Weighted Embeddings for Low-Dimensional Graph Representation

Thomas Bläsius, Jean-Pierre von der Heydt, Maximilian Katzmann, Nikolai Maas

Karlsruhe Institute of Technology (KIT), Karlsruhe, Germany

{thomas.blaesius, heydt, maximilian.katzmann, nikolai.maas}@kit.edu

Abstract

Learning low-dimensional numerical representations from symbolic data, e.g., embedding the nodes of a graph into a geometric space, is an important concept in machine learning. While embedding into Euclidean space is common, recent observations indicate that hyperbolic geometry is better suited to represent hierarchical information and heterogeneous data (e.g., graphs with a scale-free degree distribution). Despite their potential for more accurate representations, hyperbolic embeddings also have downsides like being more difficult to compute and harder to use in downstream tasks.

We propose embedding into a *weighted space*, which is closely related to hyperbolic geometry but mathematically simpler. We provide the embedding algorithm WEMBED and demonstrate, based on generated as well as over 2000 real-world graphs, that our weighted embeddings heavily outperform state-of-the-art Euclidean embeddings for heterogeneous graphs while using fewer dimensions. The running time of WEMBED and embedding quality for the remaining instances is on par with state-of-the-art Euclidean embedders.

Introduction

Embeddings are a vital concept in machine learning. They map complex objects like high-dimensional vectors (e.g., images) or symbolic entities (e.g., words of a language or nodes of a graph) into a low-dimensional vector space. Downstream tasks utilize these low-dimensional representations by perceiving objects as similar if their vector representations are similar. A natural way to measure the similarity of two vectors $x, y \in \mathbb{R}^d$ is to interpret them as points in Euclidean space and use their distance $\|x-y\|$. A common alternative is the dot product $x \cdot y$, which is closely related to spherical geometry. However, inspired by findings of the network-science community (Krioukov et al. 2010), it has been observed that some types of graphs are better represented by embedding their nodes into hyperbolic geometry (Nickel and Kiela 2017; Chamberlain, Clough, and Deisenroth 2017).

Benefits of Hyperbolic Geometry Hyperbolic space expands exponentially, i.e., the area of a disk of radius r is in $\Theta(e^r)$ for growing r. This makes it possible to embed trees or similar hierarchical structures with low distortion into the 2-dimensional hyperbolic plane (Sarkar 2011). In comparison, when embedding, e.g., a complete binary tree in the Euclidean plane, one quickly runs out of space as the number of nodes grows exponentially with the layer while the area only grows quadratically. Moreover, using hyperbolic distances as similarity measure allows nodes to be similar to many other nodes that are themselves pairwise dissimilar. This enables low-dimensional representations of heterogeneous graphs.[1] This in particular includes the prevalent so-called scale-free networks, where the degree distribution follows a power-law (Krioukov et al. 2010; Voitalov et al. 2019).

Given these benefits of hyperbolic space when it comes to representing hierarchical or heterogeneous data, it is not

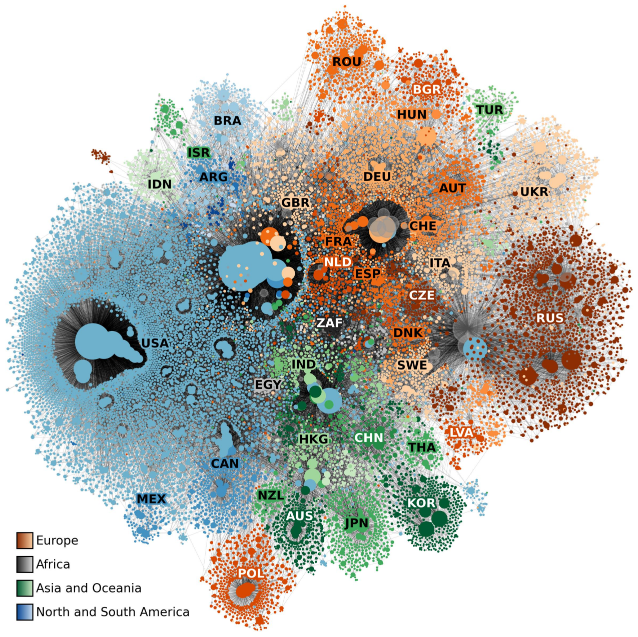

Figure 1: A 2-dimensional weighted embedding of the internet graph used by Boguñá, Papadopoulos, and Krioukov (2010). The node size indicates the weight.

[1]The term *heterogneous graph* is sometimes also used for graphs with heterogeneous labels. In this paper, graphs are always unlabeled and heterogeneity refers to the degree distribution.

surprising that hyperbolic embeddings have gained popularity in recent years. Although there have been earlier works on embedding graphs in the 2-dimensional hyperbolic plane (Boguñá, Papadopoulos, and Krioukov 2010; Papadopoulos, Psomas, and Krioukov 2015; Papadopoulos, Aldecoa, and Krioukov 2015; Wang et al. 2016; Bläsius et al. 2016), the research in the artificial intelligence community has gained traction with the seminal paper by Nickel and Kiela (2017), who demonstrated that hyperbolic embeddings can outperform their Euclidean counterpart for graph reconstruction and link prediction tasks, particularly for low-dimensional embeddings ($d \in \{10, 20\}$). This result has been extended to and reproduced for, e.g., other coordinate systems in hyperbolic space (Nickel and Kiela 2018; Law et al. 2019), to directed acyclic graphs (Ganea, Bécigneul, and Hofmann 2018; Suzuki, Takahama, and Onoda 2019), or to representing general metrics (Sala et al. 2018).

We briefly want to mention that hyperbolic geometry has also been incorporated into neural networks (Ganea, Becigneul, and Hofmann 2018; Shimizu, Mukuta, and Harada 2021) and graph neural networks (Liu, Nickel, and Kiela 2019; Chami et al. 2019), and has been used for other domains such as natural language processing (Tifrea, Bécigneul, and Ganea 2019), or in computer vision (Khrulkov et al. 2020). For an overview, see the excellent survey by Peng et al. (2022) or the collection by Yang and Zhou (2024).

Challenges with Hyperbolic Geometry There are, however, also downsides to using hyperbolic embeddings. Firstly, hyperbolic geometry is not very accessible and adjusting down-stream tasks to properly interpret hyperbolic coordinates requires quite some mathematical machinery, e.g., (Ganea, Becigneul, and Hofmann 2018; Shimizu, Mukuta, and Harada 2021). There is an effort to mitigate these difficulties, e.g., via tutorials (Choudhary et al. 2022; Zhou et al. 2023), or by providing easy-to-use libraries (Vyas et al. 2022; van Spengler, Wirth, and Mettes 2023). However, hyperbolic embeddings remain more difficult to use and reason about, in particular for someone not familiar with hyperbolic geometry.

