PRESERVING HETEROGENEOUS GRAPH SPARSIFICATION FOR REPRESENTATION LEARNING

Srilekha Geda & Chunjiang Zhu
Department of Computer Science, UNC Greensboro, Greensboro, NC 27412, USA

ABSTRACT

Graph sparsification is the task of compressing a graph with fewer edges or nodes while preserving its essential structural characteristics. It has been used in machine learning to significantly improve the computational efficiency over homogeneous graphs. In heterogeneous graphs with diverse types of nodes and edges, however, sparsification has not been extensively explored. This work develops sparsification methods that can preserve edge density across different edge types and/or edge importance in terms of eigenvector centrality, improving over existing methods. The methods have been tested on real-world networks, and the results indicate great improvements in the computational efficiency and memory cost.

1 INTRODUCTION

In recent years, the analysis of complex networks has become increasingly pivotal across various domains, ranging from social systems and biological structures to recommendation platforms. It has become evident that many interconnected systems exhibit inherent heterogeneity, involving diverse types of nodes and edges that encode multifaceted relationships. A crucial processing of heterogeneous graphs is to learn a high-quality representation/embedding for downstream learning tasks. There have been various representation learning algorithms developed over years such as (1; 2; 3; 4; 5; 6; 7). However, modern heterogeneous graphs are often of very large size (millions of nodes to billions of edges) and complex (diverse node/edge types), thus imposing high computational and storage demands. One can often use sparsification techniques to compress a graph with fewer edges or nodes while preserving its essential structural property.

In this paper, we study the under-explored sparsification problem in heterogeneous information networks and develop two novel algorithms. Unlike existing methods, our approaches can preserve edge density and/or centrality importance in the original graph. We perform extensive experiments in multiple datasets, diverse representation learning algorithms, and two graph learning tasks, link prediction and node classification, to demonstrate the improvement in the training time and memory consumption without significant loss in the quality of the learned representation.

Related Work. Several notions of graph sparsification have been proposed such as spectral sparsifiers (8) which approximate the graph spectrum and cut sparsifier that approximately preserve all graph cut values where both are not primarily focused on heterogeneous graphs and machine learning tasks. (9) developed a random sampling method where a fixed number of edges are uniformly randomly sampled for each edge type, thereby losing critical edge density and edge centrality information. Jiang et al. (10) developed edge sparsification for efficient node and schema pre-training.

2 DENSITY-PRESERVING SPARSIFICATION METHODS

The prior method *Random* (9) samples, for each node, a pre-determined number k edges out of edges of each type uniformly at random. In the sparsified graph, (most) nodes have exactly k edges from each type, completely losing the number of edges across different types, called *edge density* in the full graph. Edge density information represents the connectivity distribution of diverse edge types and should be encoded in a powerful learned representation. To this end, we first propose a density-preserving (*DP*) sparsification method called *DP-Random* that can preserve edge density by keeping a fixed percentage K of edges of each type for a node. Specifically, it performs sparsification on incoming and outgoing edges (out-edges) of a node independently. Consider out-edges. For a node u and its out-edges $E^t_{Out}(u)$ of edge type t, it first computes the number $m = K * |E^t_{Out}(u)|$ of out-edges of type t that should be in the sparsifier H, and the number x of out-edges that are already

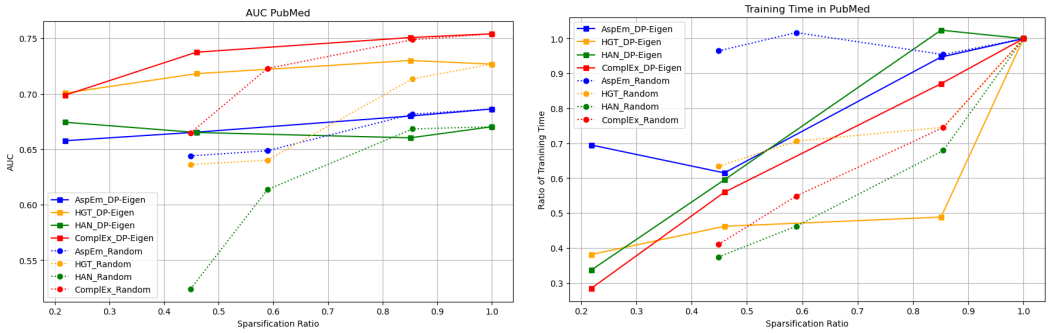
included in H. Then $m - x$ edges out of $E^t_{Out}(u) \setminus H$ are sampled uniformly at random and added to H. Keep in mind that we process nodes one at a time, and so processing a node before the one we're processing right now could have added a few edges in $E^t_{Out}(u)$ to H.

