Algorithmic Complexity Underpins When Path Information Improves Graph Neural Networks Performance on Molecular Graphs

Abdulrahman Ibraheem[1(✉)], Narsis Kiani[3,4,5], and Jesper Tegner[1,2,5,6,7]

[1] Living Systems Lab, BESE and CEMSE, KAUST, Thuwal, Saudi Arabia
rahmanoladi@yahoo.com, jesper.tegner@kaust.edu.sa
[2] SDAIA-KAUST Center of Excellence in Data Science and AI, Thuwal, Saudi Arabia
[3] Algorithmic Dynamics Lab, Karolinska Institutet, Solna, Sweden
narsis.kiani@ki.se
[4] Department of Oncology-Pathology, Karolinska Institutet, Solna, Sweden
[5] Center for Molecular Medicine, Karolinska Institutet, Solna, Sweden
[6] Karolinska University Hospital, Stockholm, Sweden
[7] Science for Life Laboratory, Solna, Sweden

Abstract. Graph Neural Networks (GNNs) are designed to process irregular relational data in recommendation systems, protein networks, social networks, and molecules. GNNs typically rely on message passing and aggregation, with some architectures incorporating graph path information in a bid to improve accuracy. However, it is unclear whether such incorporation of path information truly improves GNN accuracy in all cases. As a first step, we herein shed light on this issue for the case of molecular graphs. We evaluated Graphormer and Mix-Hop models, with and without path information on 36 molecular datasets, derived from six MoleculeNet benchmark datasets. Path information improved performance in some cases but not in other cases. This finding is important, because these two models always incorporate path information in practice, whereas the finding shows such incorporation of path information might actually be detrimental to the models' accuracies in some cases. To more deeply probe this observation, we developed a graph representation model called T-hop which allows us to further highlight the use, versus non-use, of path information. On one hand, we formulate the Path Usefulness Measure (PUM) to quantify the benefit of path information. On the other hand, we quantified the randomness of the different datasets via their algorithmic complexities, using the Block Decomposition Method (BDM). We hypothesized, and confirmed our hypothesis, that: GNN models trained on molecular datasets with less random structures (i.e. lower algorithmic complexity) should benefit more from path information (i.e. larger PUM), compared to datasets with more random structures. Put in other words, we found an inverse relationship between algorithmic complexity and the PUMs.

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2027
O. Gervasi et al. (Eds.): ICCSA 2026, LNCS 16767, pp. 18–34, 2027.
https://doi.org/10.1007/978-3-032-30491-9_2

Keywords: GNN · Algorithmic Complexity · Molecular Graphs

1 Introduction

Currently, a prominent class of models for processing graphs comprises graph neural networks (GNN) [1–4]. While most GNNs follow a message passing paradigm [13,14], some GNNs, such as graphormer [5] and Mix-Hop [7], incorporate path information, in a bid to further improve accuracy. In these models, the underlying assumption is that incorporation of graph path information should always improve models' accuracies. However, as we shall see in this work, this assumption can be wrong, because our results show that path information improves accuracies in some cases, while lowering accuracies in other cases.

In this work, we take the first steps towards characterizing conditions under which path information can improve GNN accuracies. To this end, we first examined two established GNN models: Graphormer and Mix-Hop. For each model, we designed two modes: a first mode that jettisons path information altogether and a second mode that utilizes path information. We then compared the two modes on six MoleculeNet graph datasets [11]. We found that path information did not consistently improve performance in these models. To study this phenomenon more closely, we designed a novel model, which we call T-Hop, which more aggressively amplifies the dichotomy between the use of path information on the one hand and its non-use on the other hand. Again, we found that using path information with the T-Hop model also did not improve accuracy in all cases. This prompted us to search for a deeper understanding of the reason behind these counter-intuitive results. We conjectured that the extent to which a graph dataset benefits from path information in GNN models might be inversely correlated with the level of randomness in the dataset's graphs' path connectivity patterns. To test this conjecture, we created six dataset families. Each data set family comprises one of six original MoleculeNet datasets along with five synthetic variants of the original data set. On one hand, we quantified the randomness in the dataset families' graph structures via algorithmic complexity [8,9], using the BDM [8,10]. On the other, we quantified the benefit of path information in the GNN models via the PUM score, which we introduce later in this work. We found that GNN models trained on dataset families with lower algorithmic complexities benefited more from path information and hence have higher PUMs. We underscored this observation by showcasing negative Pearson correlations between algorithmic complexities and the PUMs, thereby corroborating our conjecture. This was further consolidated by our clustering analysis, which showed good matches between the dataset clusters created using the PUMs and those created using the algorithmic complexities.

2 Related Work

To the best of our knowledge, there is no existing work in the literature that characterizes conditions under which path information is expected to improve

GNN accuracies, albeit a plethora of GNN models that incorporate path information in the same literature [5,7,16,17]. For example, the popular graphormer model [5] incorporates path information by utilizing information along the shortest path between every pair of nodes in the input graph and augmenting its attention matrices with that information. Another GNN model that incorporates path information is the Mix-Hop model [7]. It does so by utilizing a set of powered adjacency matrices whose entries encode path information. Yet another model that uses path information, although to a lesser extent than Mix-Hop, is the IGCN (Improved Graph Convolutional Network) model [16]. IGCN uses a powered symmetrically-normalised identity-shifted adjacency matrix, which can ultimately be expressed in terms of the powered adjacency matrix. Furthermore, similarly to Mix-Hop, Power-Up [17] is also a GNN model that utilizes path information. It does so by using a set of matrices that encode information about the path length of the shortest distance between all pairs of nodes in the graph in question. In addition, to incorporate path information, a line of work [18–20], explicitly considers the sequence of paths leading to the nodes of a graph. Other works incorporating path information include NBFNet (Neural Bellman-Ford Networks) [21] and methods that can be considered as its special cases [23–25].

Some earlier works already indicate that adding path information to GNNs does not always improve accuracy [16,20,27]. For example, for PathNN [20], while increasing path length improved results on the NCI1 data set of the TUDataset collection [26], it failed to improve results using the PROTEINS and ENZYMES data sets of the same collection. Likewise, for IGCN [16], experiments on the AWA2 dataset [28] showed worse performance as k increased from 1 to 3, where k can be viewed as a parameter that controls the maximum path length used in the model. Moreover, even for the more recent GraphMixer [27], it was found that using a larger receptive field, and hence path length, diminished accuracies. However, these earlier works neither asked nor answered the question of under which conditions graph path information can be expected to increase GNN accuracies. Hence, this work.

3 Methods

In this section, we outline our methodology vis-à-vis descriptions of: utilized datasets, algorithmic complexity and employed GNN models.