Another issue is that finding good hyperbolic embeddings is algorithmically challenging for multiple reasons. To explain this, we note that in hyperbolic embeddings, the distance of a node to the origin encodes its centrality in the sense that central nodes have small hyperbolic distance to many other nodes. Additionally, hyperbolic space behaves like Euclidean space close to the origin and its hyperbolic nature becomes more pronounced further out. This makes it crucial to choose the right scale for the embedding, which is in stark contrast to Euclidean space where one can scale an embedding without altering relative distances.

As geodesics between points "bend" towards the origin, moving a vertex towards a distant neighbor mainly makes it more central, which leads to loss functions with spikes and suboptimal plateaus, making gradient descend less effective (Bläsius, Friedrich, and Katzmann 2021). Existing embedding methods mitigate this issue, e.g., by having a burn-in phase in which an embedding is computed close to the origin before shifting vertices outwards (Nickel and Kiela 2017), by first embedding only a dense core of the graph close to the origin (Boguñá, Papadopoulos, and Krioukov 2010; Bläsius et al. 2016), or by optimizing the distance from the origin separately (Bläsius, Friedrich, and Katzmann 2021). In their survey, Peng et al. (2022) also point out that different optimizations for stochastic gradient descend are "designed to optimize parameters living in Euclidean space". Additionally, common techniques to reduce the running time, like negative sampling or geometric data structures for low-dimensional embeddings, are less effective or more complicated in hyperbolic space (Suzuki et al. 2021; Kisfaludi-Bak and van Wordragen 2024). Moreover, there are numerical difficulties depending on the used coordinate system (Nickel and Kiela 2018; Law et al. 2019; Mishne et al. 2023).

We believe that these challenges pose a real cost that counteracts the superiority of hyperbolic geometry in terms of its representational power. In fact, in an ideal world without these challenges, there is no reason to favor Euclidean over hyperbolic geometry as a sufficiently small ball in hyperbolic space behaves (almost) equivalent to Euclidean space. Nonetheless, the consensus in the literature is that hyperbolic embeddings are not always superior but mostly for low dimensions and hierarchical or heterogeneous data[2] (Peng et al. 2022). Moreover, one additionally has to account for the fact that the comparison is usually done with some ad-hoc Euclidean embedding method. This yields a fair comparison for evaluating the benefits of hyperbolic geometry in isolation. However, we believe that we are now at a stage where it is crucial to demonstrate usefulness compared to state-of-the-art Euclidean embedders like DEEPWALK (Perozzi, Al-Rfou, and Skiena 2014), VERSE (Tsitsulin et al. 2018), and FORCE2VEC (Rahman, Sujon, and Azad 2020). Besides the embedding quality, we believe that running time and ease of use are also important factors where hyperbolic embedding algorithms currently fall behind.

Our Contribution Hyperbolic embeddings have been inspired by insights form the network-science community on hyperbolic random graphs (Krioukov et al. 2010). More recent results by Bringmann, Keusch, and Lengler (2019) show that hyperbolic random graphs are in some sense equivalent to random graphs from a weighted geometric space. Motivated by this, we propose *weighted embeddings*, which are defined as follows. Each node v is mapped to a *position* $p_v \in \mathbb{R}^d$ and additionally to a *weight* $w_v \in \mathbb{R}^+$. As a similarity measure between vertices u and v, we use the *weighted distance*

$$\operatorname{dist}(u, v) = \frac{\|p_u - p_v\|}{(w_u w_v)^{1/d}}, \qquad (1)$$

where $\|p_u - p_v\|$ is the Euclidean distance between p_u and p_v. Although these weighted coordinates are not exactly equivalent to hyperbolic coordinates, Bringmann, Keusch, and Lengler (2019) provide an approximate translation between the two, where higher weights correspond to more central nodes closer to the origin.

[2]Although this seems to be the general consensus, there is, to the best of our knowledge, no extensive study on what type of graphs benefit from hyperbolic geometry. Most evaluations are done on just a handful of graphs.

Thus, using weighted distances lets us reap the benefits of hyperbolic space while eliminating some of the challenges coming with it. Clearly, weighted distances are mathematically very simple, which makes weighted embeddings easier to reason about or to incorporate into downstream tasks. It is obvious that using uniform weights yields the Euclidean setting as special case. Moreover, in contrast to hyperbolic geometry, one can scale a weighted embedding arbitrarily, which eliminates the algorithmic difficulty of having to choose the right scale. Also, optimizing the positions independent from the weights leads to more well-behaved gradients.

Our main contribution is the WEMBED algorithm that we engineered to efficiently compute high-quality weighted embeddings. It works out of the box without any graph-specific parameter tuning, outperforms POINCARÉEMBED (Nickel and Kiela 2017) by orders of magnitude, and is not much slower than the state-of-the-art Euclidean embedding algorithms DEEPWALK (Perozzi, Al-Rfou, and Skiena 2014), VERSE (Tsitsulin et al. 2018), and FORCE2VEC (Rahman, Sujon, and Azad 2020). In regards to embedding quality, WEMBED heavily outperforms its Euclidean competitors for low-dimensional embeddings of heterogeneous graphs. For higher dimensions or less heterogeneous networks, WEMBED produces comparable results. To give one example, for the commonly used `citeseer` network, WEMBED computes an 8-dimensional embedding in under $11\,\mathrm{s}$, which allows almost perfect reconstruction of its edges with an F1 score of 0.9999. VERSE, the strongest Euclidean competitor, achieves the scores 0.64, 0.977, and 0.992, in 8, 32, and 128 dimensions, respectively, with slightly higher running times.

Besides evaluating WEMBED on commonly used networks, we run experiments on generated graphs to evaluate its scaling behavior, and to gain insights into how weighted and Euclidean embeddings compare depending on the heterogeneity of the graph. Moreover, we evaluate WEMBED and its Euclidean competitors on a set of over 2000 real-world graphs, showing that WEMBED consistently produces high-quality embeddings. WEMBED is outperformed only slightly on few graphs but produces substantially better results on many graphs, particularly those with high heterogeneity. This demonstrates that WEMBED is often the best and generally a good choice for computing low-dimensional representations.

