Evaluating Size Generalization in Hyperbolic Graph Neural Networks

Nikola Bulat
{nikola.bulat}@studenti.unipd.it

Abstract

Graph Neural Networks (GNNs) have become the standard choice for learning from graph-structured data. Despite their success, the performance of GNNs is limited by the representational capacity of Euclidean geometry. Recent advancements in geometric representation learning have led to various Hyperbolic GNN (HGNN) architectures. In particular, hyperbolic manifolds produce high-quality, low-dimensional embeddings, leading to significant performance improvements for various downstream tasks. However, their ability to improve the size-generalization performance remains under-explored in the current literature. We address this gap by conducting carefully designed size-generalization experiments using an existing HGNN architecture for the graph classification task. Our results show that hyperbolic embeddings can obtain significant improvements in the size-generalization setting even for the datasets where they are not beneficial in the random size split.

1. Introduction

Graph Neural Networks (GNNs) have recently been generalized to non-Euclidean manifolds leading to various Hyperbolic GNN (HGNN) architectures. In the literature, we encounter different ways of defining HGNNs [11, 4, 6, 14] that obtain significant improvements in graph-related tasks. We focus on the graph classification task where hyperbolic embeddings obtain improvements using compact, low-dimensional embeddings [11]. In particular, we identify that, despite promising results on many benchmark datasets, current approaches do not leverage the benefits of high-quality, low-dimensional hyperbolic embeddings in the size-generalization scenario. We analyze the ability of hyperbolic embeddings to maintain the graph classification performance when the number of nodes between training and test graphs significantly differ. Our hypothesis is that hyperbolic embeddings can provide more significant improvements in the size-generalization scenario compared to random graph size distributions. In order to test it, we reuse the architectures introduced in [11] to conduct experiments in the size-generalization setting. Our results confirm the potential of hyperbolic embeddings in the size-generalization scenario in 2 out of 3 datasets. Experiments on the synthetic dataset modified from [11] show clear patterns of improvement using low-dimensional hyperbolic embeddings in the size-generalization setting even when we do not have the same benefits in the random-size split. Moreover, we show that decreasing the training graph sizes has the lowest negative impact on the low-dimensional hyperbolic embeddings. On the other hand, hyperbolic embeddings usually do not provide benefits in higher-dimensional spaces. In addition to synthetic graphs, we conduct experiments on two real-world datasets from the TUDataset library [12], adopting the dataset split from [3]. We achieve partially successful results, i.e., the performance on the PROTEINS dataset exhibits similar patterns as for the synthetic data, while our hypothesis is not true for the DD dataset. Further analysis is required to clearly understand what type of data is suitable for our method.

2. Related Work

Recently, hyperbolic embeddings have been proposed for processing graph data with tree-like structures or power-law distributions. The authors of [14] provide a comprehensive survey of current architectures and applications, while we briefly review the most important papers for understanding our work. In [11], the authors generalize the Graph Convolutional Networks (GCNs) [10] to operate on Riemannian manifolds for the graph classification task. Their results indicate the ability of hyperbolic embeddings to produce high-quality, low-dimensional representations. Concurrently with [11], the authors of [4] develop an attention-based hyperbolic architecture with a strong performance on node classification and link predictions tasks. While these architectures incorporate hyperbolic embeddings indirectly by mapping the manifold to its Euclidean tangent space, the authors of [6] propose an architecture that directly performs hyperbolic feature transformation and neighborhood aggregation. In this way, the architecture does not rely on the local approximation of a hyperbolic manifold leading to significant improvements across various tasks.

While the mentioned papers propose several applications for hyperbolic GNNs, their influence on size-generalization capabilities is under-explored in the literature. On the other hand, we can find different approaches trying to maintain the performance of GNNs when the number of nodes in the graph increases. Some of such methods require knowledge about the domain or access to the test set distribution [2, 15]. Conversely, the authors of [3] propose a novel, model-agnostic regularization method that operates without any assumptions or prior knowledge. Their improvements directly come from graph representations of higher quality, which is closely-linked with our idea of improving size-generalization by using compact hyperbolic embeddings.