3.1 Datasets

We used a total of thirty six molecular datasets – six original, non-synthetic MoleculeNet datasets [11] along with thirty additional synthetic datasets derived from the original six. Each MoleculeNet dataset comprises a list of molecules, represented via SMILES (Simplified Molecular Input Line Entry System) [31], along with corresponding properties of interest, such as solubility, blood-brain permeability, toxicity, etc. Table 1 summarizes the six original datasets employed.

Table 1. Description of datasets

DATASET	TOTAL NO.OF GRAPHS	PROPERTY	TASK TYPE	METRIC
FreeSolv	643	hydration free energy	regression	RMSE
ESOL	1128	solubility in water	regression	RMSE
Lipophilicity	4200	octanol/water distribution coeff.	regression	RMSE
BACE	1522	β-secretase binding	classification	ROC-AUC
BBBP	2053	blood-brain barrier crossing	classification	ROC-AUC
ClinTox	1491	clinical trial toxicity and FDA approval	classification	ROC-AUC

For processing the graphs in our datasets, we harnessed Deep Graph Library (DGL) [30] and DGL-LifeSci [29]. DGL-LifeSci helped convert MoleculeNet's SMILES (Simplified Molecular Input Line Entry System) [31] representations into DGL graph representations, which are based on a more intuitive list of edges graph representation (LOEGR). In the LOEGR, the graph's nodes correspond to the atoms of the underlying molecule, while the edges of the graph correspond to the bonds between the atoms. This conversion is illustrated in Fig. 1 for the case of methanoic acid. As illustrated by the top-most matrix in the figure, DGL's LOEGR of a graph is implemented as a Pytorch 2-d matrix, wherein each column of the 2-d matrix represents an edge of the graph. Furthermore, the figure also shows that the duo of DGL-LifeSci and DGL makes it possible to easily compute node and edge features for the graph. These are respectively represented by the middle and bottom matrices in the figure. The fact that DGL's graph representations are implemented as Pytorch matrices makes them especially suitable as inputs to the Pytorch implementations of our models.

For our experiments, we required a collection of datasets that share the same path connectivity information, but with different node and edge features. To this end, we defined *dataset families*. A dataset family comprises an original non-synthetic graph dataset, along with a collection of synthetic graph datasets derived from the original dataset. This derivation is done via the addition of pre-defined levels of Gaussian noise to the node and edge features of the graphs in the original dataset, without mutating the path connectivity patterns within those graphs. Note that this means that **datasets within the same dataset family share the same set of path connectivity patterns**. To create the dataset families, we started with the six original non-synthetic MoleculeNet datasets mentioned in the preceding paragraph. We then introduced a noise-level parameter, $\gamma \in \{0.0, 0.1, 0.2, 0.3, 0.4, 0.5\}$. Further, we let σ_0 denote the standard deviation of an original non-synthetic dataset's features. Then for each original dataset, we computed the set, $S_{\sigma_0} = \{0.0\sigma_0, 0.1\sigma_0, 0.2\sigma_0, 0.3\sigma_0, 0.4\sigma_0, 0.5\sigma_0\}$. To each original dataset, we then added zero-mean Gaussian noises with standard deviations $\sigma \in S_{\sigma_0}$ to the dataset's features. This yielded five synthetic variants of each original dataset. Here, note that $\sigma = 0.0\sigma_0$ is the noise-less case, which actually corresponds to the original non-synthetic dataset. In all, the above procedure allowed us to obtain a total of thirty six datasets: thirty synthetic datasets along with the six original non-synthetic datasets. We used scaffold splitting [11]

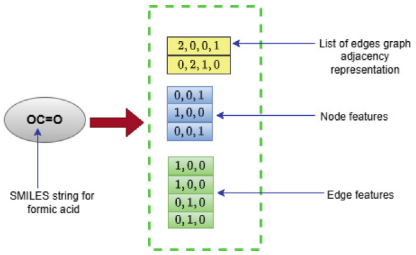


Fig. 1. DGL/DGL-LifeSci conversion of methanoic acid's SMILES representation to a graph representation comprising: 1). a list of edges representation using a Pytorch 2-d matrix with each column representing an edge (top matrix in the figure); 2) a Pytorch 2-d matrix containing the graph's nodes' features (middle matrix in the figure); 3) a Pytorch 2-d matrix containing the graph's edges' features (bottom matrix in figure).

to split the datasets into training:validation:test segments in the ratio 80:10:10. Finally, we point out that each of the 30 synthetic datasets inherit properties listed in Table 1 from its parent non-synthetic dataset.

3.2 Algorithmic Complexity and BDM

We required a way to measure the randomness (i.e. lack of regularity) of the connectivity patterns within the graphs in our datasets. We used algorithmic complexity [8,9] as a measure of this randomness, and employed the BDM [8,10] for approximating it. We favored algorithmic complexity over traditional measures of randomness, such as Shanon entropy, because the former has been demonstrated to be better at capturing the intrinsic information content of a graph's structure, beyond sheer statistical patterns [8]. In BDM [8,10], the adjacency matrix, A, of the input graph, G, is first partitioned into a set of non-overlapping $d \times d$ sub-arrays. The j-th sub-array is denoted r_j. Then, the Coding Theorem Method (CTM) [8] is applied, to compute the algorithmic complexity, $K_{CTM}(r_j)$, of r_j. This involves computing the algorithmic probability of r_j. To compute the algorithmic probability, a large collection of 2-d Turing machines (i.e. turmites) is run. The frequency with which r_j is output by the collection is then computed, and used to approximate $K_{CTM}(r_j)$. In symbols, the BDM algorithmic complexity of G, denoted $K_{BDM}(G,d)$, is expressed in terms of $K_{CTM}(r_j)$ as:

$$K_{BDM}(G,d) = \sum_{(r_j,n_j)\in A} K_{CTM}(r_j) + \log_2(n_j) \tag{1}$$

Above, n_j is the number of times r_j occurs within A. The notation $\sum_{(r_j,n_j)\in A}$, as opposed to $\sum_{r_j\in A}$, serves to emphasize that we count $K_{CTM}(r_j)$ separately for each of the n_j occurrences of r_j within A. To eschew re-inventing the wheel, we leveraged the publicly available implementation of BDM at this URL[1]

[1] https://pybdm-docs.readthedocs.io/en/latest/pybdm.html.

3.3 Models

This section describes the three models employed in this study, namely Graphormer [5], Mix-Hop [7] and T-Hop. The T-Hop model is a tensor-based graph representation model which we introduce in this work. We introduce it as a more effective model for highlighting the dichotomy between the use of path information versus the non-use of path information in GNNs.