Our core contributions can be summarized as follows.

- We propose weighted embeddings as a way to get the benefits from hyperbolic embeddings without the mathematical and algorithmic challenges coming with it.
- We engineer the efficient embedding algorithm WEMBED and demonstrate that it yields high-quality embeddings.
- We evaluate WEMBED and different state-of-the-art Euclidean embedding algorithms on a large set of graphs to see how heterogeneity affects the embedding quality.

Embedding Method

A graph $G = (V, E)$ can be perfectly reconstructed from an embedding if there is a threshold ℓ such that edges have length at most ℓ while non-adjacent vertices have larger distances. This is formalized by the following *threshold property*.

$$\forall u, v \in V\colon\ \mathrm{dist}(u, v) \le \ell \Leftrightarrow uv \in E \tag{2}$$

Achieving this is a difficult task. In fact, it is NP-hard (even $\exists\mathbb{R}$-complete) to determine whether a given graph can be embedded in the 2-dimensional Euclidean or hyperbolic plane (Breu and Kirkpatrick 1998; Schaefer 2010; Bieker et al. 2023). It is also NP-hard when using the dot product[3] (Kang and Müller 2012). Nonetheless, we can use the threshold property as a guiding principle by minimizing a loss function $\mathcal{L}$ that achieves its minimum for embeddings that satisfying it. In the following, we discuss our choice for $\mathcal{L}$, describe how we minimize it (mostly using gradient descent), and introduce a geometric data structure for the weighted space that speeds up the loss function evaluation.

Loss Function We note that directly interpreting Equation (2) as a minimization problem, i.e., trying to minimize the number of vertex pairs violating it, would yield a piecewise constant loss function, which makes using gradient descent infeasible. Instead, we define a differentiable loss function $\mathcal{L}$ consisting of an attracting component $\mathcal{L}_{\mathrm{attr}}$ that punishes too distance neighbors and a repelling component $\mathcal{L}_{\mathrm{rep}}$ that punishes too close non-neighbors. Formally, we define the loss $\mathcal{L}$ of an embedding as

$$\mathcal{L} = \sum_{uv \in E} \mathcal{L}_{\mathrm{attr}}(\mathrm{dist}(u, v)) + \sum_{uv \notin E} \mathcal{L}_{\mathrm{rep}}(\mathrm{dist}(u, v)). \tag{3}$$

We interpret the gradient of $\mathcal{L}_{\mathrm{attr}}$ as attracting forces, which decreases the distance between neighboring nodes, while the gradient of $\mathcal{L}_{\mathrm{rep}}$ is a repelling force between non-neighbors. We call them f_{attr} and f_{rep}, respectively.

In this general framework, there are two degrees of freedom to fill: the distance functions dist and choices for the loss function $\mathcal{L}_{\mathrm{attr}}$ and $\mathcal{L}_{\mathrm{rep}}$. The former is closely related to the embedding space and, as already discussed in the introduction, we use the weighted distances defined in Equation (1).

For the loss function, we considered various options that have been used in the literature; see Table 1 and Figure 2. The minimum of $\mathcal{L}_{\mathrm{attr}}(x) + \mathcal{L}_{\mathrm{rep}}(x)$ corresponds to the threshold distance ℓ in Equation (2). Based on preliminary experiments, we decided for the *linear loss function*. Together with the very similar sigmoid log-likelihood, it showed the best performance. Moreover, its simplicity has algorithmic advantages. In particular, we can use the fact there are no forces between non-adjacent vertices with distance above the threshold ℓ.

Minimizing the Loss To every vertex $v \in V$, we need to assign a weight $w_v \in \mathbb{R}$ and a position $p_v \in \mathbb{R}^d$. WEMBED first fixes the weights using a simple degree-based estimation and then optimizes the positions using gradient descend.

We set the weights to $w_v = \deg(v)^{d/8}$. Without going too much into detail, the reasoning for this is roughly as follows. Following Bringmann, Keusch, and Lengler (2019), it makes sense to set $w_v = \deg(v)$ when embedding a graph with d-dimensional latent geometry into d-dimensional space. However, for real-world networks, we do not know the true dimension, which leads to undesired effects in particular if the embedding dimension is too high. The exponent d/d' is a correction for embedding a graph that comes from d'

[3]For the dot product, higher value indicates stronger similarity, i.e., the inequality in Equation (2) has to be reversed.

Loss function	$\mathcal{L}_{\text{attr}}(x)$	$\mathcal{L}_{\text{rep}}(x)$	$f_{\text{attr}}(x)$	$f_{\text{rep}}(x)$
Fruchtermann & Reingold	$x^3/3\ell$	$-\ell^2\log(x)$	$x' \cdot x^2/\ell$	$-x' \cdot \ell^2/x$
Maxent Stress	$(x-\ell)^2/\ell^2$	$-\log(x)$	$x' \cdot 2(x-\ell)/\ell^2$	$-x' \cdot 1/x$
Sigmoid log-likelihood	$-\log(\sigma(\ell-x))$	$-\log(\sigma(x-\ell))$	$x' \cdot \sigma(x-\ell)$	$-x' \cdot \sigma(\ell-x)$
Linear	$\max(x-\ell,0)$	$\max(\ell-x,0)$	$x' \cdot \mathbf{1}_{x>\ell}$	$-x' \cdot \mathbf{1}_{x<\ell}$

Table 1: Different loss functions and their forces (Fruchterman and Reingold 1991; Gansner, Hu, and North 2013; Rahman, Sujon, and Azad 2020). The distance $\text{dist}(u, v)$ is abbreviated with x and x' is its gradient with respect to the embedding of u. The indicator function $\mathbf{1}_{x>\ell}$ is 1 if $x > \ell$ and 0 otherwise.

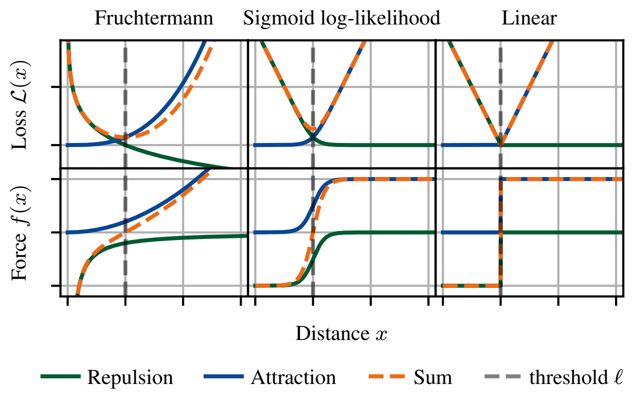

Figure 2: Different loss functions and resulting forces. Green lines correspond to $\mathcal{L}_{\text{rep}}$ and f_{rep}, blue lines to $\mathcal{L}_{\text{attr}}$ and f_{attr} and orange to their sum. The threshold ℓ is marked in gray.

dimensions into d dimensions. Clearly d' is a hyperparameter that depends on the input graph. However, $d' = 8$ robustly lead to good results in our experiments. This can be seen as an indication that the latent geometry underlying many real-world networks has quite low dimension.

