Optimizations for Matching Algorithms in Generative Graph Models

Laya Pullela
UC Santa Barbara
lpullela@ucsb.edu

Arul Saxena
UC Santa Barbara
saxena@ucsb.edu

Abstract

Matching algorithms are a critical part of generative graph machine learning models, such as Graph Variational Autoencoders, which rely on matching to align latent representations with generated graphs. The existing approach is computationally expensive and limits these models to generating small graphs. In this paper, we propose novel matching algorithms designed to improve efficiency without significant loss in matching quality.

1 Introduction

Graph-based generative models have gained significant attention in recent years due to their ability to capture complex relationships and structures, with applications ranging from social network analysis to small molecule generation. Among these, Graph Variational Autoencoders (GraphVAEs), introduced by Simonovsky and Komodakis, represent a promising approach for generating small molecular graphs [14]. GraphVAEs employ a latent variable model to encode graph structures into a continuous latent space and reconstruct graphs from this latent representation.

A critical step in the GraphVAE pipeline is graph matching, which is essential for computing the reconstruction loss. This loss ensures that the generated graph closely resembles the input graph, both in terms of structural connectivity and feature attributes of nodes and edges. However, graph matching is computationally challenging. Unlike sequences or images, graphs are permutation invariant, meaning that the same graph can be represented by many different adjacency matrices. Consequently, graph matching requires comparing every possible pairing of nodes and edges in the input and reconstructed graphs. This leads to a computational complexity of $O(n^4)$, where n is the number of nodes in the graph. This computational bottleneck significantly limits the scalability of GraphVAEs, confining their application to small graphs with $n \leq 38$.

In this work, we address this limitation by proposing a more efficient approach to graph matching. Our method leverages the inherent properties of nodes in a graph, such as degree, importance, and community structure, to reduce the number of comparisons required during matching. By focusing only on pairings that are likely to contribute to the similarity score, we aim to achieve substantial computational savings without compromising accuracy. Specifically, we introduce a binning strategy that groups nodes based on these properties and restricts comparisons to nodes within the same bin. This approach has the potential to make GraphVAEs more scalable, enabling the generation of larger molecular graphs and expanding their applicability to more complex problems.

2 Background

One of the biggest challenges to graph machine learning concerns the inherent complexity of graphs due to an explosion of equivalent representations. For example, a graph $G(V, E)$, where V represents the vertices (nodes), and E represents the edges, can have exactly $|V|!$ unique adjacency matrices.

Preprint. Under review.

It's important for representations of the adjacency matrix to be permutation invariant to ensure the model is generalizable because input graphs do not adhere to a particular ordering [8] [11].

The goal of finding a permutation π between two graphs is known as the graph isomorphism problem. More formally, it aims to find a bijective function π where $(u, v) \in G_1 \iff (\pi(u), \pi(v)) \in G_2$. Note that $\pi \in \Pi,$ and $|\Pi| = n!$, where n is the number of nodes in G_1 and G_2. The best known algorithm for this problem is quasi-polynomial time proposed by Babel in 2016 [1]. As a result, finding permutation function π is not known to be NP-Complete and computationally intractable.

Due to the intractability of finding a permutation matrix from input to output graphs, matching procedures must approximate π. One common approach to ensure the GNN model is permutation invariant or equivariant is using aggregator functions that either leave the output graph unchanged or permute it consistently with the input graph [7, 17, 10, 11].

Alternatively, Winter et al. proposed training a dedicated *permuter* alongside the encoder and decoder blocks to reorder the output graph to align with the input [16]. This method outperforms state-of-the-art matching procedures in tasks such as generation, interpolation, and classification. However, it inherits the $O(n^4)$ computational complexity of GraphVAE due to the need to compute attention across n^2 messages. The authors are able to optimize the process to $O(n^3)$ by not comparing all messages. While this approach improves accuracy, it also significantly increases the architectural complexity of the model, particularly for the subtask of computing the reconstruction loss.

2.1 GraphVAE's Graph Matching Procedure

GraphVAE addresses the challenge of permutation variance by introducing an approximate graph matching procedure. This involves a brute-force evaluation of similarity between edges in the input graph G and the reconstructed graph $\widehat{G}$. The procedure serves as the baseline for our optimizations.