3. Hyperbolic Graph Neural Networks

In this section, we briefly explain the main concepts of HGNNs, following the definitions and the notation from [11]. We introduce the notions necessary to understand our experiments by focusing on the differences with respect to vanilla GNNs. That being said, we note that many technical details are omitted due to space restrictions. Please refer to [14] for a comprehensive overview of current HGNN architectures.

3.1. Graph Convolutional Networks

GNNs based on neural message passing operate by exchanging and updating vector messages between nodes using neural networks [9]. We present the original GCN architecture introduced in [10], following the notation from [11]. In GCNs, the message from node v to its neighbor u is computed as $\boldsymbol{m}_v^{k+1} = \boldsymbol{W}^k \tilde{\boldsymbol{A}}_{uv} \boldsymbol{h}_v^k$, where $\boldsymbol{h}_v^k$ is the embedding of node v at layer k, $\boldsymbol{W}^k \in \mathbb{R}^{h \times h}$ represents the trainable weight matrix for layer k, and $\tilde{\boldsymbol{A}}$ is the normalized adjacency matrix. The normalized adjacency matrix $\tilde{\boldsymbol{A}} = \boldsymbol{D}^{-\frac{1}{2}}(\boldsymbol{A} + \boldsymbol{I})\boldsymbol{D}^{-\frac{1}{2}}$ is used to incorporate the graph structure, where $\boldsymbol{A}$ is the adjacency matrix of the graph, $\boldsymbol{I}$ is the identity matrix (for adding self-loops), and $\boldsymbol{D}$ is the diagonal degree matrix. The final updated representation for node u at layer $(k+1)$ is then computed as:

$$\boldsymbol{h}_u^{k+1} = \sigma\left(\sum_{v \in \mathcal{N}(u)} \tilde{\boldsymbol{A}}_{uv} \boldsymbol{W}^k \boldsymbol{h}_v^k\right), \qquad (1)$$

where σ is a non-linear activation function (e.g., ReLU), and $\mathcal{N}(u)$ denotes the set of neighboring nodes of u.

3.2. Riemannian Manifolds and Mapping Functions

A Riemannian manifold $(\mathcal{M}, g)$ is a real and smooth manifold equipped with a Riemmanian metric g that allows us to define the geometric properties of a space [11]. We compare the Euclidean space (zero curvature) to two different hyperbolic manifolds (negative curvature), i.e., the Poincaré ball model and the Lorentz model. In the Euclidean space, the shortest path between two points on a manifold (called a geodesic) represents a straight line calculated as $d(\mathbf{x}, \mathbf{y}) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$. The mentioned hyperbolic models measure distances differently and define more complex versions of the Riemannian metric. Please refer to [11] for details.

As we can see in Equation (1), GCN performs a series of linear transformations and pointwise non-linearities. While these operations are simple to define in the Euclidean space, their non-Euclidean counterparts can be more complex. For this reason, we will execute the functions with trainable parameters on a tangent space of a point on the Riemannian manifold, which is always Euclidean. We define two mapping functions that facilitate this [11]:

- **Logarithmic Map** ($\log_{x'}$): This map projects points from the manifold $\mathcal{M}$ to the tangent space $T_{x'}\mathcal{M}$ at a point $x' \in \mathcal{M}$. The logarithmic map translates the curved geometry into a locally flat, Euclidean-like space, making it possible to apply the necessary transformations.

- **Exponential Map** ($\exp_{x'}$): This map takes points from the tangent space $T_{x'}\mathcal{M}$ and projects them back onto the manifold $\mathcal{M}$. After applying linear transformations in the tangent space, the exponential map ensures that the updated data is returned to the original curved space, preserving the manifold's geometry.

3.3. Hyperbolic Graph Neural Networks

Since we have defined the mappings between the manifold and the tangent space in Section 3.2, we can now generalize the GNNs to operate on Riemannian manifolds, making them agnostic to the underlying space. This leads to Hyperbolic Graph Neural Networks (HGNNs), which extend classical GNNs to hyperbolic space, allowing them to better capture hierarchical structures. In contrast to classical GNNs, which perform message passing directly in Euclidean space, HGNNs operate on a curved hyperbolic manifold. The message passing in HGNNs is defined as [11]:

$$\boldsymbol{h}_u^{k+1} = \sigma\left(\exp_{x'}\left(\sum_{v \in \mathcal{N}(u)} \tilde{\boldsymbol{A}}_{uv} \boldsymbol{W}^k \log_{x'}(\boldsymbol{h}_v^k)\right)\right), \quad (2)$$

where we keep the notation from (1) adding the previously defined mappings. The main difference between HGNNs and vanilla GNNs lies in the use of these mapping functions to translate between the hyperbolic manifold and the