Although *DP-Random* can preserve edge density, it randomly samples edges to be included into the sparsifier, thus limited in overlooking the importance of different edges of the same type. To remedy for this, we propose the *DP-Eigen* method that employs edge centrality, exactly eigenvector centrality, to measure edge importance and only includes more important edges with higher centrality. Unlike *DP-Random*, *DP-Eigen* is designed to preserve both edge density and importance. Eigenvector centrality serves as a metric for the importance of nodes in a graph. We define the centrality of an edge (u, v) as the mean of the eigenvector centrality of its two endpoints u and v. *DP-Eigen* is similar to *DP-Random* except that when selecting edges from $E^t_{Out}(u) \setminus H$, $m - x$ edges with the highest eigenvector centralities are added to H. See the formal pseudocode in Appendix.

Remark that for a large heterogeneous graph, the prior method (9)'s parameter k may be hard to determine in advance, while our methods' parameter, the desirable percentage K of compression, is much easier to set. The computational time of *Random*, *DP-Random*, and *DP-Eigen* are $O(nkt)$, $O(mK)$, and $O(mK + n^{2.38})$, where m and t are the number of edges and edge types, respectively.

3 Experimental Evaluations

We used the NetworkX library (11) to compute node eigenvector centrality with an error tolerance of $1e^{-6}$ and performed all experiments in a computer with 8 cores, 2.3GHz Intel processors, and 64GB RAM. We chose the PubMed chemical dataset and Yelp review dataset from the benchmark (12) and used four diverse representation learning models AspEm (1), HAN (2), HGT (3), and ComplEx (4). The experimental setup of link prediction and node classification completely follow the baseline method (Random) (9), where the embedding size is set to 50, the AUC and MRR (mean reciprocal rank) are measures of link prediction and macro-F1 and micro-F1 are measures of node classification, and the training time and memory consumption measure the efficiency.



(a) AUC in PubMed (b) Training Time in PubMed

Figure 1: Performance Comparison of DP-Eigen and the baseline Random

We set $K = 25\%, 50\%$, and 75% to generate graphs of sparsification ratio (the ratio of the number of edges in the sparsifier and the original graph) similar to Random with $k = 1, 2, 10$ in PubMed and $k = 50, 200, 500$ in Yelp. Fig. 1(a) shows the dominating AUC improvement of DP-Eigen over Random, with an increase ranging from 2% to 14% for the 45% graph. The performance gap becomes smaller for the 86% graph and the 100% graph is exactly the original graph. Notably the performance boost is more obvious for a more aggressive sparsification: DP-Eigen's AUCs in the 21% sparsifier are even higher than Random's in the 45% sparsifier and close to the AUCs in the full graph. This verifies DP-Eigen can preserve crucial structural information in the original graph. As shown in Fig. 1(b), the graph compression translates to faster training time, especially for the performant ComplEx and HGT, which only require 28% and 38% of training time compared to the full graph for the 21% sparsifier. Similar observations can be made in Yelp where the reduction of the memory consumption is larger than in PubMed, possibly due to the larger size. Consistent with previous work (9), the impact of sparsification in the node classification is not as clear as link prediction (e.g., sparsifiers often have higher F1 values than the full graph). We also evaluate the performance of our DP-Random and DP-Eigen to show that in general DP-Eigen outperforms DP-Random especially in the realm of very small sparsifiers. See all supporting figures in Appendix.

URM STATEMENT

The authors acknowledge that the first author of this work is female and meets the URM criteria of ICLR 2024 Tiny Papers Track.

