An Empirical Study of the Expressiveness of Graph Kernels and Graph Neural Networks

Giannis Nikolentzos[1,2(✉)], George Panagopoulos[1], and Michalis Vazirgiannis[1,2]

[1] École Polytechnique, Palaiseau, France
[2] Athens University of Economics and Business, Athens, Greece
{nikolentzos,mvazirg}@lix.polytechnique.fr,
george.panagopoulos@polytechnique.edu

Abstract. Graph neural networks and graph kernels have achieved great success in solving machine learning problems on graphs. Recently, there has been considerable interest in determining the expressive power mainly of graph neural networks and of graph kernels, to a lesser extent. Most studies have focused on the ability of these approaches to distinguish non-isomorphic graphs or to identify specific graph properties. However, there is often a need for algorithms whose produced graph representations can accurately capture similarity/distance of graphs. This paper studies the expressive power of graph neural networks and graph kernels from an empirical perspective. Specifically, we compare the graph representations and similarities produced by these algorithms against those generated by a well-accepted, but intractable graph similarity function. We also investigate the impact of node attributes on the performance of the different models and kernels. Our results reveal interesting findings. For instance, we find that theoretically more powerful models do not necessarily yield higher-quality representations, while graph kernels are shown to be very competitive with graph neural networks.

Keywords: Expressive power · Graph neural networks · Graph kernels

1 Introduction

In recent years, graph-structured data has experienced an enormous growth in many domains, ranging from chemo- and bio-informatics to social network analysis. Several problems of increasing interest require applying machine learning techniques to graph-structured data. Examples of such problems include predicting the quantum mechanical properties of molecules [13] and modeling physical systems [2]. In the past years, the problem of machine learning on graphs has been governed by two major families of approaches, namely graph kernels (GKs) [26] and graph neural networks (GNNs) [35].

Recently, much research has focused on measuring the expressive power of GNNs [1,5,6,19,22–24,36]. On the other hand, in the case of GKs, there was a limited number of similar studies [17]. This is mainly due to the fact that the

I. Farkaš et al. (Eds.): ICANN 2021, LNCS 12893, pp. 139–150, 2021.
https://doi.org/10.1007/978-3-030-86365-4_12

landscape of GKs is much more diverse than that of GNNs. Indeed, although numerous GNN variants have been recently proposed, most of them share the same basic idea, and can be reformulated into a single common framework, so-called message passing neural networks [13]. These models employ a message passing procedure to aggregate local information of vertices and are closely related to the Weisfeiler-Lehman test of graph isomorphism, a powerful heuristic which can successfully test isomorphism for a broad class of graphs.

When dealing with learning problems on graphs, a practitioner needs to choose one GNN or one GK for her particular application. The practitioner is then faced with the following question: Does this GNN variant or GK capture graph similarity better than others? Unfortunately, this question is far from being answered. Most of the above studies investigate the power of GNNs in terms of distinguishing between non-isomorphic graphs or in terms of how well they can approximate combinatorial problems. However, in graph classification/regression problems, we are not that much interested in testing whether two (sub)graphs are isomorphic to each other, but mainly in classifying graphs or in predicting real values associated with these graphs. In such tasks, it has been observed that stronger GNNs do not necessarily outperform weaker GNNs. Therefore, it seems that the design of GNNs is driven by theoretical considerations which are not realistic in practical settings. Ideally, we would like to learn representations which accurately capture the similarities or distances between graphs.

A practitioner can then choose an algorithm based on its empirical performance. Indeed, GNNs and GKs are usually evaluated on standard datasets derived from bio-/chemo-informatics and from social media [21]. However, several concerns have been raised recently with regards to the reliability of those datasets, mainly due to their small size and to inherent isomorphism bias problems [15]. More importantly, it has been observed that the adopted experimental settings are in many cases ambiguous or not reproducible [8]. The experimental setup is not standardized across different works, and there are often many issues related to hyperparameter tuning and to how model selection and model assessment are performed. These issues easily generate doubts and confusion among practitioners that need a fully transparent and reproducible experimental setting.

Present Work. In this paper, we empirically evaluate the expressive power of GNNs and GKs. Specifically, we build a dataset that contains instances of different families of graphs. Then, we compare the graph representations and similarities produced by GNNs and GKs against those generated by an intractable graph similarity function which we consider to be an oracle function that outputs the true similarity between graphs. We perform a large number of experiments where we compare several different kernels, architectures, and pooling functions. Secondly, we study the impact of node attributes on the performance of the different models and kernels. We show that annotating the nodes with their degree and/or triangle participation can be beneficial in terms of performance in the case of GNNs, while it is not very useful in the case of GKs. Finally, we investigate which pairs of graphs (from our dataset) lead GNNs and GKs to the highest error in the estimated similarity. Surprisingly, we find that several GNNs and GKs assign identical or similar representations to very dissimilar graphs.