Synopsis of the Graphormer Model. Here, we describe how to use the graphormer model for the goals of this work. For a full account of the graphormer model, the interested reader is referred to the graphormer paper [5]. Essentially, Graphormer can be viewed as an analog of the popular transformer model [6] applied to graph datasets. At each layer of graphormer, the transformer-style softmax attention mechanism is applied to every pair of nodes in the input graph; followed by a skip-connection and the LayerNorm; followed by a feedforward MLP; followed again by a skip-connection and LayerNorm. In addition, the graphormer model super-imposes the structure of the underlying graph into the model by augmenting the softmax attention matrix with path information. The path information here is derived by considering the distance along the shortest path between every pair of nodes, as well as by considering the flow of edge features along those shortest paths. Consequently, for our experiments in this paper, we implemented two modes of the graphormer model: a first mode, wherein the softmax attention matrix is not augmented with any path information; and a second mode wherein the softmax attention matrix is augmented with path information as above described. Our Pytorch implementation is at this URL[2].

Synopsis of the Mix-Hop Model. The Mix-Hop model [7] harnesses powered adjacency matrices, $A, A^2, \ldots, A^{L_m}$, with L_m being the maximum path length considered in the model. Specifically, where $\|_{L=1}^{L_m}$ denotes the vector concatenation operation; where H^l denotes input feature matrix to the l-th layer; where W_L^l denotes a learnable matrix; and where σ denotes an activation function, then each layer of Mix-Hop is underpinned by the following formula:

$$H^{l+1} = \Big\|_{L=1}^{L_m} \sigma(A^L H^l W_L^l) \tag{2}$$

Since L_m denotes maximum path length, it follows that we can implement a mode that does not incorporate path information into the model by setting $L_m = 1$. Also, we can implement a mode that incorporates path information into the model by setting $L_m > 1$. Our Pytorch implementations are at this URL[3]

[2] https://github.com/rahmanoladi/kaust_path_project/tree/main/path_project/graphormer.

[3] https://github.com/rahmanoladi/kaust_path_project/tree/main/path_project/mix_hop.

The T-Hop Model: A Novel Tensor-Based Approach. This section introduces T-Hop, a model that sharply contrasts performance with and without path information. Let $G = (V, E)$ represent a graph, where $V = \{v_1, ..., v_n\}$ and $E = \{e_1, ..., e_m\}$ are the nodes and edges of G, as usual. The adjacency matrix of G is A. We consider two arbitrary nodes, v_i and v_j in G. Let $L \geq 2$, and let $\mathcal{B}^L_{i,j,k}$ be the number of paths of length L between v_i and v_j that contain v_k. We may arrange the values, $\mathcal{B}^L_{i,j,k}$, in an $n \times n \times n$ 3-d tensor denoted $\mathcal{B}^L \in \mathbb{R}^{n \times n \times n}$, such that the entry on the i-th row, j-th column and k-th depth of $\mathcal{B}^L$ is $\mathcal{B}^L_{i,j,k}$. Note that the tensor, $\mathcal{B}^L$, encapsulates path information contained in the input graph, G. Further, we scale $\mathcal{B}^L$ by $L + 1$, and denote the result as a new tensor, $\mathcal{T}^L \in \mathbb{R}^{n \times n \times n}$:

$$\mathcal{T}^L_{i,j,k} = \frac{\mathcal{B}^L_{i,j,k}}{(L+1)} \tag{3}$$

Note that, like the tensor, $\mathcal{B}^L$, the tensor $\mathcal{T}^L$ also encapsulates path information contained in the input graph, G. To proceed, we define a new $n \times n$ matrix $\mathcal{M}$ as follows:

$$\mathcal{M} = \alpha_0 A + \sum_{L=2}^{L_m} \sum_{k=0}^{n-1} \alpha_{L,k} \mathcal{T}^L_{:,:,k} \tag{4}$$

Above, A is adjacency matrix, while α_0 and $\alpha_{L,k}$ are learnable scalars. Further, L_m is the maximum path length considered in the model. Also, $\mathcal{T}^L_{:,:,k} \in \mathbb{R}^{n \times n}$ is the 2-d matrix slice associated with depth-k of $\mathcal{T}^L$. Using $\mathcal{M}$, the l-th layer of our T-Hop model can now be described as:

$$H^{l+1} = \sigma(\mathcal{M}\, H^l W^l) \tag{5}$$

Above, H^l denotes input features to the l-th layer, W^l denotes a learnable matrix of weights, and $\sigma(.)$ is a non-linear activation function. Going back to Eq. 4, we can implement a mode of the model that does not use path information by setting $\alpha_{L,k} = 0$ for all $L = 2, \ldots, L_m$ and all $k = 0, \ldots, n-1$. Under this mode, Eq. 4 distills to $\mathcal{M} = \alpha_0 A$, which shows that the model is left only with information contained in the adjacency matrix, A, to the exclusion of all the path information contained in the tensor, $\mathcal{T}^L$. On the other hand, to implement a mode that uses path information, we simply use Eq. 4 as it is. We juxtapose the two modes pictorially in Figs. 2a and 2b, for the simple case of $L_m = 2$. In Fig. 2a, we see that the diagram includes a cube representing $\mathcal{T}^L$, which encapsulates path information; whereas in Fig. 2b, the cube is not included. A Pytorch implementation of the models is at this URL.[4]

3.4 Path Usefulness Measure: A Quantifier of the Usefulness of Incorporating Path Information in Our Models

This subsection introduces the PUM, a metric designed to *quantify the extent to which incorporating path information into a given GNN model enhances its*

[4] https://github.com/rahmanoladi/kaust_path_project/tree/main/path_project/t_hop.

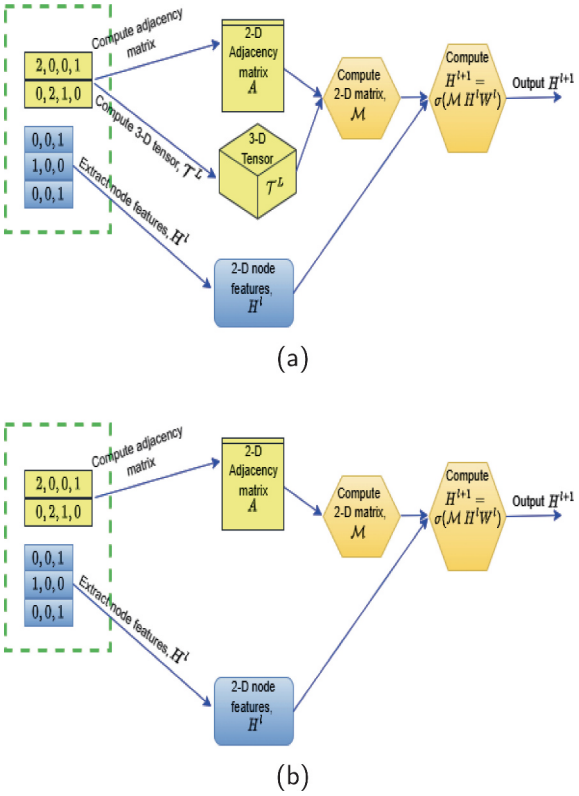


Fig. 2. Schematic depiction of the two modes of T-Hop model for the case of $L_m = 2$: (a) using path information contained in tensor $\mathcal{T}^L$ as represented by the cube in the diagram (b) *without* using path information contained in tensor $\mathcal{T}^L$ (so no cube is included in this case).