After fixing the weights, WEmbed finds positions by starting with a random initialization and then minimizing the linear loss function using batch gradient descent. We apply Adam optimizer (Kingma and Ba 2015) with an exponentially decreasing learning rate to update the embedding. The optimization process ends, when the change in position is sufficiently small or a maximum number of iterations is reached.

To compute the gradient of $\mathcal{L}$ in every step, observe that the gradient is just the sum of attracting and repulsive forces contributed by every vertex pair. We describe the gradient for the position p_u of a fixed vertex u. For every edge $uv \in E$ with $\text{dist}(u, v) > \ell$, the derivative of $\text{dist}(u, v)$ with respect to p_u yields the vector $\frac{p_v - p_u}{\|p_u - p_v\|}(w_u w_v)^{-1/d}$, which pulls u towards v. Similarly, for every non-edge $uv \notin E$ with $\text{dist}(u, v) \leq \ell$, we get the repulsive force $\frac{p_u - p_v}{\|p_u - p_v\|}(w_u w_v)^{-1/d}$ pushing u away from v.

Note that naively computing the gradient involves iterating over all $\Theta(n^2)$ vertex pairs, which is infeasible for larger graphs. A common technique to reduce the time is negative sampling, i.e., consider all m edges for the attractive forces and sample $\mathcal{O}(m)$ non-edges for the repelling forces. WEmbed does *not* use negative sampling for two reasons. First, it resulted in substantially worse weighted embeddings. Secondly, as we aim for relatively low-dimensional embeddings, it is feasible to use a geometric data structure to find all non-neighbors at distance at most ℓ. As more distant non-neighbors are not relevant for the linear loss function, this lets us compute the gradient exactly.

Geometric Data Structure For a given vertex u, we want to efficiently find all vertices with $\text{dist}(u, v) \leq \ell$, i.e., with $\|p_v - p_u\| \leq \ell \cdot (w_u w_v)^{1/d}$. A geometric data structure allowing queries of the type $\|p_v - p_u\| \leq r$ with some radius r is the R-tree (Guttman 1984). Thus, if the weights were uniform, one could simply use an R-tree to find all relevant non-neighbors of u. However, the radius r depends on w_v, which requires querying different radii for vertices of different weight. To resolve this, we use a similar technique that has been used to generate graphs (Bringmann, Keusch, and Lengler 2019; Bläsius et al. 2022). We partition the nodes into classes with similar weights, creating one R-tree for each class, and query the different R-trees with different radii. More specifically, let $V_i = \{v \in V \mid 2^{i-1} \leq w_v < 2^i\}$. We create one R-tree T_i for each non-empty weight class V_i. Querying T_i with radius $r_i(u) = \ell \cdot (w_u 2^i)^{1/d}$ yields all vertices $v \in V_i$ with weighted distance at most ℓ to u, plus some false positives with weighted distance up to 2ℓ. Thus, the union over all R-trees gives a superset of the nodes with weighted distance ℓ to u, which includes all non-neighbors relevant for repulsive forces.

Experiments

We evaluate the performance of WEmbed in comparison to different other embedding methods. Additionally we investigate the impact of a graph's heterogeneity on the embedding quality and on the relevance of using a weighted space.

Setup

We compare WEmbed with the state-of-the-art Euclidean embedders DeepWalk (Perozzi, Al-Rfou, and Skiena 2014), Verse (Tsitsulin et al. 2018), and Force2Vec (Rahman, Sujon, and Azad 2020), as well as with the hyperbolic embedder PoincaréEmbed (Nickel and Kiela 2017). We used the default parameters for DeepWalk, sigmoid and negative sampling (option 6) for Force2Vec, and adjacency similarity (verse_neigh) for Verse. The default parameters for PoincaréEmbed did not yield good results and we used lr = 0.3, epochs = 300, and batchsize = 50. All experiments were run on a machine with two AMD EPYC Zen2

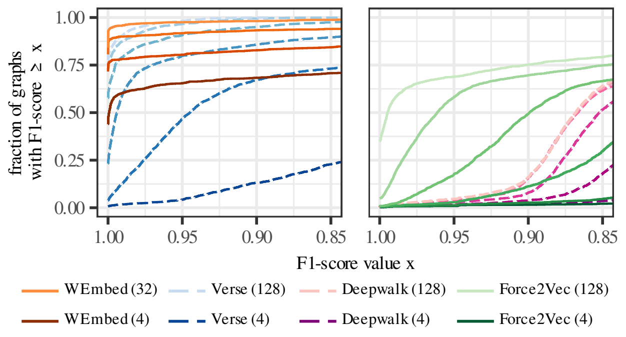

Figure 3: Embedding quality for the `real` data set. Each line represents the CDF of the F1-score (note the reversed x-axis capped at 0.85) for an embedder in a fixed dimension. Dimensions increase in powers of 2: $4, 8, 16, 32$ for WEMBED and additionally $64, 128$ for the others.

7742 (64 cores) with 2.25-3.35 GHz frequency, 1024GB RAM and 256MB L3 cache. To account for randomness, each data point is the average of five runs. To get comparable running times, computation is single-threaded. The only exception is the much slower POINCARÉEMBED, for which we compute only one embedding and use all 128 available cores. We use the R-tree implementation provided by Boost.

Data We use a set of real-world networks described by Bläsius and Fischbeck (2024) restricted to graphs with $n \leq 10\,\text{k}$ nodes and $m \leq 50\,\text{k}$ edges, which amounts to 2173 graphs. We refer to this data set as `real`. Moreover, we explicitly feature a set of 13 selected networks. This set is not disjoint to `real` but additionally contains some commonly considered larger networks. We also consider randomly generated graphs, which are described later. Each graph is considered as unlabeled, undirected, and unweighted, and is reduced to its largest connected component before embedding.