2 Related Work

Over the past years, the expressiveness of GKs was assessed almost exclusively from experimental studies. Therefore, still, there is no theoretical justification on why certain GKs perform better than others. From the early days of the field, it was clear though that the mapping induced by kernels that are computable in polynomial time is not injective [12]. Recently, a framework was developed to measure the expressiveness of GKs based on ideas from property testing [17]. It was shown that some well-established GKs such as the shortest path kernel, the graphlet kernel, and the Weisfeiler-Lehman subtree kernel cannot identify basic graph properties such as planarity or bipartitness.

With a few exceptions [28], until recently, there has been little attempt to understand the expressive power of GNNs. Several recent studies have investigated the connections between GNNs and different variants of the Weisfeiler-Lehman (WL) test of isomorphism. For instance, it was shown that standard GNNs do not have more power in terms of distinguishing between non-isomorphic graphs than the WL algorithm [22,36]. Morris *et al.* proposed in [22] a family of GNNs which rely on a message passing scheme between subgraphs of cardinality k, and which have exactly the same power in terms of distinguishing non-isomorphic graphs as the set-based variant of the k-WL algorithm. In a similar spirit, Maron *et al.* introduced in [19] k-order graph networks which are at least as powerful as the folklore variant of the k-WL graph isomorphism test in terms of distinguishing non-isomorphic graphs. These models were also shown to be universal [20], but require using high order tensors and therefore are not practical. Chen *et al.* showed in [5] that the two main approaches for studying the expressive power of GNNs, namely graph isomorphism testing and invariant function approximation, are equivalent to each other. Furthermore, the authors propose a GNN that is more powerful than 2-WL. Based on a connection between the WL algorithm and first order logic, Barceló *et al.* characterized in [1] the expressive power of GNNs in terms of classical logical languages. The impact of random features on the expressive power of GNNs is considered in [27]. Nikolentzos *et al.* showed in [24] that standard GNNs cannot identify fundamental graph properties such as triangle-freeness and connectivity, and they proposed a model that can identify these properties. Other studies take into account all possible node permutations and produce universal graph representations [6,23]. The emerging problems become intractable once the number of nodes is large and they propose approximation schemes to make the computation feasible. Some recent works have studied the generalization properties of GNNs [11,29,34].

3 Comparing Graphs to Each Other

Formally, for any two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ on n vertices with respective $n \times n$ adjacency matrices $\mathbf{A}_1$ and $\mathbf{A}_2$, we define a function $f : \mathcal{G} \times \mathcal{G} \rightarrow \mathbb{R}$ where $\mathcal{G}$ is the space of graphs which quantifies the similarity of G_1 and G_2. Note that in the literature, this problem is often referred to as *graph comparison*.

In this paper, we consider a graph comparison function which is not computable in polynomial time. The function can be expressed as a maximization problem, and is defined as follows:

$$f(G_1, G_2) = \max_{\mathbf{P} \in \Pi} \frac{\sum_{i=1}^{n} \sum_{j=1}^{n} \left[\mathbf{A}_1 \odot \mathbf{P}\mathbf{A}_2\mathbf{P}^\top\right]_{ij}}{||\mathbf{A}_1||_F \, ||\mathbf{A}_2||_F} \tag{1}$$

where Π denotes the set of $n \times n$ permutation matrices, $\odot$ denotes the element-wise product, and $||\cdot||_F$ is the Froebenius matrix norm. For clarity of presentation we assume n to be fixed (i.e., both graphs consist of n vertices). In order to apply the function to graphs of different cardinality, one can append zero rows and columns to the adjacency matrix of the smaller graph to make its number of rows and columns equal to n. Therefore, the problem of graph comparison can be reformulated as the problem of maximizing the above function over the set of permutation matrices. A permutation matrix $\mathbf{P}$ gives rise to a bijection $\pi : V_1 \rightarrow V_2$. The function defined above seeks for a bijection such that the number of common edges $|\{(u, v) \in E_1 : \big(\pi(u), \pi(v)\big) \in E_2\}|$ is maximized. Then, the number of common edges is normalized such that it takes values between 0 and 1. Solving the above optimization problem for large graphs is clearly intractable since there are $n!$ permutation matrices of size n. In this paper, we investigate how different graph comparison/representation learning approaches approximate the above-defined function from an empirical standpoint.