REFERENCES

[1] Y. Shi, H. Gui, Q. Zhu, L. Kaplan, and J. Han. Aspem: Embedding learning by aspects in heterogeneous information networks. In Proceedings of the 2018 SIAM International Conference on Data Mining, pages 144–152. SIAM, 2018.

[2] X. Wang, H. Ji, C. Shi, B. Wang, Y. Ye, P. Cui, and P. S. Yu. Heterogeneous graph attention network. In The World Wide Web Conference, pages 2022–2032, 2019.

[3] Z. Hu, Y. Dong, K. Wang, and Y. Sun. Heterogeneous graph transformer. In Proceedings of The Web Conference 2020, pages 2704–2710, 2020.

[4] T. Trouillon, J. Welbl, S. Riedel, E. Gaussier, and G. Bouchard. Complex embeddings for simple link prediction. In International conference on machine learning, pages 2071–2080. PMLR, 2016.

[5] Y. Dong, N. V. Chawla, and A. Swami. metapath2vec: Scalable representation learning for heterogeneous networks. In Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining, pages 135–144, 2017.

[6] T.-Y. Fu, W.-C. Lee, and Z. Lei. Hin2vec: Explore meta-paths in heterogeneous information networks for representation learning. In Proceedings of the 2017 ACM on Conference on Information and Knowledge Management, pages 1797–1806, 2017.

[7] A. Bordes, N. Usunier, A. Garcia-Duran, J. Weston, and O. Yakhnenko. Translating embeddings for modeling multi-relational data. Advances in neural information processing systems, 26, 2013.

[8] D. A. Spielman, and S. H. Teng. Spectral sparsification of graphs. SIAM Journal on Computing, 40(4), 981-1025, 2011.

[9] C. Chunduru, C. Zhu, B. Gains and J. Bi, "Heterogeneous Graph Sparsification for Efficient Representation Learning," in 2022 IEEE International Conference on Bioinformatics and Biomedicine (BIBM), Las Vegas, NV, USA, 2022 pp. 1891-1896. doi: 10.1109/BIBM55620.2022.9995124

[10] X. Jiang, T. Jia, Y. Fang, C. Shi, Z. Lin, and H. Wang. Pre-training on large-scale heterogeneous graph. In Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining, pages 756–766, 2021.

[11] Aric A. Hagberg, Daniel A. Schult and Pieter J. Swart. Exploring network structure, dynamics, and function using NetworkX, in Proceedings of the 7th Python in Science Conference (SciPy2008), Gäel Varoquaux, Travis Vaught, and Jarrod Millman (Eds), (Pasadena, CA USA), pp. 11–15, Aug 2008.

[12] C. Yang, Y. Xiao, Y. Zhang, Y. Sun, and J. Han. Heterogeneous network representation learning: A unified framework with survey and benchmark. IEEE Transactions on Knowledge and Data Engineering, 2020.

A FORMAL PSEUDOCODE

We start by introducing the definitions and notations we will use. A heterogeneous graph can be defined as $G(V, E, \phi, \pi, X, R)$ where V is the node set, E is the edge set, ϕ specifies the type for each node, π assigns the type of each edge, and X and R include the node and edge features respectively. ϕ, π , X, and R can be omitted from the presentation if they are clear from the context. Let $E^t_{Out}(u)$ be the set of u's out-edges of type t and $E^t_{In}(u)$ be the set of u's in-edges of type t.

Algorithm 1: DP-Random Sparsify-Graph

Data: A graph G and a parameter K
Result: A sparsified graph
1: $H = \emptyset$;
2: Sort nodes in ascending order of degree;
3: **for** *each node* u *in the sorted order* **do**
  Sparsify-Node-Out(u, G, H, K);
  Sparsify-Node-In(u, G, H, K);
end
4: return (V, H)

Algorithm 2: DP-Random Sparsify-Node-Out(-In)

Data: A node u in a graph G, an edge set H, and a percent K
Result: Updated H
1: **for** *each edge type t* **do**
  $m = K * |E^t_{Out}(u)|$;
  Compute $x = |E^t_{Out}(u) \cap H|$;
  Sample $m - x$ edges from $E^t_{Out}(u) - H$ uniformly at random; let the sampled edges be S;
  Include S into H
end
2: return (V, H)