Graph Reconstruction We evaluate the embedding quality based on reconstruction accuracy. To make this precise, let t be some threshold and let $E(t)$ be the set of *predicted edges* with respect to t, i.e., all pairs of nodes with embedding distance at most t. The *precision* $\text{Prec}(t) := \frac{|E \cap E(t)|}{|E(t)|}$ is the fraction of predicted edges that are actual edges. The *recall* $\text{Rec}(t) := \frac{|E \cap E(t)|}{|E|}$ is the fraction of actual edges that are predicted. The *F1-score* is the harmonic mean of precision and recall where t is chosen to maximize the score. For large graphs we compute an approximated F1-score by sampling $10 \cdot m$ non-edges. We note that we also considered mean average precision as well as link prediction. We do not report these scores here as they do not provide additional insights.

Evaluation

The embedding quality on the `real` data set is shown as the cumulative distribution function (CDF) of the F1-scores in Figure 3. A line going through the point (x, y) indicates that a fraction of y instances has an F1-score of x or higher. For example, WEMBED (8) yields an F1-score very close to 1 for over 75 % of graphs. We can also observe the trend that a higher number of dimensions allows for better accuracy.

Comparing the embedders, VERSE clearly has the best accuracy among the Euclidean approaches. In the comparison to WEMBED, note that WEMBED uses at most 32 dimensions while we show results for VERSE using up to 128 dimensions. Comparing equal dimensions, WEMBED clearly dominates VERSE. Moreover, even WEMBED (32) is comparable with VERSE (128), with WEMBED (32) getting very high F1-scores for more graphs, while VERSE (128) excels for lower scores, i.e., on graphs that are difficult to embed.

Table 2 reports detailed results for 13 selected graph. The comparison with VERSE matches the previous observation: WEMBED outperforms VERSE in low dimensions, sometimes by a lot. Moreover, while VERSE excels for 128 dimensions on the difficult-to-embed graph `BlogCatalog`, WEMBED often achieves better results, even in fewer dimensions.

The hyperbolic embedder POINCARÉEMBED yields better accuracy than VERSE for $d = 8$, confirming the superiority of weighted and hyperbolic over Euclidean geometry. However, the comparison between POINCARÉEMBED and WEMBED goes clearly in favor of WEMBED, with `BlogCatalog` being the only exception. Additionally, POINCARÉEMBED is much slower; usually by multiple orders of magnitude, despite using 128 cores. This worse performance is also the reason why we limited the POINCARÉEMBED experiments to 8 dimensions and a smaller selection of graphs.

The running time of VERSE is usually better then that of WEMBED, though the exact comparison depends on the graph. For low dimensions, the running times are in the same order of magnitude. Moreover, we note that a fair comparison is probably to compare the $d = 32$ for WEMBED with $d = 128$ for VERSE, where the embeddings have similar quality.

The Effect of Heterogeneity Here we assess the assumption that weighted embeddings are better suited for graphs with heterogeneous degree distribution. Figure 4 shows a scatter plot comparing WEMBED to Euclidean approaches, with color indicating the *heterogeneity* of the graph, which is defined as the $\log_{10}$ of the coefficient of variation of the degree distribution (Bläsius and Fischbeck 2024). Comparing WEMBED to itself but with uniform weights (Figure 4 left) shows a clear trend that graphs with higher heterogeneity benefit more strongly from using weights. The same trend can be seen in comparison to FORCE2VEC and DEEPWALK. The effect is also present for VERSE but less pronounced.

As the real-world networks differ in more than just their heterogeneity, we additionally use generated graphs that let us control the heterogeneity while keeping everything else fixed. We consider *geometric inhomogeneous random graphs* (GIRGs) (Bringmann, Keusch, and Lengler 2019), using the generator by (Bläsius et al. 2022). We generate GIRGs with $n = 10\,\text{k}$ nodes, average degree 15, temperature 0.1, and dimension 4. Additionally we vary the power-law exponent β, which controls the heterogeneity of the degree distribution (high heterogeneity close to $\beta = 2$ and almost uniform for $\beta = 10$). Figure 5 shows the embedding quality of the different embedders depending on the embedding dimension and the power-law exponent. One can clearly see the power-law exponent does not make a difference for WEMBED, while

	WEmbed			Verse				PoincaréEmbed (8)
Graph	8	16	32	8	16	32	128	
AstroPh1	0.776	0.969	1	0.425	0.804	0.955	0.999	–
$\lvert V\rvert = 14\,845,\ \lvert E\rvert = 119\,652$	610	2080	3644	172	200	266	908	–
AstroPh2	0.619	0.836	0.997	0.278	0.609	0.868	0.997	0.565
$\lvert V\rvert = 17\,903,\ \lvert E\rvert = 196\,972$	989	3678	5907	217	252	338	1741	$38\,848^*$
BlogCatalog	0.309	0.389	0.467	0.335	0.359	0.435	0.834	0.408
$\lvert V\rvert = 10\,312,\ \lvert E\rvert = 333\,983$	660	1517	3408	138	164	220	428	$64\,326^*$
CAIDA	0.737	0.981	0.999	0.319	0.721	0.899	0.994	0.458
$\lvert V\rvert = 26\,475,\ \lvert E\rvert = 53\,381$	897	3393	9667	276	316	418	2910	$11\,116^*$
Citeseer	0.999	1	1	0.641	0.924	0.977	0.993	0.669
$\lvert V\rvert = 2110,\ \lvert E\rvert = 3668$	13	26	61	24	26	31	65	569^*
CondMat	0.736	0.989	1	0.275	0.758	0.946	0.999	–
$\lvert V\rvert = 21\,363,\ \lvert E\rvert = 91\,286$	971	4065	7498	251	283	362	2239	–
Cora	1	1	1	0.646	0.930	0.986	1	0.676
$\lvert V\rvert = 2485,\ \lvert E\rvert = 5069$	17	40	100	28	30	36	76	813^*
DBLP	0.562	0.857	1	0.255	0.564	0.844	0.989	–
$\lvert V\rvert = 12\,495,\ \lvert E\rvert = 49\,563$	358	1410	2497	133	153	198	474	–
GrQc	0.981	1	1	0.689	0.965	0.999	1	–
$\lvert V\rvert = 4158,\ \lvert E\rvert = 13\,422$	50	98	254	44	50	61	130	–
HepPh	0.869	0.989	1	0.516	0.840	0.957	0.999	0.757
$\lvert V\rvert = 11\,203,\ \lvert E\rvert = 117\,618$	376	1064	2065	127	146	188	402	$22\,384^*$
Internet	0.724	0.962	1	0.334	0.718	0.899	0.990	0.520
$\lvert V\rvert = 23\,748,\ \lvert E\rvert = 58\,414$	821	2839	6929	248	280	375	2836	$12\,174^*$
Pubmed	0.705	0.989	1	0.217	0.637	0.907	0.994	0.422
$\lvert V\rvert = 19\,717,\ \lvert E\rvert = 44\,324$	601	2420	5657	216	245	310	1745	9167^*
RouteViews	0.905	1	1	0.537	0.887	0.985	1	0.542
$\lvert V\rvert = 6474,\ \lvert E\rvert = 12\,572$	100	232	590	62	70	92	199	2624^*