Similarity Function. To compute the permutation X from $\widehat{A}$ to A, a similarity function S is defined for edges in G and $\widehat{G}$:

$$S : (i, j) \times (a, b) \to \mathbb{R}^+, \quad i, j \in G,\, a, b \in \widehat{G}.$$

The function is given by:

$$S((i, j), (a, b)) = A_{i,j}A_{a,b}A_{a,a}A_{b,b}[i \neq j \wedge a \neq b] + (F_{i,\cdot}^T F_{a,\cdot})A_{a,a}[i = j \wedge a = b],$$

where A is the adjacency matrix of the graph and F represents node features.

This brute-force approach evaluates all edge pairs in G and $\widehat{G}$, incurring a time complexity of $O(n^4)$ for n nodes in the graphs. This is the calculation we aim to optimize.

S is then passed through the Max-Pool Matching [5] procedure to obtain a continuous permutation matrix, which is then disretized using the Hungarian algorithm [9]. The resulting permutation matrix X is then applied to input adjacency graph: $A' = XAX^T$, and A' is used in loss calculations.

3 Theory/Model/Methods

We propose various optimizations for the original GraphVAE brute force matching approach. Instead of computing similarity on *all* n^4 edge to edge pairings, we leverage algorithms with faster time complexities to infer certain node properties in A and $\widehat{A}$, and only search on edges whose nodes exhibit similar properties.

3.1 Notation

We define graph $G(V, E)$ to denote the input graph, and $\widehat{G}(\widehat{V}, \widehat{E})$ to denote the reconstructed output. V and E are sets of vertices and edges respectively. A and $\widehat{A}$ are adjacency matrices representing G and $\widehat{G}$ respectively. Note that $\widehat{A}$ is a probabilistic graph, while A is discrete. Let $n = |V| = |\widehat{V}|$. Let $d(x)$ denote the degree of node x.

3.2 Binning Method

We propose to only check similarities between nodes $a \in \widehat{A}$ and $i \in A$ if $d(i) \approx d(a)$. We use the idea that if $(a, b) \in \widehat{E}$ and $(i, j) = (\pi(a), \pi(b)) \in E$, then the degree of the nodes is preserved, i.e., $d(a) = d(i)$.

Proof:

Let $\widehat{A}^D$ denote the discretized reconstructed adjacency matrix, defined as: $\widehat{A}^D_{i,j} = 1$ if $\widehat{A}_{i,j} \geq 0.5$, and 0 otherwise. Suppose there exists a bijective function π mapping the nodes of $\widehat{A}^D$ to A such that $(a, b) \in \widehat{E} \mapsto (\pi(a), \pi(b)) = (i, j) \in E$, where $\widehat{E}$ and E are the edge sets of $\widehat{A}^D$ and A, respectively.

Let P be the binary permutation matrix encoding π. Then $\widehat{A}^D = PAP^T$. Let x be an all-ones vector of size n. Then $d(\widehat{A}^D) = \widehat{A}^D x = PAP^T x = PAx = Pd(A)$. So, $d(\widehat{A}^D) = Pd(A)$, which means $d(a) = d(i)$ for $i = \pi(a)$.

Since permutations preserve node degree, our optimization is as follows. For nodes $i, j \in A$, $a, b \in \widehat{A}$, we compute $S(i, j, a, b)$ only if $|d(i) - d(a)| \leq 1$ and $|d(j) - d(b)| \leq 1$. Otherwise, $S(i, j, a, b) = 0$.

3.3 Centrality Based Methods

3.3.1 PageRank Binning Method

We optimize similarity computation using a rank-based binning method. The rank function,

$$\text{rank}(x, \tilde{A}) : V \mapsto \{0, 1, \ldots, n-1\},$$

maps each node x in graph $\tilde{A}$ to a unique rank based on the PageRank algorithm [13], where n is the total number of nodes. PageRank ranks nodes by considering both the quantity (degree) and quality (importance) of their connections.