accuracy on a specific dataset family. Consider a set of K GNN models, $\mathcal{M} = M_1, M_2, \ldots, M_K$, and a collection of N dataset families, $\mathcal{D} = D_1, D_2, \ldots, D_N$. Each dataset family D_j includes an original dataset along with several synthetic variants generated by adding noise. For each variant $D_j^\gamma \in D_j$, associated with a noise level γ, we evaluate whether the inclusion of path information improves the model's accuracy. Define an indicator function $\mathcal{F}(M_i, D_j^\gamma)$ such that:

$$\mathcal{F}(M_i, D_j^\gamma) = \begin{cases} 1, & \text{if incorporating path information improves accuracy} \\ 0, & \text{otherwise} \end{cases}$$

PUM for a given model M_i and dataset family D_j is then:

$$U_{ij}(M_i, D_j) = \frac{\sum_{D_j^\gamma \in D_j} \mathcal{F}(M_i, D_j^\gamma)}{|D_j|} \tag{6}$$

where $|D_j|$ is the total number of variants in the family. This score represents the fraction of variants within the dataset family for which path information led to improved accuracy. In essence, PUM estimates the probability that incorporating path information enhances model performance for a particular model-dataset pair, providing a normalized measure to compare across datasets and models.

4 Experiments

Having established the theoretical foundations, we now experimentally explore how the use of graph path information impacts the accuracies of GNNs. Our results show that path information does not consistently boost GNN accuracies. Moreover, our results show an inverse correlation between the benefit of using path information and the level of randomness/irregularities in the graphs.

4.1 Incorporating Path Information Does Not Yield Consistent Gains in Accuracies on the Original Six Datasets

In this subsection, we report our experimental results assessing how useful including graph path information is for enhancing GNN performance on the six non-synthetic molecular datasets introduced in Sect. 3.1. We consider three model types discussed in Sect. 3 and two modes per model—one without path information and one with—resulting in a total of $6 \times 3 \times 2 = 36$ experimental configurations. For each setup, we used the Optuna library [32] to perform hyper-parameter sweeps on the validation subset to identify optimal parameters. For each of the three model types, details of the Optuna hyper-parameters associated with the model can be found under the directory associated with the model on this project's github[5]. Using these hyper-parameters, we trained each model for 200 epochs on the corresponding training set, then evaluated performance on the test set, recording the relevant metric: RMSE for regression tasks and ROC-AUC for classification tasks. To ensure robustness, each experiment was run three times, and we report the mean and standard deviation of the results. These are shown in Table 2, with the standard deviations placed in parentheses beneath the associated means. Analyzing these results across all configurations, it becomes clear that adding path information does not yield consistent improvements. The additional computational cost therefore is not always justified. Also, this shows that a section of the users of the Graphormer and Mix-Hop models might be able to improve the accuracies of their models simply by switching off the use of path information in those models.

4.2 Measuring PUM on the Models and Dataset Families

Experiments from the previous sub-section showed that incorporating path information does not consistently lead to gains in performance on the three models

[5] https://github.com/rahmanoladi/kaust_path_project.

Table 2. Comparison of Models with and without path information across all six original datasets. Path information does not help in all Cases.

DATASET	RMSE (Lower is better)						ROC-AUC (Higher is better)					
	FREESOLV		ESOL		LIPOPHILICITY		BACE		BBBP		CLINTOX	
	WITHOUT PATH	WITH PATH	WITHOUT PATH	WITH PATH	WITHOUT PATH	WITH PATH	WITHOUT PATH	WITH PATH	WITHOUT PATH	WITH PATH	WITHOUT PATH	WITH PATH
Graphormer												
	3.55 (0.4539)	**2.91** (0.1556)	1.20 (0.0762)	**1.01** (0.0018)	**0.91** (0.0147)	0.99 (0.0357)	**0.782** (0.0002)	0.735 (0.0283)	0.686 (0.0172)	**0.709** (0.0089)	0.602 (0.0128)	**0.779** (0.0065)
Mix-Hop												
	2.45 (0.0714)	2.61 (0.1790)	1.01 (0.0138)	**0.97** (0.0373)	0.96 (0.0024)	**0.95** (0.0105)	**0.799** (0.0047)	0.798 (0.0018)	**0.707** (0.0201)	0.678 (0.0088)	**0.596** (0.0098)	0.591 (0.0228)
T-Hop												
	2.87 (0.3241)	**2.73** (0.0385)	**1.00** (0.0330)	1.00 (0.0367)	**0.78** (0.0210)	0.88 (0.0210)	**0.852** (0.0087)	0.746 (0.0346)	**0.649** (0.0032)	0.623 (0.0363)	**0.785** (0.0079)	0.752 (0.1003)

and on the six original molecular datasets considered. Yet, the experiments did not provide any quantification of the extent to which path information helped. Recall from Sect. 3.4 that the PUM gives us a criterion for quantifying the extent to which path information helps on a given dataset family, with respect to a given model. Consequently, in this section, we will describe experiments performed to obtain PUMs for the various combinations of (model, dataset family) pairs relevant to this work. Specifically, for our experiments here, we used all the thirty six synthetic and non-synthetic molecular datasets within all the six dataset families listed in the earlier sub-section on datasets (i.e. Section 3.1). However, since we already dealt with the case of the six original non-synthetic datasets in the preceding sub-section, then for those cases, we simply copied the relevant results from the preceding sub-section. The remaining thirty synthetic datasets spawned a total of $30 \times 3 \times 2 = 180$ experimental cases, because we trained three different model types (i.e. Graphormer, Mix-Hop and T-hop) at two different modes (i.e. without path information and with path information) on each of the thirty synthetic datasets. Ideally, we should perform a separate hyper-parameter sweep for each of the 180 cases, but that would be too computationally expensive. Hence, we reused the hyper-parameters from the original datasets for the synthetic variants to reduce computation. Specifically, suppose e_n represents an experimental case associated with a non-synthetic dataset, and e_s represents an experimental case associated with a synthetic dataset. We applied the hyper-parameters associated with e_n to e_s as long as e_s and e_n share the same model type, dataset family and mode (mode here means whether the experimental case involves the use of path information or not). Based on this, for each of the 180 synthetic-dataset experimental cases, we trained the pertinent model associated with the case for 200 epochs on the training segment of the associated dataset, and used the trained model to obtain scores of a relevant accuracy metric (R.M.S.E for regression datasets and ROC-AUC for classification datasets) on the test segment of the dataset. To address statistical fluctuations, we actually ran each experimental case thrice, and computed the mean and standard deviation of the accuracies of the three runs. We present the means and standard deviations in Table 3, where the standard deviations are shown in parentheses

