Adapting Graph Layout Algorithms to New Objective Functions

TABLE OF CONTENTS

TABLE OF CONTENTS—*Continued*

LIST OF FIGURES

LIST OF FIGURES—*Continued*

LIST OF FIGURES—*Continued*

LIST OF FIGURES—*Continued*

LIST OF TABLES

Chapter 1

INTRODUCTION

Graphs (networks) are powerful models for real-world relationships, but are often very large or complicated and so visualizing them becomes important way to gain an understanding of the data. A popular abstraction for graph data is the node link diagram, where the vertices of the graph are represented by shapes in the plane and the edges between vertices drawn as curves. To compute an embedding of a graph using this abstraction, there exist many embedding algorithms.

Many node-link embedding algorithms are rooted on the Gestalt principle of proximity (whether or not directly stated); that objects near each other are perceived as similar and so vertices that are adjacent should be drawn near each other. However, this is not always enough as there are other graph properties that may be emphasized by choice of embedding, intentional or otherwise [44]. Properties such as shape, density, vertex neighborhood, cluster identity, symmetry, and others can be heavily influenced by a embedding techique. This is a fact that is often overlooked. Too, the embedding space of a graph visualization is often assumed to be Euclidean. Though there is a very good reason for this (the computer screen, printed papers, and world around us are Euclidean, after all), other geometries can offer novel benefits for drawing node-link diagrams.

Spherical geometry lessens the centering bias that is unavoidable in the plane; For any Euclidean graph drawing one must choose a center. The choice of center is likely unintentional, and may imply importance that does not exist. This is best illustrated with two drawings of K_4, the complete graph on four vertices. Two typical Euclidean drawings are A) three vertices in a triangle with the fourth in the center or B) all four vertices on a rectangle. Drawing A avoids edge crossings, which is generally desirable, but implies that the central vertex is somehow important. However, all four vertices are equally connected and we could have chosen any one vertex to be central arbitrarily. The opposite for drawing B, no central node but we must have crossings in a straight line drawing. The sphere has no notion of a center, and we can draw K_4 using straight lines on the sphere without crossings *and* with no center, something not possible in the plane. A more real-world example can be found in Figure 2.1, where each cluster can be translated to be the center of the drawing. Related, in group level node-link diagrams where graph clusters are drawn as filled shapes, these shapes can be interpreted as region boundaries and drawing them on a sphere offers familiarity in interaction and interpretation.

Hyperbolic geometry is less intuitively understood, but no less useful. Like the Euclidean plane, the hyperbolic plane is infinite. However, the area of a hyperbolic circle increases more quickly than that of a Euclidean circle. This property allows exponentially expanding structures such as trees or hierarchies to be placed in the hyperbolic plane with

zero error [64, 68, 78]. Some real world complex networks such as social networks [64] and gene expression networks [110] have been shown to admit embeddings with lower error (distance distortion) in hyperbolic space. Hyperbolic projections to the Euclidean plane have interesting and useful properties as well. Notably, the Poincaré projection maps the infinite hyperbolic plane to a finite Euclidean circle, and provides a focus+context effect not unlike a fish-eye lens. Objects near the center of the projection maintain high detail and accuracy, while objects closer to the perimeter become small and distorted. The entirety of the space can stay in view though, allowing one to translate the focus while maintaining the surrounding context.

Although non-Euclidean geometries offer benefits for graph visualization, few algorithms exist to compute a layout for a general graph in these spaces. Of those that do, no algorithms performed distance based computation purely within the respective geometries; either using projections to the plane [62], performing a constrained 3-dimensional Euclidean computation [85], or transforming the distances to use Euclidean methods [92].

There are several considerations when designing such an algorithm that make this a non-trivial task; The dilation (scale) of the distances must be addressed, a measure of accuracy employed, and an optimization scheme must be carefully chosen (e.g. exploding gradient is a bigger problem in hyperbolic computation). Graph embedding techniques are needed that address these issues to allow for accessible graph visualization in non-Euclidean spaces.

Revisiting Euclidean space, dimension reduction and force-directed based embedding techniques emphasize some aspects of the data, and tend to obscure others. In most cases, visualization designers are forced into choosing a single embedding technique for their data and so forced into emphasizing those particular aspects. Two popular techniques that are adapted in graph visualization are (metric) multi-dimensional scaling (MDS) [25, 66] and t-distributed stochastic neighbor embedding (t-SNE) [69]. The goals of these two algorithms are somewhat orthogonal: MDS seeks to preserve all pairwise distances, paying more attention to the objects that have larger distance, while t-SNE preserves the likelihood of two points being near each other in the embedding, given that they were near each other in the original space. MDS is said to preserve *global* structure, while t-SNE is said to preserve *local* neighborhoods [35].

These ideas are directly applicable to graph visualization, where we can define high-dimensional distances as the graph theoretic distances, e.g., via all pairs shortest paths computation. In the graph layout literature, MDS is often referred to as stress minimization [42, 109], and t-SNE has been adapted to graph layout in an algorithm known as tsNET [65] and later DRGraph [111]. Choosing the "best" graph embedding algorithm depends on the graph structure and on the goal of the visualization. For example, highly structured or mesh-like graphs can be drawn well using MDS, while tsNET can fail at accurately capturing their global structure. Similarly, highly clustered and dense graphs benefit from tsNET but MDS struggles to produce good layouts as it is dominated by "long-distance" information.

Algorithms and techniques are needed to both discover whether a global or local embedding better suits the data, and to preserve intermediate structure by e.g. allowing clusters to be less separated by white space but ensuring the distance between those clusters is data faithful. Additionally, metrics are needed to measure this intermediate structure faithfulness.

1.1 Overview

Chapter 2 contains a literature review of the current state-of-the-art of non-Euclidean and local/global graph embedding techniques for visualization. To date, I have authored three papers to address the above problems: a paper on spherical graph embedding, another paper on hyperbolic graph embedding, and a third paper on preserving intermediate structure in graph embeddings. These works are the basis for Chapter 3, each paper taking a subsection detailed below. Chapter 4 details the projects I am currently working on and my current plan and road-map to submit them.

Spherical Graph Embedding: In this paper we present a scalable adaptation to MDS that embeds data on the sphere, Spherical Multidimensional Scaling (SMDS). We evaluate the technique by comparing speed and embedding faithfulness to a related approach. We address the dilation problem in spherical space by providing an optimization scheme to find a good dilation factor and propose a heuristic that works well on our graph benchmark. This work was published at the International Symposium on Graph Drawing 2022 [72] and the contents of the paper found in section 3.1, with the dilation problem and solution detailed in section 3.1.4.

Hyperbolic Graph Embedding: In this paper we implement three separate approaches for hyperbolic graph embedding: a projection based method, a re-implementation of a tangent plane method, and a generalization of MDS to hyperbolic space, Hyperbolic Multidimensional Scaling (HMDS). We provide an evaluation of each method and show that HMDS is able to embed graphs with less error than the other methods, and we show that some graphs achieve lower error in a hyperbolic embedding than a Euclidean MDS embedding. This work was published at IEEE PacificVis 2022 [73] and is the source of section 3.2.

Graph Embedding Intermediate Scales: This work sees the introduction of a new graph (data) embedding algorithm, L2G, which uses a single parameter to determine how 'local' or how 'global' an embedding to produce. We evaluate the method both on how well the algorithm smoothly transitions between local and global outputs, and introduce a new metric called cluster distance (CD) which measures how well the distance between clusters is preserved. We show that L2G, for some parameter values, outperforms competing state-of-the-art method for preserving this intermediate structure via CD. This paper is under review at IEEE VIS 2023 [71] and is the source of section 3.3.

Chapter 2

RELATED WORK

We review related work in non-Euclidean geometry and graph layout methods.

2.1 Non-Euclidean Geometry

Non-Euclidean geometries are a special case of Riemannian geometries, which are spaces that are locally "smooth": one can define an inner product on the tangent space at each point. Spherical and hyperbolic non-Euclidean geometries are similar to Euclidean geometry, except for one axiom.

Euclid's *Elements* specify five axioms/postulates upon which all true statements about geometry should be proved. The fifth axiom is quite lenghty and it was hypothesized for centuries to be provable using the first four. In 1892 Lobachevsky and Bolyai independently discovered and published their formulation of hyperbolic geometry by inverting an equivalent statement to Euclid's fifth axiom, Playfair's axiom: *In a plane, given a line and a point not on it, at most one line parallel to the given line can be drawn through the point*. Replace "*at most one line*" with "*at least two distinct lines*" to get hyperbolic geometry. Replace "*at most one line*" with "*there does not exist a line*" to arrive at spherical geometry.

Spherical geometry has benefits in the context of data visualization. In Euclidean (or hyperbolic) layouts, one is forced to choose a "center" of the embedding, intentionally or not, whereas on the sphere there is no notion of a center. A perceptual side effect of centered embeddings is that vertices near the center seem more important, while vertices away from the center seem more peripheral. This problem does not occur in spherical space, where simple rotation can place any vertex in the center of the view (a feature that is very useful when visualizing social networks, or networks of research fields); see Fig. 2.1. Additionally, many spherical projections into Euclidean space, such as the stereographic projection, provide a desirable focus+context effect.

Distances in non-Euclidean geometries generalize the concept of a straight line to that of a geodesic, defined as an arc of shortest length (not necessarily unique) that contains both endpoints. The distance between two endpoints is then the length of that curve.

A point on a sphere of radius R is uniquely represented by a pair of angles, (ϕ, λ), where $0 \leq \phi \leq \pi$ is known as the *latitude* and $0 \leq \lambda \leq 2\pi$ is the *longitude*. Given two points $(\phi_1, \lambda_1), (\phi_2, \lambda_2)$ on the sphere with radius R, the geodesic distance is then derived by the spherical law of cosines:

$$\delta((\phi_1, \lambda_1), (\phi_2, \lambda_2)) = R * \arccos(\sin\phi_1 \sin\phi_2 + \cos\phi_1 \cos\phi_2 \cos(\lambda_1 - \lambda_2)) \qquad (2.1)$$

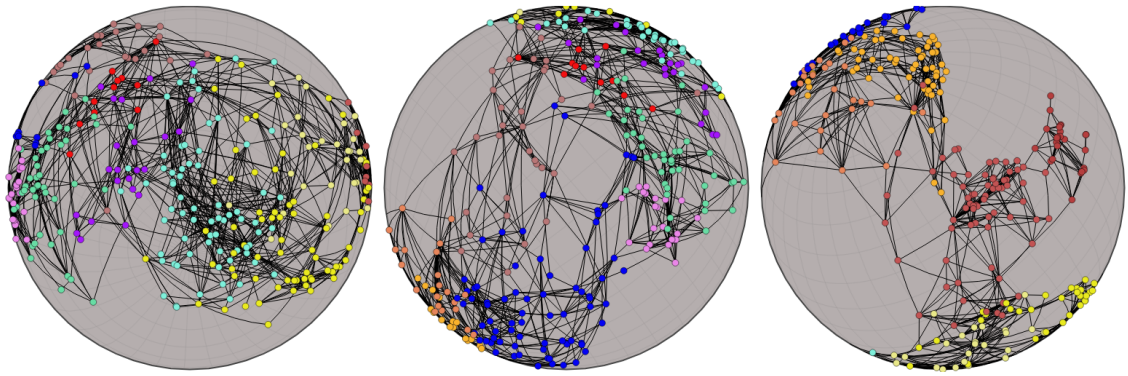


Figure 2.1: The Maps of Science dataset [15] laid out using our SMDS algorithm, from three different perspectives. Each color represents a different field of science (nodes are subfields), and their relationships exhibit a ring-like structure. Any field can be placed in the center of the view.

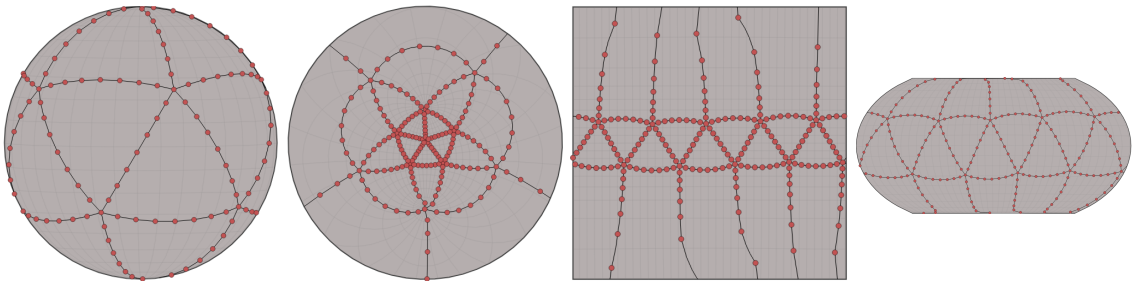


Figure 2.2: A subdivided isocahedron graph embedded on the sphere, displayed with the orthographic, stereographic, Mercator, and equal Earth projections.

where $\delta(X_i, X_j)$ denotes the geodesic distance between points X_i and X_j, assuming X is an $n \times 2$ matrix whose rows correspond to spherical coordinates.

It is known that the surface of a sphere cannot be perfectly preserved in any 2-dimensional Euclidean drawing, due to its curvature. One can preserve various combintations of angles, areas, geodesics, or distances but not all of these simultaneously. The orthographic projection, or the "view from space" projects the sphere onto a tangent plane with point of perspective from outside the sphere. While half of the sphere is obscured and shapes and area are distorted near the boundary, geodesics through the origin are preserved and it gives the impression of a 3-dimensional globe. The stereographic projection is similar but instead with a point of projection looking through the sphere, and preserves angles. The Mercator projection is a common cylindrical map projection with heavy area distortion near the poles. The equal Earth projection preserves area and gives the impression of a spherical shape. Examples are shown in Fig. 2.2. We primarily use the orthographic and equal Earth projections in this paper.

One of the earliest approaches of graph drawing in the hyperbolic plane is by Lamping

et al. [68] and embeds hierarchies into the hyperbolic plane by recursively placing each node's children evenly spaced around the arc of a circle. This is possible thanks to the exponential expansion intrinsic to the geometry. They make use of the Poincaré projection to display the graph on the computer monitor, which also provides the now well known 'focus+context' effect. Navigating the hierarchy is done by re-centering the projection at a new node in the hyperbolic plane. The embedding can be computed in linear time and arbitrary graphs can also be visualized using this approach by utilizing a spanning tree of the graph and 'filling in' the rest of the edges later.

Table 2.1: Hyperbolic browsing systems

System	Date	Description
H2 Tree Browser	1995	2 dimensions, hierarchy viewer
HVS	2007	Hiearchy visualization application
Js InfoVis Toolkit	2013	Web-based data vis suite
Treebolic	2014	Java Hyperbolic Poincaré visualization
d3-hypertree	2018	Hyperbolic tree visualization library
H3	2000	3 dimensions, hierarchy viewer
walrus	2000	Re-implementation of H3 in Java
h3py	2015	Re-implementation of H3 in Python

A bioinformatics-motivated java application by Bingham and Sudarsanam [7] uses a similar approach to visualize phylogenetic trees. Andrews *et al.* [2] also rely on Lamping *et al.*'s work in their Hierarchy Visualization System as do Baumgartner and Waugh [5] who visualize Roget's thesaurus. The Java InfoVis Toolkit also implements a hyperbolic hierarchy browser [6] and TreeBolic implements the hyperbolic tree layout [13]. More recently, Glatzhofer developed a hyperbolic hierarchy browser utilizing d3.js, a javascript graphics library which works in the browser, and can display large hierarchies smoothly with different layout algorithms [45, 46, 47].

While most prior work considers the 2-dimensional hyperbolic plane, Munzner has also used 3D hyperbolic space to visualize hierarchies with the help of the Beltrami-Klein projection [74, 75, 76, 78]. Here geodesics are mapped to straight lines rather than the circular arcs of the Poincaré projection. Munzner's work has been re-implemented in two subsequent systems: Walrus [55] and h3py [108].

Classical MDS has been explored in hyperbolic space, by replacing the conversion to similarities with an appropriate hyperbolic scaling function [21, 92]. Using a similar idea, metric and non-metric MDS have been generalized to hyperbolic space by incorporating hyperbolic geodesic distance into the cost function [102, 103, 104, 110].

It has been shown that some graphs can be embedded with lower error in hyperbolic space than in Euclidean space [8]. Zhou and Sharpee [110] show that hyperbolic MDS (H-MDS) can be used to detect the underlying geometry of a dataset, when comparing its embedding error to Euclidean non-metric MDS. They go further to show that the underlying space of genomes is hyperbolic. Krioukov *et al.*'s [64] work indicates that hyperbolic geometry may underlie complex networks and hierarchical networks, such as phylogenetic trees and the internet.

Greedy embeddings also appear to have a close relationship with hyperbolic geometry. Indeed, any connected, finite graph admits a greedy embedding in hyperbolic space, which is not generally true in Euclidean geometry [59]. Greedy embeddings of graphs allow for greedy routing, which is particularly useful when a node may not know the global topology, but only its own position and that of its neighbors such as in social networks and the internet [33].

An open-source hyperbolic visualization tool, RogueViz [17], includes different projections and educational tools, although its restriction to tessellations of the hyperbolic plane makes it less than ideal for general graphs. Self-organizing maps have been generalized to hyperbolic space, but are restricted to lattices [83].

Stress-based approaches have been explored in other Riemannian spaces such as the torus [19, 18], as different spaces provide different visualization advantages. Unlike in the plane, the torus allows larger classes of graphs can be drawn without crossings. Human subject studies show that these spaces are no worse than Euclidean space for common navigation tasks [28, 19].

One can achieve a similar focus+context effect by using lens effects [99, 98]. In particular, at first glance the Poincaré disk appears to resemble a fisheye lens. However, a lens effect generally applies only to a subset of the visible data, scaling or warping it to bring it into focus. The focus+context view is applied across all of the data in the Poincaré disk, with an exponential decrease in data size away from the center, but the entirety of the object remaining in view.

Note also that there are theoretical limits on the effectiveness of hyperbolic embeddings for general graphs. Some graphs can be embedded trivially with a low, constant embedding error (e.g., as cycles and square lattices) but have non-trivial embedding error in the hyperbolic plane [31, 101]. However, other graphs such as trees and hyperbolic tilings can be embedded better in hyperbolic space than in Euclidean space. For example, while Euclidean geometry only admits 3 regular tessellations (triangles, squares, hexagons), the hyperbolic plane admits infinitely many.

2.2 Multidimensional Scaling

There are three different types of Multidimensional Scaling (MDS): classical, metric, and non-metric (although these labels are not used consistently in different fields). Here, we refer to Torgerson's MDS as classical MDS, where the input distances are converted to similarities, and principal component analysis (PCA) is used to obtain the embedding [100]. Metric MDS minimizes a loss function, commonly known as *stress* [94]. Finally, in non-metric MDS the input distances are not necessarily distances, but can be ranks [66].

Using graph-theoretic distances to determine a graph layout dates back to the Kamada-Kawai algorithm [57]. A more general embedding approach from a given set of distances is the multi-dimensional scaling (MDS) [94] which has extensively been applied to graph layout; see [40, 42, 109]. Both the Kamda-Kawai and (metric) MDS algorithms aim to

minimize the *stress* function, which is the sum of residual squares between the given and the embedded distances of each pair of datapoints. Formally, given a graph $G = (V,E)$ with the graph theoretic distances between its n vertices $\left(d_{ij}\right)_{i,j=1}^{n,n}$, where the vertices are labeled $1,2,\ldots,n$ MDS aims to embed the graph in $\mathbb{R}^d$ by minimizing the following *stress* function to find the locations for its vertices:

$$\sigma(X) = \sum_{i<j} w_{ij}(||X_i - X_j|| - d_{ij})^2. \tag{2.2}$$

The resulting solution of $X_1, X_2, \ldots, X_n \in \mathbb{R}^d$ represents the coordinates of the embedded graph vertices.

Various forms of MDS have been analyzed. Metric MDS was first studied by Shepard [94] (see equation in (3.1)), and the related non-metric MDS by Kruskal [66]. Classical MDS is similar but uses an objective function called *strain*.

The classical MDS has a closed form solution while the metric MDS and non-metric MDS rely on solving an optimization problems to minimize the corresponding stress functions. Many approaches have been proposed to solve (metric) MDS including stress majorization [41] and (stochastic) gradient descent [10].

When used for the purposes of visualization, the embedding space for MDS is almost always 2 dimensional Euclidean, as that is the space of a flat sheet of paper, or the flat screen of a computer monitor. The natural measure of distance is then the Euclidean norm.

In this work we will focus on metric MDS, defined in (2.2) but instead of embedding the graph in Euclidean space, we embed it directly on the sphere. The MDS approach has already been applied to embed graphs on spherical [30] and hyperbolic [73] spaces. Our contribution is to solve the proposed optimization problem faster and be able to handle larger graphs, address the dilation/resizing problem, as well as analyze the approach on wider range of graphs and provide a working and easy to use implementation.

2.2.1 Graph Layout in non-Euclidean Geometry

Non-Euclidean graph visualization has been studied by Munzner, with an emphasis on trees and hierarchies [74, 75, 76, 78], and the following link `treevis.net`, provides several examples of hyperbolic and sphere based tree visualizations [93]. Spherical layouts have been investigated in an immersive setting such as virtual reality [67, 107]. Self-organizing maps have been developed for both spherical and hyperbolic geometries [83]. Several other examples of spherical graph visualization include the Map of Science [15], the "Places and Spaces" [14], and "Worldprocessor" [49] exhibitions. Some limitations of the existing algorithms for hyperbolic graph visualization are discussed in [32].

Force-directed algorithms model the nodes and edges as a physical system, and provide a layout by minimizing the total energy. These algorithms are popular in part due to their conceptual simplicity and quality layouts [61]. A general technique for generalizing force-directed algorithms to non-Euclidean spaces is described in [62]. However, it only works

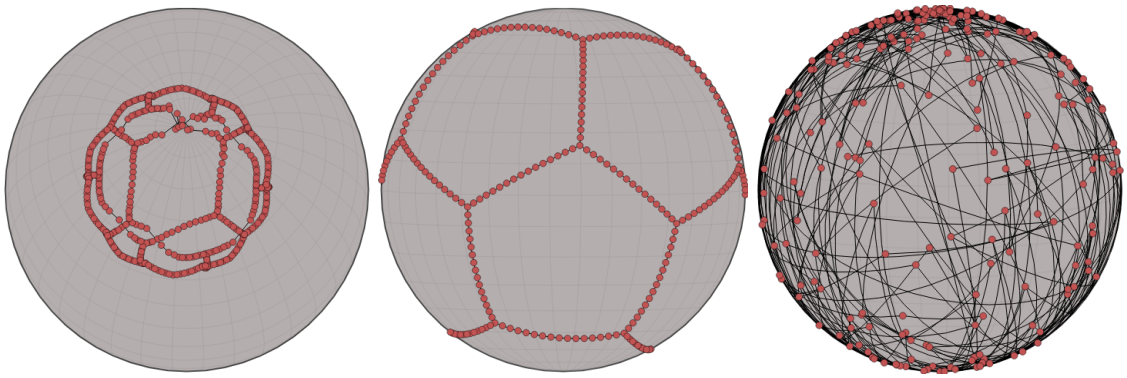


Figure 2.3: A dodecahedron subdivision graph. Left: a small dilation factor forces the layout on a small patch of the sphere. Middle: a correct dilation factor using our heuristic discussed in Section 3.1.4, allows the graph to make use of the spherical geometry. Right: a large dilation factor makes the distances unachievable.

for small graphs as for larger ones it is too computationally expensive and is unlikely to escape local minima.