4 Empirical Evaluation

4.1 Dataset

Since the function defined in (1) is intractable for large graphs, we generated graphs consisting of at most 9 vertices. Furthermore, each graph is connected and contains at least 1 edge. We generated 191 pairwise non-isomorphic graphs. The dataset consists of different types of synthetic graphs. These include simple structures such as cycle graphs, path graphs, grid graphs, complete graphs and star graphs, but also randomly-generated graphs such as Erdős-Rényi graphs, Barabási-Albert graphs and Watts-Strogatz graphs. Table 1 shows statistics of the synthetic dataset that we used in our experiments. Figure 1 illustrates the distribution of the similarities of the generated graphs as computed by the proposed measure. There are $^{191*192}/_{2} = 18,336$ pairs of graphs in total (including pairs consisting of a graph and itself). Interestingly, most of the similarities take values between 0.5 and 0.8.

4.2 Selected Approaches

Suitable GKs and GNNs were selected according to the following criteria: (1) publicly available implementations, (2) strong architectural differences, (3) popularity, and (4) peer reviewed. We next present the GKs and GNNs that were

Table 1. Summary of the synthetic dataset that we used in our experiments.

Synthetic dataset	
Max # vertices	9
Min # vertices	2
Average # vertices	7.29
Max # edges	36
Min # edges	1
Average # edges	11.34
# graphs	191

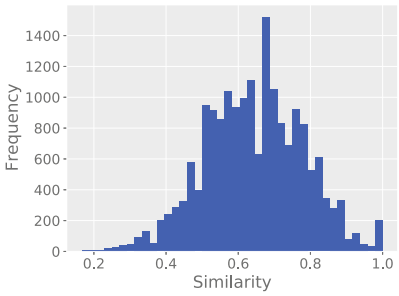


Fig. 1. Distribution of similarities between the synthetically generated graphs.

included into our evaluation. For a detailed description of each kernel and each GNN, we refer the reader to their respective papers.

We selected the following 6 GKs: (1) random walk kernel (RW) [12], (2) shortest path kernel (SP) [4], (3) graphlet kernel (GR) [31], (4) Weisfeiler-Lehman subtree kernel (WL) [30], (5) pyramid match kernel (PM) [25], and (6) GraphHopper kernel [9]. Note that GR can only operate on unlabeled graphs. The rest of the kernels can handle graphs with discrete node labels, while GraphHopper is the only kernel that can deal with continuous multi-dimensional node features.

We also selected the following GNNs: (1) GCN [16], (2) GAT [33], (3) 1-GNN [22], (4) GG-NN [18], (5) GraphSAGE [14], (6) GIN [36], (7) ChebNet [7], (8) ARMA [3], (9) 1-2-GNN [22], (10) k-hop GNN [24], and (11) Provably Powerful GNN [19]. To produce graph representations, we use 3 pooling functions, namely the sum aggregator, the mean aggregator and the max aggregator.

4.3 Baselines

We utilize some simple baselines whose purpose is to understand the extent to which GKs and GNNs can indeed learn representations that capture graph similarity. The first baseline is a function that randomly computes the similarity of two graphs by sampling a number from $[0, 1]$ with uniform probability. The second baseline is a constant function (output equal to 1). Using such simple baselines as a reference is crucial since they can provide feedback on the effectiveness of GKs and GNNs in the considered task. If the performance of a GNN or a GK is close to that of one of the baselines, that would mean that the GNN/GK fails to encode accurately graph representations and similarities.

4.4 Experimental Settings

Normalization. As discussed above, the function defined in (1) gives an output in the range $[0, 1]$. The similarity of two graphs G_1 and G_2 is equal to 1 if and only if G_1 and G_2 are isomorphic to each other. We normalize the

obtained kernel values as follows such that they also take values in the range $[0, 1]$: $k_{ij} = {}^{k_{ij}}/_{\sqrt{k_{ii}}\sqrt{k_{jj}}}$. where k_{ij} is the kernel between graphs G_i and G_j. For the GNNs, we compute the cosine similarity between the graph representations of the penultimate layer as follows: ${}^{\mathbf{z}_i^\top \mathbf{z}_j}/_{||\mathbf{z}_i|| \, ||\mathbf{z}_j||}$ where $\mathbf{z}_i$ is the representation of graph G_i. Note that the ReLU function has already been applied to these representations, and thus the cosine similarity also takes values between 0 and 1. In fact, the two employed normalization schemes (i.e., for kernels and GNNs) are equivalent since a kernel value corresponds to an inner product between the representations of two graphs in some Hilbert space.