beneath the associated means; for ease of reference, the table also copies results for the cases associated with the non-synthetic datasets from Table 2. In addition to the R.M.S.E and ROC-AUC for each experimental case, the table also displays the PUMs associated with each dataset family and model. Because there are six datasets within each dataset family, the PUMs lie in the range $[\frac{0}{6}, \frac{6}{6}]$. A PUM of $\frac{6}{6}$ indicates complete certainty that the given model always yields better results with path information on the given dataset family, while a PUM of $\frac{0}{6}$ indicates complete certainty that the given model yields better results without path information on the given dataset family. Now, we see from the table that, most of the times, the PUMs are neither equal to $\frac{6}{6}$ nor equal to $\frac{0}{6}$, which shows that, even within a single dataset family, it is difficult to predict whether a given model will benefit from the incorporation of path information. Nonetheless, a closer look at the PUMs for each model type shows that the values associated with the T-Hop model are either very close to $\frac{6}{6}$ (i.e. $\frac{5}{6}$ on FreeSolv dataset family) or very close to $\frac{0}{6}$ (e.g. $\frac{1}{6}$ on ESOL family and $\frac{0}{6}$ on Lipophilicity family). This shows that, unlike the other models, T-Hop allows us to more easily distinguish between when path information is helpful versus when it is not helpful. In other words, T-Hop is more effective at highlighting the dichotomy between dataset families that benefit from path information versus those that do not. To quantify this observation, we may compute a *dichotomy score*, Φ_i, for the i-th model according to $\Phi_i = \frac{1}{6}\sum_{D_j \in \mathcal{D}} \max\{U_{ij}, 1 - U_{ij}\}$. Here, U_{ij} is PUM of applying model M_i on dataset family D_j, and $\mathcal{D}$ denotes our collection of six dataset families. Using this confidence score, we obtained scores of $\frac{29}{36}$, $\frac{24}{36}$ and $\frac{33}{36}$ for the Graphormer, Mix-Hop and T-Hop models respectively. We see that T-Hop very closely approaches a perfect score of 1.

4.3 PUM Shows Inverse Correlation with Algorithmic Complexity

A centerpiece of the results from the preceding sub-sections is that it is difficult to predict with certainty when path information can be expected to help for a given model and molecular dataset family. Nonetheless, having a means of making such a prediction would be truly beneficial for deploying GNNs in a more resource-efficient manner. Towards this, we hypothesized that the less random the connectivity pattern of a molecular graph is, the more the path information contained in that graph can be expected to boost the accuracies of GNN models. Since algorithmic complexity is a measure of randomness in graphs [9], this hypothesis implies that we expect an inverse correlation between the algorithmic complexity of our dataset families, and the PUMs associated with the families. In this sub-section, we will empirically validate this hypothesis from two perspectives. The first perspective will employ Pearson correlations between PUM and algorithmic complexity, while the second will use clustering analysis. From our work in the previous sub-section, we already have all the required PUMs. Thus, we now turn to how to obtain the algorithmic complexities for our six dataset families. To obtain them, we used BDM [8,10]. Here, we point out that all members of the same dataset family share similar values of algorithmic complexity. This is because algorithmic complexity only examines the connectivity patterns

Table 3. PUMs for all model types and dataset families. It is difficult to use the PUMs to predict whether or not path information will boost accuracies.