There are several different approaches to embedding a graph on the sphere. A simple idea is to generate a 2D Euclidean layout and project it onto the sphere through a linear map [28, 85], however, this embedding will not make full use of spherical geometry. Another approach is to embed the graph in 3D Euclidean space and modify it to force it on the surface of a sphere [26, 85], but this is quite mathematically involved and complicates the optimization. A more natural method directly computes a 2D spherical embedding (in latitude and longitude) such that the geodesic distances on the sphere and graph-theoretic distances between pairs of vertices are closely matched [30]. We focus on this approach and make it scalable by adopting stochastic gradient descent for the optimization phase and by solving the dilation/resizing problem specified below.

In the graph drawing literature, the normalized stress of a layout is a standard quality measure [40, 65, 111]. This is perfectly acceptable in Euclidean space where a layout is not meaningfully changed when the layout is resized. For non-Euclidean graph layouts there is a possible issue of *dilation or resizing*. Formally, a dilation is a function on a metric space M, $f : M \to M$ that satisfies $d(f(x), f(y)) = rd(x,y)$ for $x, y \in M$, $r > 0 \in \mathbb{R}$ and $d(x,y)$ being the distance between x and y. In non-Euclidean spaces, such as the sphere, the size of a layout can have drastic effects; see Fig. 2.3. At small dilation, a graph embedded on the sphere takes only a small patch and the sphere patch behaves like a piece of the Euclidean plane. At large dilation, a graph embedded on the sphere wraps over itself. At some optimal dilation the embedded graph fits on the sphere with low distortion. Choosing the size of the sphere is important to accurately represent the data. We are unaware of any work regarding this problem in spherical embedding, and propose a heuristic and optimization scheme to solve it in Section 3.1.4.

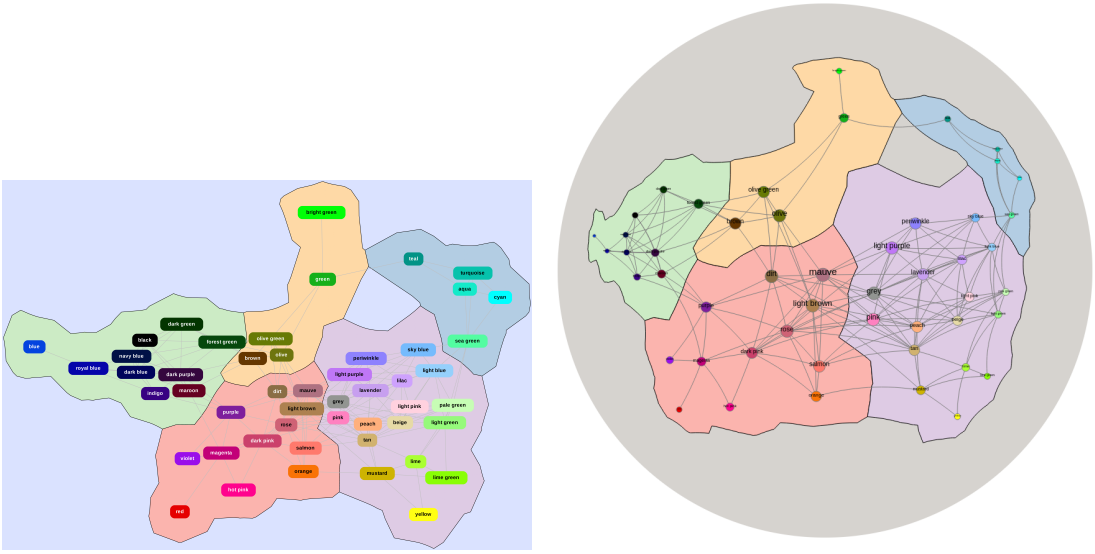


Figure 2.4: An example of a GMap Euclidean layout (left) and its hyperbolic realization (right) via inverse projection.

As stress is difficult to interpret between geometries, we use a more fair comparison metric called *distortion* [73, 92] defined later in Section 3.1.3.

We review prior work on DR methods and graph embeddings.

2.2.2 Dimensionality Reduction

Dimensionality reduction (DR) refers to a large family of algorithms that map a set of high-dimensional datapoints in low-dimensionsal space. Different DR algorithms aim to preserve different properties of the dataset, including total variance, global distances, local distances, etc. Depending on the application, the target lower dimension might vary. In the context of visualization, the dataset is typically projected onto 2D or 3D Euclidean space. A recent survey by Espadoto *et al.* [35] classifies DR algorithms into several categories, two of which are "input type" and "neighborhood." A DR algorithm's input type belongs to either *sample* or *distance*. Sample methods such as Principal Component Analysis (PCA) [36, 56] project the high dimensional data down to the embedding space. While not generally applicable to graph embedding, one can first embed the graph in a high dimensional space and apply such a method [50, 63]. Distance methods do not operate on the actual high-dimensional vectors, but on the distances between them and therefore are applicable to datasets with distance metrics. For graph embedding, a natural notion of distance is the graph-theoretic distance (usually obtained via all-pairs-shortest-paths computation).

Local DR algorithms attempt to preserve the nearby neighbors of a data point in the low dimensional embedding, while global DR algorithms attempt to retain all pairwise distances. Popular techniques in the local category include t-distributed stochastic neighbor

embedding (t-SNE) [69] and uniform manifold approximation and projection (UMAP) [70], but also includes Locally Linear Embedding (LLE) [88] and IsoMap [97]. Multidimensional scaling (MDS) has variants, but here we mean metric MDS which minimizes stress [94].

Few techniques attempt to capture both global and local structure. Chen and Buja [20] propose to adapt MDS to look for local structure. They choose a subset of pairs of which to preserve distance (via a k-nearest neighbor search from each data point) and assume all other pairs of vertices have an arbitrarily large ideal distance. The underlying idea is similar to what we propose, although it does not provide a framework to cover the spectrum from local to global as our method does.

While t-SNE's perplexity parameter determines the number of neighbors expected around a point, increasing perplexity does not lead to global structure preservation [106]. Anchor-t-SNE attempts to improve the global structure preservation by anchoring a set of points to use as a skeleton for the rest of the embedding [38] but does not provide a framework to cover the spectrum from local to global.

Uniform Manifold Approximation and Projection (UMAP) [70] is a non-linear dimension reduction method that also aims to preserve local distances. UMAP allows the use of labels (or partial labels) for supervised or semi-supervised dimensional reduction tasks. A review of different variants of UMAP and its applications is summarized in [43]. While UMAP claims to preserve the global structure of the dataset better than t-SNE, we show this is not universally true for graph data in section 3.3.4.

2.2.3 Graph Embedding

The graph embedding problem aims to assign a set of low dimensional vectors to the vertices of a graph in a way that captures the graph structure. More formally, given a graph $G = (V,E)$ and a dimension $d < |V|$, find a representation of the graph as a set of d-dimensional vectors corresponding to the elements of V, which preserves some properties as much as possible [16] (e.g., pairwise distances in MDS). Aesthetic criteria are often used to evaluate the quality of a graph embedding: the number of edge crossings, average edge length, overall symmetry, etc. [86]. While aesthetic criteria are important for readability and task facilitation, it is also important to ensure that the embedding properly represents the underlying information. This refers to the concept of information *faithfulness*, that a visualization should accurately show all of the underlying data, regardless of task [77, 79] and graph embeddings provide a nice benefit by directly optimizing graph structure preservation.

Graph structure is a nebulous term; here we mean inherent properties of the underlying graph such as local/global distances. Global distance preservation methods attempt to geometrically capture the topological structure of the graph, by closely matching embedded distances to graph-theoretic distances. This is ideal for connectivity based tasks such as finding or approximating a shortest path, and reveals what the data looks like at a 'global' scale and often captures shape. Local structure preservation methods instead attempt to preserve the immediate neighborhood around each vertex, better capturing clusters or densely

connected subgraphs. While one can generally trust that vertices near each other in the embedding are similar, the relative distance between two distant vertices may be meaningless; note the long edges in the local embedding column in Fig. 2.5.

Force-directed layouts: The class of force-directed algorithms for graph layouts have achieved popularity due to their conceptual simplicity and aesthetically pleasing results [61]. These algorithms model the graph as a physical system in 2D or 3D space, computing a layout by minimizing the energy of the system. They may or may not attempt to explicitly preserve the underlying graph structure and often struggle with local minima when the graph is large.

Popular force-directed models, such as Fruchterman-Reingold's algorithm [37], define two types of forces: attractive and repulsive. Nodes attract each other if they are somehow similar, meanwhile a repulsive force between all nodes ensures node separation. The relative magnitude of these forces has a great effect on the resulting layout. The lin-log energy layout algorithm is a force-directed algorithm that does well at revealing clusters in a graph [81]. The model relies on the observation that large repulsive forces between nodes (or edges) create well separated clusters and is particularly useful for drawing dense graphs. While our method utilizes repulsion, we did not pursue a force-directed approach as we want to explicitly preserve underlying graph structures.

Graph layout by dimensionality reduction: Early work in graph layout proposed a natural evaluation; a good layout closely matches the drawn distance to the graph-theoretic distance between vertices [58]. While this can be formulated as a force-directed model, minimizing this sum of differences turns out to be very similar to MDS. This observation has been used to apply MDS to graph embedding [42]. Stress is minimized using different methods such as majorization [42] or stochastic gradient descent (SGD) [109].

The MDS approach suffers from a neccessary all-pairs-shortest-path computation, which usually relies on Floyd-Warshall's $O(|V|^3)$ algorithm, or on Johnson's $O(|V|^2 log|V| + |E||V|)$ algorithm. The maximum entropy model (MaxEnt) [40] adds a negative entropy between vertices in the graph. The motivation for MaxEnt is to provide an improvement in asymptotic complexity. Only using the edge set avoids pre-processing for shortest paths, leaving only the $O(|V|^2)$ update pattern of the optimization. The MaxEnt model attempts to place graph neighbors close together while maximizing the distance between all vertices. This is conceptually similar to the LMDS algorithm of Chen and Buja [20]. Our approach differs from the MaxEnt model in motivation: We aim to create an algorithm that can capture local structure, global structure, or balance between the two, whereas MaxEnt is primarily concerned with speed. We cannot avoid an all-pairs shortest path computation in general, and make use of SGD to optimize our objective function in lieu of majorization.

While minimizing or majorizing stress produces good layouts for many graphs, it can fail to capture local structure; see Fig. 2.5. To capture the local strudcure, the t-distributed stochastic neighbor embedding (t-SNE) [69] has been adapted to graph embedding, known as tsNET [65]. The tsNET, much like the underlying t-SNE algorithm, defines two prob-

ability distributions for pairs of vertices in the graph theoretic space and pairs of vertices in the realized 2D space. A layout is found by updating the realized distances so that the Kulback-Leibler (KL) divergence is minimized. The tsNET also adds a repulsive force between vertices to achieve cluster separation. The tsNET framework has been sped up by making use of negative sampling and sparse approximation to avoid the all-pairs-shortest-path computation [111]. Nocaj et al. [82] achieve effects similar to tsNET by weighting edges based on "edge embeddedness" (how important a given edge is to the graph structure), and perform MDS on the now weighted graph.

A metric known as neighborhood preservation (NP) has been used to evaluate locally preserving methods. NP is the average Jaccard similarity of vertex graph-theoretic neighborhoods and an embedded neighborhood of the same size [65, 111]. For NP, a perfect preservation score is 1. However, to compare this score with stress which has perfect preservation at 0, we define the Neighborhood Error (NE) as the Jaccard dissimilarity, which we define formally in Section 3.3.1.

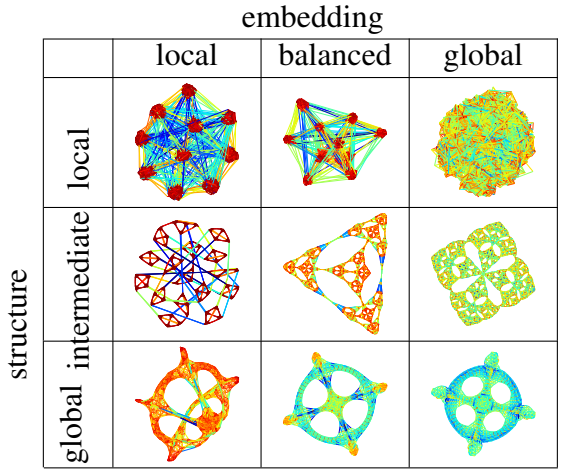


Figure 2.5: (1st row) block_2000 graph; (2nd row) sierpinksi_3d graph; (3rd row) dwt_1005 graph; (Left column) tsNET embedding; (Middle column) L2G embedding; (Right column) MDS embedding. While local embedding methods perform well on graphs with distinct local structure, they can distort the global shape of the graph. Global methods capture the shape (when it exists), but may miss important local structures. Some graphs, e.g., sierpinski_3d exhibit both local and global structure and a balanced L2G(100) embedding works best.

Chapter 3

CURRENT WORK

3.1 Graph drawing on the sphere by multidimensional scaling

Node-link diagrams are typically created by embedding the vertices and edges of a given graph in the Euclidean plane and different embedding spaces are rarely considered. Multi-Dimensional Scaling (MDS), realized via stress minimization or stress majorization, is one of the standard approaches to embedding graphs in Euclidean space. The idea behind MDS is to place the vertices of the graph in Euclidean space so that the distances between them are as close as possible to the given graph theoretic distances. Due to the nature of Euclidean geometry, this cannot always be done without some distortion (e.g., while K_3 naturally lives in 2D, K_4 does not, as there are no four equidistant points in the Euclidean plane). Moreover, some graphs "live" naturally in manifolds other than the Euclidean plane. For example 3-dimensional polytopes, or triangulations of 3-dimensional objects can be better represented on the sphere, while trees and special lattices are well-suited to hyperbolic spaces.

When visualizing graphs in Euclidean space, common techniques include adapting off-the-shelf dimensionality reduction algorithms to the graph setting. Such algorithms include the Multi-Dimensional Scaling (MDS) [94], Principal Component Analysis (PCA) [36], t-distributed Stochastic Neighbor Embedding (t-SNE) [69], and Uniform Manifold Approximation Projection (UMAP) [70]. The popularity of graph visualisation, and the fact that some of the underlying datasets are easier to embed in non-Euclidean spaces, led to some visualization techniques for spherical geometry [30, 85] and hyperbolic geome-

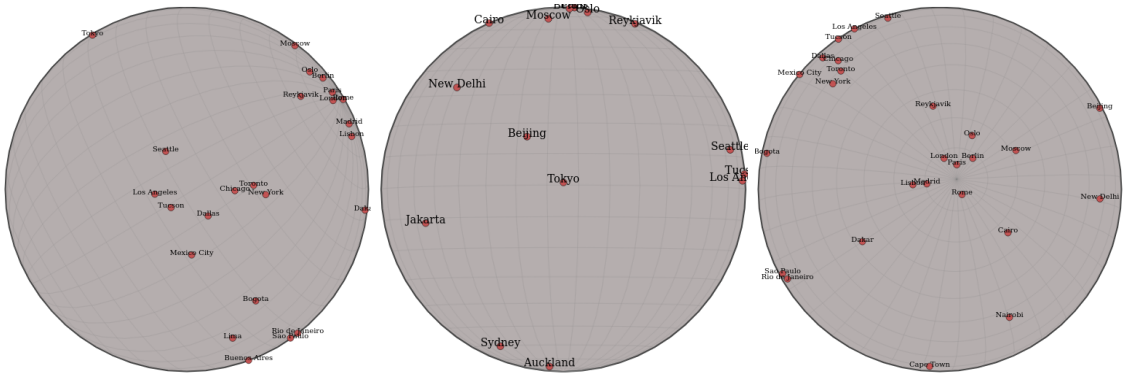


Figure 3.1: Applying spherical MDS to embed 30 cities from around the Earth (given pairwise distances between the cities). The spherical MDS recovers the underlying geometry.

try [64, 73, 92]. Most of the existing non-Euclidean graph visualization approaches, however, either lack in accuracy or do not scale to larger graphs.

With this in mind, we propose and analyze a stochastic gradient descent algorithm for spherical MDS. Specifically, we present a scalable technique to compute graph layout directly on the sphere, adapting previous work for general datasets [30] and applying stochastic gradient descent [87, 109]. We provide an evaluation of the technique by comparing its speed and faithfulness to the exact gradient descent approach. We also investigate differences in graph layouts between the consistent geometries (Euclidean, spherical, hyperbolic) by first showing that *dilation or resizing* has a large effect on layouts in spherical and hyperbolic geometry, and second by showing some structures can be better represented in one geometry than the other two. All sourcecode, datasets and experiments, as well a web based visualization tool are available on GitHub: `https://github.com/Mickey253/spherical-mds`.

Note that the proposed method is not restricted to graphs, but is applicable to any dataset specifying a set of objects and pairwise distances between them.

3.1.1 Multidimensional Scaling in S^2

Our spherical multi-dimensional scaling algorithm (SMDS) resembles that of other stress based graph layout algorithms. That is, we first compute a graph-theoretic distance matrix via an all-pairs-shortest-paths algorithm and then use this distance matrix as an input to minimize the generalized stress function and compute vertex coordinates on the sphere. This differs from standard Euclidean MDS in that Euclidean distances between points are replaced by geodesic distances between the points on sphere. The corresponding stress function defined on the sphere is

$$\sigma_S(X) = \sum_{i<j} w_{ij}(\delta(X_i, X_j) - d_{ij})^2 \tag{3.1}$$

Where $\delta(X_i, X_j)$ denotes the geodesic distance between vertices i and j, d_{ij} is the graph-theoretic distance between vertex i and j, and w_{ij} is a weight, typically set to d_{ij}^{-2}. However, one can also give preferred weights based on the importance of the points and based on the application. Another typical choice is binary weights, where w_{ij} is either 0 or 1. Unless otherwise specified, δ corresponds to the geodesic distances on the unit sphere and δ_R the geodesic distances on a sphere with radius R.

We minimize equation (3.1) by stochastic gradient descent (SGD), which we experimentally show converges in fewer iterations while achieving lower distortion than exact gradient descent for sufficiently large graphs. SGD is a minimization approach in the gradient descent family of algorithms. Fully computing the exact gradient can be too expensive and SGD instead repeatedly performs a constant time approximation of the gradient, by considering only a single term of the sum (or subset of terms for mini-batch stochastic), which allows it to make more updates. As a result, SGD tends to converge in fewer iterations while more consistently finding the global minimum [89].

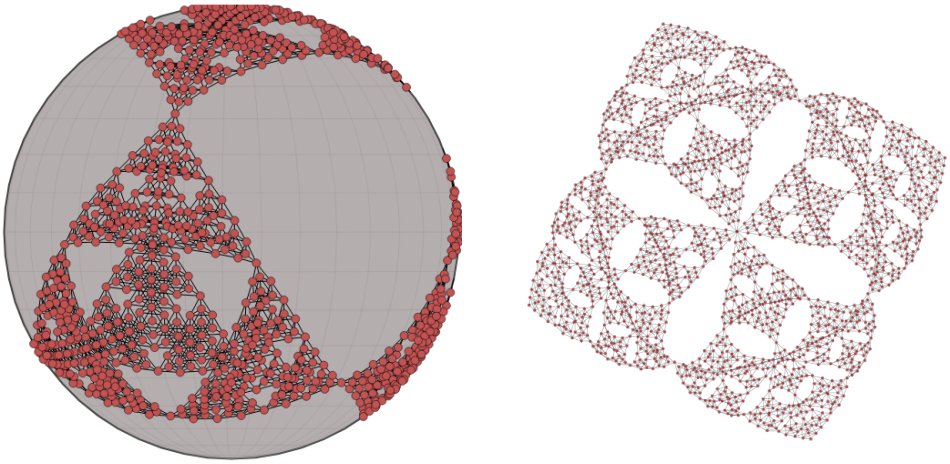


Figure 3.2: The Sierpinski3d [65] graph on the sphere (left) and in the plane (right). While the Euclidean drawing on the right is aesthetically pleasing, it looks deceptively like a 2D structure and implies a center. The sphere more accurately captures the structure.

To perform SGD on the stress function, we approximate the gradient by looking at only a single pair of vertices. Note that this corresponds to one summand of the full stress function. If we rewrite equation (3.1) as $\sigma_S(X) = \sum_{i<j} f_{i,j}(X)$ then we can more simply write the full gradient in terms of f. Apply the chain rule to see we will need to derive the partial gradient of the geodesic distance function:

$$\frac{\partial \delta(X_i,X_j)}{\partial \phi_i} = \frac{-\cos\phi_i \sin\phi_j - \sin\phi_i \cos\phi_j \cos(\lambda_i - \lambda_j)}{\sqrt{1-\cos^2\left(\delta(X_i,X_j)\right)}}$$

$$\frac{\partial \delta(X_i,X_j)}{\partial \lambda_i} = \frac{\cos\phi_i \cos\phi_j \sin(\lambda_i - \lambda_j)}{\sqrt{1-\cos^2\left(\delta(X_i,X_j)\right)}}$$

Unlike in Euclidean space, the gradient of the spherical distance function is not symmetric, i.e., $\frac{d\delta(X_i,X_j)}{d(\phi_i,\lambda_i)} \neq -\frac{d\delta(X_i,X_j)}{d(\phi_j,\lambda_j)}$. Writing out the full gradient:

$$\frac{\partial f_{i,j}}{\partial X_k} = \begin{cases} 2w_{i,j}(\delta(X_i,X_j) - d_{ij}) \left(\frac{\partial \delta(X_i,X_j)}{\partial \phi_i}, \frac{\partial \delta(X_i,X_j)}{\partial \lambda_i} \right) & k = i \\ 2w_{i,j}(\delta(X_i,X_j) - d_{ij}) \left(\frac{\partial \delta(X_i,X_j)}{\partial \phi_j}, \frac{\partial \delta(X_i,X_j)}{\partial \lambda_j} \right) & k = j \\ 0 & \text{otherwise} \end{cases} \tag{3.2}$$

We can apply SGD to equation (3.1) by selecting pairs i, j in random order, and updating X by $X - \eta \frac{\partial f_{i,j}}{\partial X_k}$ where η is the learning rate; see Algorithm 1.