Euclidean tangent space. The logarithmic map moves node features to the locally flat tangent space, allowing linear transformations to be performed. Afterwards, the exponential map projects the transformed features back onto the hyperbolic manifold, preserving the manifold's geometry.

4. Datasets

In our experiments, we use both the synthetic and real-world datasets. For the synthetic dataset generation, we follow [11]. Real-world datasets are taken from the TUDataset library [12], following [11, 3].

4.1. Synthetic Graphs

The authors of [11] design a synthetic experiment to test whether hyperbolic embeddings capture structural information better than their Euclidean counterpart. The task is to classify graphs based on the underlying generation algorithm. They chose three algorithms: Erdős-Rényi [7], Barabási-Albert [1], and Watts-Strogatz [13]. Synthetic generation allows us to control the amount of structural information inserted into the graphs.

The graphs are generated as follows. For each graph generation algorithm, a number of nodes is uniformly sampled between 100 and 500, after which the algorithm is applied to construct the graph. In the case of Barabási-Albert graphs, the number of edges connecting a new node to existing nodes is randomly chosen between 1 and 100. For Erdős-Rényi graphs, the edge creation probability is set between 0.1 and 1. For Watts-Strogatz graphs, each node is connected to between 1 and 100 nearest neighbors in the ring topology, with the edge rewiring probability set between 0.1 and 1.

The total number of generated graphs is $18,000$ with each of the datasets (i.e., training, validation, and test) containing 6000 instances. The classes are perfectly balanced, i.e., we have 2000 graphs per class in each of the datasets. Finally, the original split from [11] randomly distributes the graph sizes between training, validation, and test datasets.

4.2. TUDatasets

We experiment on two of the datasets used in [3], i.e., PROTEINS and DD. In both datasets, each graph represents a protein, with nodes as amino acids and edges denoting chemical interactions. While the task is similar for both datasets, i.e., a binary classification into one of the two structural classes, the DD dataset consists of larger and more complex graphs. The dataset statistics are reported in Table 1 and Table 2.

	All	**Smallest 50%**	**Largest 10%**
Class A	59.56%	41.97%	90.17%
Class B	40.43%	58.02%	9.82%
Num of graphs	1113	567	112
Avg graph size	39	15	138

Table 1. PROTEINS dataset statistics.

	All	**Smallest 50%**	**Largest 10%**
Class A	58.65%	35.47%	79.66%
Class B	41.34%	64.52%	20.33%
Num of graphs	1178	592	118
Avg graph size	284	144	746

Table 2. DD dataset statistics.

5. Method

5.1. Size-Generalization Split

In order to test our hypothesis that hyperbolic embeddings can provide additional benefits in the size-generalization setting, we use dataset splits specifically designed for such scenarios. The authors of [11] randomly split the synthetically generated graphs between the training and the test set in terms of the number of nodes. We generate datasets with the size-generalization split and compare the relative improvements of using hyperbolic embeddings in both settings. That is, we train on graphs with $100-200$ nodes, while the test sets will have several ranges to analyze different levels of scale increase. In particular, we test on the following ranges for the number of nodes: $200-400, 400-500$, and $100-500$. The first two ranges aim to test the influence of hyperbolic node embeddings when we have 2 and 3 times average size increase, respectively. The third range (i.e., $100-500$) reuses the original test dataset graph sizes to analyze the performance degradation of using smaller training graphs. The number of graphs in the datasets and all the other hyperparameters are exactly the same as in the random split to ensure that the only difference lies in graph sizes.

For the TUDatasets [12], we compare the random split with the size-generalization split where we train on the 50% smallest graphs and test on the 10% largest graphs, as in [3]. Please note that this generalization split uses only 60% of the datasets. To ensure fair comparison, we use the 60% of all the graphs in the dataset for the random split as well.