Algorithm 3: DP-Eigen Sparsify-Graph

Data: A graph G and a parameter K
Result: A sparsified graph
1: $H = \emptyset$;
2: Compute Eigenvector centrality;
3: Sort nodes in ascending order of degree;
4: **for** *each node* u *in the sorted order* **do**
  Sparsify-Node-Out(u, G, H, k);
  Sparsify-Node-In(u, G, H, k);
end
5: return (V, H)

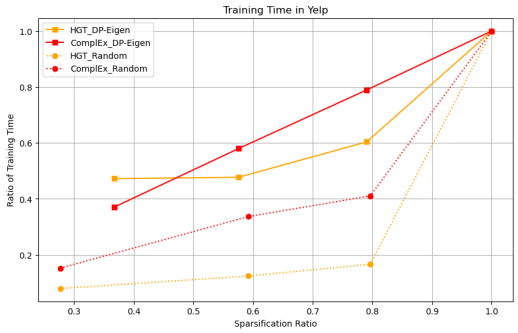
Algorithm 4: DP-Eigen Sparsify-Node-Out(-In)

Data: A node u in a graph G, an edge set H, and a percent K
Result: Updated H
1: **for** *each edge type t* **do**
  $m = K * (E^t_{Out}(u))$;
  Compute $x = |E^t_{Out}(u) \cap H|$;
  Sort $m - x$ edges from $E^t_{Out}(u) - H$ based on its importance by eigenvector centrality; let the sorted edges be S;
  Include S into H
end
2: return (V, H)

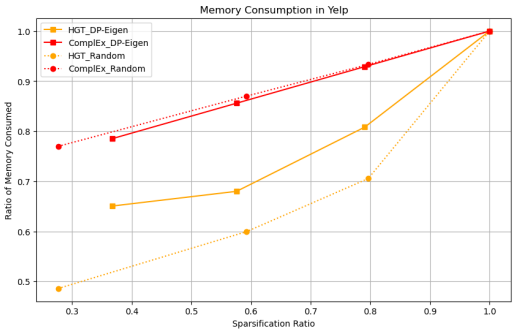
B MORE EXPERIMENTAL RESULTS

Table 1: Statistics of Tested Real-world Networks

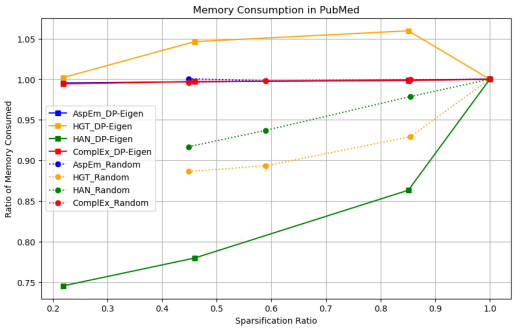
Dataset	***n***	***m***	***Node Types***	***Edge Types***
PubMed	63,109	236,458	4	10
Yelp	82, 465	16, 274, 179	4	4