We randomly initialize the placement of vertices uniformly on the sphere, as other work has shown that SGD is consistent across initialization strategies [9, 73, 109].

Algorithm 1 Stochastic gradient descent algorithm for spherical MDS

procedure STOCHASTIC GRADIENT DESCENT(d)
  $X \leftarrow$random initialization
  while $\Delta(\text{stress}(X)) > \varepsilon$ or max iterations is reached **do**
    for (i,j) in random order **do**
      $X_i \leftarrow X_i - \eta \frac{df_{i,j}}{dX_i}$, according to (3.2)
      $X_j \leftarrow X_j - \eta \frac{df_{j,i}}{dX_j}$, according to (3.2)
    end for
  end while
  return X
end procedure

3.1.2 Evaluation

We first investigate the various parameters that effect SGD's optimization, then compare our results to exact gradient descent.

Hyper-parameters There are several hyper-parameter choices to be made when using SGD. The learning rate η (also known as step size, annealing rate) has a large effect on the resulting embedding. If the learning rate is too large, the optimization will "overstep" and either fluctuate around a minimum, or diverge. If the learning rate is too small, the optimization may require many iterations to converge and is more likely to converge to a local minimum. A better strategy is to employ a learning rate schedule, where at early iterations the learning rate is large but decreases over time to allow for convergence. This is known to converge to a stationary point (could be a saddle) under certain conditions: $\sum g(t) = \infty$ and $\sum g(t)^2 < \infty$ [11].

We investigate a limited subset of possible learning rate schedules, a fixed learning rate at 0.05, a piece-wise schedule similar to that of Zheng et al. [109], a fractional decay of $\Theta(t^{-1})$, and a slower fractional decay of $\Theta(t^{-.5})$. The piece-wise schedule begins with an exponential decay function, with large initial values and switches to $\Theta(t^{-1})$ once below a threshold. There are a few changes we needed to make to the piece-wise schedule. Firstly, while Zheng et al. [109] upper bound their learning rate by 1, this upper bound is too large for SMDS. The upper bound for the Euclidean algorithm was derived from the geometric structure, and a value of 1 reduces the stress between a single pair of vertices to 0. The latitude and longitude of the sphere are angles and so do not have this property. We instead need a relatively small upper bound, noting that large movements of a pair vertices on the sphere that need to be moved apart can actually bring them closer together (by wrapping around the sphere). We investigated values in the range 0 to 1, and settled on 0.1 as it achieves low stress quickly while not being so small as to fall into local minima; see Fig. 3.3.

Randomization is a to select pairs i, j in the stochastic optimization function. While

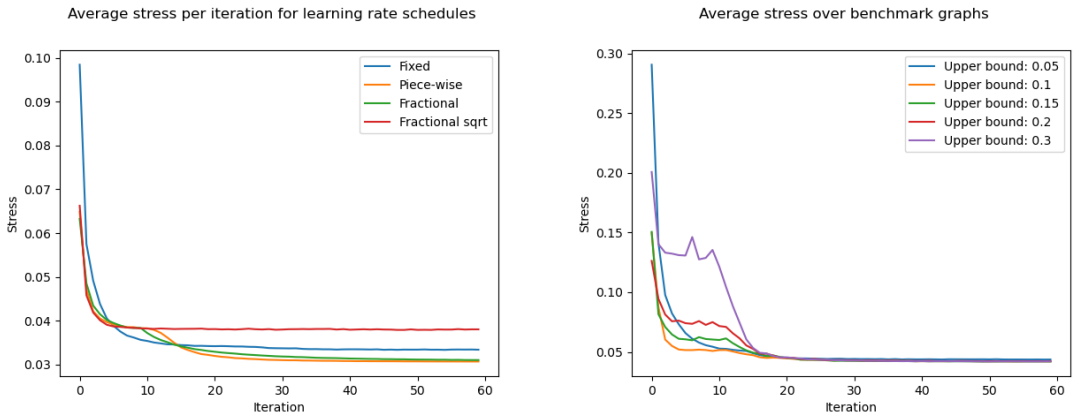


Figure 3.3: (Left) Effect of the learning rate schedule on the optimization. The piece-wise schedule adapted from [109] arrives at a minimum faster on average. (Right) Effect of the upper bound on the learning rate on the optimization. An upper bound of 0.1 behaves predictably. Values for both are averaged over all graphs in our benchmark.

SGD was originally done using sampling with replacement, random reshuffling has been shown to converge in fewer total updates [48]. To use random reshuffling in stress minimization, we enumerate all pairs $i < j$ of vertices in a ordered list and shuffle this list after every iteration. This ensures that the order in which we visit pairs is random, but that each pair is visited before we sample the same pair again.

A stopping condition is how the algorithm determines to terminate, either by converging or by reaching a maximum number of iterations. We measure convergence by tracking the change in the value of the optimization function between iterations. In the figures and evaluation results below, we set the convergence threshold to $1e^{-7}$ or a balance between speed and quality.

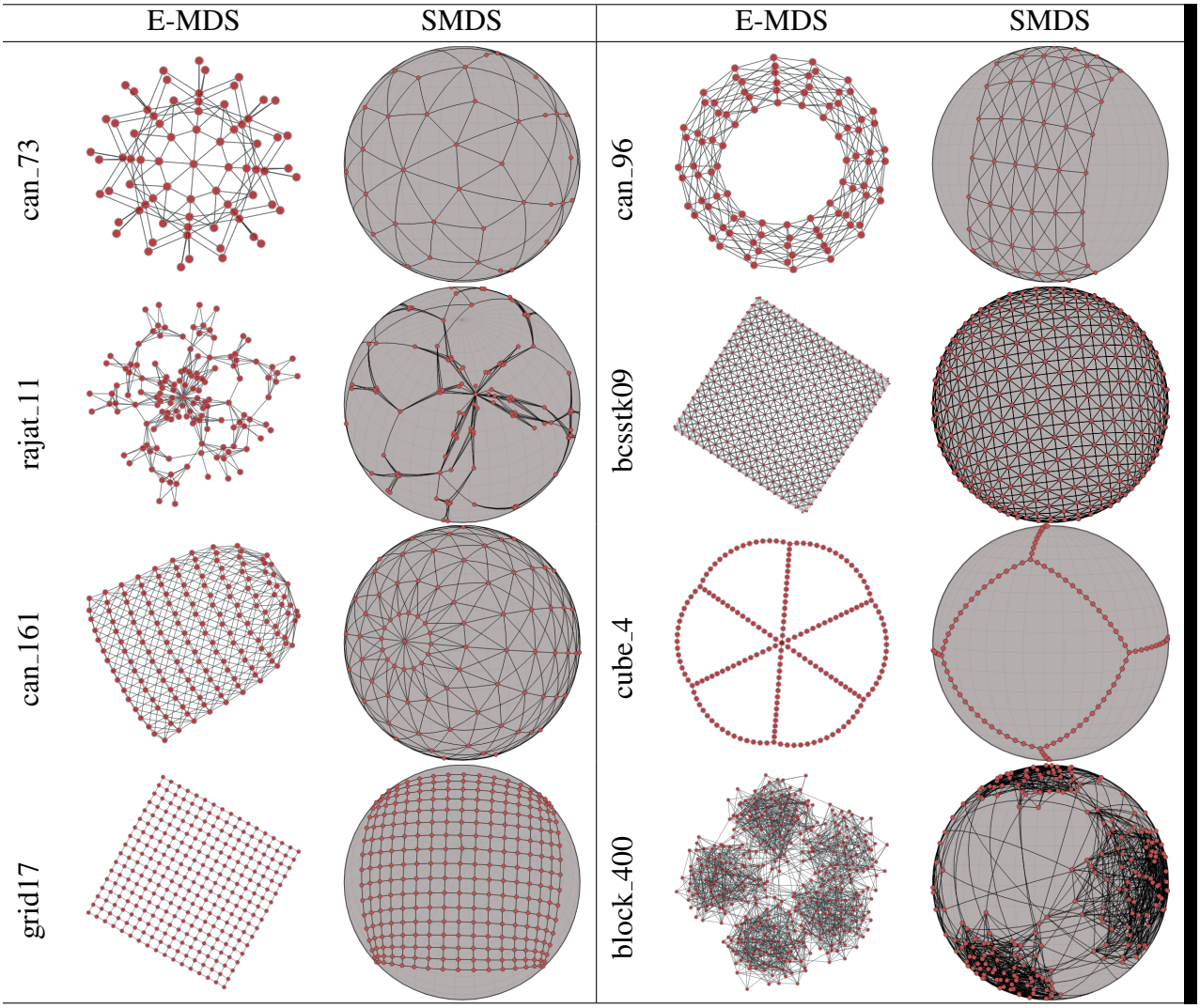
Evaluation Our code is open source and written in Python. Experiments are performed on an Intel® Core™ i7-3770 CPU @ 3.40GHz × 8 with 32 GB of RAM with a 64 bit installation of Ubuntu 20.04.3 LTS.

We use a set of 40 graphs to evaluate our SMDS algorithm: 34 from the *SuiteSparse Matrix Collection* [24] and the remaining 6 from skeletons of 3-dimensional polytopes. We use the cube, dodecahedron, and isocahedron, and subdivide them 4 times each to obtain cube_4, dodecahedron_4 and isocahedron_4. We present spherical layouts of a subset of our benchmark graphs; see Table 3.1. The remaining layouts can be found in the full paper. We see that there are several graphs particularly well suited to spherical layout: the 3D polytopes and their subdivisions have much lower distortion on the sphere than in the plane. 3-dimensional meshes and triangulations of surfaces such as dwt_307 and delaunay_n10 also have lower error on the sphere.

The SGD optimization scheme performs better than exact GD on both time to conver-

gence and stress as the size of the data becomes large as we expect; see Fig. 3.4.

Table 3.1: Layouts



3.1.3 Geometry Comparison

Here we discuss some possible drawbacks of graph embedding in spherical space and compare graph embeddings between Euclidean, spherical and hyperbolic spaces. Stress works well for producing layouts, but directly comparing stress scores between geometries are difficult to interpret. Layouts are often uniformly scaled so that stress is minimum before reporting (see [40, 65]) which works fine in Euclidean space, but becomes a problem in spherical and hyperbolic spaces. In order to more fairly compare embedding error across

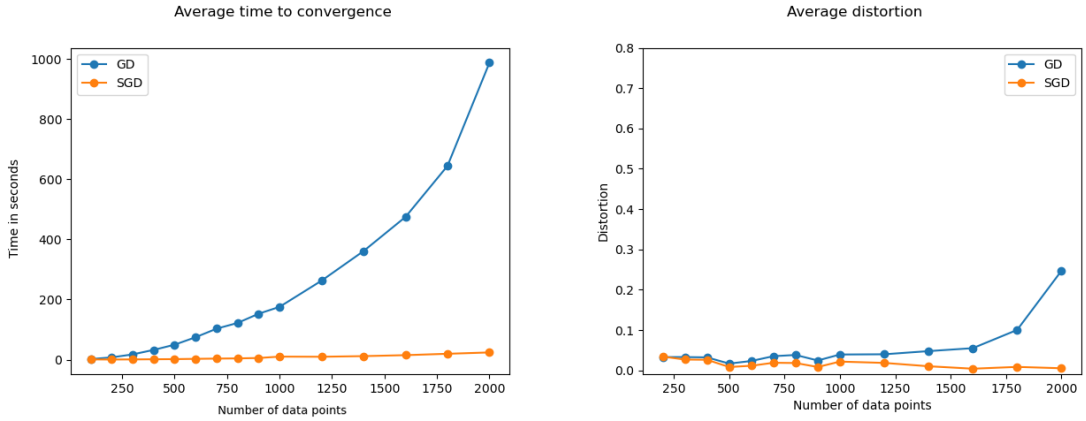


Figure 3.4: How the SGD optimization scheme fairs compares to the exact GD in terms of time (left) and error (right). The larger the size of the graph, the more benefit is seen from the use of SGD.

geometries, we use the *distortion* [73, 92] metric, defined as

$$\text{distortion}(X) = \frac{1}{\binom{|V|}{2}} \sum_{i<j} \frac{|\delta(X_i, X_j) - d_{i,j}|}{d_{i,j}}. \tag{3.3}$$

3.1.4 Dilation of Distances

It is known that Euclidean MDS is invariant to dilation, that is if one multiplies the given distances by a positive real number, the corresponding MDS solution is the original MDS solution multiplied by the same scalar factor (up to rotation). However, this is not true for spherical and hyperbolic spaces. Moreover, spherical space is bounded, unlike Euclidean space. For example, on the 2D unit sphere the maximum distance that can be achieved between two points is π (assuming that between any two points we always take the shortest geodesic distance). Any graph with diameter (longest shortest path) longer than π cannot possibly be embedded on the unit sphere with zero error. A reasonable solution is to dilate the input distances so that all the given distances are less than or equal to π. That is, to multiply the distance matrix, d, by $\frac{\max d}{\pi}$. This heuristic appears to work well in practice; see Fig. 3.6. For all of our experiments and layouts, we use this heuristic. However, this has no guarantees of being optimal.

One possible approach to the dilation problem is to make the radius of the sphere also a parameter to optimize. The problem would then become finding the best radius so that the defined stress function is as small as possible. This can be captured by reformulating Eq. (3.1) to also optimize the radius:

$$\arg \min_{R, X_1, \ldots X_n \in S^2_R} \sum_{i,j=1}^{N} \left(\delta_R(X_i, X_j) - d_{ij}\right)^2. \tag{3.4}$$

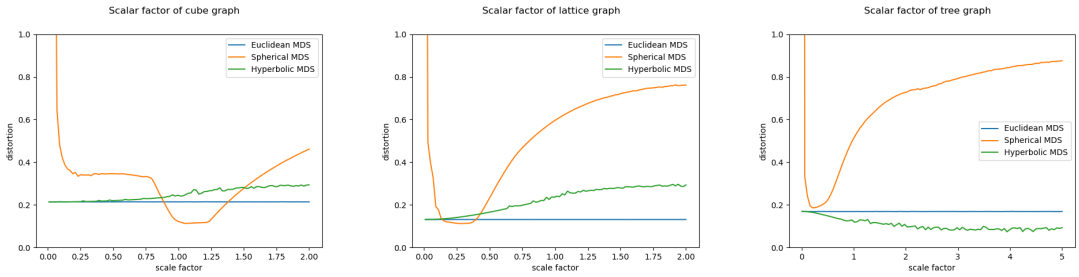


Figure 3.5: Behavior of distortion on selected graphs with respect to dilation factor in each geometry.

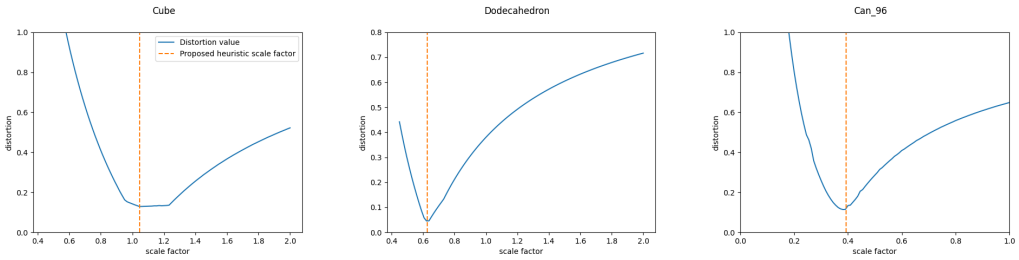


Figure 3.6: Effect of dilation on distortion. Our proposed heuristic (orange line) is often very close to the minimum (of the blue curve).

Here $\delta_R(X_i, X_j)$ corresponds to the geodesic distance on the sphere with radius R between points X_i and X_j. We derive the gradient for R and update it along with the vertex positions at each update step.

To the best of our knowledge none of the existing algorithms for spherical embedding consider this dilation/resizing problem. However, we believe that it is a crucial parameter while embedding/drawing a graph on the sphere.

Choosing Between Geometries One reason to consider embedding graphs on different manifolds (Euclidean, hyperbolic, spherical) is to be able to preserve and visualize important properties of the given graph. Some graphs achieve lower distortion on the sphere, others in hyperbolic space. In this section we investigate how spherical graph layouts differ from other consistent geometries. We choose a selection of graphs from the sparse matrix collection, and lay them each out using the Euclidean, spherical, and hyperbolic variants of MDS and measure the distortion. We repeat the layout 5 times each, and report the average distortion for each graph in each geometry. We make use of [109] for the Euclidean MDS implementation and [73] for the hyperbolic MDS (HMDS) implementation.

The hypothesis we test here is that some graphs have a dramatically lower distortion in a particular geometry. For instance, rectangular lattices can be embedded with constant error in Euclidean space [101], regular 3D polytopes can be thought of as tesselations of the sphere, and trees have been described as "discrete hyperbolic spaces" [64]. The

	E-MDS	SMDS	HMDS
cube	0.21494	0.11287	0.28334
isocahedron	0.23256	0.03905	0.2612
dodecahedron	0.21762	0.04262	0.48258
dwt_72	0.12862	0.14406	0.20609
can_73	0.21831	0.20114	0.35937
lesmis	0.22548	0.25536	0.1959
can_96	0.19802	0.11498	0.41292
rajat11	0.18752	0.20923	0.25801
can_144	0.11089	0.07133	0.28803
can_161	0.18532	0.16809	0.26348
dwt_162	0.09281	0.08898	0.32336
cube_4	0.20793	0.05715	0.54074
jazz	0.24723	0.25819	0.23984
dwt_221	0.08002	0.08593	0.50755
visbrazil	0.17988	0.20071	0.21925
grid1_dual	0.17266	0.18365	0.47681
grid17	0.10002	0.09049	0.28645
dwt_307	0.21835	0.11481	0.82275
dwt_361	0.07689	0.08097	0.51507
netscience	0.17617	0.19067	0.38224
block_400	0.27969	0.27782	0.25253

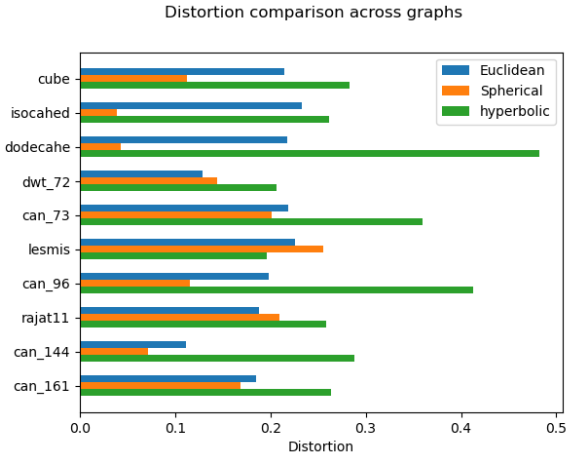


Figure 3.7: The left subfigure shows a subset of results from the direct comparison for distortion in Euclidean, spherical and hyperbolic space. The right subfigure plots the first 10 rows. We note that 3D polytopes and meshes (the can graphs) are particularly well suited to the sphere, the LesMis graph is a complex network which is best embedded into hyperbolic space, and Euclidean space is better for the remaining ones.

results are summarized in Fig. 3.7 with additional data in the full paper. We observe that spherical geometry is in fact able to embed polytopes and 3D meshes with lower distortion. Further, hyperbolic geometry is able to embed networks with "small-world" properties such as lesmis and block_400 with lower distortion. In graphs with 2D structure, Euclidean space is the clear winner.

In Fig. 3.8 we go beyond graphs to verify the different nature of the three geometries. We sample points randomly from each space, and use these points to define the distance matrices. We expect the corresponding geometry's MDS to embed the data with much lower distortion and this is indeed the effect we see.

3.1.5 Conclusions and Future Work

We described an efficient method for embedding graphs in spherical space. The method generalizes beyond graphs to embedding high-dimensional data. We studied (quantitatively and qualitatively) the difference between spherical embeddings of graphs and embeddings in Euclidean and hyperbolic spaces. We discussed the issue of dilation and proposed an approach that seems to work well in practice. Furthermore, we compared how structures are preserved in different geometries. The algorithm is implemented and fully functional and we provide the source code, experimental data and results, and a web based visualization tool on GitHub: `https://github.com/Mickey253/spherical-mds`.

While our proposed algorithm is much faster than exact gradient descent (5 seconds for

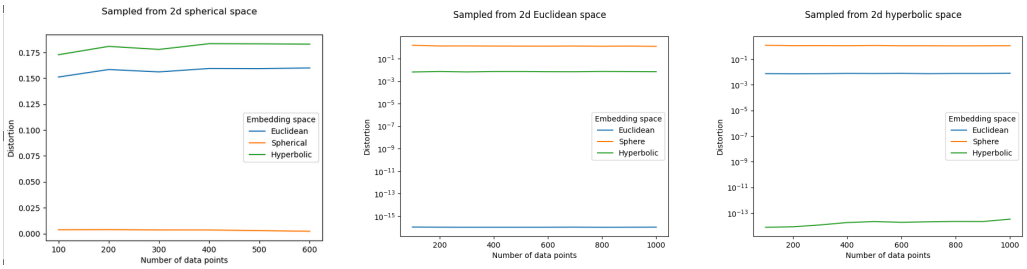


Figure 3.8: Results from sampling data uniformly at random from each consistent geometry: as expected SMDS, MDS and HMDS perform dramatically better on data that comes from the geometry it embeds in.

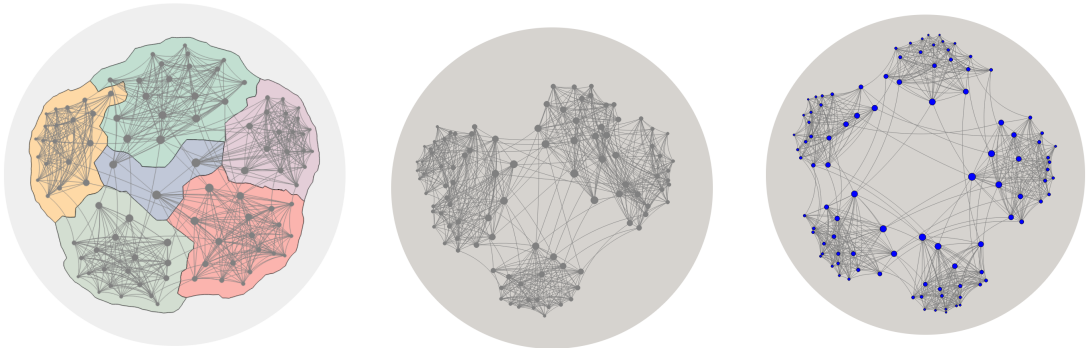


Figure 3.9: Example layouts of the same graph, generated by the three hyperbolic graph embedding algorithms discussed in this paper: inverse projection (left), force-directed (center) and hyperbolic-MDS (right).

a 1000-vertex graph), it still requires an all-pairs-shortest-paths computation as a preprocessing step, which cannot be done faster than quadratic time in the number of vertices. This is a bottleneck computation for any graph-distance based approach and coming up with a strategy (e.g., sampling a subset of distances) is a problem whose solution can impact many existing algorithms. Another direction for future work is to quickly determine the best embedding space for a given graph. That is given a graph, decide the best manifold to embed it in: Euclidean, spherical or hyperbolic. We considered stress and distortion measures here, but exploring other graph drawing aesthetics across different geometries seems to be a worthwhile direction to explore.