We compute the similarity $S(i, j, a, b)$ for nodes $i, j \in A$ and $a, b \in \widehat{A}$ only if:

$$|\text{rank}(i, A) - \text{rank}(a, \widehat{A})| \leq b \quad \text{and} \quad |\text{rank}(j, A) - \text{rank}(b, \widehat{A})| \leq b,$$

where b is a hyperparameter (e.g., $b = 2$) representing an error threshold, and $2b + 1$ is the nodes per bin. If these conditions are not met, we set $S(i, j, a, b) = 0$.

This method ensures even distribution of nodes across bins due to the injectivity of the rank function, unlike degree-based methods, which can place up to all nodes in the same bin. The computational complexity is further reduced by limiting similarity evaluations to nodes within the same bin.

3.3.2 Betweenness Binning Method

We simply replace the PageRank algorithm in 3.3.1 with betweenness centrality ranking [6][12][4]. Otherwise, the implementations are identical.

3.4 Community Binning Methods

We bin nodes in A and $\widehat{A}$ using community detection algorithms. Modularity is used as a metric to determine the strength of each community [12].

Using the following properties, we condense our similarity computations to nodes within the same community.

1. If nodes (x, y) are in the same community, they are more likely to share an edge.
2. If nodes $(i, j) \in A$ are assigned community c_1 and $(a, b) \in \widehat{A}^D$ are assigned community c_2, then $i = \pi(a)$ and $j = \pi(b)$ only if $\text{mod}(c_1) = \text{mod}(c_2)$, where mod is the modularity function, and π is a permutation function.

   Since permutations preserve degree (section 3.2), and modularity depends on degree, permutations preserve modularity.

3.4.1 Louvain Binning

Procedure:

1. Find Louvain communities [2] for A and $\widehat{A}$.
2. Compute the modularity of each community.
3. Normalize the modularities of communities in each graph.
4. Compute the similarity $S(i, j, a, b)$ for nodes $i, j \in A$ and $a, b \in \widehat{A}$ only if:
   - (i, j) are in the same community C_1,
   - (a, b) are in the same community C_2, and
   - the distance $|\text{mod}(C_1, A) - \text{mod}(C_2, \widehat{A})|$ is minimized.
   - Otherwise, $S(i, j, a, b) = 0$

3.4.2 Spectral Binning

This procedure is the same as the Louvain Binning method, except we use spectral clustering [15] to find communities in Step 1.

4 Dataset Description

We test on graphs in the enzymes dataset [3] and set the maximum graph size to 38 to match GraphVAE[14].

5 Results

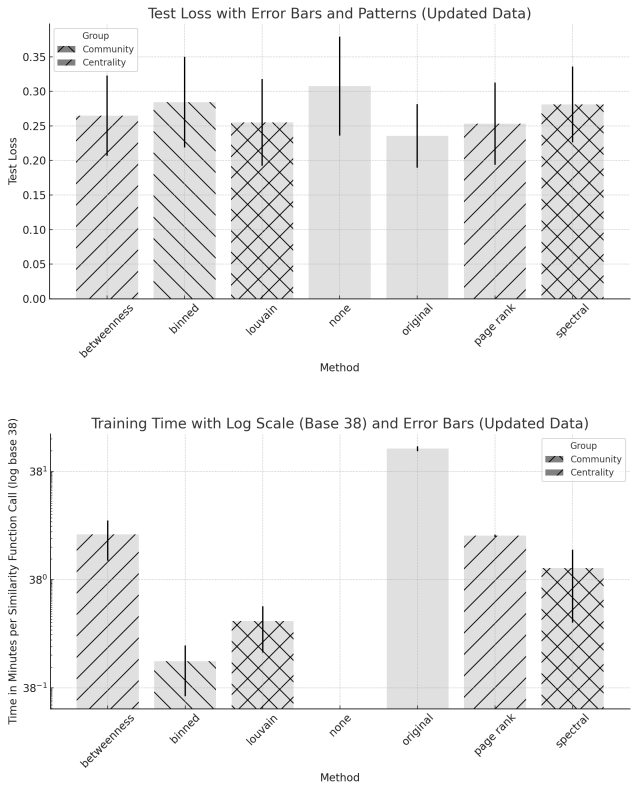

Figure 1: Accuracy (right) and compute time (left) for various graph matching methods