(a) Training Time in Yelp

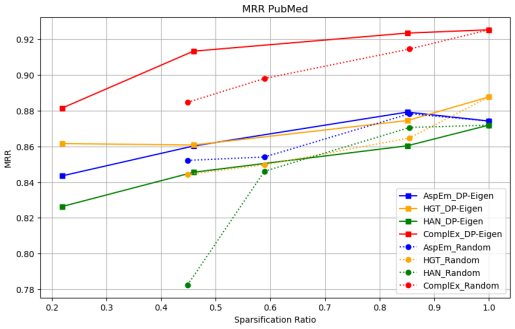


(b) Memory Consumption in Yelp

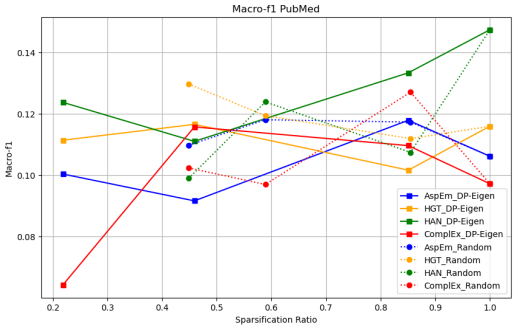


(c) Memory Consumption in PubMed

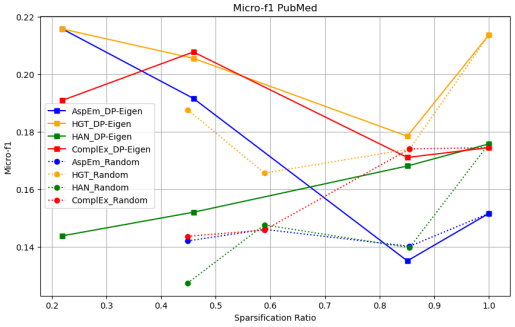
Figure 2: Comparison of Training Time and Memory Consumption