3.2 Graph Drawing in Hyperbolic Geometry

Node-link representations of graphs in the 2-dimensional Euclidean plane are the most typically used graph visualizations. The structure of many graphs, notably planar graphs, can be realized well in the plane, but others are better represented in non-Euclidean geometries. For example, 3-dimensional polytopes are well represented in spherical space, while large

hierarchies such as trees can be cleanly embedded in hyperbolic space. Standard hyperbolic projections into Euclidean space also provide a natural 'focus+context' view of the graph, with parts of the graph near the center of the view shown large and those far from the center progressively smaller, with the entire graph being in the view.

A recent work [64] suggests that hyperbolic geometry underlies complex networks, in a similar way as spherical geometry underlies geographic data.

Though there has been some work on visualizing hierarchies using hyperbolic space in the browser [46], there are no tools that support browser-based hyperbolic visualization of general graphs.

We describe three methods for laying out graphs in the 2-dimensional hyperbolic space, H^2. The first method relies on taking a pre-computed Euclidean layout of a graph and projecting it into hyperbolic space, providing standard map interactions, such as pan, zoom, re-center, click and drag. We implement this method in a web based system that provides several layout algorithms for node-link and map-based visualization. This allows us to view and interact with GMaps, MapSets, BubbleSets, and LineSets in hyperbolic space. The second method makes use of a generalization of force-directed algorithms to Riemannian geometries [62]. We exploit the locally Euclidean properties of hyperbolic space so that with the help of Möbius transformations we can accurately model the forces. In particular, this approach allows us to compute layouts where distances between nodes in hyperbolic space correspond to the underlying graph-theoretic distances between them. The third method attempts to directly realize graph distances in H^2 through a hyperbolic generalization of multidimensional scaling (MDS) [104, 23]. For this method we adapt stochastic gradient descent (SGD) to hyperbolic space, as SGD has been shown to be efficient and produce high-quality layouts in Euclidean space [109]. To the best of our knowledge, there are no prior methods to adapt Euclidean layouts to hyperbolic space, nor any hyperbolic SGD approaches. All three methods are available online. The projection method is available through GMap at `http://gmap.cs.arizona.edu`. The other two methods are available as a webapp on GitHub at `https://github.com/Mickey253/hyperbolic-space-graphs`.

3.2.1 Projection-based Method

The first method we present is based on the idea of starting with a precomputed layout and projecting it to the hyperbolic plane. The implementation is available on the web, in a browser based graph visualization system. The system offers several layout algorithms, clustering algorithms and visualization styles, with a focus on map-like representations such as GMaps [39], MapSets [29], BubbleSets [22], and LineSets [1]. Several human-subject studies suggest that such map-like visualizations are at least as good as traditional node-link diagrams when it comes to task performance, memorization, and recall of the data [90, 91].

The Euclidean layouts are computed then saved in the graphviz DOT file format which includes graph-wide attributes, a node list, and an adjacency list [**?**]. GMap computes the layout and stores node positions as Cartesian coordinates. This is sufficient to draw node-

link diagrams. Polygons given as a set of vertices are stored as a graph-wide attribute along with their colors for the other map-like layouts: GMaps, MapSets, BubbleSets and LineSets. Parsing the polygons is done as in [85].

The system relies on two different layout algorithms for computing a Euclidean layout: *sdfp* is a multi-level force-directed algorithm [53] and *neato* is a implementation of the Kamada-Kawai algorithm [57]. We show examples of the *colors graph* drawn as a node-link, GMap, BubbleSet, and LineSet diagram; see Fig. 3.10. This is a graph of the 38 most popular RGB colors, courtesy of xkcd[1].

We make use of a javascript library called Hyperbolic Canvas [4]. It is a mathematical model of the Poincaré disk projection of hyperbolic space that allows lines and shapes to be drawn using an HTML canvas. The projection-based pipeline below is based on the approach by Perry *et al.* for browser-based visualization of graphs on the sphere [85].

The Projection-based Pipeline Given a pre-computed 2-dimensional Euclidean layout, the projection-based method can be summarized as follows:

1. Calculate geometric mean of the 2-d Euclidean layout
2. Apply an inverse hyperbolic Lambert azimuthal projection centered on the geometric mean
3. Project back into the Euclidean plane of the browser using the Poincaré projection (providing the look and feel of hyperbolic space).

Hyperbolic Projections: It is well known that non-Euclidean spaces (such as spherical and hyperbolic spaces) can not be perfectly projected to the Euclidean plane. No matter what type of projection is used, something will get lost in the translation: distances are distorted, or region areas are distorted, or angles are distorted. This problem is well studied in cartography in the context of projecting the sphere onto the 2-d Euclidean plane.

Knowing that a perfect embedding in the plane is impossible, useful maps can still be created by choosing which information to preserve. Three well-known projections of the sphere are gnomonic, orthographic, and stereographic projections. The gnomonic projection preserves straight lines; geodesics of the sphere are shown as straight lines in the projection. This is particularly useful in flight planning, and is said to be the oldest map projection. The orthographic projection resembles the view of the Earth from space, and preserves scale at the center of the projection, making it useful in visualization. Finally, the stereographic projection preserves angles and has its roots in star charts used in sailing [95].

Hyperbolic surface is curved (negatively) just like spherical space is curved (positively), resulting in similar problems when attempting to display it in the plane of a monitor or on a piece of paper. Just as the sphere has many projections that serve different purposes, so there exists many hyperbolic projections to the plane, although they are not as well studied. These projections can be thought of as analogous to their spherical counterparts

[1] https://xkcd.com/color/rgb/

and can often be derived in an analogous way. For instance, the Beltrami-Klein projection is analogous to the gnomonic projection of the sphere; they both preserve geodesics as straight lines. Similarly, the Gans model of the hyperbolic plane is analogous to the orthographic projection, in that they both have a point of perspective at infinity. The Poincaré projection is a spherical analogue of the stereographic projection as they both preserve angles.

Hyperbolic Lambert Azimuthal Projection Since we know we are projecting node-link and map diagrams, it seems reasonable to choose to preserve areas. One way this can be accomplished is through a less common hyperbolic analogue to the Lambert azimuthal projection, which has been called the hyperbolic Lambert azimuthal projection. This projection is equi-areal, so area is preserved. The hyperbolic analogue can be derived in much the same way as the sphere.

Consider two disks: one in the 2-dimensional Euclidean space and the other in the 2-dimensional hyperbolic space. Denote the area of the Euclidean disk of radius r as $e(r)$ and the area of a hyperbolic disk of the same radius $h(r)$. We can then define the function $f(r)$ such that $e(r) = h(f(r))$. Assuming unit curvature, then

$$h(r) = 2\pi(cosh(r) - 1)$$

$$e(r) = \pi r^2$$

$$f(r) = arccosh(\frac{1}{2}r^2 + 1)$$

where *arccosh* is the inverse hyperbolic cosine. This maps the Euclidean plane to the hyperbolic plane and gives us the transformation $(r, \theta) \rightarrow (f(r), \theta)$, which preserves areas, but distorts angles and shapes. The further away a shape is from the projection center the greater the distortion, so centering about the geometric mean reduces this effect; see Fig. 3.11.

Poincaré Projection Recall that the Poincaré projection of the hyperbolic plane is similar to the stereographic projection of the sphere, in that it preserves angles. The infinite hyperbolic plane is mapped to the inside of the unit disk with hyperbolic lines corresponding to either arcs of circles orthogonal to the boundary of the disk, or diameters of the disk if the line passes through the origin. The Poincaré disk intrinsically provides the look and feel of hyperbolic space in the browser. The 'focus+context' mentioned before is due to the Poincaré projection. A small area near the border of the disk represents a very large area in hyperbolic space, while the same size area near the center of the disk represents a small area of hyperbolic space. This can be seen mathematically in the transformation that takes the hyperbolic plane to the Poincaré disk

$$(r, \theta) -> (\frac{e^r - 1}{e^r + 1}, \theta)$$

The exponentiation in the Poincaré transformation implies a practical limit on the hyperbolic radius of about 700, as larger values require dealing with large numbers and lead to numerical overflow.

Visualization Considerations In this section we discuss the interactive features, the parameters, and the task considerations.

Navigating the Map: One of the main reasons for using map-like visualization for graphs is our familiarity with map interactions such as pan, zoom, click and drag. In the Poincaré disk, clicking and dragging brings new nodes and regions into focus, allowing the viewer to exploit the 'focus+context' property of the projection. We accomplish this by making use of Möbius transformations.

A Möbius transformation is a complex function of the form $f(z) = \frac{az+b}{cz+d}$ where z is a complex variable and $ad - bc \neq 0$. Möbius transformations have many uses in complex analysis and geometry, but one subgroup is especially useful for our purposes; the class of transformations that map the open unit disk to itself. In particular the transformation

$$f(z) = \frac{z - z_0}{-\tilde{z_0}z + 1}$$

takes z_0 to the origin and preserves the Poincaré projection of the hyperbolic plane, i.e., the transformation recenters the Poincaré projection at z_0.

We can obtain transitions that look smooth to the human eye by repeatedly applying the above transformation at a point some ε distance from the previous origin in the direction the mouse is being dragged. Two still images centered at different points in a random graph are shown in Fig. 3.13, but interacting with the actual visualization in GMap better conveys the idea. Fig. 3.14 additionally shows the difference between panning to the edge of a map in Euclidean space and hyperbolic space.

3.2.2 Force-directed Method

Our projection-based hyperbolic visualization method uses a precomputed 2-dimensional Euclidean layout, but it uses hyperbolic space just for the visualization and 'focus+context' effect, rather than for the actual graph embedding. Properly embedding the graph in hyperbolic space would allow us to take advantage of the underlying hyperbolic geometry. Algorithms for directly embedding special classes of graphs in hyperbolic space, such as trees and hierarchies, can better take advantage of the properties of the space and obtain better embeddings than via projections. It is also possible to modify the standard force-directed algorithm for operation in Riemannian geometries (such as hyperbolic and spherical) by taking advantage of the locally Euclidean properties of such spaces [62]. The implementation, which provides visualization in the browser, and is made available in a browser based system through GitHub.

The idea is to compute a tangent plane at each vertex embedded in the non-Euclidean Riemannian space, mapping every other vertex to that plane, performing a step of a force-directed algorithm in the plane, and projecting back the resulting node position changes to the Riemannian space. While conceptually simple, this method allows the graph to make use of the properties of the corresponding non-Euclidean geometry.

For instance, on the sphere, layout methods that correctly make use of the geometry allow 3D polytopes to wrap 'around' the sphere. Thus, compared to the plane, a more accurate realization of their structure is possible; see method two of [85].

We apply this idea to the Kamada-Kawai type of force-directed graph layout algorithm, for its conceptual simplicity and its desirable property of capturing graph structure (e.g., graph distances between pairs of nodes) in the embedding (e.g., realized distances between pairs of nodes in the non-Euclidean space). Specifically, we compute the graph theoretic distances between all pairs of nodes and these define desired distances in the layout. Spring forces, proportional to the squared Euclidean distance between nodes in the layout, are used to gradually improve a given initial layout to one in which realized distances match the graph theoretic distances [57]. Formally, there is an attractive or repulsive force (similar to stress) defined for any pair of edges based on the difference between the graph theoretical distance and the realized distance in the current embedding. Specifically, the total energy of the system is modeled as:

$$E = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{1}{2} k_{ij} (|p_i - p_j| - d_{ij})^2$$

where given a pair of nodes i and j, d_{ij} is the graph theoretic distance between them, $|p_i - p_j|$ is the current realized distance in the embedding between them, and k_{ij} is the strength of the spring forces between them. The layout is obtained by reducing the energy of the system via gradient descent.

Tangent Plane In order to compute a tangent plane at some node x in H^2, we need to set the distance between x and every other node in the plane to the hyperbolic distance between them, and ensure the angle between the nodes stay the same [62]. The Poincaré disk preserves angles, so we only need to map hyperbolic to Euclidean distances. In the Poincaré model, hyperbolic distance is simplest to compute from the origin, so we first apply the Möbius transformation that takes x to the origin. The distance between x and any node y is

$$d_h(x,y) = ln(\frac{1+|y|}{1-|y|}) = 2arctanh\,|y|$$

where arctanh is the inverse hyperbolic tangent. Then, let x be the center of the disk and for every node y, let the transformation $(|y|,\theta) \rightarrow (d_h(x,y),\theta)$ be its location in the tangent Euclidean plane, using polar coordinates.

Once the tangent plane is computed and a step of the force-directed algorithm has completed, the central node must be placed back into hyperbolic space. This is accomplished through an inverse of the above equations. Let y' be the new location of the moved node. We apply the transformation

$$(|y'|,\theta) -> (tanh\,\frac{|y'|}{2},\theta)$$

The following Möbius transformation takes the disk back to its former origin, z_0

$$f^{-1}(y) = \frac{-y - z_o}{-\tilde{z_o}y - 1}$$

Precision While hyperbolic geometry poses many interesting challenges for graph drawing, the most notable we encountered was the issue of precision. It is well known that floating point numbers are not arbitrarily precise and that this can cause problems when the number of significant bits needed is large. This effect is pronounced on the Poincaré disk, as the number of bits needed to accurately reflect the hyperbolic position increases exponentially as one approaches the border of the disk. We choose to trade accuracy for stability: using the Euclidean coordinates of the unit disk, if a node is pushed to within 0.001 of the border, the node is 'pulled back' to avoid errors from 'ideal' points of magnitude ≥ 1. This effectively creates a bounding region defined by a circle of (Euclidean) radius 0.999 from the center of the Poincaré disk. This precision could be increased to allow for larger nodes extremely far away from the center, or decreased to keep the visual distortion of the drawing small.

Maps Once we have computed a layout for a node-link diagram, we can compute a map-like representation for it by projecting it to the plane and running the existing GMap/BubbleSets/MapSets/L algorithm to obtain the needed groups and polygons.

It should be possible to compute the map-like representations directly in hyperbolic space. For example, the cluster regions (polygons) for GMaps are computed using Voronoi diagrams and it has been shown that Voronoi diagrams for 2-dimensional points generalize to hyperbolic space and can be computed in $O(n \log n)$ time [80]. Similarly, LineSets requires Bezier curves between nodes in a cluster, which should also be computable in hyperbolic space.

3.2.3 Multidimensional Scaling in H^2

In the force-directed approach, we compute the graph embedding with the help of many tangent plane computations, so that we can use the standard force computations in Euclidean space. Here, we consider a simple embedding: hyperbolic multidimensional scaling (H-MDS).

Recall that metric multidimensional scaling is a dimensionality reduction technique that attempts to preserve relationships between n data points by finding a set of n points in the target space whose distances match observed distances. MDS can be naturally formulated as a graph drawing problem by computing the pairwise distances through an all-pairs-shortest-paths computation. Metric MDS is then typically solved by minimizing an objective function. The most common function used is known as stress

$$Stress = \sum_{i<j} w_{ij}(\|X_i - X_j\| - d_{ij})^2 \tag{3.5}$$

where d_{ij} is the given distance between two nodes in the graph, $\|X_i - X_j\|$ is the distance between them in the target space, and w_{ij} is a normalization factor (typically 1 if the given distances are of the same order, or d_{ij}^{-2} if the given distances include both very large and very small distances). The stress function is non-convex and classic optimization techniques are not guaranteed to find the global optimum. However, existing techniques, such as gradient descent, stochastic gradient descent and stress majorization, achieve sufficiently good results in practice.

Early approaches for H-MDS suggest using gradient descent [103]. However, gradient descent is too slow in practice even for relatively small graph sizes. We adapt stochastic gradient descent (SGD) to provide reasonable runtimes for the browser.

The SGD algorithm considers random pairs of nodes, calculates the minimum distance the pair needs to be moved to realize its observed distance d_{ij}, then steps along this direction by a distance proportional to the learning rate. As we aim to find an embedding in hyperbolic space, in order to evaluate the stress function and perform gradient descent, we need to choose an appropriate coordinate system. We use a coordinate system known as Lobachevsky coordinates in order to solve the H-MDS problem via SGD. Lobachevsky coordinates are defined as a pair of real numbers (x, y), where for a given point, x is its distance along a geodesic horizontal axis and y is its perpendicular distance to that axis. Lobachevsky coordinates for hyperbolic space are analogous to Cartesian coordinates for the Euclidean space. For example any pair of real numbers represents a point in hyperbolic space and any point in hyperbolic space can be represented by a pair of real numbers.

Parameters The SGD algorithm depends on several parameters and tuning these parameters can have non-trivial impact. Here we discuss these parameters and how we set the default values.

Randomization

Computing the stress function for a given embedding requires $O(|V|^2)$ time by definition; see (3.5). Thus the gradient computation also requires quadratic runtime per iteration. SGD allows for better runtimes in practice, using constant-time computations at each step by only calculating the gradient of a specific pair at a time, although this pairwise gradient calculation needs to be performed many more times. The classic SGD method samples the original data with replacement [87]. In our case, this would mean choosing two nodes at random every step until complete. On the opposite end, a method known as random reshuffling enumerates all possible data subsets (in our case all pairs) and shuffles this list. This ensures that while the order of pairs moved is random, each pair is guaranteed to be moved once in a fixed number of steps. Under certain conditions, random reshuffling outperforms and converges faster than classical replacement [48] and has been shown to work well for Euclidean SGD [109]. A third method known as index shuffling randomizes the indices in place, and pairs are chosen from this new ordering.

We investigate these three randomization methods in the context of H-MDS: classical sampling with replacement, shuffling of indices, and random reshuffling. We demonstrate

that random reshuffling tends to reach the lowest stress values; see Fig. 3.16. We use random reshuffling and define an *iteration* as a full pass through all pairs and show, in Section 3.2.3, that we only need a constant number of iterations.

Initialization As the stress function (3.5) is non-convex, there are no convergence guarantees for gradient descent or for SGD.

Recent work has shown that 'smart initialization' is not necessary for SGD in Euclidean space, as the algorithm is consistent regardless of initial embedding [9]. To see if this holds true for hyperbolic space, we performed a small-scale analysis on a selection of graphs (chosen from the sparse matrix collection [24]) and compared our smart initialization to random initialization.

Knowing the Euclidean algorithm is good at escaping local minima, we run Euclidean SGD on the graph for 5 iterations, then project this layout into hyperbolic space to obtain our smart initialization. Random initialization is obtained by placing each node uniformly at random in a circle of hyperbolic radius 1. While we initially saw small improvements, there was no statistically significant benefit of smart initialization over using a random initialization, confirming the results of [9].

Learning Rate Another important parameter for SGD is the learning rate. At each iteration of gradient descent, we move the value being optimized along the steepest direction of the gradient by a size proportional to the learning rate, η. If the learning rate is very small, the algorithm might take too long to converge; if the learning rate is too large, the algorithm might not converge. Thus, the proper choice of learning rate is crucial for both the accuracy and the speed of the algorithm.

Generally, it is good for η to be large for the initial steps to move the system quickly to a lower energy configuration, but η should tend toward zero as the number of iterations increases so that the algorithm converges. Computing a good η is a research topic all on its own and is important for SGD's effectiveness [89, 9]. We upper bound the product $\eta w_{ij} \leq 1$ as in [109]. This allows us to use a larger initial rate to 'jump' out of bad neighborhoods and possible local optima, but still converge as η goes to 0. We set a maximum and a minimum learning rate, a function $s(t)$ that outputs a learning rate η at time step t_i. $s(t_0) = \eta_{max} = d_{max}^2$ and $s(t_{max} = \eta_{min} = \varepsilon d_{min}^2$ where d_{max} and d_{min} correspond to the longest and shortest shortest paths of the input graph, respectively.

Euclidean SGD works particularly well with an exponential decay learning rate [109]. To test if hyperbolic SGD behaves the same way, we compare this exponetial decay learning rate with two additional schedules: $\Theta(1/t)$ and $\Theta(1/\sqrt{t})$ schedules. We define the exponential schedule according to [109] using $\eta_{max}e^{-bt}$, the traditional $\Theta(1/t)$ as $\frac{a}{1+bt}$ and the $\Theta(1/\sqrt{t})$ schedule as $\frac{a}{\sqrt{1+bt}}$. We set $a = d_{min}^2$ and $b = -(t_{max})log\frac{\eta_{min}}{\eta_{max}}$.

As expected, the $\Theta(1/t)$ schedule struggles to step out of local minima. It is somewhat surprising that the $\Theta(1/\sqrt{t})$ schedule appears to achieve lower minima for some classes of graphs; see Fig. 3.17. This could be due to the function's larger learning rates allowing the system to avoid local minima.

Stopping Condition Gradient descent algorithms terminate either if they converge or if

they reach a maximum number of iterations. The convergence is reached when the change in objective function value is less than some tolerance. However, computing the stress value at each iteration is time consuming and we avoid doing this for SGD. Instead, we measure the max change in pairwise distance per iteration.

For our web focused application, we primarily investigate the use of fixed number of iterations, although one can select to iterate until convergence under 'advanced options.' We set $t_{max} = 20$ using the exponential learning rate described above, after experimenting with different input graphs. We observe that there is little improvement after 20 iterations; see Fig. 3.19

Evaluation Similar to SGD for Euclidean space, we see similar improvements in time and quality using SGD in hyperbolic space. Experiments were conducted using a desktop machine with an Intel Core i7-3770 CPU @ 3.40GHz x 8 processor, 32 GB of memory, and NVidia GeForce gt 640 graphics running Ubuntu 20.04.3 LTS. Both the GD and SGD algorithms are implemented in Python, making use of the Numpy, Graph-tool, and Numba libraries.

As mentioned in section 3.2.3, while the overall complexity of SGD is no different than GD, the run time is significantly faster; see Fig 3.16. We conduct this experiment by generating a single random graph on n nodes, then computing an embedding using the classic GD and SGD, and recording the average time over 30 runs. Each graph of n nodes has $3n$ edges selected at random. At 500 nodes, GD takes over a minute but SGD takes only about 1.5 seconds.