Evaluation Metrics. To assess how well the different approaches approximate the similarity function defined in (1), we employed two evaluation metrics: the Pearson correlation coefficient and the mean squared error (MSE). The Pearson correlation coefficient measures the linear relationship between two variables. It takes values between -1 and 1, while a value of 1 denotes total positive linear correlation. In our setting, a high value of correlation would mean that the approach under consideration captures the relationships between the similarities (e.g., whether the similarity of a pair of graphs is greater or lower than that of another pair). The second measure, MSE, is equal to the average squared difference between the estimated values and the actual values. A very small value of MSE denotes that the derived similarities are very close to those produced by the function defined in (1). A credible graph representation learning/similarity approach would yield both a high correlation and a small MSE. Note that the correlation between the output of a constant function and the output of (1) is not defined since the values produced by the constant function have a variance equal to zero.

Hyperparameters. For RW, we set λ to 0.01. For GR, we count all the graphlets of size 3. For the WL kernel, we choose the number of iterations from $\{1, 2, \ldots, 6\}$. For PM, the dimensionality of the embeddings d and the number of levels L are set equal to 6 and 4, respectively. For the GraphHopper kernel, a linear kernel is used on the continuous-valued attributes.

For the GNNs, we use 2 neighborhood aggregation layers. For GraphSAGE, we use the mean function to aggregate the neighbors of a node. For ChebNet, we use polynomials of order 2, and for ARMA, we set the number of stacks K to 2 and the depth T also to 2. The hidden-dimension size of the neighborhood aggregation layers is set equal to 64. We apply the ReLU activation function to the output of each neighborhood aggregation layer. As mentioned above, we use 3 common readout functions: sum, mean and max. The output of the readout function is passed on to a fully-connected layer with 32 hidden units followed by ReLU nonlinearity. The output of the ReLU function corresponds to the representation of the input graph (i.e., each graph is represented by a 32-dimensional vector). For Provably Powerful GNN, we use a network suffix that consists of an invariant readout function followed by 2 fully connected layers. To train all neural networks, we use the Adam optimizer with learning rate 0.001. We set the batch size to 32 and the number of epochs to 100. We store the model that achieved the best validation accuracy into disk. At the end of training, the model is retrieved from the disk, and we use it to generate graph representations.

Protocol. For deterministic approaches, we compute the graph similarities once and report the emerging correlation and MSE. For the remaining approaches (e.g., GNNs, since their parameters are randomly initialized), we repeat the experiment 10 times and report the average correlation, the average MSE and the corresponding standard deviations.

Implementations. For the GKs, we used the implementations contained in the GraKeL library [32]. For 1-2 GNN, k-hop GNN, and Provably Powerful GNN, we use the implementations provided by the authors. The remaining GNNs were implemented with the Pytorch Geometric library [10].

4.5 Results

No Features. In the first set of experiments, we do not use any pre-computed features, and therefore, the emerging representations/similarities rely solely on the representational capabilities of the different approaches. Note that all GNNs except Provably Powerful GNN and most GKs require the nodes to be annotated with either continuous attributes or discrete labels. Therefore, for these approaches, we annotate each node with a single feature (i.e., the discrete label 1 or the real value 1.0).

For GNNs, we consider two alternatives: (1) we randomly initialize the parameters of the models and we perform a feedforward pass to generate the graph representations or (2) we randomly initialize the parameters of the models, we train the models on some independent dataset and then we perform the feedforward pass to generate the representations. GKs are, in a sense, unsupervised, and cannot be trained on any dataset. The obtained results for the two aforementioned cases are illustrated in Fig. 2. Note that in the second case, the models were trained on the IMDB-BINARY graph classification dataset.