MODEL	RMSE (Lower is better)						ROC-AUC (Higher is better)					
	FREESOLV		ESOL		LIPOPHILICITY		BACE		BBBP		CLINTOX	
	WITHOUT PATH	WITH PATH	WITHOUT PATH	WITH PATH	WITHOUT PATH	WITH PATH	WITHOUT PATH	WITH PATH	WITHOUT PATH	WITH PATH	WITHOUT PATH	WITH PATH
Graphormer												
$\gamma = 0$	3.55 (0.4539)	**2.91** (0.1556)	1.20 (0.0762)	**1.01** (0.0018)	**0.91** (0.0147)	0.99 (0.0357)	**0.782** (0.0002)	0.735 (0.0283)	0.686 (0.0172)	**0.709** (0.0089)	0.602 (0.0128)	**0.779** (0.0065)
$\gamma = 0.1$	4.219 (0.3890)	**3.54** (0.0853)	1.23 (0.1130)	**1.19** (0.1868)	**1.02** (0.0474)	1.05 (0.0132)	**0.779** (0.0033)	0.761 (0.0340)	**0.693** (0.0160)	0.661 (0.0321)	**0.594** (0.0094)	0.582 (0.0064)
$\gamma = 0.2$	4.16 (0.3905)	**3.42** (0.0912)	1.31 (0.0743)	**1.16** (0.0750)	**1.03** (0.0559)	1.04 (0.0203)	**0.781** (0.0028)	0.767 (0.0283)	**0.687** (0.0122)	0.653 (0.0264)	**0.594** (0.0100)	0.575 (0.0080)
$\gamma = 0.3$	4.24 (0.3831)	**3.44** (0.0721)	1.36 (0.0956)	**1.22** (0.0803)	**1.02** (0.0262)	1.06 (0.0298)	**0.780** (0.0013)	0.768 (0.0267)	**0.688** (0.0148)	0.663 (0.0341)	**0.580** (0.0141)	0.578 (0.0046)
$\gamma = 0.4$	4.02 (0.4484)	**3.47** (0.0434)	**1.34** (0.1136)	1.51 (0.2614)	**1.03** (0.0312)	1.06 (0.0165)	0.776 (0.0047)	**0.783** (0.0089)	**0.684** (0.0157)	0.636 (0.0245)	0.579 (0.0240)	**0.580** (0.0053)
$\gamma = 0.5$	4.20 (0.7253)	**3.49** (0.0349)	**1.41** (0.2020)	1.62 (0.1524)	**1.00** (0.0391)	1.07 (0.0122)	0.777 (0.0037)	**0.781** (0.0059)	**0.670** (0.0122)	0.641 (0.0206)	**0.581** (0.0176)	0.577 (0.0052)
PUM		PUM= $\frac{6}{6}$		PUM= $\frac{4}{6}$		PUM= $\frac{0}{6}$		PUM= $\frac{2}{6}$		PUM= $\frac{1}{6}$		PUM= $\frac{2}{6}$
Mix-Hop												
$\gamma = 0$	**2.45** (0.0714)	2.61 (0.1790)	1.01 (0.0138)	**0.97** (0.0373)	0.96 (0.0024)	**0.95** (0.0105)	**0.799** (0.0047)	0.798 (0.0018)	**0.707** (0.0201)	0.678 (0.0088)	**0.596** (0.0098)	0.591 (0.0228)
$\gamma = 0.1\%$	**2.43** (0.0838)	2.82 (0.2120)	**1.19** (0.1675)	1.27 (0.1040)	1.05 (0.0052)	**1.02** (0.0117)	0.737 (0.0195)	**0.780** (0.0066)	**0.671** (0.0094)	0.656 (0.0215)	**0.581** (0.0690)	0.559 (0.0440)
$\gamma = 0.2\%$	2.87 (0.3401)	**2.73** (0.2040)	1.28 (0.1640)	**1.23** (0.0140)	1.16 (0.0377)	**1.11** (0.0416)	0.671 (0.0833)	**0.691** (0.0734)	**0.657** (0.0274)	0.629 (0.1393)	**0.589** (0.0148)	0.539 (0.0329)
$\gamma = 0.3\%$	**3.16** (0.2525)	3.23 (0.5875)	1.48 (0.2173)	**1.41** (0.0625)	1.23 (0.0448)	1.23 (0.0763)	0.674 (0.0400)	**0.752** (0.0417)	0.602 (0.0198)	**0.614** (0.0230)	**0.578** (0.0367)	0.555 (0.0263)
$\gamma = 0.4\%$	3.74 (0.1838)	**3.18** (0.2319)	1.68 (0.2307)	**1.57** (0.0652)	**1.33** (0.0829)	1.39 (0.1044)	0.630 (0.1005)	**0.700** (0.0621)	0.592 (0.0259)	**0.599** (0.0303)	**0.627** (0.0057)	0.463 (0.0334)
$\gamma = 0.5\%$	3.89 (0.1518)	**3.32** (0.2235)	1.83 (0.3599)	**1.63** (0.1241)	**1.45** (0.1081)	1.51 (0.1752)	**0.676** (0.0268)	0.663 (0.0423)	0.567 (0.0310)	**0.603** (0.0269)	**0.550** (0.0619)	0.493 (0.0641)
PUM		PUM= $\frac{3}{6}$		PUM= $\frac{5}{6}$		PUM= $\frac{3}{6}$		PUM= $\frac{4}{6}$		PUM= $\frac{3}{6}$		PUM= $\frac{0}{6}$
T-Hop												
$\gamma = 0$	2.87 (0.3241)	**2.73** (0.0385)	**1.00** (0.0330)	1.00 (0.0367)	**0.78** (0.0210)	0.88 (0.0210)	**0.852** (0.0087)	0.746 (0.0346)	**0.649** (0.0032)	0.623 (0.0363)	**0.785** (0.0079)	0.752 (0.1003)
$\gamma = 0.1$	3.68 (0.7068)	**3.07** (0.0387)	**1.09** (0.0307)	1.32 (0.0524)	**0.82** (0.0052)	1.01 (0.0090)	**0.819** (0.0167)	0.764 (0.0136)	**0.645** (0.0066)	0.618 (0.0077)	**0.826** (0.0061)	0.681 (0.0367)
$\gamma = 0.2$	**3.34** (0.1859)	3.35 (0.2472)	**1.26** (0.1582)	1.30 (0.0215)	**0.88** (0.0383)	1.00 (0.0078)	**0.827** (0.0228)	0.770 (0.0033)	**0.642** (0.01160)	0.620 (0.0262)	**0.806** (0.0023)	0.591 (0.0518)
$\gamma = 0.3$	3.80 (0.8851)	**3.34** (0.3717)	**1.22** (0.0781)	1.34 (0.0678)	**0.92** (0.0241)	1.01 (0.0525)	**0.804** (0.0251)	0.687 (0.0718)	**0.632** (0.0195)	0.612 (0.0201)	**0.799** (0.0151)	0.627 (0.0567)
$\gamma = 0.4$	3.41 (0.4146)	**3.34** (0.3023)	**1.50** (0.3625)	1.52 (0.1281)	**0.93** (0.0286)	1.02 (0.0142)	**0.804** (0.0219)	0.774 (0.0359)	**0.631** (0.0222)	0.611 (0.0202)	**0.784** (0.0212)	0.667 (0.0466)
$\gamma = 0.5$	3.91 (0.1905)	**3.55** (0.1852)	1.62 (0.1876)	**1.42** (0.1401)	**0.93** (0.0333)	1.04 (0.0137)	**0.789** (0.0044)	0.763 (0.0223)	0.624 (0.0154)	**0.634** (0.0242)	**0.773** (0.0095)	0.538 (0.0673)
PUM		PUM= $\frac{5}{6}$		PUM= $\frac{1}{6}$		PUM= $\frac{0}{6}$		PUM= $\frac{0}{6}$		PUM= $\frac{1}{6}$		PUM= $\frac{0}{6}$

of the underlying graphs, without looking at the node and edge features. Hence, to obtain the algorithmic complexity of a dataset family, D_j, it suffices to compute the algorithmic complexity of its original non-synthetic member, $D_j^0 \in D_j$. Therefore, for each of the six dataset families, we computed the algorithmic complexity of every single molecular graph in its non-synthetic dataset member, D_j^0. We averaged the algorithmic complexity values over all molecular graphs in D_j^0. We display the results in the first row of Table 4, wherein we have arranged the results in ascending order from left to right, thereby obtaining ascending order algorithmic complexities (AOAC). Further, Table 4 also contains the PUMs associated with all three models and all six datasets, copied from Table 3.

For each dataset family, we also computed the average PUM accross all model types; we placed those average PUMs in the *across all models* row of the table. To

Table 4. Exhibition of inverse correlation between AOAC and PUM

	DATASETS' AOAC AND MODELS' PUMS						CORRELATION BETWEEN AOAC and PUM
	FREESOLV	ESOL	BBBP	CLINTOX	LIPOPHILICITY	BACE	
DATASETS' AOAC:	105.61	216.63	488.75	510.71	577.11	717.38	
MODELS' PUMs:							
Graphormer	$\frac{6}{6}$	$\frac{4}{6}$	$\frac{1}{6}$	$\frac{2}{6}$	$\frac{0}{6}$	$\frac{2}{6}$	-0.84
Mix-Hop	$\frac{3}{6}$	$\frac{5}{6}$	$\frac{3}{6}$	$\frac{0}{6}$	$\frac{3}{6}$	$\frac{4}{6}$	-0.19
T-Hop	$\frac{5}{6}$	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{0}{6}$	$\frac{0}{6}$	$\frac{0}{6}$	-0.81
Across all models	$\frac{14}{18}$	$\frac{10}{18}$	$\frac{5}{18}$	$\frac{2}{18}$	$\frac{3}{18}$	$\frac{6}{18}$	-0.82

validate our hypothesis from a first perspective, we computed the Pearson correlation between the PUMs, listed in Table 4, and the corresponding algorithmic complexities. We display the computed Pearson correlations in the last column of the table. First, according to the correlations in the table, we note that each model showed negative correlation. The graphormer model showed the strongest negative correlation, followed closely by the T-Hop model, with the Mix-Hop model coming third. Moreover, from the table, we also see that the correlation between the average PUMs and their associated algorithmic complexities is also strongly negative (See the *across all models* row of the table). We illustrate these observations graphically in Fig. 3. As shown in the figures's legend, three of the plots are for the three individual models, while the fourth plot is for the average across all models. All four plots in the figure exhibit negative correlations between PUM and algorithmic complexity. Hence, it becomes obvious that low algorithmic complexity of a molecular dataset family is an indicator that GNN models are more likely to benefit from path information on the datasets within the family. This result is reasonable, because if the path connectivity pattern in a graph is to be useful for machine learning prediction tasks, then we should want the pattern to be as far as possible from being random/unpredictable. That is, we should want it to have low algorithmic complexity.