Consistent with the findings in Euclidean space, hyperbolic SGD also performs better than GD in regards to quality; see Fig. 3.19. We show a selection of 4 graphs from the sparse matrix collection [24] and plot the stress minimization curves as each algorithm proceeds. Often just a few iterations of SGD is enough to 'untangle' the layout and the curve often bottoms out quite quickly.

3.2.4 Scale Invariance

It is known that the Euclidean MDS is invariant to scale. That is, given a distance matrix and its corresponding embedding by MDS, if one scales all distances by the same scalar and applies MDS to the scaled distances, the achieved embedding should be the scaled version of the initial one. However, this property does not hold for spherical-MDS (S-MDS) and H-MDS. This can perhaps be most intuitively seen by looking at the non-Euclidean analogues of the Pythagorean theorem (assuming unit curvature).
Euclidean: $a^2 + b^2 = c^2$,
Spherical: $cos(a) + cos(b) = cos(c)$,
Hyperbolic: $cosh(a) + cosh(b) = cosh(c)$.
While we can multiply both a and b by the same constant k to obtain k^2c^2 in Euclidean space, the same property does not hold for hyperbolic and spherical spaces.

So then, our objective function for H-MDS becomes

$$Stress = \sum_{i<j} w_{ij}(gdist(X_i, X_j) - \alpha d_{ij})^2,$$

where the $gdist((X_i, X_j))$ is the geodesic distance in hyperbolic space between nodes X_i and X_j.

Spherical space is even more problematic when considering embedding scales, as for any given radius of the sphere, the maximum distance that one can achieve on the sphere is finite (rather than infinite in Euclidean and hyperbolic space). This leads to a natural heuristic scale value: $\alpha = \frac{\pi}{d_{max}}$, where d_{max} is the diameter (longest shortest path) of the graph. This normalizes d to a maximum distance of π, which is the longest distance possible on the unit sphere.

In the hyperbolic space, although one can achieve arbitrarily large distances, similar to the S-MDS, scaling the data or considering a different hyperbolic radius can drastically affect the embedding. Thus, there is a need to find an appropriate scaling parameter α for which the achieved embedding best captures the underlying graph distances. If α is very small, the layout occupies a small fraction of the hyperbolic space, resulting in an embedding that is similar to Euclidean space, and thus does not capture the focus+context effect. If α is large, then most of the graph is located at the periphery, making it hard to see. We can find a good scaling parameter for any given graph using binary search for the value of α that achieves lowest embedding distortion and this is indeed an available option under 'advanced options.' We show an example of an optimized α compared to a naive $\alpha = 1$; see Fig. 3.20. By default we set $\alpha = \frac{10}{d_{max}}$, where d_{max} is the length of the longest shortest path in the graphs. This caps the largest distance to a hyperbolic unit length of 10 and the resulting embeddings tend to capture the focus+context effect and do not place large parts of the graph near the periphery.

3.2.5 Discussion, Limitations and Future Work

We described three methods for visualizing graphs in hyperbolic space, which are illustrated in Fig. 3.9. We present a small-scale comparison of the three approaches by comparing time and distortion values; see Fig. **??**. The projection-based method allows us to show any 2D Euclidean graph representation in hyperbolic space, where we can take advantage of the 'focus+context' properties of the space while still relying on standard map interactions. Related work has been limited to standard node-link representations, but this method can be applied to *any* graph visualization metaphor, which we show with GMaps, MapSets, BubbleSets, and LineSets. The method currently relies on Lambert azimuthal projections and the Poincaré disk model. We have not yet explored other projections or the Beltrami-Klein model. Finally, this method does not fully take advantage of the underlying geometry of the space.

The inherent distortion of shapes and angles introduced when using the inverse Lambert projection to the hyperbolic plane implies that at some threshold the outer regions of the

layout become too distorted to be of use. This is already apparent in the MusicLand example from GMap as shown in Fig. 3.14, with around 250 nodes. Even though our method can handle larger graphs, it is clear that larger graphs pose additional challenges. A multi-level representation of the graph might be useful to provide 'semantic zooming' where we start with a high level overview of the graph and zooming in brings up more details, following Schneiderman's mantra (overview first, zoom and filter, details on demand).

When moving through a curved space, an inherent property causes an observer to incur rotation. This could be desirable, as it gives several different perspectives on the same layout, but it could potentially be confusing when navigating large maps. Specifically, moving the layout in the Poincaré disk, incurs a rotation in the layout (clockwise or counter-clockwise): consider translating a layout some fixed distance up, the same distance to the right, then again down, and back to the left. In 2D Euclidean geometry, the layout would be identical after these transformations, while in the Poincaré disk (and hyperbolic geometry in general) this causes a 90-degree rotation. An orientation correcting transformation could be applied after translating the layout, but in our prototype we only provide the 'reset button,' which restores the original layout.

The force-directed method utilizes the geometry of hyperbolic space, but is not as efficient as our projection-based method. The underlying Kamada-Kawai algorithm is already rather computationally expensive, due to the all-pairs shortest path calculations and many tangent plane computations. There are several scalable force-directed algorithms for Euclidean space, which can be adapted to the hyperbolic setting, but this remains as future work.

Our third method lays out a graph directly into the hyperbolic plane using H-MDS, a generalization of multidimensional scaling. The algorithm is implemented, fully functional and available online on GitHub. In order to optimize the stress function of H-MDS we employed stochastic gradient descent, which not only significantly improves the runtime, but often finds a better minimum when compared to gradient descent. H-MDS also requires an all-pairs-shortest-paths computation, but relatively few iterations. Adapting a sparse approximation method for graphs in which the pre-processing is prohibitively expensive could be a direction for future work.

In this work we visualized the hyperbolic space by using the Poincaré disk model, as it provides the look and feel of hyperbolic space. Other models such as the Beltrami-Klein model or Poincaré half-plane model may provide additional benefits for visualization.

While we have addressed the computational scalability of hyperbolic layouts with hyperbolic SGD, visual scalability remains an open problem. Recent work has pointed to limitations on hyperbolic graph embeddings [28, 31], but it is also known that some graphs can be embedded in hyperbolic space with lower error [64] Determining whether a lower distortion in a geometry corresponds to better task support remains a promising direction for future work.

As discussed in Section 3.2.4, scaling is crucial for hyperbolic and spherical embeddings and a robust algorithm to efficiently determine the correct scaling parameter for H-

MDS and S-MDS is needed. We provide an efficient heuristic for setting the scale and provide a more computationally expensive method to find a scale that minimizes distortion, but a closed form solution remains an open problem.

A possible promising application for hyperbolic/spherical visualization are virtual reality and augmented reality, as prior work seems to have only considered spherical space in this context [67].

A potentially interesting question is how the hyperbolic geometry may change the layout's aesthetic properties. Wang *et al.* optimize edge orientation to better facilitate navigation tasks on graphs using a fish-eye lens [105]. Perhaps optimizing over additional aesthetic criteria could improve the readability of hyperbolic graph layouts.

3.3 Graph Drawing between local and global scales

Graphs and networks are a powerful tool to encode relationships between objects. Graph embeddings, which map the vertices of a graph to a set of low dimensional vectors (real valued coordinates), are often used in the context of data visualization to produce node-link diagrams. While many layout methods exist [96], dimension reduction (DR) techniques have had success in providing desirable layouts, by capturing graph structure in reasonable computation times. DR methods are used to project high-dimensional data into low-dimensional space and some of these methods only rely on the relationships between the datapoints, rather than *datapoint coordinates* in higher dimension. These techniques are applicable for both graph embeddings and visualization.

As the "best" embedding algorithm depends on the the graph structure, it is difficult to automate the process without resorting to trial and error which is time consuming and inefficient. With this in mind, we present a Local-to-Global (L2G) method that provides a a framework that spans the spectrum from local embedding to global embedding. L2G relies on a parameter, which determines the level of local-global structure that we aim to preserve. Smaller values of this parameter result in embeddings where the local structures are preserved and larger values of this parameter results in global structure preservation. L2G also allows us to explore the local/global trade-off between the two extremes, highlighting cases where there is a meaningful middle ground. We introduce a new metric called *cluster distance* to measure how well this intermediate structure is preserved. Everything described in this paper is available on Github: `https://anonymous.4open.science/r/L2G-6304`. We provide a video and additional layouts and analysis in supplemental material.

3.3.1 The Local-to-Global (L2G) Algorithm

Local embedding methods such as t-SNE aim to preserve local neighborhoods, while global embedding methods such as MDS attempt to capture all pairwise distances. We propose the Local-to-Global (L2G) algorithm that achieves the following 3 goals:

G1 A single parameter that controls the balance between local and global embeddings

G2 When this parameter is small, the resulting embedding preserves local neighborhoods

G3 When this parameter is large, the embedding preserves the global structure

In this paper, by "local neighborhood" of a vertex we refer to the immediate neighbors of the vertex being considered. If the nearest neighbors of each vertex in embedding match well with the nearest neighbors in the actual graph, we will say that the embedding accurately preserves the local structures. By "global structure" we refer to the preservation of all pairwise graph distances (including long ones) in the embedding. Finally, "intermediate structure refers to capturing both local neighbors and global structure. Fig. 2.5 shows graphs exemplifying local, intermediate and global structures and Sec. 3.3.1 defines formal embedding measures: neighborhood error, cluster distance, and stress.

In Sec. 3.3.1 we describe how to select such a balance parameter k and an objective function such that the solution of the proposed objective function behaves according to the goals described above. To meet **G1**, we modify the distance preservation of MDS to preserve distances in a neighborhood of varying size, where the locality measure is defined according to parameter k. Thus, when we preserve distances for large neighborhoods (e.g., every other point) we necessarily satisfy **G3**. This leaves a question for **G2**: Does applying distance preservation to a subset of pairs result in locally faithful embeddings?

Adapting Stress Minimization for Local Preservation We define a parameter, k, that represents the size of a neighborhood surrounding each vertex. A straightforward approach would involve simply selecting the k-nearest vertices for every given vertex (as in [20]). However, the graph-theoretic distance in an undirected graph is a discrete measure, which can create complications. For example, consider the local structure graph (top row) in Fig. 2.5. Although the within-cluster density is high, there are many edges between different clusters. A given vertex may have many neighbors within a cluster, but is also likely to have one or two out-of-cluster neighbors all of which are the same graph-theoretic distance away. Unfortunately, there is no simple way to test if an edge is within cluster or out-of-cluster. In order to produce tsNET-like embeddings, which should pay more attention to local structures, we must avoid preserving such out-of-cluster edges.

Instead of directly considering the distance, for each vertex we find its top k most connected vertices based on the following hypothesis: If there are many possible walks between two vertices, it is more likely that these two vertices are similar to each other (in the same neighborhood or cluster); see an example in Fig. 3.23. Even though vertices v_A and v_B do not share an edge, they have the same set of neighbors. If vertex v_A and vertex v_B are both neighbors to a set of vertices (e.g., v_c, v_d, etc.), then we can say with greater confidence that v_A and v_B are similar; their similarity is "confirmed" by their shared proximity to v_c, v_d, etc. [34].

To formalize the discussion above we use a well-known result from spectral graph theory – the c-th power of an adjacency matrix $(A_G^c)_{i,j}$ encodes the number of c-length walks from vertex i to vertex j. We can use this property to find the top k "most connected"

vertices for each vertex by the following procedure. Given an adjacency matrix of an undirected graph, A_G, raise it to the c-th power, taking the sum of all powers $\mathbf{A}^* = \sum_{\mathbf{1}\leq \mathbf{i} \leq \mathbf{c}} \mathbf{A}_\mathbf{G}^\mathbf{i}$, we obtain a matrix whose (i, j)-th element tells us the number of walks from i to j of length less or equal than c. Since each row in $\mathbf{A}^*$ corresponds to a vertex in G, we find the k largest values in row i (e.g., by sorting). We define these top k vertices to be the "most connected" neighborhood $N_k(v_a)$ of vertex v_a; see Fig. 3.23 for an illustration. To not overvalue long walks, we weight the power of the matrix with a decaying weight factor, $s, 0 < s < 1$, such that $A^* = \sum_{1\leq i\leq c} s^i A_G^i$. We investigate a limited range of values for s, and set $s = 0.1$; see the supplemental material for results.

When the number of vertices is large, computing the powers of $\mathbf{A}_G$ is time-consuming, as matrix multiplication is costly. We propose the following procedure to reduce the number of matrix multiplications to compute $\mathbf{A}^* = \sum_{1\leq i\leq c} \mathbf{A}_G^i$. For a given symmetric affinity matrix $\mathbf{A}_G$ we compute its eigendecomposition $[\mathbf{Q}, \Lambda] = eig(\mathbf{A}_G)$, where the rows of $\mathbf{Q}$ contain the eigenvectors and the diagonal elements of Λ contain the corresponding eigenvalues of $\mathbf{A}_G$, $\lambda_1, \lambda_2, \ldots, \lambda_n$. Next, we use the well-known fact from linear algebra, that for any $1 \leq i$, $\mathbf{A}_G$ and $\mathbf{A}_G^i$ share the same set of eigenvectors with corresponding eigenvalues $\lambda_1^i, \lambda_2^i, \ldots, \lambda_n^i$. Next, we observe that,

$$\mathbf{A}^* = \sum_{i=1}^{c} \mathbf{A}_G^i = \sum_{1\leq i\leq c} s^i \mathbf{Q}\Lambda^i\mathbf{Q}^T = s^i\mathbf{Q}\left(\sum_{i=1}^{c}\Lambda^i\right)\mathbf{Q}^T$$

Computing A^* naively requires c matrix multiplications, but in this way we only use two. We remark, that Λ is a diagonal matrix and one can compute its i-th power simply by considering the diagonal matrix Λ^i with diagonal entries $\lambda_1^i, \lambda_2^i, \ldots, \lambda_n^i$.

Note that only preserving distances of a subset of pairs will result in poor embeddings: e.g., two vertices that cannot "see" each other can be placed arbitrarily close with no penalty. A second term is needed in the objective function to prevent this, and we add an entropy repulsion term as in [20, 40], to force pairs of vertices away from each other.

Objective Function For a given the pairwise distance matrix $\left[d_{ij}\right]_{i,j=1}^{n}$ we define the following generalized stress function as an objective function:

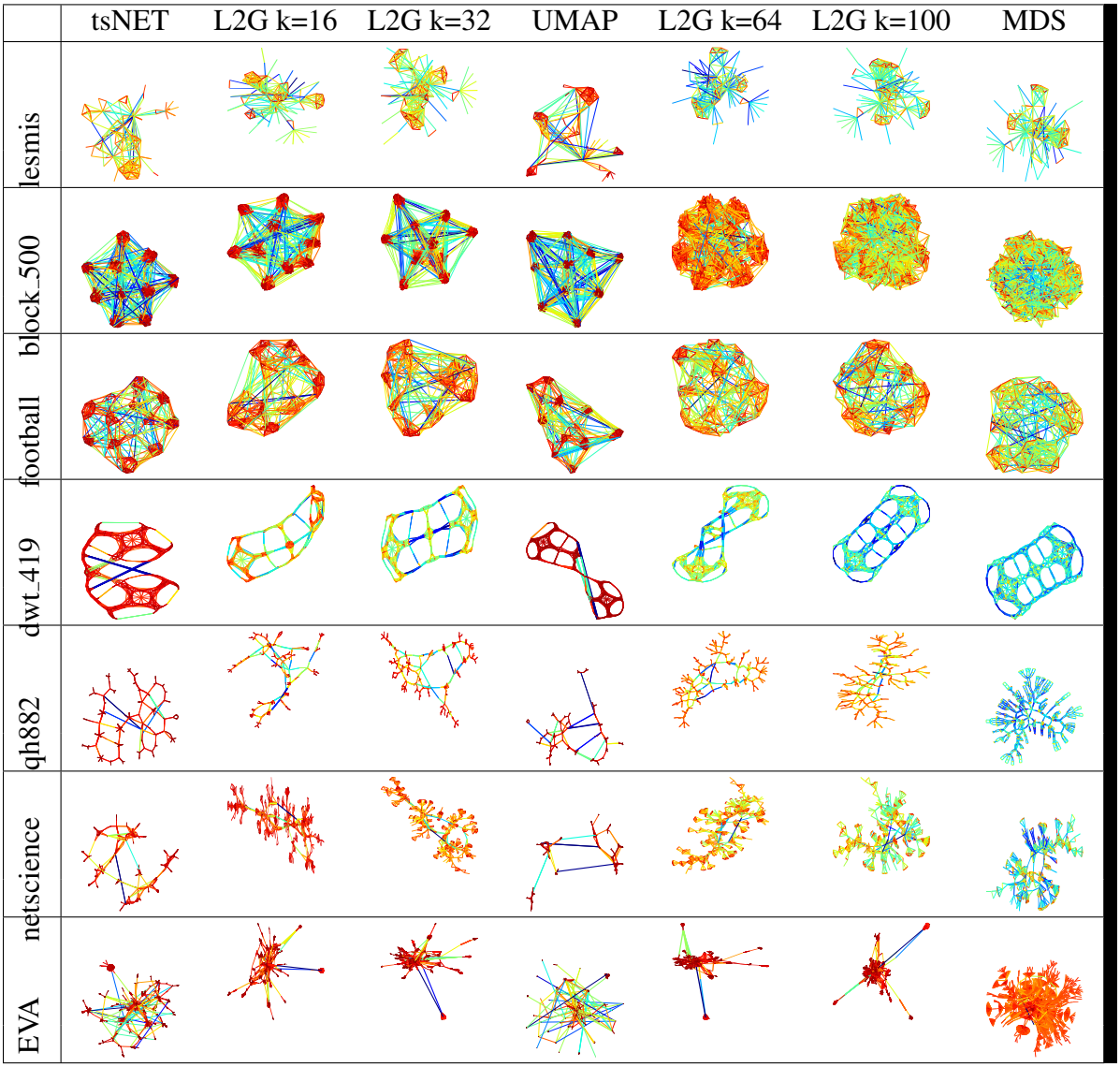
$$\sigma(X) = \sum_{(i,j)\in N_k} (\|X_i - X_j\| - d_{ij})^2 - \alpha \sum_{(i,j)\notin N_k} ln\|X_i - X_j\|, \tag{3.6}$$

where X_i is the embedded point in $\mathbb{R}^d$, α is a fixed constant parameter that controls the weight of the logarithmic term, and N_k corresponds to the neighborhood that we aim to preserve in the embedded space.

This objective function ensures that distances are preserved between the most-connected neighborhoods, while maximizing entropy. We use the negative logarithm of the distance between points, so that the repulsive force is relatively strong at small distances, but quickly decays (so that distant points are not forced to be too distant from each other). While similar to LMDS [20] and MaxEnt [40], the proposed objective function in Eq. 3.6 differs in (1)

how the set N_k is selected (LMDS uses a k-nearest neighbor search and MaxEnt preserves distances between two vertices if and only if they share an edge) and (2) LMDS and MaxEnt cannot be easily parameterized to balance local and global structure preservation. We minimize the objective function by SGD which works well for stress minimization [109, 9].

Table 3.2: Example embeddings. The first and last columns show the two extremes tsNET (local) and MDS (global); the middle column shows UMAP. The remaining columns show a gradual increase of L2G's k parameter, moving from local to global distance preservation (left to right).



Optimization Parameters

The parameters that we use to optimize the objective function (3.6) by using SGD include: **Learning rate**, **Stopping condition** and **Randomization**. We start by defining an 'epoch' as a complete partial gradient calculation of all $\binom{n}{2}$ pairs of vertices in random order.

For the learning rate, while an exponential regime was shown to work well for SGD in the full stress model, the added entropy potentially allows for exploding gradients. However, we did not observe this to become a problem on the graphs that were tested with L2G, as the objective function stabilizes within 10-15 epochs. We adopt a learning rate schedule similar to that of [109]: large learning rates initially, with an exponential decay schedule, followed by a $\Theta(1/t)$ schedule, to satisfy the necessary properties for converge to a stationary point [12].

The stopping condition for L2G is triggered when either a pre-specified maximum number of epochs is reached, or when convergence occurs before reaching the maximum number. We experimentally determine the default maximum number of epochs to be 60. To declare that the algorithm has converged we need a specific convergence threshold value. As we are using SGD, we want to avoid computing the full cost function (which is computationally expensive). Therefore, we record the maximum distance any vertex moves in an epoch and when it falls below a certain threshold over the entire epoch, we declare convergence. By default, we set this convergence threshold to a small value 10^{-7}.

Randomization refers to the method used to select the order of pairs updated in an epoch, which has a non-trivial effect on the resulting embedding. While sampling with replacement was originally used for SGD schemes [51], it has recently been shown that random reshuffling leads to higher quality in fewer epochs [48], so we implement random reshuffling for L2G.

L2G for High Dimensional Data We can extend the idea of L2G from graph data to high dimensional data. Similar to t-SNE [69] and UMAP [70], for a given dataset we can construct a weighted graph that represents the given dataset. The vertices of the graph correspond to datapoints and pairwise as

$$w_{ij} = e^{-\frac{d^2(x_i,x_j)}{\sigma^2}}$$

where σ^2 is a dataset specific constant (can be the sample variance or just set to be 1) and $d(x_i,x_j)$ is the Euclidean distance between the i-th and j-th datapoints. We further normalize the weights so that the rows of the corresponding weight matrix add to 1.

This leaves us with a matrix where the weights for small distances approach 1 and weights for large distances approach 0. We can interpret these as the probability of randomly moving from datapoint i to j in one 'hop.' Similar to a graph adjacency matrix, the c^{th} power of the matrix represents the probability of making that jump in c hops. We can then apply the L2G method using the transformed probability matrix in place of the adjacency matrix; see more details in Sec. 3.3.3.

Evaluation Metrics Here we discuss the metrics used to evaluate the embedding algorithms. We have a measure for local neighborhood error (NE) score, another for intermediate structure (CD), and one for global distances (Stress).

NE Metric: Neighborhood hits (NH) measures how well an embedding preserves local structures [20, 35]. Specifically, NH is the average Jaccard similarity of the neighbors in the high-dimensional and low-dimensional embedding. Let Y be an $n \times d$ dimensional dataset, X be its $n \times 2$-dimensional embedding, and a radius r defines the size of the neighborhood one intends to measure; then NH is defined as:

$$NH(Y,X,r) = \frac{1}{n}\sum_{i=1}^{n} \frac{|N_Y(p_i,r) \cap N_X(p_i,r)|}{|N_Y(p_i,r) \cup N_X(p_i,r)|} \tag{3.7}$$

where $N_Y(p_i,r)$ denotes the r nearest points to point p_i in Y and $N_X(p_i,r)$ the r nearest points to point p_i in X.