In terms of correlation, 1-GNN and GIN are the best-performing approaches followed by GG-NN and one variant of Chebnet. The 6 GKs along with 2-hop GNN and Provably Powerful GNN perform slightly worse than the above models. In terms of MSE, the GKs outperform in general the GNNs. Notably, some GKs such as WL and PM achieve very low values of MSE. On the other hand, most GNNs fail to outperform the random baseline. Specifically, the only models that outperform this baseline are 1-GNN, 2-hop GNN, GG-NN with sum pooling, Chebnet with sum pooling and Provably Powerful GNN with max pooling. With regards to the three pooling functions, the sum operator seems to outperform the others. Furthermore, it is important to mention that more powerful architectures (e.g., 1-2-GNN, 2-hop GNN, Provably Powerful GNN) do not necessarily lead to higher correlation and lower MSE than standard, less expressive GNNs. We next investigate how the performance of the GNNs changes when the models are first trained on the IMDB-BINARY dataset. The results are shown in Fig. 2 (Bottom). We observe that there is a decrease in correlation, and no GNN achieves higher correlation than the ones achieved by the WL and PM kernels anymore. On the other hand, the MSE of most GNNs also decreases. For instance, most GNNs now yield lower MSEs than the random baseline. However, still, GCN, GAT and GraphSAGE fail to outperform this baseline.

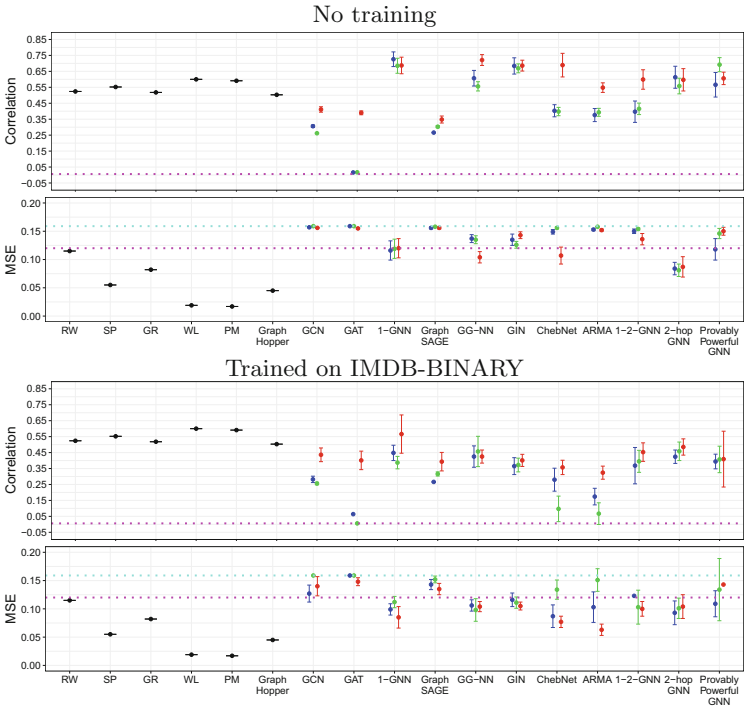


Fig. 2. Performance of the different approaches without node features. The GNNs were either not trained (Top) or trained on the IMDB-BINARY dataset (Bottom). For GNNs, the different colors indicate the three pooling functions: sum (•), mean (•), and max (•). The horizontal lines correspond to the two baselines (random, constant) (Color figure online).

The Effect of Node Features. In GNN literature, it is common practice to use local features (e.g., degree) as node attributes. In previous studies, it has been reported that using node degrees as input features leads to an increase in performance on almost all graph classification datasets [8]. We next investigate what is the impact of such features on the learned graph representations. Specifically, each node is annotated with a 2-dimensional vector where the two elements correspond to its degree and to the number of triangles in which it participates. Note that the GR kernel cannot handle node labels/attributes, and hence, it is excluded from the evaluation. Furthermore, all the other GKs except GraphHopper can only handle discrete node labels, and thus we map each unique pair of features (i.e., degree and triangle participation) to a natural number.

Figure 3 illustrates the obtained results for the case of randomly initialized GNNs and GNNs trained on IMDB-BINARY. We observe that GraphHopper, the only kernel that can naturally handle multi-dimensional continuous node features, takes advantage of the node degree and triangle participation information since it exhibits a very high correlation and a small MSE. On the other hand, the quality of the representations learned by the remaining GKs seems to be lower