To validate our hypothesis from a second perspective, we performed binary clustering analysis. We first clustered the dataset families into two categories based on their associated algorithmic complexities. We then repeated the clustering based on PUMs instead of algorithmic complexities. The results are shown in Table 5. In the table, we display the algorithmic complexities (and PUMs), and place the cluster labels, in parentheses, next to the algorithmic complexities (and PUMs). For the algorithmic-complexities-based clustering, we used 0 to label dataset families corresponding to low algorithmic complexities, and used 1 to label those corresponding to high algorithmic complexities. However, since we

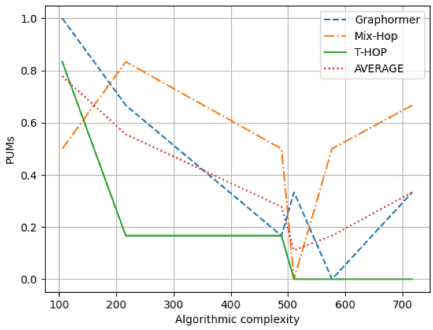


Fig. 3. Plots of PUMs versus algorithmic complexity.

hypothesize an inverse relationship between algorithmic complexity and PUM, we inverted the labelling scheme in the case of the PUM-based clustering. For the PUM-based clustering, we used 0 to label dataset families corresponding to high PUMs, and used 1 to label those corresponding to low PUMs. Our hypothesis is that the dataset-family clustering produced via the algorithmic complexity values, should match with the dataset-family clustering produced via the PUM. By looking at the *across all models* row in Table 5, we find that the cluster labels there perfectly match the cluster labels in the algorithmic complexities row of the table, thereby validating our hypothesis. For individual models, we found that the Graphormer model also agrees with the hypothesis, with the T-Hop model coming next in agreement, and the Mix-Hop model coming third in agreement. The observed disparities in the levels of agreement with the hypothesis are not far fetched, since the models incorporate path information in different ways. In the table, we also included a column that contains the Silhouette scores for the clustering produced by the three models. We found that the T-Hop model got the highest Silhouette score, which is likely to be due to its heavy and aggressive use of all the available path information contained in the graphs, whereas the graphormer model, for example, only considers path information along shortest paths between graph nodes.

4.4 Computational Run-Times with and Without Path Information

We juxtapose the compute times required for our models without using path information versus using path information. The single-epoch training times, in seconds, for all six non-synthetic datasets, across all three models are shown in Table 6. As can be clearly seen from the table, in all cases, running the models without path information is more efficient, as expected. For each model type, the model sizes were kept constant across all datasets. Table 7 shows the pertinent hyper-paramaters defining the model sizes. We listed "None" as the maximum path length of the Graphormer model because the model considers all pair-wise node interactions in its attention mechanism, regardless of how far apart the nodes are in the underlying graph topology.

Table 5. Two ways of clustering dataset families: 1) based on algorithmic complexities and 2). based on PUMs. Clusterings produced via the two ways match remarkably.

	DATASETS' AOAC, MODELS' PUMS AND CLUSTER LABELS IN PARENTHESES						SILHOUETTE SCORES
	FREESOLV	ESOL	BBBP	CLINTOX	LIPOPHILICITY	BACE	
DATASETS' AOAC & CLUSTER LABELS:	105.61 **(0)**	216.63 **(0)**	488.75 **(1)**	510.71 **(1)**	577.11 **(1)**	717.38 **(1)**	0.71
MODELS' PUMs & CLUSTER LABELS:							
Graphormer	$\frac{6}{6}$ **(0)**	$\frac{4}{6}$ **(0)**	$\frac{1}{6}$ **(1)**	$\frac{2}{6}$ **(1)**	$\frac{0}{6}$ **(1)**	$\frac{2}{6}$ **(1)**	0.60
Mix-Hop	$\frac{3}{6}$ **(0)**	$\frac{5}{6}$ **(0)**	$\frac{3}{6}$ **(0)**	$\frac{0}{6}$ **(1)**	$\frac{3}{6}$ **(0)**	$\frac{4}{6}$ **(0)**	0.61
T-Hop	$\frac{5}{6}$ **(0)**	$\frac{1}{6}$ **(1)**	$\frac{1}{6}$ **(1)**	$\frac{0}{6}$ **(1)**	$\frac{0}{6}$ **(1)**	$\frac{0}{6}$ **(1)**	0.72
Across all models	$\frac{14}{18}$ **(0)**	$\frac{10}{18}$ **(0)**	$\frac{5}{18}$ **(1)**	$\frac{2}{18}$ **(1)**	$\frac{3}{18}$ **(1)**	$\frac{6}{18}$ **(1)**	0.62

Table 6. Comparison of computational run-times with and without path information across all six original datasets.

	FREESOLV		ESOL		LIPOPHILICITY		BACE		BBBP		CLINTOX	
	WITHOUT PATH	WITH PATH	WITHOUT PATH	WITH PATH	WITHOUT PATH	WITH PATH	WITHOUT PATH	WITH PATH	WITHOUT PATH	WITH PATH	WITHOUT PATH	WITH PATH
T-Hop	**0.46**	0.69	**0.70**	6.94	**2.42**	172.43	**0.95**	38.76	**1.52**	133.37	**1.23**	100.32
Graphormer	**0.93**	4.88	**1.80**	25.96	**10.69**	547.71	**3.39**	301.83	**6.23**	191.69	**4.70**	275.44
Mix-Hop	**0.56**	0.65	**0.88**	0.95	**3.23**	3.56	**1.22**	1.35	**2.14**	2.33	**1.69**	1.90

Table 7. Comparison of computational run-times with and without path information across all six original datasets.