For graph embeddings, this notion is called neighborhood preservation (NP) [65, 111, 40], with the main difference being that the radius r now refers to graph-theoretic distance: all vertices with shortest path distance $\leq r$ from vertex v_i. Specifically, NP measures the average Jaccard similarity of a vertex's graph-theoretic neighborhood of radius r and an equally sized neighborhood of that vertex's closest embedded neighbors. Since NH and NP measure accuracy, it is desirable to maximize these values. To facilitate comparison with the other two metrics (where lower scores mean better embeddings), we use Jaccard dissimilarity instead and refer to it as Neighborhood Error (NE).

Cluster Distance Metric: We develop a new metric to measure how well intermediate structures are captured in an embedding. Motivated by the fact that distances between clusters in t-SNE cannot be interpreted as actual distances [106], while clusters in MDS embeddings are often poorly separated, we measure how faithful the relative distances between cluster centers are represented in the embedding. When actual cluster information is given as part of the input (e.g., labels, classes), we can use them to define distances between the clusters. When cluster information is not given, we can compute it with k-means clustering in the high-dimensional data case, or with modularity clustering in the graph case. The distances between clusters in the high-dimensional case is given the Euclidean distance between the cluster centers. In the graph case, we measure the distance between each cluster by first taking the normalized count of edges between them, then subtracting the normalized count to convert similarity into dissimilarity. This gives the cluster-distance matrix, δ. Let $C_1,\ldots,C_n$ be the set of vertices belonging to cluster $1,\ldots,n$, then

$$\delta_{i,j} = 1 - \frac{1}{|E|}\sum_{u \in C_i, v \in C_j} \mathbb{1}(u,v \in E)$$

Once δ is computed, we can compare how well clusters are represented in the embedding. We compute the geometric center of each embedded cluster (using the same clustering as before) and compute the cluster-level stress between the graph-level-cluster distances and

realized-cluster distances. This measure will be small when similar clusters (that share many edges) are placed near each other and dissimilar clusters are placed far apart. Formally, the cluster distance (CD) is:

$$CD(\delta,\chi) = \sum_{i,j} \left(\frac{\delta_{i,j} - ||\chi_i - \chi_j||}{\delta_{i,j}} \right)^2 \tag{3.8}$$

where $\delta_{i,j}$ is the dissimilarity measure between cluster i and cluster j and χ_i is the geometric center of cluster i in the embedding.

Stress Metric:

Normalized stress has been used in many earlier global-distance-based approaches. It is defined as follows:

$$\text{stress}(d,X) = \sum_{i,j} \left(\frac{d_{i,j} - ||X_i - X_j||}{d_{i,j}} \right)^2 \tag{3.9}$$

where d is the high dimensional Euclidean distance, or the graph-theoretic distance matrix, and X is the given embedding. In order to fairly compare across all embedding algorithms, we scale their embeddings to the same size before computing stress, as in [40, 65, 111].

3.3.2 L2G Embedding of Graphs

We start with a visual analysis and discussion of layouts produced by L2G. Following the convention, several embeddings of the same graph are displayed side-by-side with increasing the value of k from left to right, going from local to global. Underneath the L2G embeddings, we place t-SNE, UMAP, and MDS embeddings of the same graph.

For all graph embeddings provided in this paper, we use the jet color scheme to encode edge length. An edge length of 1 (ideal for unweighted graphs) is drawn in green, while red indicates that edge has been compressed (length < 1) and blue indicates the edge is stretched (length > 1). This makes clusters easy to spot as bundles of red edges, and global structure preservation apparent when most edges are green.

Similar to tsNET, low values of k capture local neighborhoods well, by allowing some longer edges. As a result, clusters tend to be well separated. Note that tsNET allows even longer edges in an embedding, occasionally breaking the topology; see Fig. 3.22. Higher values of k result in L2G behaving similar to MDS, where edges tend to be more uniform in length. As a result global structures (e.g., mesh, grid, lattice) can be seen, but clusters might be missed.

Grid_cluster is a synthetic example with 900 vertices (9 clusters of size 100) and 10108 edges, created to illustrate the notion of cluster distance preservation. It is an instance of a stochastic block model (SBM) graph. Within cluster edges are created with probability 0.8. We distinguish between two types of out-of-cluster edges. Clusters are first placed on a lattice. Out-of-cluster edges are created with probability 0.01 if they are adjacent in the lattice (no diagonals) and 0.001 otherwise. The layouts of this graph are in Fig. 3.24. Note

the visual similarities between L2G(32) and tsNET; both seperate each cluster into dense sub-regions and place them seemingly randomly in the plane. Also note the similarities between L2G(200) and MDS, where both methods tend to miss the clusters. UMAP also fails to capture the intermediate structure built into this network: while there is a single cluster placed in the middle of the other eight, the surrounding shape is not a square. Meanwhile, L2G(100) is able to accurately place each cluster in the appropriate position, making it the only one that clearly shows the 3x3 underlying lattice. Although the clusters are not as dense and separable as the more local drawings, they are more faithful in terms of cluster placement.

Connected_watts_1000 is a Watts-Strogatz random graph on a 1000 vertices and 11000 edges. It first assigns the vertices evenly spaced around a cycle with the nearest (k=7) vertices connected by an edge. Then, with low probability, some random 'chords' of the cycle are added by rewiring some of the local edges to other random vertices. This type of graph models the small-world phenomenon (small average shortest path length) seen in real-world examples, such as social networks. The embeddings of connected_watts_1000, obtained by L2G, tsNET, UMAP, and MDS are are in Fig. 3.22. We observe that both tsNET and UMAP embeddings accurately capture the existence of a one dimensional structure, but twist and break the circle to varying degrees. Meanwhile, MDS overcrowds the space, forming a classic 'hairball' where there is no discernible structure. For intermediate values of k in L2G, the circular structure in the data and the numerous chord connections become clearly visible.

Sierpinksi_3d models the Sierpinski pyramid with 2050 vertices and 6144 edges – a finite fractal object with recursively smaller recurring patterns (the pyramid itself is built out of smaller pyramids). These fractal properties are ideal for showcasing the L2G algorithm at work, as small local structures build upon each other to create a global shape. We observe that tsNET captures the smallest structures well but places them arbitrarily in the embedding space. UMAP does better at placing the local structures in context but still creates long edges and twists not present in the data. While MDS visually captures the fractal motifs, it 'squishes' local structures. L2G can be used to balance these extremes.

fpga is a circuit simulation network with 1220 vertices and 2807 edges from the SuiteSparse matrix collection [24], and contains dense interesting local structures, arranged in a circular manner like spokes of a wheel. tsNET captures these clusters but misses their circular arrangement, placing them arbitrarily, as shown in Fig. 3.27. Similarly, L2G also misses the circular arrangement for low values of k. UMAP packs the clusters very tightly, but still misses this regular placement of the clusters. While MDS finds the circular backbone, it spreads out the individual clusters, making it less clear there are indeed dense clusters. L2G(150) finds a good middle ground for this graph. The clusters are obvious (note the red edges) and they are correctly places around the circular backbone.

btree9 is an example of a graph (binary tree with 1023 vertices) without an optimal balance, but one that allows L2G to show the gradual transition from smaller to larger clusters; see Fig. 3.28. tsNET finds many small clusters, while MDS creates a radial layout.

L2G starts with 16 small clusters, and as k increases we see 8 clusters, then 4 clusters, and so on, capturing the recursive nature of the underlying graph.

We demonstrate several other examples in Table 3.2, with additional embeddings available in the supplemental material. Note that for lower values of k the embedding obtained by L2G visually resembles the output of tsNET, while for larger values of k the obtained embedding is more similar to the outputs of MDS. We see this reflected numerically in many graphs; see Fig. 3.25 and Table 3.3. For small values of k the NE value is small (local preservation) and similar to that of tsNET and UMAP. With larger values of k L2G transitions close to MDS with low stress (global preservation). For intermediate values of k, L2G often outperforms tsNET, UMap and MDS with respect to the intermediate structure preservation, measured by cluster distance (CD).

3.3.3 L2G Embeddings of High-Dimensional Data

To illustrate L2G on non-graph data, we use a high-dimensional dataset. Specifically, we consider the Human Activity Recognition (HAR) [3] dataset: a collection of experimental data of subjects performing 6 different tasks (walking, walking upstairs, walking downstairs, sitting, standing, laying) while wearing a smartphone sensor to collect data. We use the pre-processed data available from [35] which has 735 samples with 561 features (dimensions). A typical task for this type of data is to identify which physical activity is being performed based on the data. We use L2G to produce a low-dimensional (2D) embedding where the activity classes can clearly be seen, and where we can also see the similarity between the classes.

The embeddings shown in Fig. 3.30 suggest the presence of three distinct clusters within the data: one related to walking activities, another comprising of sitting and standing, and a final one representing lying down. L2G successfully captures the similarity between the clusters corresponding to sitting/standing and laying by placing them near each other without merging them, as observed in MDS. Interestingly, t-SNE and UMAP do not show any clear organization of the three clusters.

There are a small number of outliers in the data, shown in green. They are distinctly apparent and separated in all of the L2G embeddings, and we can also see the points individually due to local distance preservation. Of the other algorithms, only UMAP makes these outliers apparent and even then UMAP condenses these data to an overlapping point so it is not as visually clear.

We can verify that L2G captures this intermediate structure appropriately by observing the metrics; see Fig. 3.29. L2G manages to be no worse NE than MDS and no worse than UMAP in stress, but is able to achieve consistently lower CD values on the HAR dataset. For additional experiments with real-world high dimensional datasets see the supplemental material.

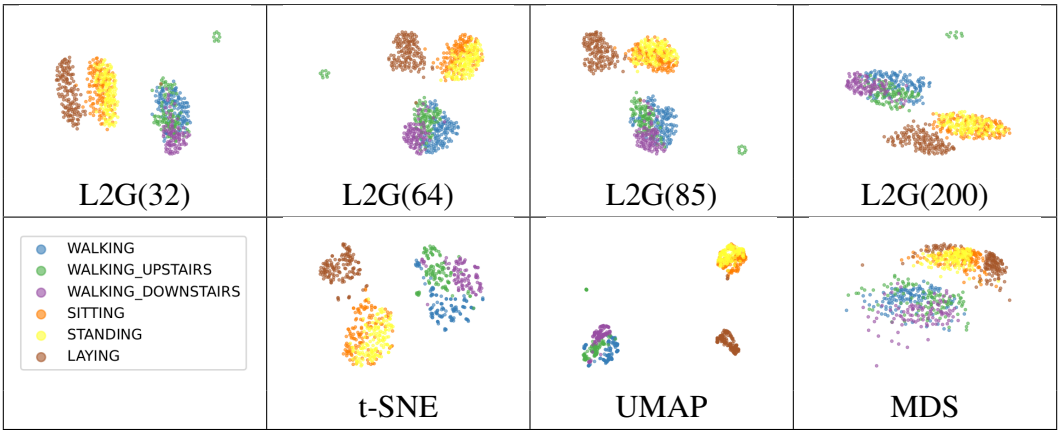


Figure 3.30: Embeddings of the HAR dataset by L2G (Top) and t-SNE, UMAP, MDS (Bottom). MDS mixes the sitting/standing and laying clusters, but all other methods capture the three separable clusters present in the data. Note that t-SNE and UMAP clusters are placed arbitrarily far apart, but L2G correctly places the sitting/standing and laying clusters nearby but separable.

3.3.4 Evaluation

We test the L2G algorithm on a selection of real-world graphs from the SuiteSparse Matrix collection [24] and a subset used in [65, 111]. We additionally evaluate a selection of synthetic graphs with high density or clustering coefficients. The full list of graphs we evaluated can be found in the supplemental material. We numerically demonstrate the application of L2G algorithm in two different scenarios (a) graphs with existing global structures such as meshes and (b) graphs with existing local neighborhoods such as graphs with clear clusters.

Algorithmic Parameters We investigate L2G's parameters c, as defined in Sec. 3.3.1 and α, as defined in equation (3.6), that have an impact on our objective function: $\mathbf{c}$, the power we raise the adjacency matrix to, in pre-processing and α, the repulsion term's strength in the modified stress equation. The power c, has a non-trivial impact on the layout. We experimented with the graphs in our benchmark to determine suitable values for c, and found that large values have diminishing returns on the quality of the layout; see Fig. 3.31 (Top). With this in mind, we set the default value of c to 10, but note that small values may result in better local structure (NE) for for non-dense graphs.

While the MaxEnt model proposes a decaying repulsive force, we fix the coefficient of the repulsive force, similar to tsNET. As Fig. 3.31 (Bottom) demonstrates, the value of α does not have a large effect on the resulting layout as long as it is positive. By default, we set α to 0.2.

Experiments We compare L2G against state-of-the-art techniques for local and global embeddings. Specifically, we use the implementations of tsNET [65], UMAP [70] and MDS [109], with their default parameters. Our implementation of L2G is also available online. The experiments were performed on an Intel® Core™ i7-3770 machine (CPU @ 3.40GHz × 8 with 32 GB of RAM) running Ubuntu 20.04.3 LTS.

NE, CD and Stress Values and Trends

To evaluate how well L2G preserves local neighborhoods, we compare the average NE scores over several runs and present our results in Table 3.3(a). We can see a general trend: although, tsNET performs better with respect to NE values, L2G has consistently lower NE values than MDS. Additionally, as we increase the size of the neighborhood parameter, the NE values tend to increase by bringing the layouts closer to those of MDS. Interestingly, UMAP also tends to fall somewhere between tsNET and MDS on this metric. As expected, in many cases, L2G 'transitions' from tsNET to UMAP then finally to MDS as one goes from left to right, increasing k.

Next, we compute and report the averaged stress scores in Table 3.3(c). We remark, that MDS is consistently good at minimizing the stress (as it is the function that MDS optimizes), but we see a salient trade-off between the stress scores of L2G's tsNET-like embeddings with low k values and L2G's MDS-like embeddings with high k values. When we look at small neighborhoods such as $k = 16$, we tend to see high stress values, however, the values decrease as we expand the neighborhoods. UMAP does not seem to capture global structure well for these graphs, often having the highest stress values.

Finally, we report the average CD scores for the graphs in our benchmark in Table 3.3(b). Unlike the NE values which increase as we increase k and the stress values which decrease as we increase k, the best CD values are obtained for intermediate values of k. This confirms that a balance between local and global optimization is needed to capture intermediate structures.

Effect of k on Evaluation Metrics:

To help visually explain how our algorithm behaves, we plot a few examples of how NE, CD, and stress behave with respect to k. In Fig. 3.25 (Top), we demonstrate two separate plots for each graph. It can be seen that we often fall in between the values of NE and stress that tsNET and MDS reach. Nevertheless, these plots show what we expect to see: as k increases NE increases and stress decreases.

In Fig. 3.25 (Bottom), we plot the CD values of our layout with tsNET, UMAP, MDS for comparison. Note that in many of the layouts where L2G seeks to capture intermediate structure, L2G indeed has a the lowest CD score.

3.3.5 Discussion and Limitations

We described L2G: an adaptable algorithmic framework for embeddings that can prioritize local neighborhoods, global structure, or a balance between the two. L2G offers flexibility in choosing what structures, while producing embeddings that are comparable in quality to earlier single-purpose (just local, or just global) methods. L2G outperforms state-of-the art

methods in realizing intermediate structures, as captured by the cluster distances preservation. The algorithm is implemented and fully functional and we provide sourcecode and experimental data on GitHub: `https://anonymous.4open.science/r/L2G-5D1B/`.

There are many limitations, starting with the fact that results presented here are based on a small number of graphs and even smaller number of high-dimensional datasets. Further and more systematic experimentation can strengthen the argument for the utility of L2G. L2G is based on a modified MDS objective function, to allow it to consider different size neighborhoods. One could consider modifying the Kullback-Leibler divergence cost function of t-SNE in a simiar fashion (to consider different size neighborhoods). Note that t-SNE's perplexity parameter ostensibly controls the size of a neighborhood, but high perplexity values do not result in global structure preservation [106].

The parameter space of the L2G algorithm is rather large with both c and α having an effect on the resulting visualization independent of k. We provide experimentally derived defaults for both c and α, and leave k as a truly free parameter. Our intention is for a visualization designer to adjust k as needed; to generate a spectrum of embeddings to get a sense of both local and global properties of a dataset. Note that we have not yet realized an interactive version of L2G.

While L2G runs in seconds for graphs with a few thousand vertices, the running time can become untenable for larger instances, due to the $O(|V|^2)$ optimization per epoch, and pre-processing with all-pairs-shortest-paths. While L2G's runtime is comparable with those of tsNET and MDS (see Fig. 3.32) both techniques can be sped up through the use of approximations [38, 84, 111]. Speeding up L2G can be accomplished using similar ideas but remains as future work.

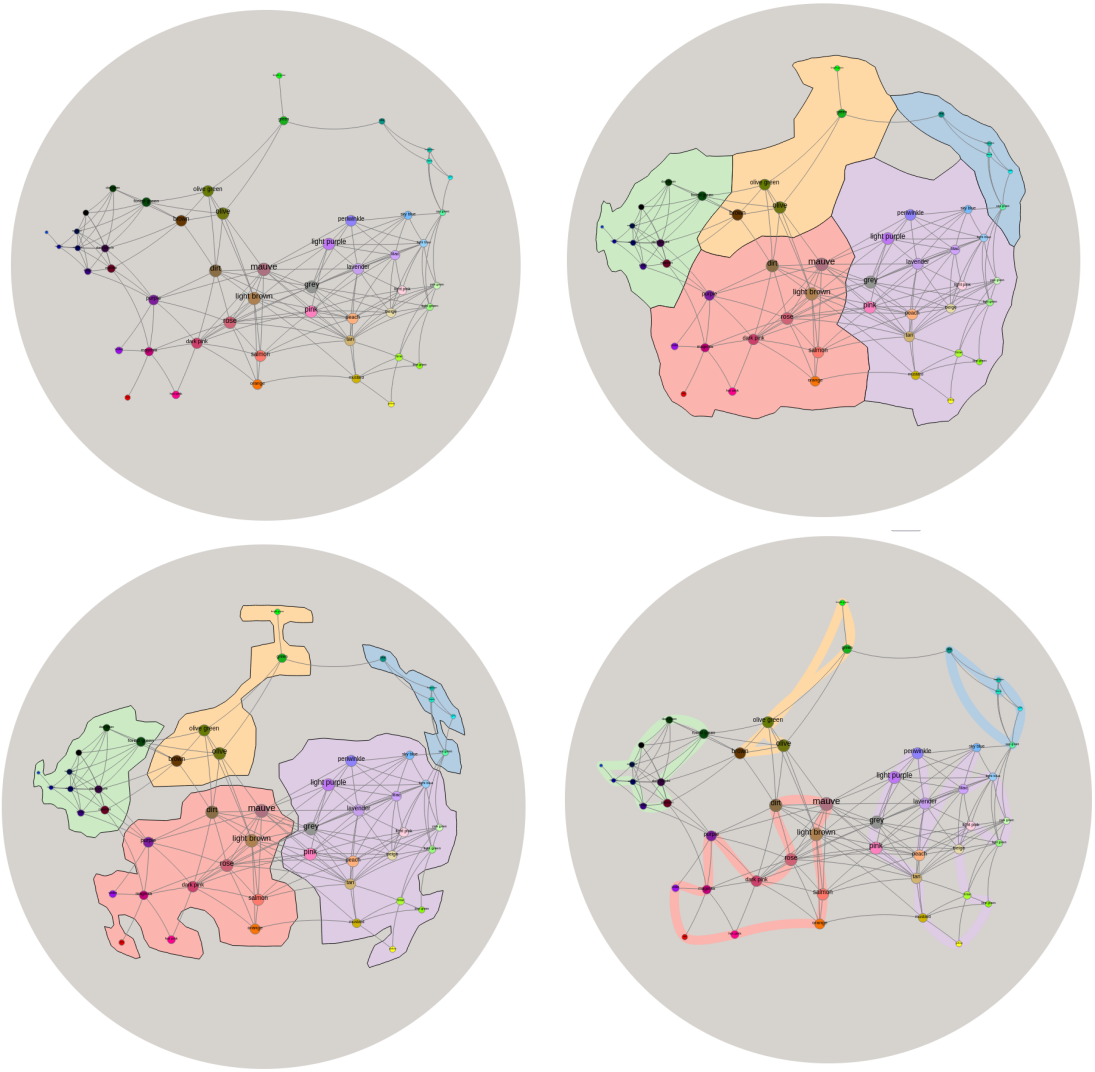


Figure 3.10: Different GMap drawing options for the same graph using inverse projection from Euclidean to hyperbolic space.

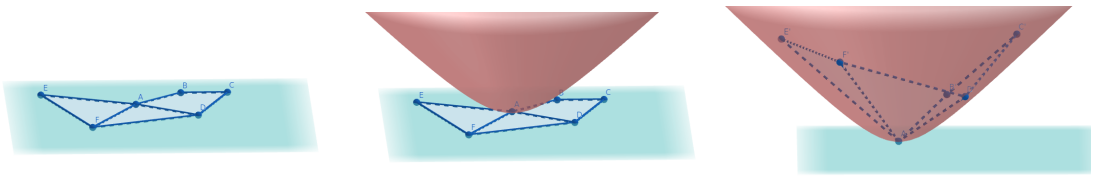


Figure 3.11: Illustration of an inverse projection: wrapping a plane drawing on a hyperboloid.

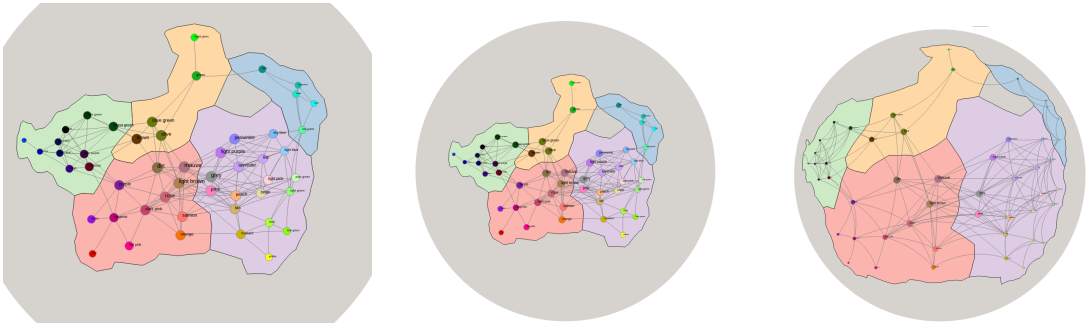


Figure 3.12: An example of the default (center), increased zoom (left), and increased coverage (right) for the same graph.