5.2. Implementation

The original implementation[1] provided by the authors of [11] contains a few issues regarding efficiency, reproducibility, and consistency with the information in the pa-

[1]https://github.com/facebookresearch/hgnn.git

per. Most importantly, the authors do not report the architecture hyperparameters in the paper, while they differ for the hyperbolic and Euclidean manifolds in the code. As our goal is to test only the influence of different manifolds, we want to maintain all the other architectural components exactly the same. For this reason, we use the updated implementation [2] that addresses the concerns we have noticed. Throughout this work, we are using our reproduced results from the updated implementation as the baseline. We build on top of the updated code base by adding our data loading pipelines using PyTorch Geometric library [8]. Our code is available here.

6. Experiments

6.1. Experimental Setup

We test the influence of hyperbolic node embeddings by constructing exactly the same architectures except for the manifold choice, i.e., we vary between Euclidean, Poincaré, and Lorentz manifolds. Following the authors of [11], we also vary the node embedding sizes between $3, 5, 10, 20$, and 256, which is especially important considering our size-generalization analysis. For the synthetic graph dataset, we use the F1 score as the metric following [11]. On the other hand, we use the Matthews correlation coefficient (MCC) for the TUDatasets, which is more reliable in imbalanced settings caused by the size-generalization split [3, 5]. We also report the variance of the MCC for 10 independent runs for the TUDatasets, while we run only once for the synthetic graphs due to the dataset size. We train and test our models for both the random and the size-generalization split to analyze relative differences of using hyperbolic embeddings.

6.2. Synthetic Dataset Results

Figure 1 depicts the results of our synthetic dataset experiments. We group the models by embedding sizes and compare the performance of different manifolds. The upper left corner, i.e., a random split in terms of graph sizes, represents the reproduced results from the updated implementation. Our results match the ones reported by the authors, i.e., there are slight differences due to the fact that we generated datasets from scratch (they do not provide their dataset), but the results exhibit very similar patterns confirming that we properly conducted the experiments. In the random split scenario, our results do not indicate benefits of using the hyperbolic embeddings. In fact, the performance slightly varies between different manifolds, but we do not see clear patterns of improvement.

On the other hand, our size-generalization split results show the ability of hyperbolic embeddings to obtain high-quality low-dimensional representations. Focusing on the two lowest embedding dimensions (i.e., 3 and 5), we can see a clear pattern of improvement using hyperbolic embeddings. In fact, the 5-dimensional Poincaré embeddings achieve the best performance in all of the 3 generalization settings. On the other hand, in higher-dimensional spaces, hyperbolic embeddings do not perform better than their Euclidean counterparts. These results indicate the benefits of compact hyperbolic representations in terms of size generalization even for the dataset where they do not improve the performance in the random split setting.

To further emphasize the benefits of using hyperbolic embeddings in the size-generalization setting, we show Figure 2. It depicts the absolute decrease in F1 score when we lower the training graph sizes from $100 - 500$ nodes to $100 - 200$ nodes. Given that the test set remains exactly the same, this analysis shows how the performance of each manifold degrades when we decrease the size of the training graphs. As we can see, the low-dimensional hyperbolic embeddings (3 and 5) have the lowest absolute decrease in the F1 score, indicating that they maintain the performance significantly better than other architectural settings. That being said, we have to take into account that synthetic experiments are ran only once, which means that some of the improvements could be due to variance. However, comparison with the authors results (updated implementation) leads to similar patterns (not reported due to space restrictions). The gaps for the low-dimensional embeddings are very large in both cases, making us confident that the differences would remain significant if we were to average the results over multiple runs.

6.3. TUDatasets Results

As already mentioned, we perform 10 independent runs for the TUDatasets [12] experiments, carefully setting the same seeds for the random and size-generalization settings to ensure fair comparison. We report the average MCC across all runs along with the standard deviation.