Table 2: F1-score and embedding time (gray) in seconds on selected real-world graphs. Results are reported for different dimensions. The experiments on PoincaréEmbed were run in parallel and are marked with an $*$.

its variant with uniform weights, as well as Force2Vec and DeepWalk have more difficulties with lower β (i.e., higher heterogeneity). Interestingly, the quality for Verse is independent of β. We mention potential reasons for this when discussing the target space in the next section.

Discussion and Conclusion

Our experiments clearly demonstrate that WEmbed is superior to other state-of-the-art embedding algorithms for low dimensions. With this, WEmbed represents two types of contributions. First, it is a new tool for efficiently computing low-dimensional representations of high quality. Secondly, it serves as a proof-of-concept to understand the usefulness of weighted embeddings. That being said, we believe that both perspective, i.e., engineering new tools as well as deepening our understanding of embeddings, still leave room for improvement. In the following, we discuss which future directions we find promising in this regard. Moreover, we address further minor insights we gained while building WEmbed.

Target Space for Heterogeneous Graphs Despite the close connection between hyperbolic geometry and Euclidean space with weights, there are clearly some differences, e.g., the weighted distance in Equation (1) is not actually a metric as it does not satisfy the triangle inequality. The impact of these differences on embeddings is not yet fully understood. One difference becoming clear with this paper is that using weighted rather than hyperbolic spaces makes the embedding problem algorithmically more tractable. This raises the question whether the gained simplicity comes at the cost of representative power. Our experiments clearly agree with the insight from network science that weighted space is as suited to represent heterogeneous graphs as hyperbolic geometry. We conjecture that this equivalence is also given for hierarchical structures like trees, even if their degree distribution is uniform. We note that WEmbed in its current state would struggle to embed trees with uniform degree distribution, as it would assign every vertex the same weight yielding just a Euclidean embedding. One way to mitigate this is to take the transitive closure of the tree. Interestingly, this technique has also been used for hyperbolic embeddings by Nickel and Kiela (2017). Thus, struggling to embed uniform hierarchical structures is not unique to WEmbed and we believe that the reason is purely algorithmic and not due to the target space. It is an interesting direction to rigorously study these differences between hyperbolic and weighted space and how they can represent different structures.

Another observation related to the target space is that Verse is not heavily impacted by the heterogeneity. Verse uses the dot product as similarity measure. For normalized

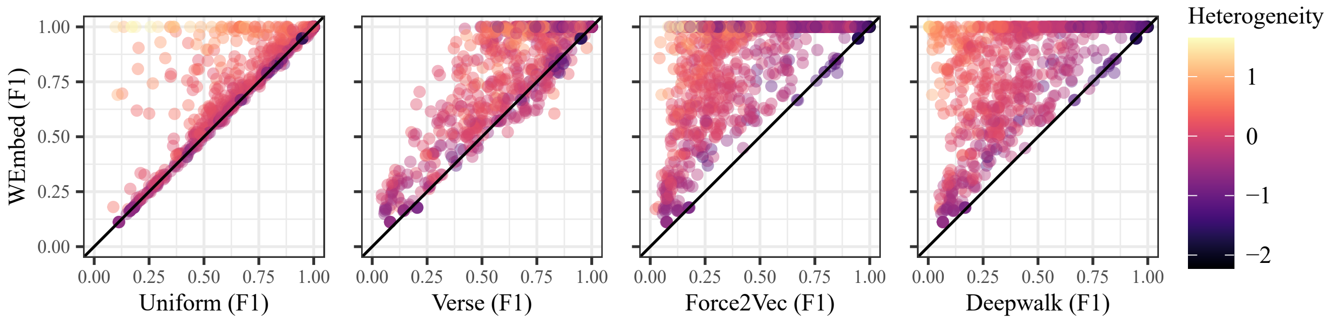

Figure 4: Comparing the quality for 8-dimensional embeddings for the `real` data set. Each point is one graph with the F1-score achieved by WEMBED on the y-axis and the F1-score by Euclidean approaches on the x-axis. The color indicates the heterogeneity of the degree distribution. Uniform (first column) is WEMBED but with uniform weight.

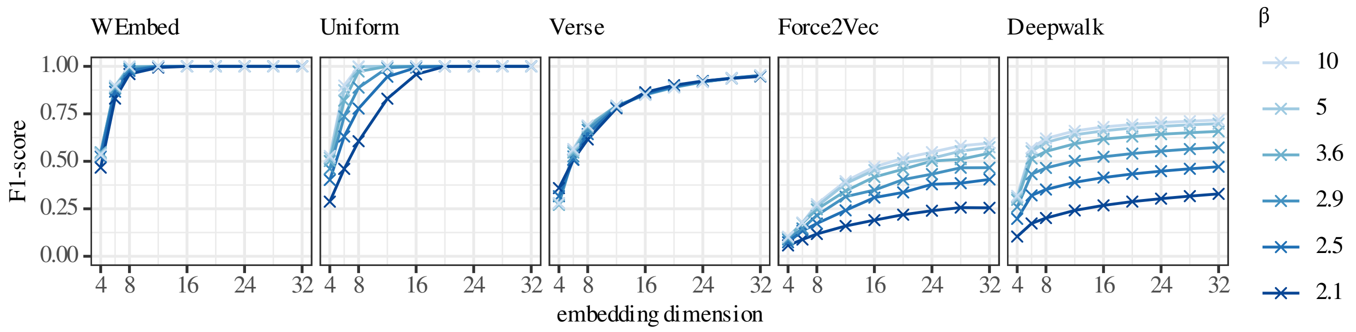

Figure 5: Embedding quality for GIRGs with different power-law exponents. Uniform refers to WEMBED with uniform weights.

vectors, the dot product just measures the angle between vectors and thus there is a close connection to distances on a sphere, which behaves close to Euclidean. However, VERSE is not restricted to normalized vectors, and thus the similarity measure is scaled by the length of the vectors. We believe that this is the reason that VERSE can handle heterogeneous graphs to some degree. It seems fruitful to better understand how the dot product compares to hyperbolic or weighted space and to further explore related spaces.