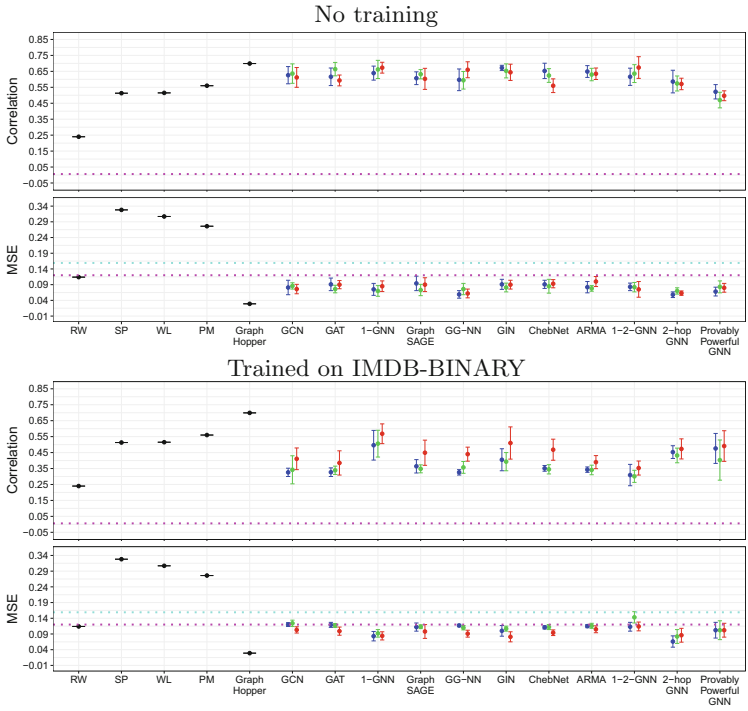
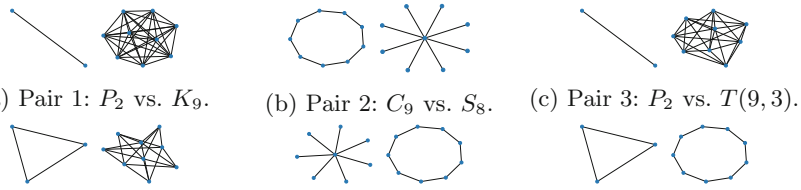


Fig. 3. Performance of the different approaches with node features. The GNNs were either not trained (Top) or trained on the IMDB-BINARY dataset (Bottom). For GNNs, the different colors indicate the three pooling functions: sum (●), mean (●), and max (●). The horizontal lines correspond to the two baselines (random, constant) (Color figure online).

when these features are taken into account. The addition of the features leads to slight decrease in correlation, and also to a large increase in MSE. This suggests that for GKs that are not designed to operate on graphs with continuous node attributes, using these features may result into a decrease in performance. In the case of randomly initialized GNNs, we observe an increase in the correlation between most models and the considered graph similarity function. 1-2-GNN, 1-GNN and GG-NN achieve the highest levels of correlation, while 2-hop GNN, GG-NN and Provably Powerful GNN are the best-performing models in terms of MSE. Again, more complicated models do not necessarily outperform simpler models. When trained on IMDB-BINARY, the correlation between the models and (1) decreases. At the same time, the models yield slightly higher MSEs.

Which Pairs are Hard? We next find for each approach the similarity that deviates most from the one computed using (1). These pairs of graphs can be thought of as the most challenging for a model or kernel. For each approach, we find the pair of graphs which maximizes the absolute difference between the similarity computed using (1) and the one produced by the model/kernel. Note that for GNNs, we focus on the worst-case scenario independent of the pooling



(a) Pair 1: P_2 vs. K_9. (b) Pair 2: C_9 vs. S_8. (c) Pair 3: P_2 vs. $T(9,3)$.

(d) Pair 4: C_3 vs. $T(9,2)$. (e) Pair 5: S_7 vs. C_9. (f) Pair 6: C_3 vs. C_9.

Fig. 4. Examples of challenging pairs of graphs for certain GNNs/GKs.