Table 1: Comparison of Test Loss and Training Time for Different Methods

Method	Test Loss	Std Loss	Training Time (min)	Std Training Time (min)
Betweenness	0.2647	0.0582	4.620	2.730
Binned	0.2845	0.0656	0.065	0.045
Louvain	**0.2552**	0.0627	0.249	0.162
None	0.3076	0.0714	0.000	N/A
Original	**0.2356**	0.0460	82.201	6.726
PageRank	**0.2530**	0.0595	4.385	0.184
Spectral	0.2810	0.0549	1.487	1.251

6 Results

Our results indicate that the PageRank and Louvain procedures show promising potential in approximating our similarity matching function with significantly reduced computation time. While neither procedure surpassed the original brute force approach in accuracy, these findings validate the feasibility of using node-property-inspired binning methods to cut computational costs. We suspect the reason for Louvain's fast run time is a large number of communities, which cubically decreases the computation. PageRank provided the highest scoring similarity approximation but ran about $23\times$ slower than Louvain. Louvain, in particular, provided the close second best approximation of the original similarity function, while achieving computational gains exceeding an order of magnitude compared to the brute force method.

Our heuristic time complexity analysis suggests that Louvain's efficiency is maximized when communities are well-distributed among nodes. In such scenarios, the time complexity scales as c^3 in the average case: $c \cdot \left(\frac{n}{c}\right)^4 = \frac{n^4}{c^3}$, where c is the number of communities. On the other hand, for the PageRank algorithm, which ensures well-distributed bins of size determined by hyperparameter k, the time complexity is reduced to $n^2 \cdot k^3$, where k is a relatively small constant compared to n. For instance, in a graph of size $n = 38$ with $k = 5$, the expected speedup is approximately $13\times$, and this scale factor increases exponentially as n grows linearly. However, our results show a speed up of about $19\times$. We attribute this to a higher frequency of cases where nodes $i = j$ and $a = b$, which simplifies our computation by a factor of k.

7 Intrinsic Evaluation of Matching Functions

In addition to evaluating the performance of the graph similarity functions in the context of the original task, we conducted an analysis of the similarity functions in isolation to better understand their behavior and properties. Specifically, we evaluated the matching functions on randomly generated graphs, examining how their similarity scores capture correlations in the following structural graph descriptors: degree distribution, clustering coefficient distribution, density, average path length, and diameter. This evaluation highlights the sensitivities of the matching functions to different topological features of graphs.

7.1 Evaluation Procedure

The evaluation procedure involves the following steps to assess the behavior of graph similarity functions on randomly-generated graphs.

1. **Generate Random Graphs:** Randomly generate two graphs G_1 and G_2 of the following types:
   - Erdős–Rényi random graph
   - Scale-free network using preferential attachment, via Barabási–Albert model
   - Small-world network, via Watts-Strogatz model
   - Random geometric graph (RGG)

   Each graph is initialized with randomly assigned features.

2. **Ensure Graph Connectivity:** Verify that each graph is fully connected. If not, apply a minimum spanning tree to enforce connectivity.
3. **Compute Graph Descriptors and Similarities:** For each graph pair G_1 and G_2:
   (a) Calculate the following graph descriptors for G_1 and G_2:
      - Degree distribution
      - Clustering coefficient distribution
      - Density
      - Average path length (APL)
      - Diameter
   (b) Compute the similarity between the descriptors of G_1 and G_2:
      - For density, APL, and diameter, use the absolute difference:

$$\text{Similarity} = |d_{G_1} - d_{G_2}|$$

      - For degree distribution and clustering coefficient distribution, use the Kullback–Leibler (KL) divergence:

$$\text{Similarity} = D_{\text{KL}}(P_{G_1} \parallel P_{G_2})$$

4. **Apply Similarity Function:** Run the similarity function of interest on G_1 and G_2. Let S denote the similarity matrix produced by the function. Compute the total similarity score as the mean of the entries in S:

$$\text{Total Similarity Score} = \frac{1}{|S|} \sum_{i,j} S_{i,j}$$

Using the above strategy, we generated plots comparing each similarity function class's score to the correlations in the graph descriptors across 500 random graph pairs.