	LAYERS	HIDDEN DIM	ATTENTION HEADS	MAX. PATH LENGTH(L_m)	
				WITHOUT PATH	WITH PATH
T-Hop	5	600	None	1	4
Graphormer	5	600	6	None	None
Mix-Hop	5	600	None	1	3

5 Conclusion and Future Work

This work advances our understanding of conditions under which the incorporation of path information into graph neural networks improves accuracies on molecular graph datasets. Specifically, our results demonstrate that molecular graphs with higher algorithmic complexity scores, which reflect more random patterns, show minimal or no benefit from the inclusion of path information, thereby questioning the cost-effectiveness of incorporating such information in these cases. Indeed, future work can parlay this finding into a scheme that conserves computational resources, by not expending the extra computation involved in incorporating path information on datasets that are known to have high algorithmic complexity. This is especially important in large-scale or resource-constrained environments, where high computational costs are pro-

hibitive. Finally, while the current study was limited to the domain of molecular graphs, we believe that the lessons learnt here will spark researchers in other domains to test our hypothesis on graphs occurring within their domains.

References

1. Shen, J., Ain, Q.T., Liu, Y., Liang, B., Qiang, X., Kou, Z.: GTAT: empowering graph neural networks with cross attention. Sci. Rep. **15**(1), 4760 (2025)
2. Wang, J., Hou, K., Chen, H., Fang, J., Li, H.: Dual self-attention fusion message neural network for virtual screening in drug discovery by molecular property prediction. J. Bionic Eng. **22**(1), 354–369 (2025)
3. Yu, Z., Huang, W., Liu, Y.: Rigid protein-protein docking via equivariant elliptic-paraboloid interface prediction. arXiv preprint arXiv:2401.08986 (2024)
4. Kipf, T.: and Welling. M. Semi-supervised classification with graphconvolutional networks. ICLR (2017)
5. Ying, C., et al.: Do Transformers really perform bad for graph representation? In: Advances in Neural Information Processing Systems, pp. 28877–28888 (2021)
6. Vaswani, A., et al.: Attention is all you need. In: Advances in Neural Information Processing Systems, pp. 5998–6008 (2017)
7. Abu-El-Haija, S., et al.: Higher-order graph convolutional architectures via sparsified neighborhood mixing. ICML, MixHop (2019)
8. Zenil, H., Kiani, N.A., Tegnér, J.: Algorithmic Information Dynamics: A Computational Approach to Causality with Applications to Living Systems. Cambridge University Press (2023)
9. Zenil, H., Kiani, N.A., Tegnér, J.: A review of graph and network complexity from an algorithmic information perspective. Entropy **20**(8), 551 (2018)
10. Zenil, H., Hernández-Orozco, S., Kiani, N.A., Soler-Toscano, F., Rueda-Toicen, A., Tegnér, J.: A decomposition method for global evaluation of shannon entropy and local estimations of algorithmic complexity. Entropy **20**(8), 605 (2018)
11. Wu, Z., et al.: MoleculeNet: a benchmark for molecular machine learning. Chem. Sci. **9**, 513–530 (2018)
12. Gori, M., Monfardini, G., Scarselli, F.: A new model for learning in graph domains. In: Proceedings. IEEE International Joint Conference on Neural Networks, vol. 2, pp. 729–734 (2005)
13. Gilmer, J., Schoenholz, S., Riley, P., Vinyals, O., Dahl, G.: Neural message passing for quantum chemistry. arXiv preprint arXiv:1704.01212 (2017)
14. Yang, K., Swanson, K., J.W.: Analyzing learned molecular representations for property prediction. J. Chem. Inf. Model. **59**, 3370–3388 (2019)
15. Veličković, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., Bengio, Y.: Graph attention networks. In: International Conference on Learning Representations (2018)
16. Li, Q., Wu, X., Liu, H., Zhang, X., Guan, Z.: Label efficient semi-supervised learning via graph filtering. In: CVPR, pp. 9582–9591 (2019)
17. Jin, M., Chang, H., Zhu, W., Sojoudi, S.: Power up! robust graph convolutional network via graph powering. In: AAAI (2021)
18. Kong, L., Chen, Y., Zhang, M.: Geodesic graph neural network for efficient graph representation learning. arXiv preprint arXiv:2210.02636 (2022)
19. Sun, Y., et al.: Beyond homophily: structure-aware path aggregation graph neural network. In: Proceedings of the 31st International Joint Conference on Artificial Intelligence, pp. 2233–2240 (2022)

20. Michel, G., Nikolentzos, G., Lutzeyer, J., Vazirgiannis, M.: Path neural networks: expressive and accurate graph neural networks. In: Proceedings of the ICML (2023)
21. Zhu, Z., Zhang, Z., Xhonneux, L.P., Tang, J.: Neural bellman-ford networks: a general graph neural network framework for link prediction. In: Advances in Neural Information Processing Systems, vol. 34 (2021)
22. Bellman, R.: On a routing problem. Q. Appl. Math. **16**(1), 87–90 (1958)
23. Katz, L.: A new status index derived from sociometric analysis. Psychometrika **18**(1), 39–43 (1953)
24. Liben-Nowell, D., Kleinberg, J.: The link-prediction problem for social networks. J. Am. Soc. Inform. Sci. Technol. **58**(7), 1019–1031 (2007)
25. Page, L., Brin, S., Motwani, R., Winograd, T.: The pagerank citation ranking: bringing order to the web. Technical report, Stanford InfoLab (1999)
26. Morris, C., Kriege, N.M., Bause, F., Kersting, K., Mutzel, P., Neumann, M.: TUDataset: a collection of benchmark datasets for learning with graphs. In: Graph Representation Learning and Beyond Workshop (2020)
27. Cong, W., et al.: Do we really need complicated model architectures for temporal networks? In: ICLR (2023)
28. Xian, Y., Lampert, C., Schiele, B., Akata, Z.: Zero-shot learning-a comprehensive evaluation of the good, the bad and the ugly. IEEE Trans. Pattern Anal. Mach. Intell. (2018)
29. Li, M., et al.: DGL-LifeSci: an open-source toolkit for deep learning on graphs in life science. ACS Omega (2021)
30. Wang, M., et al.: Deep graph library: a graph-centric, highly-performant package for graph neural networks. arXiv preprint arXiv:1909.01315 (2019)
31. Weininger, D.: Smiles, a chemical language and information system. 1. introduction to methodology and encoding rules. J. Chem. Inf. Comput. Sci. **28**(1), 31–36 (1988)
32. Akiba, T., Sano, S., Yanase, T., Ohta, T., Koyama, M.: Optuna: a next-generation hyperparameter optimization framework. In: KDD '19: Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 2623–2631 (2019)
33. Liaw, R., Liang, E., Nishihara, R., Moritz, P., Gonzalez, J.E., Stoica, I.: Tune: a research platform for distributed model selection and training. arXiv preprint arXiv:1807.05118