function. The different pairs of graphs are illustrated in Fig. 4. Surprisingly, a lot of methods fail to accurately estimate the similarity of structurally very dissimilar graphs. For instance, GCN, GAT, GraphSAGE, ARMA, Chebnet, SP and GraphHopper all found the first two graphs (the path graph P_2 and the complete graph on 9 vertices) to be identical to each other (i.e., similarities equal to 1), while the output of (1) is 0.166. In the case of the two GKs, this is because the implicit vector representation of one graph is a positive scalar multiple of the representation of the other. With regards to the GNNs, note that the above models use the mean function to aggregate the representations of their neighborhoods. When such neighborhood aggregation approaches are followed by mean or max pooling functions, they produce identical representations for the P_2 and K_9 graphs. Another pair of graphs which turns out to be challenging for several approaches is the one consisting of the cycle graph with 9 vertices and the star graph S_8. The output of (1) for this pair is equal to 0.235, while the similarities produced by all the following approaches are greater than 0.900: GIN, 1-GNN, GG-NN, 2-hop GNN and RW. The next four pairs of graphs correspond to the worst-case scenarios for Provably Powerful GNN, 1-2-GNN, PM and WL, respectively. The similarity between the path graph P_2 and the Turán graph $T(9,3)$ (i.e., 3^{rd} pair) is 0.192 according to (1), while the representations generated by Provably Powerful GNN yielded a similarity equal to 0.955. Likewise, the output of (1) is 0.258 for the 4^{th} pair of graphs, while 1-2-GNN produced a value equal to 0.922. The star graph S_7 along with the cycle graph with 9 vertices (i.e., 5^{th} pair) led PM to the worst similarity estimation. While (1) gave a similarity of 0.251, the normalized kernel value was equal to 0.773. The last pair of graphs, the cycle graph with 3 vertices and the cycle graph with 9 vertices, turns out to be the hardest for the WL kernel. While the value of (1) is 0.384, the normalized kernel value is equal to 1. Again, this is due to the fact that the representation of one graph is a positive scalar multiple of the representation of the other.

5 Conclusion

In this paper, we studied the expressive power of GNNs and GKs from an empirical standpoint. The produced representations and similarities were compared against those generated by an intractable graph similarity function. The results showed that theoretically more powerful GNNs do not necessarily yield higher-quality representations, while GKs were found to be competitive with GNNs.

References

1. Barceló, P., Kostylev, E.V., Monet, M., Pérez, J., Reutter, J., Silva, J.P.: The logical expressiveness of graph neural networks. In: International Conference on Learning Representations (2020)
2. Battaglia, P., Pascanu, R., Lai, M., Rezende, D.J., et al.: Interaction networks for learning about objects, relations and physics. In: Advances in Neural Information Processing Systems, pp. 4502–4510 (2016)
3. Bianchi, F.M., Grattarola, D., Alippi, C., Livi, L.: Graph neural networks with convolutional ARMA filters. arXiv preprint arXiv:1901.01343 (2019)
4. Borgwardt, K.M., Kriegel, H.P.: Shortest-path kernels on graphs. In: Proceedings of the 5th IEEE International Conference on Data Mining, pp. 74–81 (2005)
5. Chen, Z., Villar, S., Chen, L., Bruna, J.: On the equivalence between graph isomorphism testing and function approximation with GNNs. In: Advances in Neural Information Processing Systems, pp. 15894–15902 (2019)
6. Dasoulas, G., Santos, L.D., Scaman, K., Virmaux, A.: Coloring graph neural networks for node disambiguation. In: Proceedings of the 29th International Joint Conference on Artificial Intelligence, pp. 2126–2132 (2020)
7. Defferrard, M., Bresson, X., Vandergheynst, P.: Convolutional neural networks on graphs with fast localized spectral filtering. In: Advances in Neural Information Processing Systems, pp. 3844–3852 (2016)
8. Errica, F., Podda, M., Bacciu, D., Micheli, A.: A fair comparison of graph neural networks for graph classification. In: 8th International Conference on Learning Representations (2020)
9. Feragen, A., Kasenburg, N., Petersen, J., de Bruijne, M., Borgwardt, K.: Scalable kernels for graphs with continuous attributes. In: Advances in Neural Information Processing Systems, pp. 216–224 (2013)
10. Fey, M., Lenssen, J.E.: Fast graph representation learning with PyTorch geometric. arXiv preprint arXiv:1903.02428 (2019)
11. Garg, V.K., Jegelka, S., Jaakkola, T.: Generalization and representational limits of graph neural networks. In: Proceedings of the 37th International Conference on Machine Learning (2020)
12. Gärtner, T., Flach, P., Wrobel, S.: On graph kernels: hardness results and efficient alternatives. In: Schölkopf, B., Warmuth, M.K. (eds.) COLT-Kernel 2003. LNCS (LNAI), vol. 2777, pp. 129–143. Springer, Heidelberg (2003). https://doi.org/10.1007/978-3-540-45167-9_11
13. Gilmer, J., Schoenholz, S.S., Riley, P.F., Vinyals, O., Dahl, G.E.: Neural message passing for quantum chemistry. In: Proceedings of the 34th International Conference on Machine Learning, pp. 1263–1272 (2017)
14. Hamilton, W.L., Ying, R., Leskovec, J.: Inductive representation learning on large graphs. arXiv preprint arXiv:1706.02216 (2017)
15. Ivanov, S., Sviridov, S., Burnaev, E.: Understanding isomorphism bias in graph data sets. arXiv preprint arXiv:1910.12091 (2019)
16. Kipf, T.N., Welling, M.: Semi-supervised classification with graph convolutional networks. arXiv preprint arXiv:1609.02907 (2016)
17. Kriege, N.M., Morris, C., Rey, A., Sohler, C.: A property testing framework for the theoretical expressivity of graph kernels. In: In Proceeding of the 27th International Joint Conference on Artificial Intelligence, pp. 2348–2354 (2018)
18. Li, Y., Tarlow, D., Brockschmidt, M., Zemel, R.: Gated graph sequence neural networks. arXiv preprint arXiv:1511.05493 (2015)