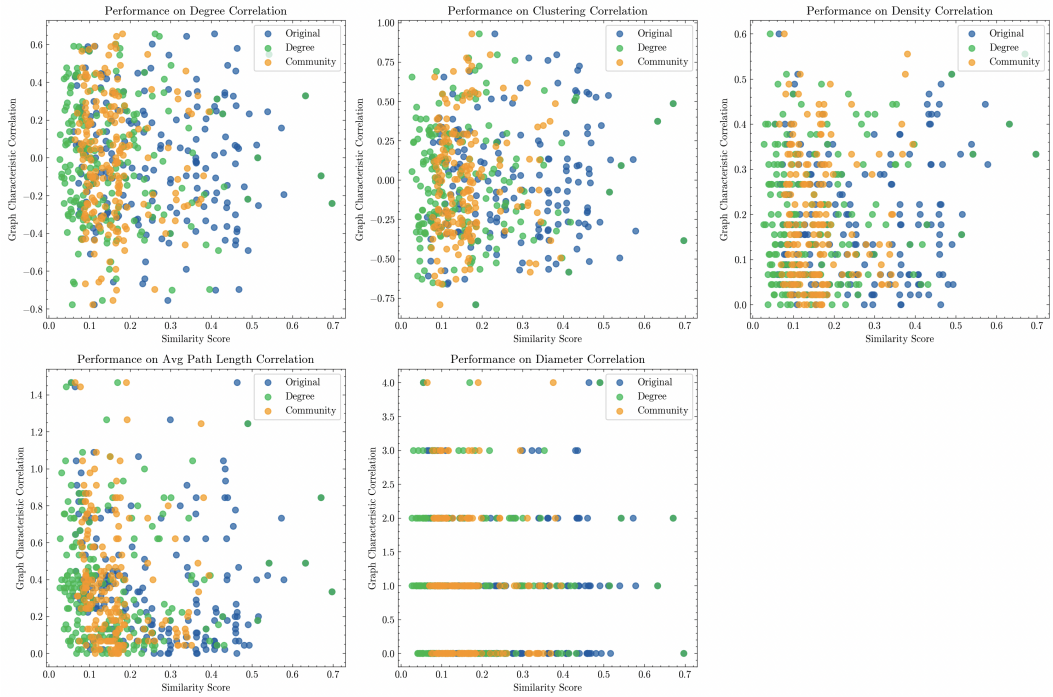

Figure 2: Comparison of correlation in graph characteristics (degree distribution, correlation coefficient distribution, density, APL, diameter) against the similarity scores (method classes: original, degree-based, community-based) on pairs of randomly-generated graphs. Characteristic correlations of smaller magnitude indicate greater levels of similarity in that metric. For example, a degree distribution correlation of 0 indicates that two graphs have identical degree distributions.

According to Figure 2, the degree-based and community-based methods demonstrated a performance comparable to the original similarity function in this evaluation. These functions effectively captured the correlations between APL, diameter and density, as their similarity scores increased consistently with stronger correlations in these descriptors. However, their ability to capture degree distribution correlation and clustering coefficient correlation appeared noticeably weaker, with less pronounced trends in the corresponding similarity scores.