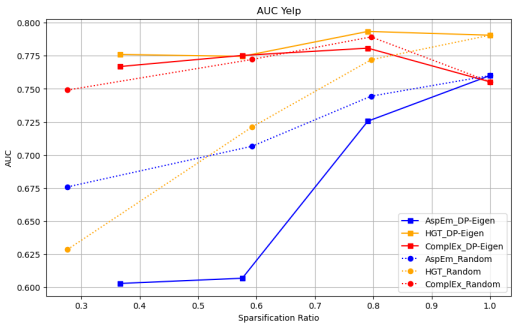
(a) MRR in PubMed



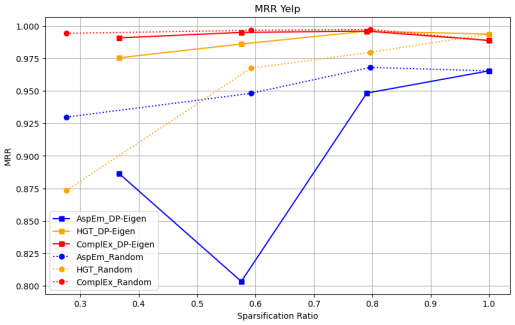
(b) Macro-F1 in PubMed



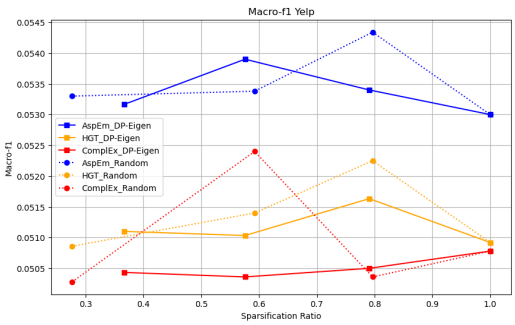
(c) Micro-F1 in PubMed



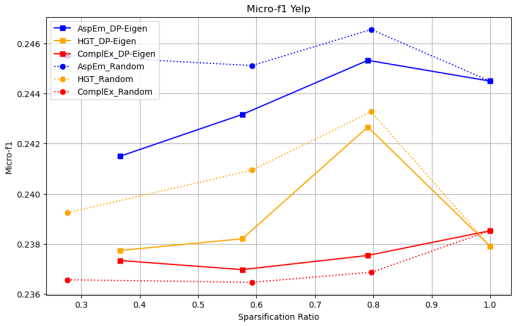
(d) AUC in Yelp



(e) MRR in Yelp

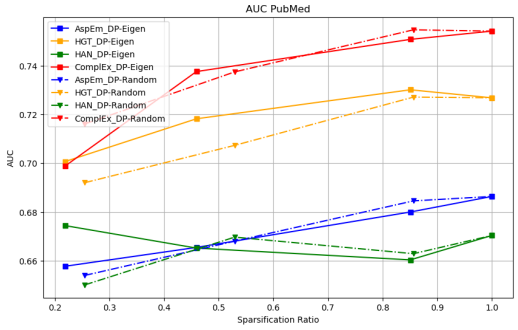


(f) Macro-F1 in Yelp

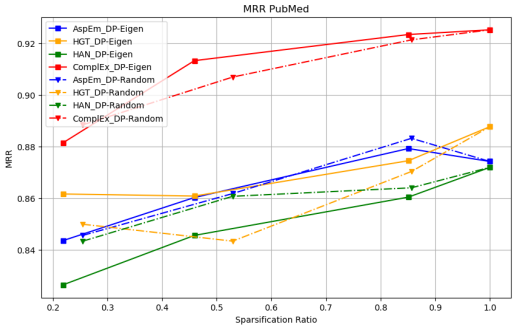


(g) Micro-F1 in Yelp

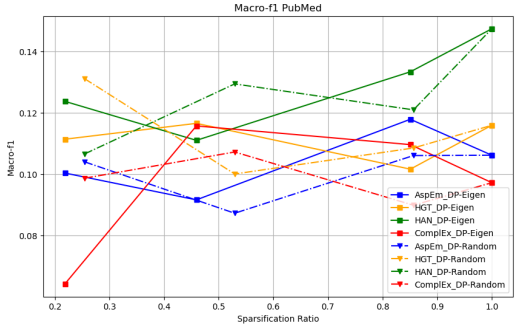
Figure 3: Performance Comparison of DP-Eigen and Random



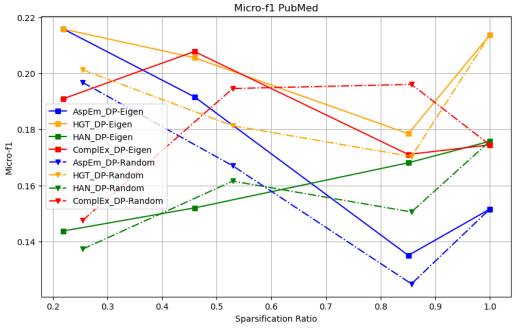
(a) AUC in PubMed



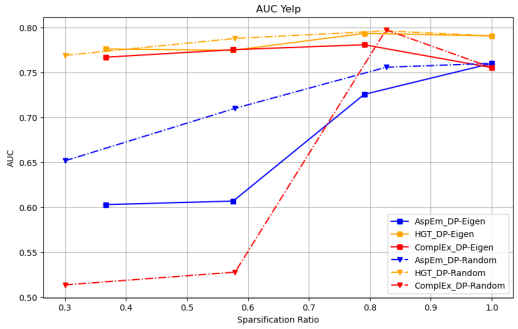
(b) MRR in PubMed



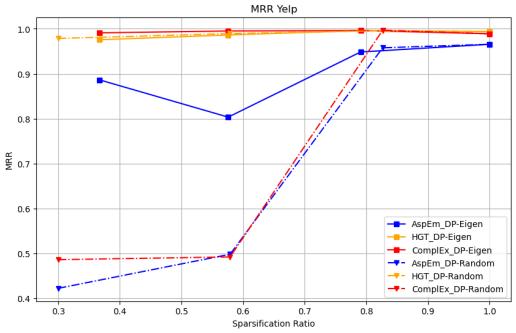
(c) Macro-F1 in PubMed



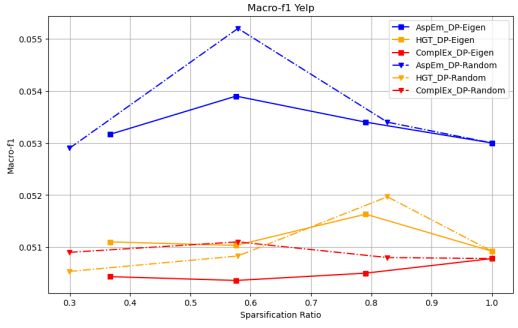
(d) Micro-F1 in PubMed



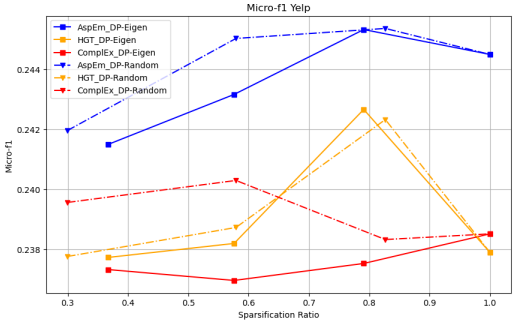
(e) AUC in Yelp



(f) MRR in Yelp



(g) Macro-F1 in Yelp



(h) Micro-F1 in Yelp

Figure 4: Performance Comparison of DP-Eigen and DP-Random