6.3.1 PROTEINS Dataset

Results for the PROTEINS dataset are reported in Table 3. We group the results by the manifold, embedding size, and the split ("Random" or "Size-Gen"). Observing the value for the node embedding sizes of 3 and 5, we notice similar patterns as for the synthetic graph dataset. In the random split scenario, we can see slight improvements using hyperbolic embeddings, while they get more prominent in the size-generalization setting. In fact, focusing on the dimensionality 5, we see a very strong advantage of using the Poincaré manifold. This architecture has the best overall performance for both settings, but the relative improvement with respect to the best Euclidean architecture is more

[2] https://github.com/rubenwiersma/hyperbolicgnn.git

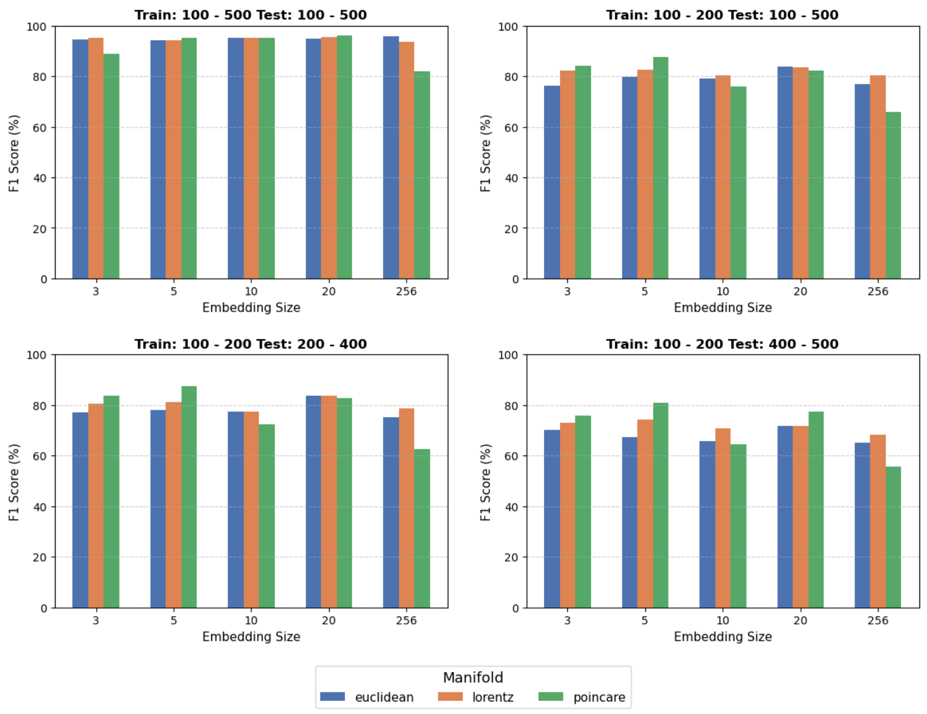

Figure 1. F1 score for the synthetic dataset experiments grouped my manifolds and node embeddings sizes. In the random-size split, we do not obtain improvements using hyperbolic embeddings. On the other hand, we see the patterns of better performance for the low-dimensional hyperbolic embeddings (3 and 5) in all the size-generalization scenarios.

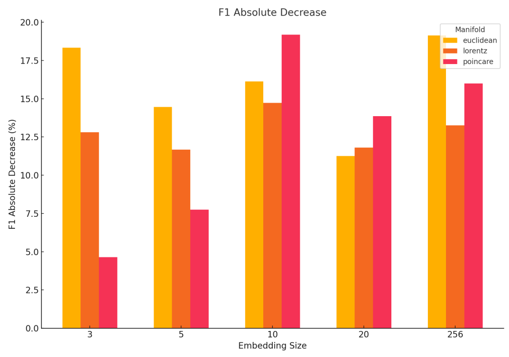

Figure 2. Absolute decrease in the F1 score between the models trained on graphs with $100 - 500$ nodes and the models trained on graphs with $100 - 200$ nodes. The test dataset and all the other hyperparameters are kept the same to ensure fair comparison. Low-dimensional hyperbolic embeddings (3 and 5) are the best at maintaining the performance when the training graph sizes are decreased.

significant in the size-generalization scenario. The values of top-performing Euclidean and hyperbolic manifolds (considering them together) are bolded to emphasize this difference. Finally, similarly to the synthetic dataset, hyperbolic embeddings usually do not provide benefits for large node embedding sizes (except for the embedding size of 20 in the size-generalization setting).