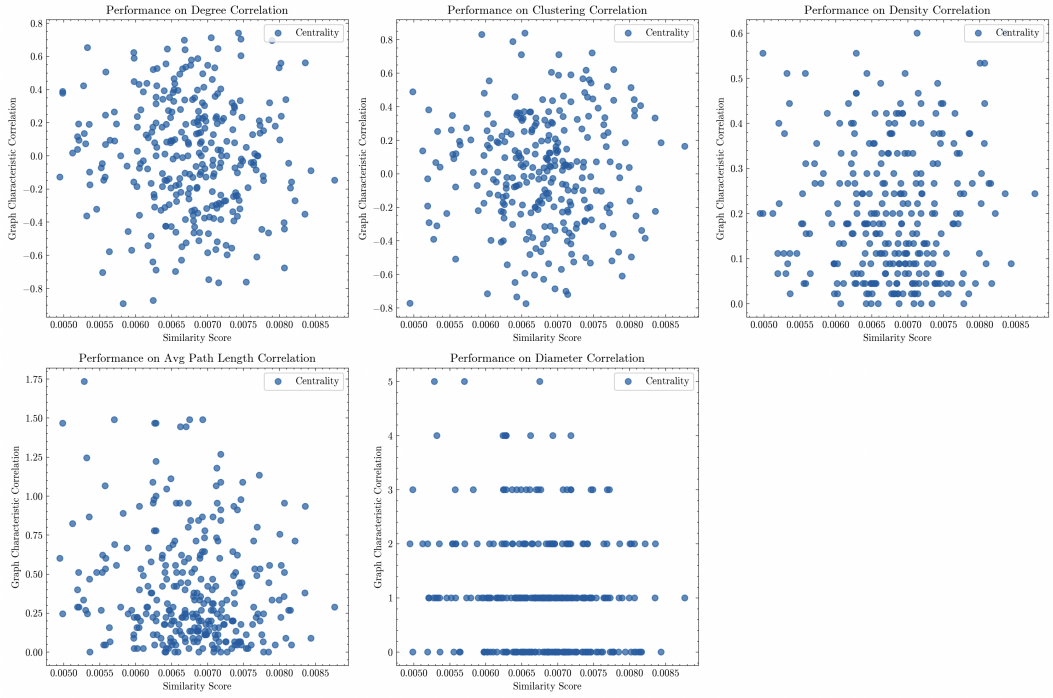

Figure 3: Same comparison as in Figure 2, but for Centrality-based methods

In contrast, Figure 3 shows that the centrality-based similarity function (PageRank, betweenness) struggled to capture any of the graph descriptors tested effectively. Data across all graphs exhibited a relatively uniform distribution and similarity scores remained largely consistent, regardless of the degree of similarity between the graphs in terms of the descriptors tested. This suggests that centrality-based similarity functions are less sensitive to the structural features examined in this evaluation.

The difference in performance of the centrality-based methods can be attributed to their design, which focuses on identifying nodes of importance or influence within a graph. Random graphs typically lack well-defined or pronounced centers of influence, limiting the ability of PageRank and betweenness centrality to capture meaningful structural information in this context. However, this limitation does not hinder their effectiveness in the original task of faster graph matching, as evidenced by strong performance in terms of test loss and train time.

8 Conclusion and Future Directions

The Louvain and PageRank approaches show strong potential as efficient methods for approximating the similarity tensor by selectively comparing edges in the input and reconstructed graphs. Louvain demonstrated the most effective approximation of the original similarity function while significantly reducing computational time. Additionally, the betweenness ranking method performed within 3% of Louvain, highlighting the utility of leveraging node centrality for approximations.

Our preliminary testing involved two centrality-based and two community-based approaches for ranking and clustering nodes. While positive results were observed, no clear preference emerged between the two categories in terms of approximating the solution. However, the experiments were

limited to 10% of the enzyme dataset and capped at five epochs due to computational constraints, as all simulations were conducted on a single processor to maintain experimental control.

We also attempted to understand which graph features each method was particularly suited for capturing. The degree-based and community-based similarity functions showed similar sensitivities, effectively capturing many of the same structural metrics as the original similarity function. In contrast, the centrality-based method exhibited a lack of sensitivity to any of the tested descriptors, indicating that their focus on node importance did not align with the structural features we tested.

Future work could involve a more comprehensive analysis, including evaluating how loss evolves as a function of training time, epochs, and graph size. We hope to further test the underperforming methods, including binning and spectral, to understand if and how their performance changes with different thresholds or longer training times. We also hope to test additional methods from both centrality-based and community-based approaches to understand which node properties are most useful for this task. Additionally, scaling experiments to larger datasets and higher epochs could better elucidate the trade-offs between accuracy and computation. These directions have the potential to deepen our understanding of the scalability and robustness of these methods in practical applications.