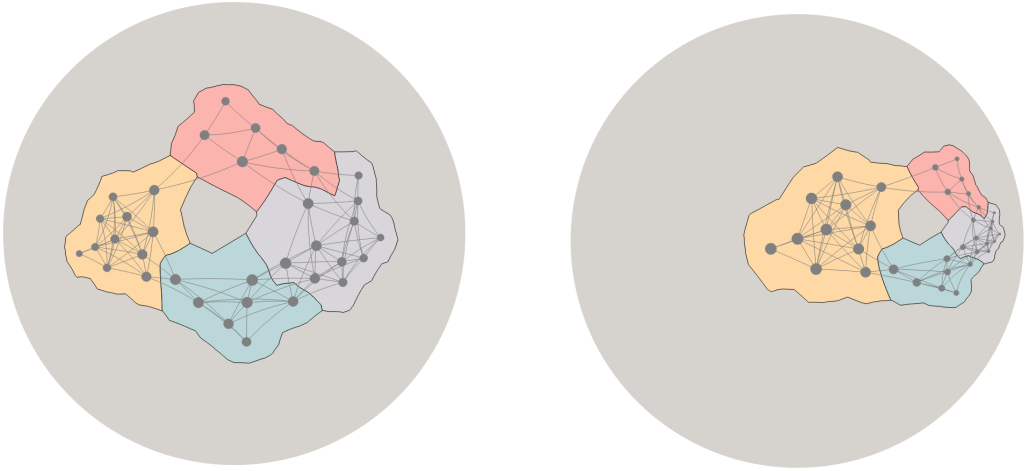


Figure 3.13: The same graph centered about two different origins.

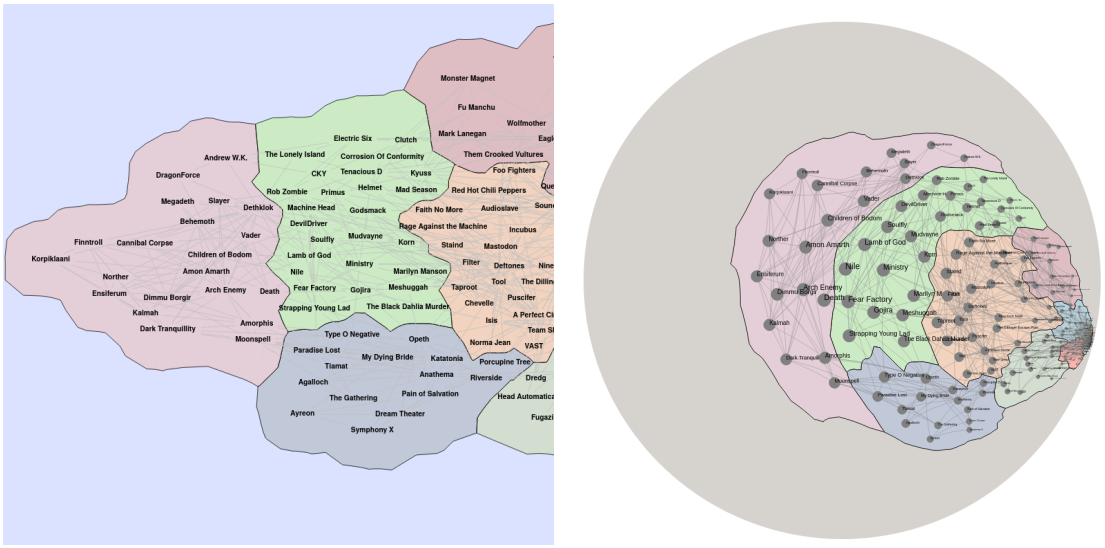


Figure 3.14: A 2D Euclidean GMap instance of the MusicLand graph (left) and its hyperbolic realization (right).

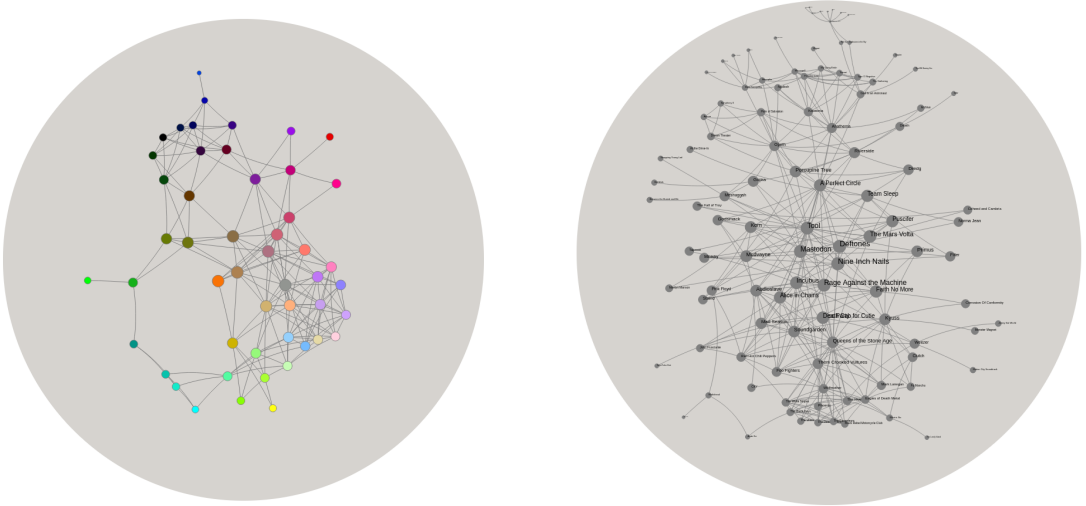


Figure 3.15: Force-directed colors (left) and MusicLand (right) graphs.

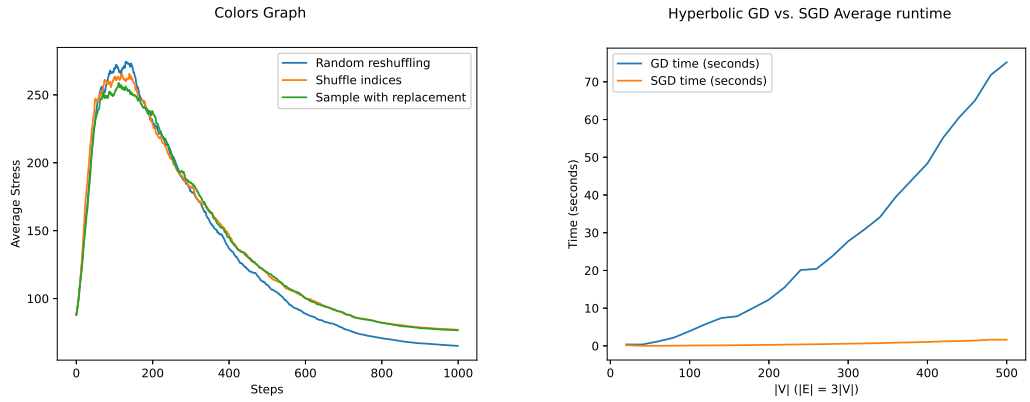


Figure 3.16: Effect of randomization techniques (left) described in 3.2.3, showing average stress over 30 runs at each step on the Colors graph and average runtime (right) of HMDS using GD and SGD over 30 runs (pre-processing omitted). Similar results were found on other graphs.

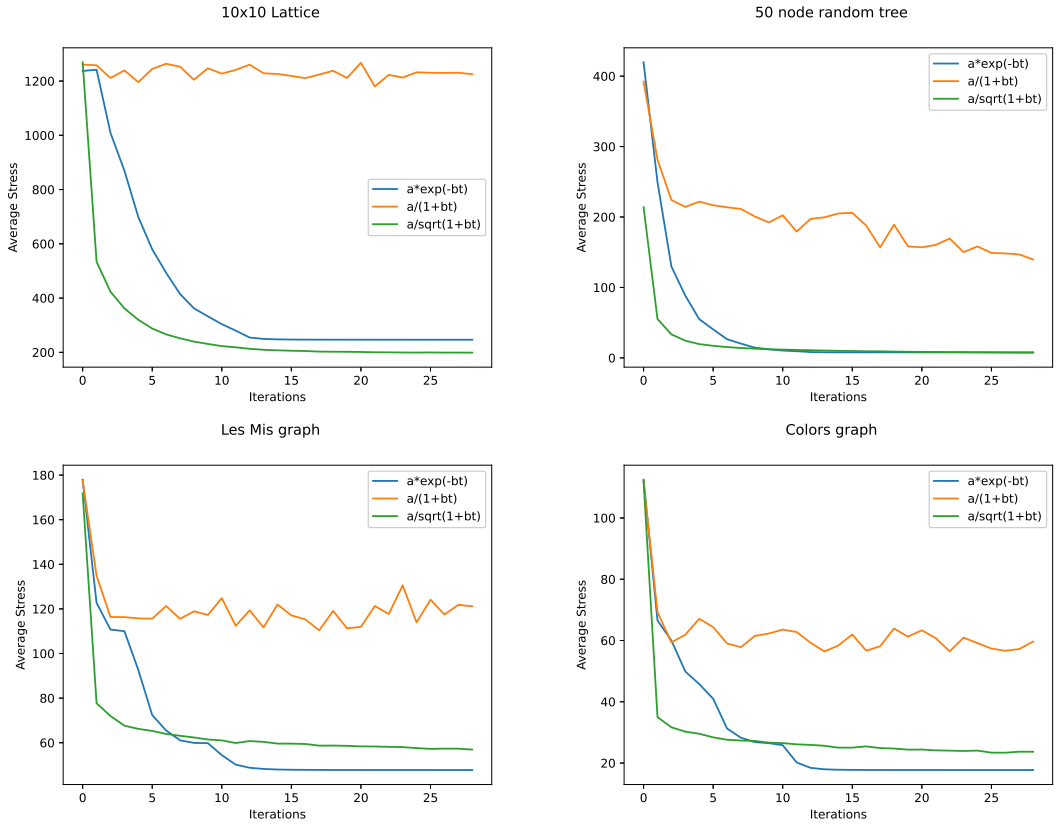


Figure 3.17: Effect of learning rate on various classes of graphs (average over 15 runs each). Graphs used are a 10x10 grid (top left), 50 node random trees (top right), the Les Mis graph [60] (bottom left), and the colors graph (bottom right).

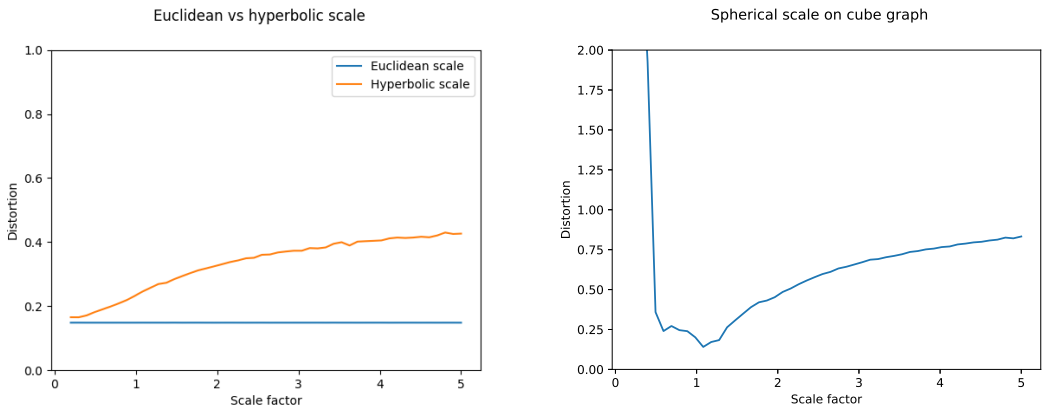


Figure 3.18: Left: Distortion on triangular lattice graph shown in Fig. 3.20. Hyperbolic space gets worse as the scale increases, but Euclidean can embed the graph with constant error. Right: The effect of scale on the sphere on a cube graph. For this example, there is a noticeable optimum at around $\pi/3$ (note that the diameter of a cube graph is 3).

Figure 3.19: Average stress plots of GD and SGD. Initial stress values are omitted.

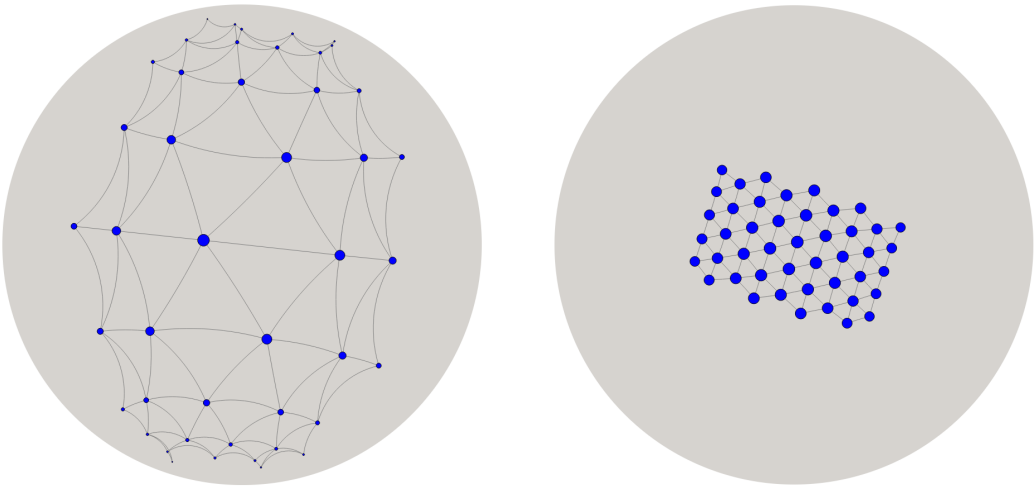


Figure 3.20: Triangular lattice with scaling factor $\alpha = 1$ (left) and optimized $\alpha = 0.22$ (right).

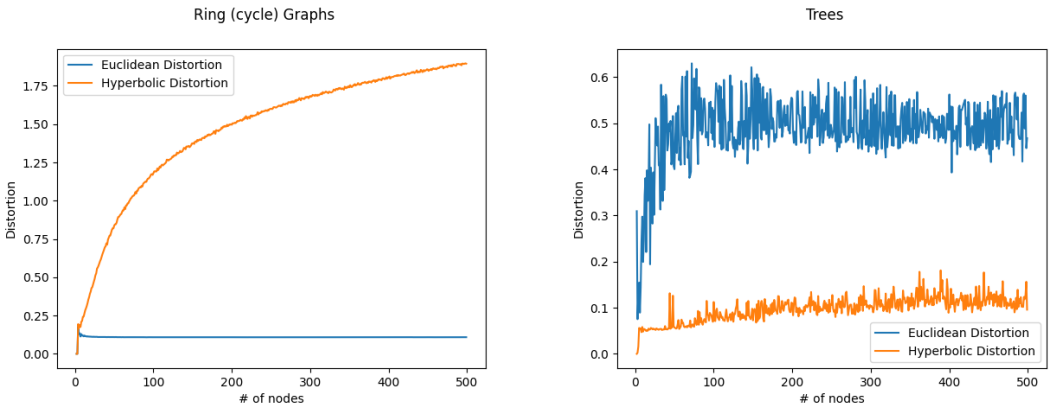


Figure 3.21: Euclidean and hyperbolic embedding distortion on rings (left) and trees (right). It can be seen that the number of nodes in a ring in Euclidean space does not matter, but distortion gets worse with size of the ring in hyperbolic space. The inverse is true for trees, they can be embedded with constant distortion in hyperbolic space but not Euclidean.

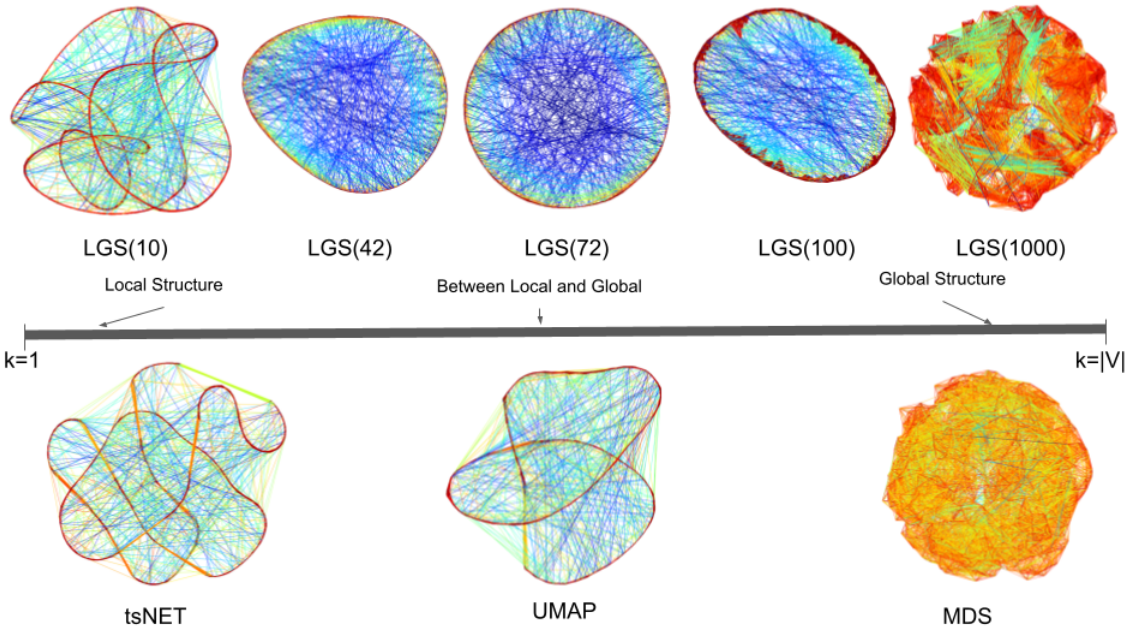


Figure 3.22: Example embeddings of the connected_watts_1000 graph, a small-world network generated by the Watts-Strogatz random graph model, where vertices are placed evenly around one cycle, connected by an edge to their nearest neighbors in space, then some edges are rewired as 'chords' in the cycle. The top row shows L2G embeddings – from local to global – listing the value of the parameter k specifying the size of the neighborhood considered around each vertex. The L2G(72) layout captures the underlying model that generated this network. The bottom row shows three state-of-the-art embedding techniques, tsNET [65], UMAP [70], and MDS [109], for the same graph. Both tsNET and UMAP capture the one dimensional topology, but create bends and intersections of the circle. Finally MDS creates a 'hairball' that is difficult to read, though does capture the high-level circular shape of the graph. Edges are colored with red indicating a 'compressed' edge, blue indicating a 'stretched' edge and green indicating a 'correct' edge.

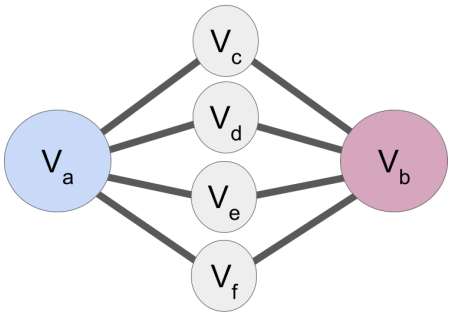


Figure 3.23: An example of how we may skip over immediate neighbors when selecting neighborhoods to preserve. In this case, $c = 2$. There is only one unique walk of length ≤ 2 from v_a to v_c, v_d, v_e, v_f, but there are 4 such walks from v_a to v_b. In this case, v_b would be the first vertex added to v_a's most connected neighborhood.

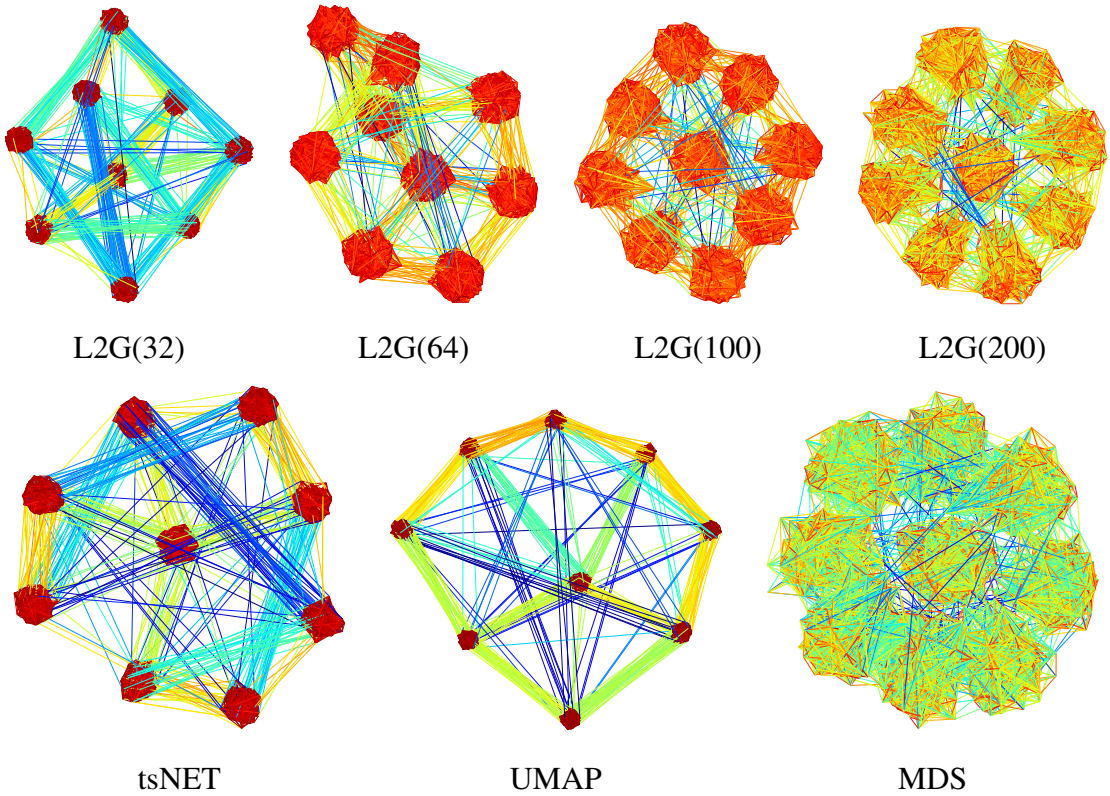


Figure 3.24: The grid_cluster graph is generated so that each cluster has many out-of-cluster edges to its neighbors in a 3x3 lattice, providing a recognizable intermediate structure. While tsNET and UMAP do not place clusters on a grid, MDS mixes the clusters; L2G(100) captures the 3x3 grid and shows dense, well-separated clusters.