Algorithmic Techniques Although WEMBED already produces high quality embeddings and is reasonably efficient, there is still room for improvements. The most obvious lies in optimizing the weights, which are currently chosen with a simple heuristic solely based on the node degrees. We note, however, that optimizing the weights is not trivial. Preliminary experiments indicated that using gradient descend does not work well for the weights. Nonetheless, we believe that there is quite some potential using other techniques. In particular, we would be interested to see whether using other centrality measures than just the node degree yields good weights also for hierarchical structures with uniform degrees.

Another approach for improving the embedding quality commonly used for visualization (i.e., very low dimensional embeddings) is a multi-level approach (Hachul and Jünger 2005; Meyerhenke, Nöllenburg, and Schulz 2018). The core idea is to hierarchically group the nodes into clusters and embed the rough structure by finding positions for the clusters before placing individual nodes. This helps to untangle the graph. Preliminary experiments with WEMBED showed that this only helps in very low dimensions. This makes sense as, intuitively, nodes are less likely to get trapped in local optima as there are more directions to get around an obstacle. Thus, the gradient becomes more well-behaved in higher dimensions, eliminating the need for a multi-level approach.

Finally, we believe that exploring more advanced negative sampling techniques as well as improving the geometric data structure has the potential to improve the running time.

Applications We demonstrated that WEMBED can produce very accurate low-dimensional representation of graphs by embedding them into a weighted space. An obvious next question is how the insight that weighted space is well suited to host heterogeneous data, as well as the tool WEMBED itself, can be utilized for various applications. This involves down-stream tasks on graphs like node classification, the extension to, e.g., graph neural networks, and the applicability to other domains like word embeddings for natural language processing or computer vision, where hyperbolic embeddings have been observed to be beneficial. We believe that weighted embeddings are simpler to adapt and that the resulting models are easier to optimize, while providing the same benefits that hyperbolic geometry promises.

Acknowledgements

This work was supported by funding from the pilot program Core–Informatics of the Helmholtz Association (HGF).

This work has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No. 882500).

References

Bieker, N.; Bläsius, T.; Dohse, E.; and Jungeblut, P. 2023. Recognizing Unit Disk Graphs in Hyperbolic Geometry is $\exists\mathbb{R}$-Complete. *CoRR*, abs/2301.05550.

Bläsius, T.; and Fischbeck, P. 2024. On the External Validity of Average-case Analyses of Graph Algorithms. *ACM Trans. Algorithms*, 20(1): 10:1–10:42.

Bläsius, T.; Friedrich, T.; and Katzmann, M. 2021. Force-Directed Embedding of Scale-Free Networks in the Hyperbolic Plane. In *Symposium on Experimental Algorithms (SEA)*, Leibniz International Proceedings in Informatics (LIPIcs), 22:1–22:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

Bläsius, T.; Friedrich, T.; Katzmann, M.; Meyer, U.; Penschuck, M.; and Weyand, C. 2022. Efficiently Generating Geometric Inhomogeneous and Hyperbolic Random Graphs. *Network Science*, 10(4): 361–380.

Bläsius, T.; Friedrich, T.; Krohmer, A.; and Laue, S. 2016. Efficient Embedding of Scale-Free Graphs in the Hyperbolic Plane. In *European Symposium on Algorithms (ESA)*, 16:1–16:18.

Boguñá, M.; Papadopoulos, F.; and Krioukov, D. 2010. Sustaining the Internet with Hyperbolic Mapping. *Nature Communications*, 1(62).

Breu, H.; and Kirkpatrick, D. G. 1998. Unit disk graph recognition is NP-hard. *Comput. Geom.*, 9(1-2): 3–24.

Bringmann, K.; Keusch, R.; and Lengler, J. 2019. Geometric Inhomogeneous Random Graphs. *Theor. Comput. Sci.*, 760: 35–54.

Chamberlain, B. P.; Clough, J. R.; and Deisenroth, M. P. 2017. Neural Embeddings of Graphs in Hyperbolic Space. *CoRR*, abs/1705.10359.

Chami, I.; Ying, Z.; Ré, C.; and Leskovec, J. 2019. Hyperbolic Graph Convolutional Neural Networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 4869–4880.

Choudhary, N.; Rao, N.; Subbian, K.; Sengamedu, S. H.; and Reddy, C. K. 2022. Hyperbolic Neural Networks: Theory, Architectures and Applications. In *International Conference on Knowledge Discovery and Data Mining (KDD)*, 4778–4779. Association for Computing Machinery.

Fruchterman, T. M. J.; and Reingold, E. M. 1991. Graph Drawing by Force-directed Placement. *Softw. Pract. Exp.*, 21(11): 1129–1164.

Ganea, O.; Bécigneul, G.; and Hofmann, T. 2018. Hyperbolic Entailment Cones for Learning Hierarchical Embeddings. In *International Conference on Machine Learning (ICML)*, volume 80 of *Proceedings of Machine Learning Research*, 1632–1641. PMLR.

Ganea, O.; Becigneul, G.; and Hofmann, T. 2018. Hyperbolic Neural Networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 31. Curran Associates, Inc.

Gansner, E. R.; Hu, Y.; and North, S. C. 2013. A Maxent-Stress Model for Graph Layout. *IEEE Trans. Vis. Comput. Graph.*, 19(6): 927–940.

Guttman, A. 1984. R-Trees: A Dynamic Index Structure for Spatial Searching. In *SIGMOD International Conference on Management of Data (SIGMOD)*, 47–57. ACM Press.

Hachul, S.; and Jünger, M. 2005. Drawing Large Graphs with a Potential-Field-Based Multilevel Algorithm. In *Graph Drawing*, 285–295. Springer Berlin Heidelberg.

Kang, R. J.; and Müller, T. 2012. Sphere and Dot Product Representations of Graphs. *Discret. Comput. Geom.*, 47(3): 548–568.

Khrulkov, V.; Mirvakhabova, L.; Ustinova, E.; Oseledets, I. V.; and Lempitsky, V. S. 2020. Hyperbolic Image Embeddings. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 6417–6427. Computer Vision Foundation / IEEE.