Codebase

Project code can be found in the following GitHub repository:
`https://github.com/lpullela/graph_vae`.
We used the [18] benchmark implementation as we could not find the original GraphVAE code.

Contributions

Arul and Laya both ran and implemented the solutions. Laya set up the code base. Arul developed the graph property analysis simulation. For the paper, Arul wrote the introduction and background sections, Laya authored the methods section, and both Arul and Laya contributed to the results and conclusion sections. Both Arul and Laya developed the in-class presentation.

References

[1] László Babai. Graph isomorphism in quasipolynomial time [extended abstract]. In *Proceedings of the 48th Annual ACM Symposium on Theory of Computing (STOC)*, pages 684–697. ACM, 2016.

[2] Vincent D. Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. Fast unfolding of communities in large networks. *Journal of Statistical Mechanics: Theory and Experiment*, 2008(10):P10008, 2008.

[3] Karsten M Borgwardt, Cheng Soon Ong, Stefan Schönauer, S V N Vishwanathan, Alex J Smola, and Hans-Peter Kriegel. Protein function prediction via graph kernels. *Bioinformatics*, 21(suppl_1):i47–i56, 2005.

[4] Ulrik Brandes. A faster algorithm for betweenness centrality. *Journal of Mathematical Sociology*, 25(2):163–177, 2001.

[5] Myungjin Cho, Jinwoo Sun, and Jean Ponce. Finding matches in a haystack: A max-pooling strategy for graph matching in the presence of outliers. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2091–2100, 2014.

[6] Linton C Freeman. A set of measures of centrality based on betweenness. *Sociometry*, 40(1):35–41, 1977.

[7] William L Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems*, pages 1024–1034, 2017.

[8] Zhongyu Huang, Chaozhuo Li, Yingheng Wang, and Huiguang He. Going deeper into permutation-sensitive graph neural networks. *Journal Placeholder*.

[9] H. W. Kuhn. The hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2(1-2):83–97, 1955. The preparation of this report was supported, in part, by the ONR Logistics Project, Department of Mathematics, Princeton University.

[10] Haggai Maron, Heli Ben-Hamu, Nadav Shamir, and Yaron Lipman. Invariant and equivariant graph networks. In *Proceedings of the 7th International Conference on Learning Representations*, 2019.

[11] Christopher Morris, Gaurav Rattan, Matthias Fey, William L. Hamilton, Jan Eric Lenssen, and Martin Grohe. Relational pooling for graph representations. *arXiv preprint arXiv:1911.08613*, 2019.

[12] M. E. J. Newman. Modularity and community structure in networks. *Proceedings of the National Academy of Sciences*, 103(23):8577–8582, 2006. Edited by Brian Skyrms, University of California, Irvine, CA, and approved April 19, 2006.

[13] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The pagerank citation ranking: Bringing order to the web. Technical report, Stanford InfoLab, 1999.

[14] Martin Simonovsky and Nikos Komodakis. Graphvae: Towards generation of small graphs using variational autoencoders. In *Proceedings of the 27th International Conference on Artificial Neural Networks (ICANN)*, pages 412–422. Springer, 2018.

[15] Ulrike von Luxburg. A tutorial on spectral clustering. *Statistics and Computing*, 17(4):395–416, 2007. The original publication is available at www.springer.com.

[16] Robin Winter, Frank Noé, and Djork-Arné Clevert. Permutation-invariant variational autoencoder for graph-level representation learning. *35th Conference on Neural Information Processing Systems*.

[17] Rex Ying, Jiaxuan You, Christopher Morris, Xiang Ren, William L Hamilton, and Jure Leskovec. Hierarchical graph representation learning with differentiable pooling. In *Advances in Neural Information Processing Systems*, pages 4800–4810, 2018.

[18] Jiaxuan You, Rex Ying, Xiang Ren, William L Hamilton, and Jure Leskovec. Graphrnn: Generating realistic graphs with deep auto-regressive models. In *International Conference on Machine Learning*, pages 5708–5717. PMLR, 2018.