19. Maron, H., Ben-Hamu, H., Serviansky, H., Lipman, Y.: Provably powerful graph networks. In: Advances in Neural Information Processing Systems, pp. 2156–2167 (2019)
20. Maron, H., Fetaya, E., Segol, N., Lipman, Y.: On the universality of invariant networks. In: Proceedings of the 36th International Conference on Machine Learning, pp. 4363–4371 (2019)
21. Morris, C., Kriege, N.M., Bause, F., Kersting, K., Mutzel, P., Neumann, M.: TUDataset: a collection of benchmark datasets for learning with graphs. arXiv preprint arXiv:2007.08663 (2020)
22. Morris, C., et al.: Weisfeiler and leman go neural: higher-order graph neural networks. In: Proceedings of the AAAI Conference on Artificial Intelligence, pp. 4602–4609 (2019)
23. Murphy, R., Srinivasan, B., Rao, V., Ribeiro, B.: Relational pooling for graph representations. In: Proceedings of 36th the International Conference on Machine Learning, pp. 4663–4673 (2019)
24. Nikolentzos, G., Dasoulas, G., Vazirgiannis, M.: k-hop graph neural networks. Neural Netw. **130**, 195–205 (2020)
25. Nikolentzos, G., Meladianos, P., Vazirgiannis, M.: Matching node embeddings for graph similarity. In: Proceedings of the 31st AAAI Conference on Artificial Intelligence, pp. 2429–2435 (2017)
26. Nikolentzos, G., Siglidis, G., Vazirgiannis, M.: Graph kernels: a survey. arXiv preprint arXiv:1904.12218 (2019)
27. Sato, R., Yamada, M., Kashima, H.: Random features strengthen graph neural networks. arXiv preprint arXiv:2002.03155 (2020)
28. Scarselli, F., Gori, M., Tsoi, A.C., Hagenbuchner, M., Monfardini, G.: Computational capabilities of graph neural networks. IEEE Trans. Neural Netw. **20**(1), 81–102 (2008)
29. Scarselli, F., Tsoi, A.C., Hagenbuchner, M.: The Vapnik-Chervonenkis dimension of graph and recursive neural networks. Neural Netw. **108**, 248–259 (2018)
30. Shervashidze, N., Schweitzer, P., Van Leeuwen, E.J., Mehlhorn, K., Borgwardt, K.M.: Weisfeiler-Lehman graph kernels. J. Mach. Learn. Res. **12**(9) (2011)
31. Shervashidze, N., Vishwanathan, S., Petri, T., Mehlhorn, K., Borgwardt, K.: Efficient graphlet kernels for large graph comparison. In: Proceedings of the 12th International Conference on Artificial Intelligence and Statistics, pp. 488–495 (2009)
32. Siglidis, G., Nikolentzos, G., Limnios, S., Giatsidis, C., Skianis, K., Vazirgiannis, M.: GraKeL: a graph kernel library in python. J. Mach. Learn. Res. **21**(54), 1–5 (2020)
33. Veličković, P., Cucurull, G., Casanova, A., Romero, A., Lio, P., Bengio, Y.: Graph attention networks. arXiv preprint arXiv:1710.10903 (2017)
34. Verma, S., Zhang, Z.L.: Stability and generalization of graph convolutional neural networks. In: Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, pp. 1539–1548 (2019)
35. Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C., Philip, S.Y.: A comprehensive survey on graph neural networks. IEEE Trans. Neural Netw. Learn. Syst. **32**(1), 4–24 (2020)
36. Xu, K., Hu, W., Leskovec, J., Jegelka, S.: How powerful are graph neural networks? In: International Conference on Learning Representations (2019)