Additionally, we compare our results to SizeShiftReg [3] to confirm the relevance of our analysis. To be clear, the goal is not to directly compare the performances because the GCN architecture hyperparameters differ. Instead, we emphasize that our conclusions are not drawn from the analysis with trivial results, but with comparable values to recent papers. The GCN architecture from [3] obtains the average MCC of 0.21 for the PROTEINS dataset. Applying their method, the average MCC increases to 0.29. Looking at Table 3, we see that our best Euclidean GCN obtains the value of 0.22, while we improve it to 0.30 by changing the manifold. Again, a fair comparison would require further experimentation, but similar improvements with a different method support the significance of our results.

6.3.2 DD Dataset

Results for the DD dataset experiments are reported in Table 4. Again, we bold the top-performing Euclidean and hyperbolic architectures in both settings. The first thing we notice is the absence of the previously encountered pattern for the small embeddings sizes (i.e., 3 and 5). For the DD dataset, hyperbolic embeddings are not relatively better in the size-generalization scenario. In fact, they do not show clear improvement in either case, being especially

Dimensionality	**3**	**5**	**10**	**20**	**256**
Euclidean (Random)	0.33 ± 0.19	**0.38** ± 0.09	0.33 ± 0.06	**0.38** ± 0.10	0.19 ± 0.14
Lorentz (Random)	0.39 ± 0.10	0.39 ± 0.09	0.39 ± 0.09	0.29 ± 0.13	0.17 ± 0.18
Poincaré (Random)	0.40 ± 0.09	**0.41** ± 0.08	0.37 ± 0.09	0.34 ± 0.10	0.20 ± 0.17
Euclidean (Size-Gen)	0.15 ± 0.18	0.09 ± 0.18	**0.22** ± 0.19	0.08 ± 0.18	0.07 ± 0.14
Lorentz (Size-Gen)	0.22 ± 0.18	0.09 ± 0.16	0.20 ± 0.19	0.15 ± 0.22	0.06 ± 0.13
Poincaré (Size-Gen)	0.26 ± 0.19	**0.30** ± 0.19	0.20 ± 0.14	0.20 ± 0.14	0.09 ± 0.16

Table 3. MCC (mean and standard deviation) for the PROTEINS dataset, comparing random split and size-generalization split across manifolds and embedding dimensions. The best values for the hyperbolic (both of the models considered together) and the Euclidean architecture are bolded for both splits, separately.

Dimensionality	**3**	**5**	**10**	**20**	**256**
Euclidean (Random)	0.35 ± 0.05	0.33 ± 0.12	0.35 ± 0.08	**0.36** ± 0.08	0.31 ± 0.09
Lorentz (Random)	0.25 ± 0.14	**0.37** ± 0.07	0.35 ± 0.08	0.35 ± 0.09	0.23 ± 0.14
Poincaré (Random)	0.29 ± 0.08	0.34 ± 0.09	0.31 ± 0.13	0.27 ± 0.07	0.09 ± 0.14
Euclidean (Size-Gen)	0.22 ± 0.08	0.22 ± 0.08	**0.23** ± 0.07	0.17 ± 0.10	0.07 ± 0.10
Lorentz (Size-Gen)	0.07 ± 0.11	0.15 ± 0.11	**0.21** ± 0.09	0.18 ± 0.05	0.12 ± 0.11
Poincaré (Size-Gen)	0.15 ± 0.12	0.15 ± 0.10	0.19 ± 0.12	0.13 ± 0.16	-0.01 ± 0.02

Table 4. MCC (mean and standard deviation) for the DD dataset, comparing random split and size-generalization split across manifolds and embedding dimensions. The best values for the hyperbolic (both of the models considered together) and the Euclidean architecture are bolded for both splits, separately.

bad in the size-generalization setting in comparison to the other two datasets. Looking at the bolded table values, we see that hyperbolic embeddings still obtain comparable performance to the Euclidean manifold, but the results indicate that our hypothesis is not true for the DD dataset. Understanding why hyperbolic embeddings do not bring improvements in this case requires analysis out of the scope of this work. We discuss the dataset choice in Section 7.