Kingma, D. P.; and Ba, J. 2015. Adam: A Method for Stochastic Optimization. In *International Conference on Learning Representations (ICLR)*.

Kisfaludi-Bak, S.; and van Wordragen, G. 2024. A Quadtree, a Steiner Spanner, and Approximate Nearest Neighbours in Hyperbolic Space. In *International Symposium on Computational Geometry (SoCG)*, volume 293 of *Leibniz International Proceedings in Informatics (LIPIcs)*, 68:1–68:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

Krioukov, D.; Papadopoulos, F.; Kitsak, M.; Vahdat, A.; and Boguñá, M. 2010. Hyperbolic Geometry of Complex Networks. *Phys. Rev. E*, 82: 036106.

Law, M. T.; Liao, R.; Snell, J.; and Zemel, R. S. 2019. Lorentzian Distance Learning for Hyperbolic Representations. In *International Conference on Machine Learning (ICML)*, volume 97 of *Proceedings of Machine Learning Research*, 3672–3681. PMLR.

Liu, Q.; Nickel, M.; and Kiela, D. 2019. Hyperbolic Graph Neural Networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 8228–8239.

Meyerhenke, H.; Nöllenburg, M.; and Schulz, C. 2018. Drawing Large Graphs by Multilevel Maxent-Stress Optimization. *IEEE Trans. Vis. Comput. Graph.*, 24(5): 1814–1827.

Mishne, G.; Wan, Z.; Wang, Y.; and Yang, S. 2023. The Numerical Stability of Hyperbolic Representation Learning. In *International Conference on Machine Learning (ICML)*, volume 202 of *Proceedings of Machine Learning Research*, 24925–24949. PMLR.

Nickel, M.; and Kiela, D. 2017. Poincaré Embeddings for Learning Hierarchical Representations. In *Advances in Neural Information Processing Systems (NeurIPS)*, 6338–6347.

Nickel, M.; and Kiela, D. 2018. Learning Continuous Hierarchies in the Lorentz Model of Hyperbolic Geometry. In *International Conference on Machine Learning (ICML)*, volume 80 of *Proceedings of Machine Learning Research*, 3776–3785. PMLR.

Papadopoulos, F.; Aldecoa, R.; and Krioukov, D. 2015. Network Geometry Inference Using Common Neighbors. *Phys. Rev. E*, 92: 022807.

Papadopoulos, F.; Psomas, C.; and Krioukov, D. 2015. Network Mapping by Replaying Hyperbolic Growth. *IEEE/ACM Transactions on Networking*, 23(1): 198–211.

Peng, W.; Varanka, T.; Mostafa, A.; Shi, H.; and Zhao, G. 2022. Hyperbolic Deep Neural Networks: A Survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(12): 10023–10044.

Perozzi, B.; Al-Rfou, R.; and Skiena, S. 2014. DeepWalk: online learning of social representations. In *International Conference on Knowledge Discovery and Data Mining (KDD)*, 701–710. ACM.

Rahman, M. K.; Sujon, M. H.; and Azad, A. 2020. Force2Vec: Parallel Force-Directed Graph Embedding. In *International Conference on Data Mining (ICDM)*, 442–451. IEEE.

Sala, F.; Sa, C. D.; Gu, A.; and Ré, C. 2018. Representation Tradeoffs for Hyperbolic Embeddings. In *International Conference on Machine Learning (ICML)*, volume 80 of *Proceedings of Machine Learning Research*, 4457–4466. PMLR.

Sarkar, R. 2011. Low Distortion Delaunay Embedding of Trees in Hyperbolic Plane. In *International Symposium on Graph Drawing (GD)*, volume 7034 of *Lecture Notes in Computer Science*, 355–366. Springer.

Schaefer, M. 2010. Complexity of Some Geometric and Topological Problems. In *Graph Drawing*, 334–344. Springer Berlin Heidelberg.

Shimizu, R.; Mukuta, Y.; and Harada, T. 2021. Hyperbolic Neural Networks++. In *International Conference on Learning Representations (ICLR)*.

Suzuki, A.; Nitanda, A.; wang, j.; Xu, L.; Yamanishi, K.; and Cavazza, M. 2021. Generalization Bounds for Graph Embedding Using Negative Sampling: Linear vs Hyperbolic. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 34, 1243–1255. Curran Associates, Inc.

Suzuki, R.; Takahama, R.; and Onoda, S. 2019. Hyperbolic Disk Embeddings for Directed Acyclic Graphs. In *International Conference on Machine Learning (ICML)*, volume 97 of *Proceedings of Machine Learning Research*, 6066–6075. PMLR.

Tifrea, A.; Bécigneul, G.; and Ganea, O. 2019. Poincare Glove: Hyperbolic Word Embeddings. In *International Conference on Learning Representations (ICLR)*. OpenReview.net.

Tsitsulin, A.; Mottin, D.; Karras, P.; and Müller, E. 2018. VERSE: Versatile Graph Embeddings from Similarity Measures. In *International Conference on World Wide Web (WWW)*, 539–548. ACM.

van Spengler, M.; Wirth, P.; and Mettes, P. 2023. HypLL: The Hyperbolic Learning Library. In *International Conference on Multimedia (MM)*, 9676–9679. Association for Computing Machinery.

Voitalov, I.; van der Hoorn, P.; van der Hofstad, R.; and Krioukov, D. 2019. Scale-Free Networks Well Done. *Phys. Rev. Res.*, 1: 033034.

Vyas, A.; Choudhary, N.; Khatir, M.; and Reddy, C. K. 2022. GraphZoo: A Development Toolkit for Graph Neural Networks with Hyperbolic Geometries. In *The ACM Web Conference (WWW)*. Association for Computing Machinery.

Wang, Z.; Li, Q.; Jin, F.; Xiong, W.; and Wu, Y. 2016. Hyperbolic Mapping of Complex Networks Based on Community Information. *Physica A: Statistical Mechanics and its Applications*, 455: 104–119.

Yang, M.; and Zhou, M. 2024. Hyperbolic Representation and Deep Learning: A Comprehensive Collection. https://github.com/marlin-codes/Awesome-Hyperbolic-Representation-and-Deep-Learning. Accessed: 2024-05-01.

Zhou, M.; Yang, M.; Xiong, B.; Xiong, H.; and King, I. 2023. Hyperbolic Graph Neural Networks: A Tutorial on Methods and Applications. In *International Conference on Knowledge Discovery and Data Mining (KDD)*, 5843–5844. Association for Computing Machinery.