6.4. Runtime Overhead

The authors of [11] report relatively minor computational overhead of using hyperbolic node embeddings (below 1.1 times increase with respect to Euclidean manifolds for the datasets of similar size as in our experiments). As we already described, the only architectural differences are in the logarithmic and exponential maps which require computationally cheap operations. Due to space restrictions, we do not report the runtime analysis, but our measurements confirm that we observe a similar overhead as reported by the authors [11]. Moreover, the ability of hyperbolic manifolds to obtain high-quality, low-dimensional embeddings can provide relative runtime decrease in certain cases.

7. Conclusion

We conducted carefully designed experiments to analyze the influence of hyperbolic node embeddings in the size-generalization setting. Our results are partially successful, but we believe that they show the potential for a new application of HGNNs. We emphasize that we used one of the simplest hyperbolic architectures, i.e., the only difference with respect to a vanilla GNN is in the use of logarithmic and exponential maps. This was done on purpose to marginally test the influence of the hyperbolic space, but our work can be directly extended by incorporating novel HGNN architectures. Moreover, the real-world datasets in our experiments were not chosen specifically to accommodate the benefits of hyperbolic embeddings. This choice was made to put the emphasis on the comparison between the random split and the size-generalization split. Experimenting on datasets with specific features that compliment hyperbolic embeddings' properties presents another potential extension. Finally, we propose an adaptation of SizeShiftReg [3] as a direction of future work. This method deals with the same goal of identifying the underlying features that are relevant across graph sizes through high-quality graph representations. We laid the foundations for combining these two ideas by achieving similar effects with a different method, and we believe that their combination could lead to interesting results.

References

[1] Albert-László Barabási and Réka Albert. Emergence of scaling in random networks. *science*, 286(5439):509–512, 1999.

[2] Beatrice Bevilacqua, Yangze Zhou, and Bruno Ribeiro. Size-invariant graph representations for graph classification extrapolations. In *International Conference on Machine Learning*, pages 837–851. PMLR, 2021.

[3] Davide Buffelli, Pietro Liò, and Fabio Vandin. Sizeshiftreg: a regularization method for improving size-generalization in graph neural networks. *Advances in Neural Information Processing Systems*, 35:31871–31885, 2022.

[4] Ines Chami, Zhitao Ying, Christopher Ré, and Jure Leskovec. Hyperbolic graph convolutional neural networks. *Advances in neural information processing systems*, 32, 2019.

[5] Davide Chicco and Giuseppe Jurman. The advantages of the matthews correlation coefficient (mcc) over f1 score and accuracy in binary classification evaluation. *BMC genomics*, 21:1–13, 2020.

[6] Jindou Dai, Yuwei Wu, Zhi Gao, and Yunde Jia. A hyperbolic-to-hyperbolic graph convolutional network. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 154–163, 2021.

[7] Paul Erdos, Alfréd Rényi, et al. On the evolution of random graphs. *Publ. math. inst. hung. acad. sci*, 5(1):17–60, 1960.

[8] Matthias Fey and Jan Eric Lenssen. Fast graph representation learning with pytorch geometric. *arXiv preprint arXiv:1903.02428*, 2019.

[9] William L Hamilton. *Graph representation learning*. Morgan & Claypool Publishers, 2020.

[10] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

[11] Qi Liu, Maximilian Nickel, and Douwe Kiela. Hyperbolic graph neural networks. *Advances in neural information processing systems*, 32, 2019.

[12] Christopher Morris, Nils M Kriege, Franka Bause, Kristian Kersting, Petra Mutzel, and Marion Neumann. Tudataset: A collection of benchmark datasets for learning with graphs. *arXiv preprint arXiv:2007.08663*, 2020.

[13] Duncan J Watts and Steven H Strogatz. Collective dynamics of ‘small-world’networks. *nature*, 393(6684):440–442, 1998.

[14] Menglin Yang, Min Zhou, Zhihao Li, Jiahong Liu, Lujia Pan, Hui Xiong, and Irwin King. Hyperbolic graph neural networks: A review of methods and applications. *arXiv preprint arXiv:2202.13852*, 2022.

[15] Gilad Yehudai, Ethan Fetaya, Eli Meirom, Gal Chechik, and Haggai Maron. From local structures to size generalization in graph neural networks. In *International Conference on Machine Learning*, pages 11975–11986. PMLR, 2021.