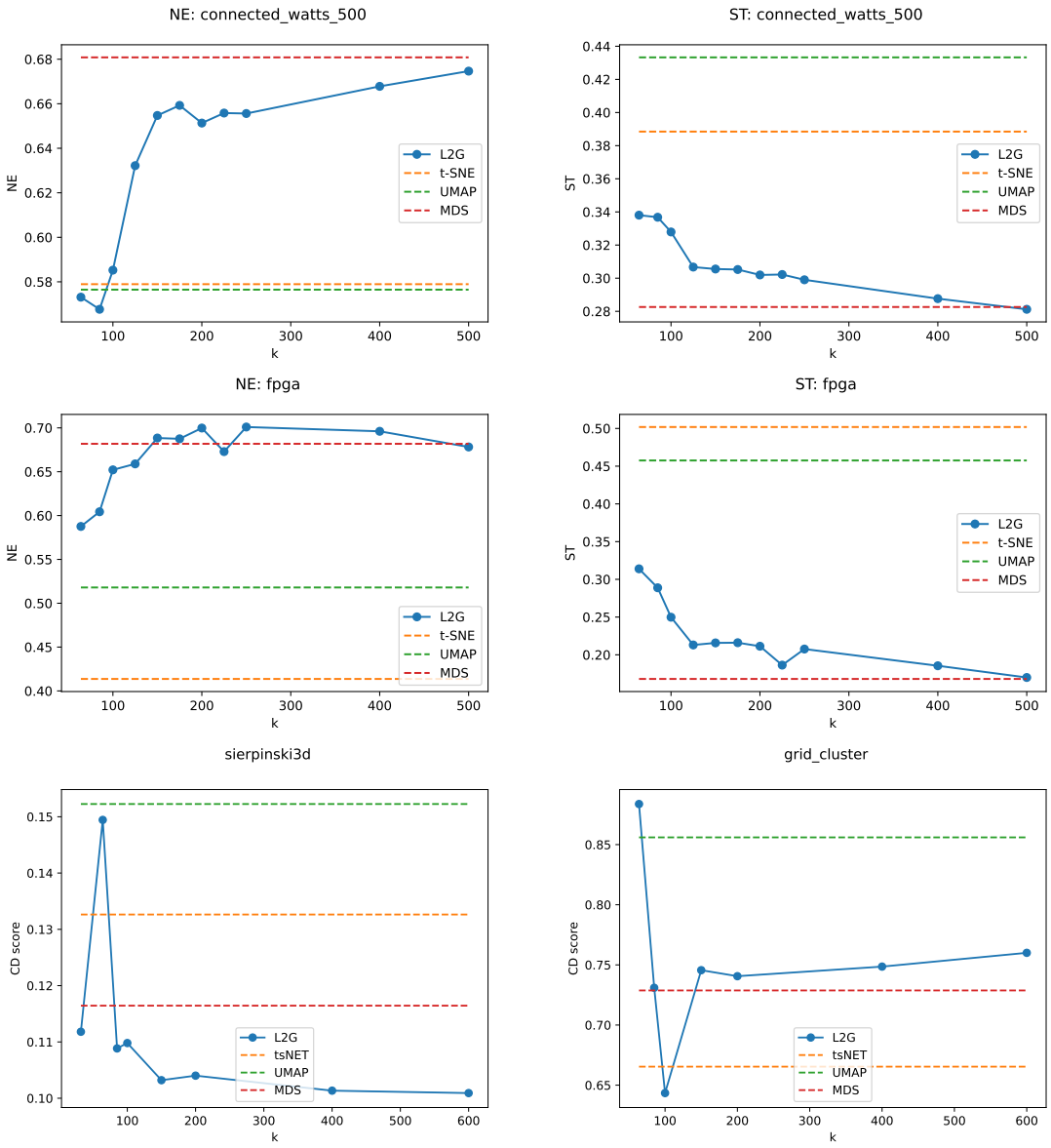


Figure 3.25: (1st and 2nd row) Behavior of NE and stress: as k increases NE gets worse and stress gets better (L2G transitions from preserving local to global structure); tsNET, UMAP, and MDS values are shown as dotted lines for comparison. (3rd row) CD values for the Sierpinski_3d and grid_cluster graphs, showing L2G can capture the intermediate structure for some value of k.

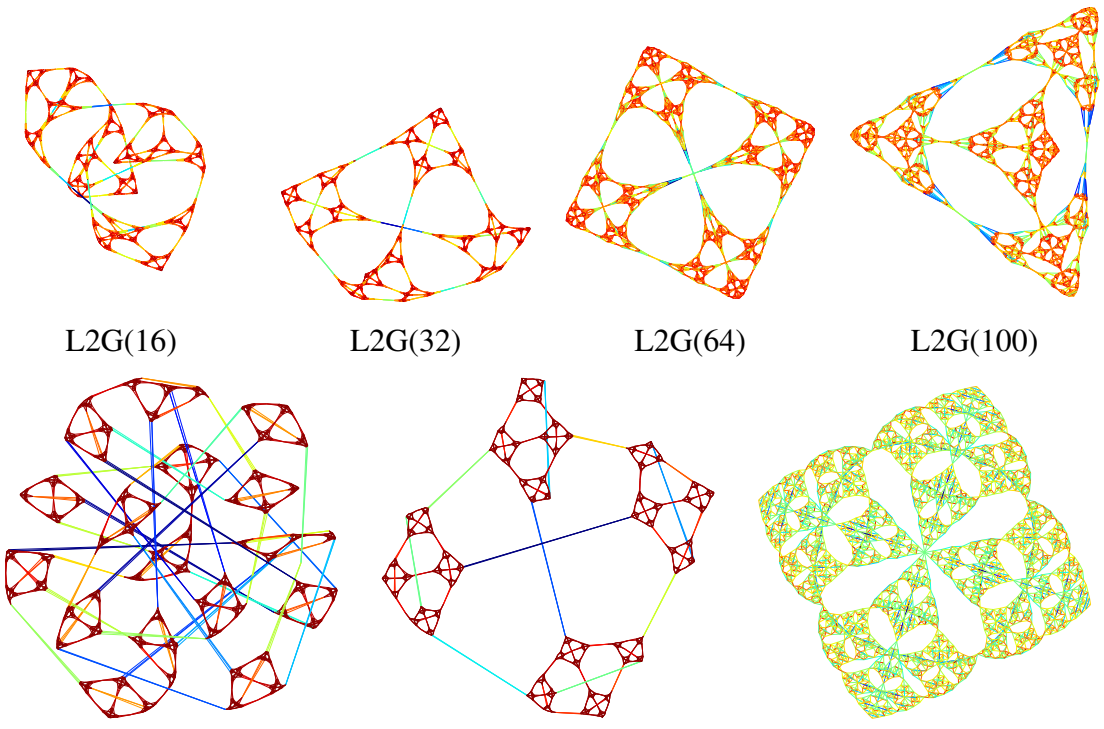


Figure 3.26: Sierpinksi3d graph is a fractal pyramid with regular local and global structure. L2G manages to capture the recursive (fractal) nature of the underlying structure. tsNET, UMAP miss the global placement of small pyramids and MDS stretches them.

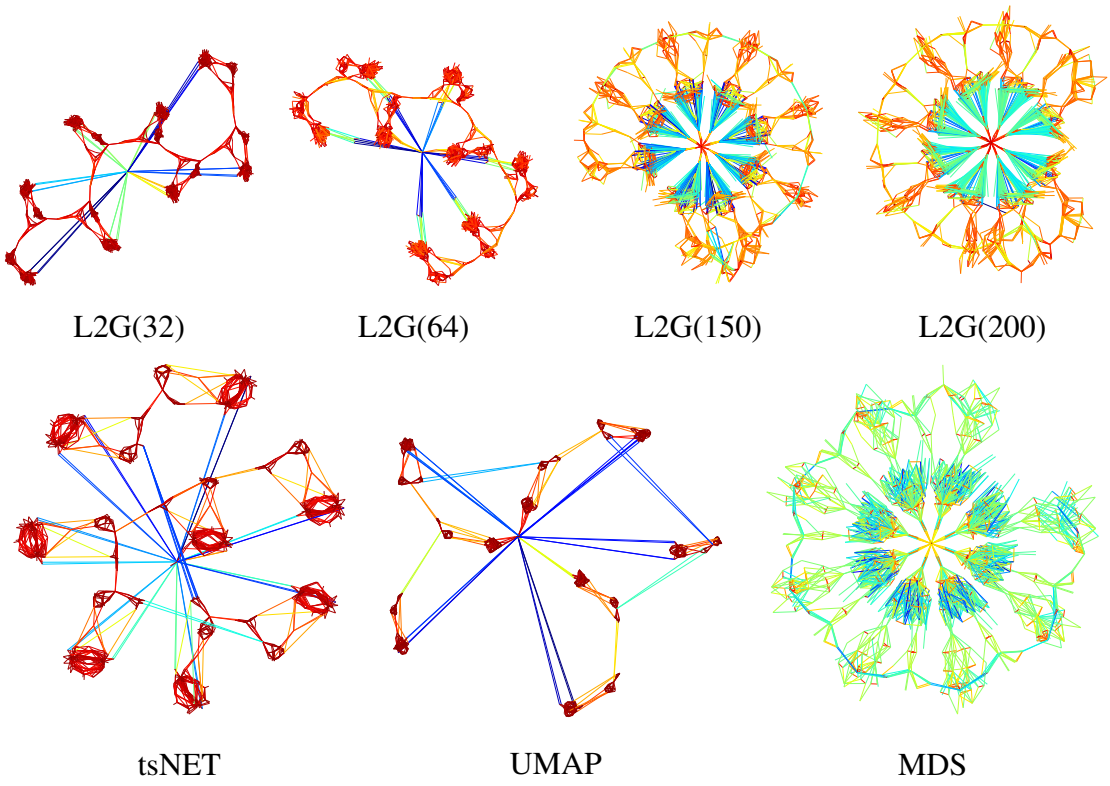


Figure 3.27: The fpga graph contains regularly spaced clusters around a circle with both local and global structure. Local methods miss the relative placement of the clusters while gloabal methods miss the clusters.

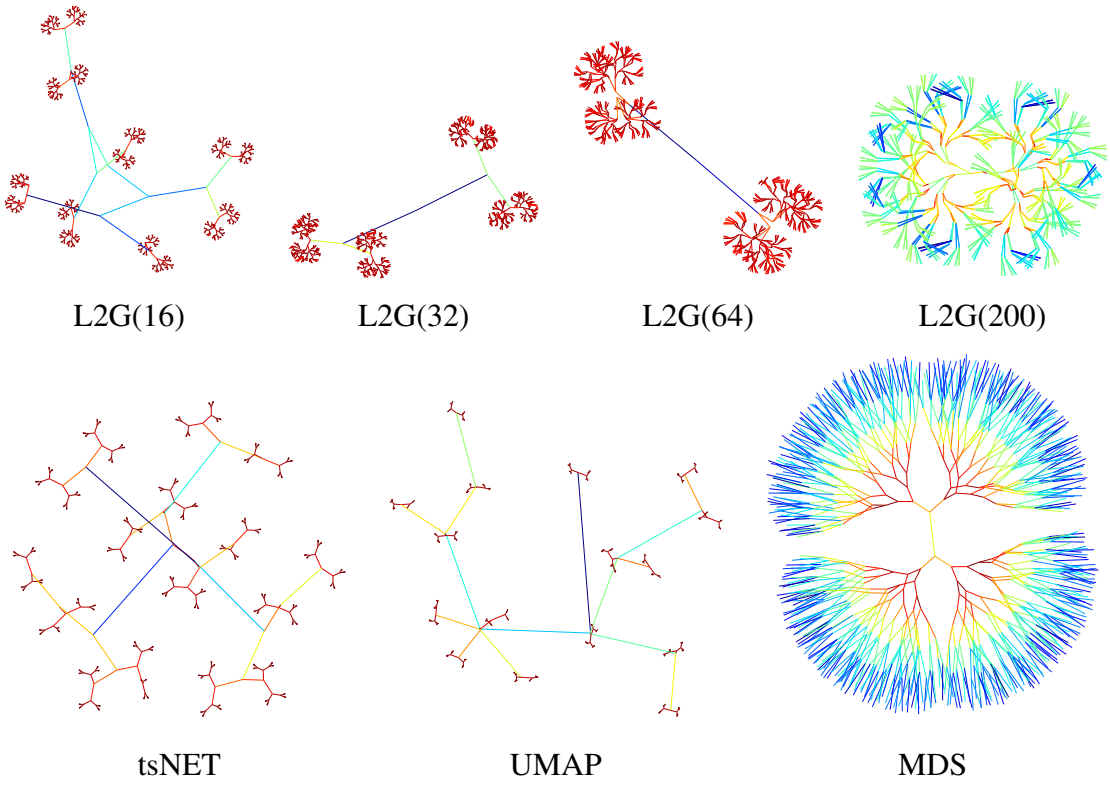


Figure 3.28: The btree9 (binary tree of depth 9) graph embedded with L2G. The value of k increases from left to right. We see gradually larger and larger clusters being shown in the drawing, indicating that there are hierarchical groupings in the data.

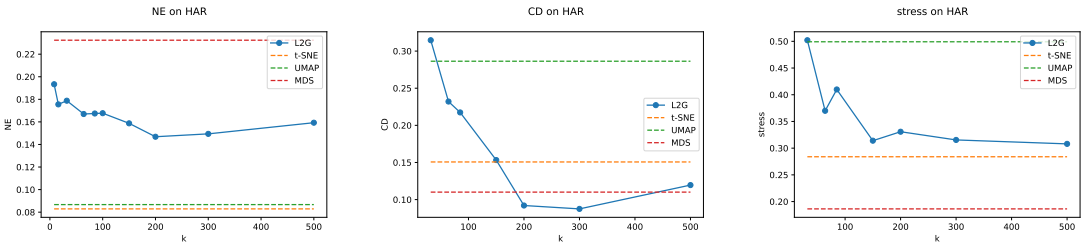


Figure 3.29: Metric values for the HAR dataset. The x axis corresponds to k, the number of nearest neighbors we consider in L2G. t-SNE, UMAP, and MDS are fixed with default parameters and shown as dotted lines.

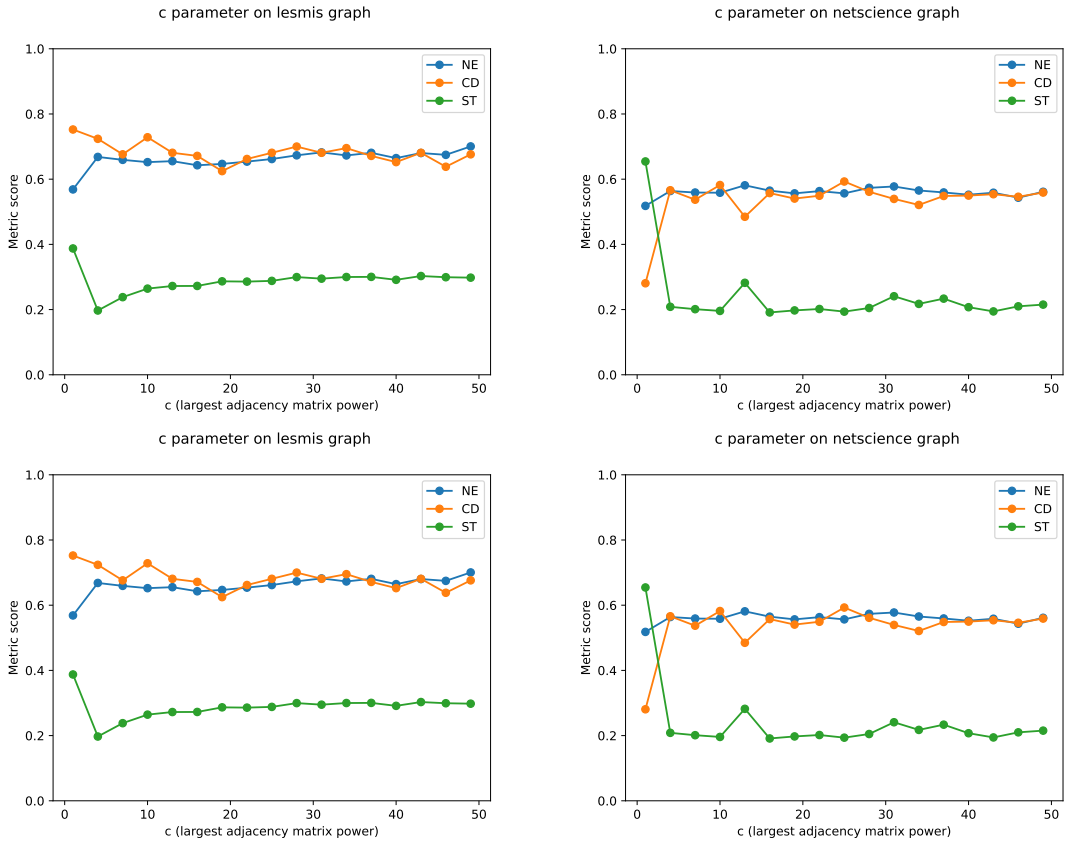
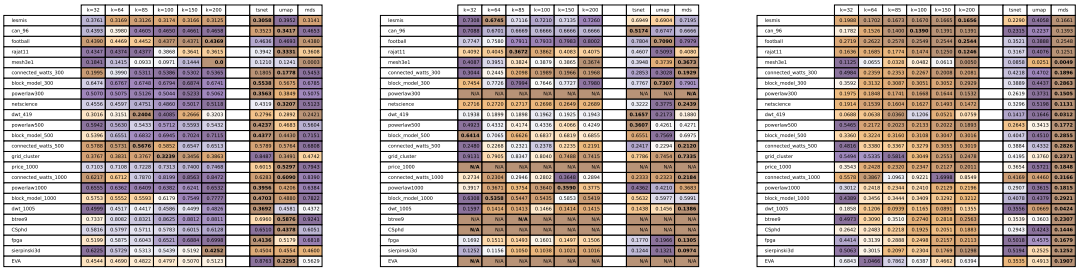


Figure 3.31: (Top): The effects of the c parameter with $k = 25$ on different graphs. The effects appear to be consistent after around 10 or 20, so we set the default to 10. (Bottom): The effects of the α parameter with $k = 25$ on different graphs. As long as $\alpha > 0$, there is no large effect on the layout. We set the default to 0.2



NE scores CD scores Stress scores

Table 3.3: NE (left), CD (middle), and stress (right) scores on benchmark graphs. Cell color is normalized by row, from orange (lowest) to white (middle) to purple (highest); lower is good. Note the pattern in the NE table; The gradient goes orange to purple from left to right. The inverse is true for the stress table, where the gradient goes from right to left.

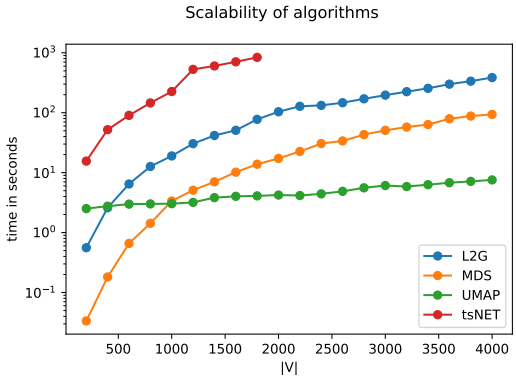


Figure 3.32: Times of all algorithms with increasing size of random graph. The x-axis encodes the number of vertices and the y-axis is the time to complete for the algorithm in seconds.

Chapter 4

FUTURE WORK

I plan to continue to work on graph layout problems and have plans to make a few submissions soon and more in the next year of my PhD.

4.1 Multilayer graph visualization (LIMGraph)

In my work with Pacific Northwest National Laboratory (PNNL), we developed a way to display a layer-disjoint multilayer graph in the plane and utilizing traditional graph layout techniques. This paper has gone through a few revisions, but we plan to submit it to GD this year. Briefly, a multilayer graph is a graph in which the vertices and edges belong to distinct layers. For example, you might have a social network containing friendships, colleagues, and families as different edge types creating a graph of three layers. We are interested in a special class of multilayer graphs known as layer disjoint graphs, where each vertex belongs to one and only one layer. These occur in biology (protein interaction networks) and in infrastructure (power-water-transportation networks).

Our idea is straightforward, to break each layer into an independent graph embedding, then place the independent drawings in such a way to minimize between layer edge crossings. The novel parts of this project are the arrangement of the layers to minimize edge crossings and the domain application of the data. We both rotate the independent embeddings using a gradient descent scheme and reorder the embeddings to achieve better results.

4.2 Motif shape perception

I attended a Dagstuhl seminar earlier this year and started a small project with my working group. We aim to answer the questions of whether the exact representation of a substructure effects a users perception. For example, if you draw a cycle as a rough circle and again as

Paper	Venue	Date
LIMGraph	GD2023	June 2023
Motif Shape Perception	GD2023	June 2023
ENS-t-SNE	PacificVis 2023	October 2023
Non-Euclidean Journal	TVCG	December 2023
Dagstuhl Graph Application	PacificVis 2023	October 2023

Table 4.1: Timeline of current projects and ideal venue to submit to

a line with a long curve from the beginning to end, do people correctly interpret these as the same abstract structure?

We have developed an experimental design that goes as follows: Generate graphs with an inserted motif drawn in a particular shape. Highlight this figure in the drawing and show two drawings side by side. Participants are then asked whether they think the two motifs are identical, similar, or different and must answer quickly to ensure the pre-attentive response is what's captured. There are four different types of motif pairs: motifs that have both the same shape and same structure, motifs with the same shape and different structure, different structure but same shape, and different shape with different structure. We hypothesize that same shape-same structure pairs and different shape-different structure pairs will be correctly identified as identical/different, but that the other two types can be used to 'lie' by either implying two motifs with different graph structure are the same or that two identical motifs have different structure.

I was responsible for creating the graph drawings for user experiments, creating networks, inserting motifs and ensuring they are drawn in the defined shapes while the surrounding network layout is a realistic one. Additionally I am creating the online survey to collect the data and will be running the analysis for the paper submission.

4.3 ENS-t-SNE

ENS-t-SNE (Embedding Neighborhoods Simultaneously with t-SNE) [54], is a dimensionality reduction project I have been apart of as part of my lab at Arizona. It is an extension of an earlier work [52] and, very briefly, attempts to capture multiple subspaces of a high dimensional dataset by embedding the whole dataset in 3d, and finding 2d projections which are faithful to a subspace.

We submitted this work to EuroVis 2023 and though rejected we received concrete, constructive, and actionable reviewer comments. Namely, ideas for datasets to better show the method's real-world use, ways to increase the scalability, and to further extend evaluation against MPSE, 3d t-SNE, and 3d UMAP. I will address these changes over the summer and aim for a resubmission to PacificVis in October though my co-authors and I have discussed submitting straight to a journal (either way these comments need to be addressed).

4.4 Following up on non-Euclidean Graph Embeddings

We should write a journal version of the two non-Euclidean graph embedding papers, including the spherical and hyperbolic MDS descriptions and discussing the dilation problem. Something I think left unaddressed by the conference versions of the paper is whether low distortion (embedding error) corresponds at all to task performance or accuracy. For instance, just because an embedding is more mathematically faithful in hyperbolic geometry, is that still useful if users perform better on a Euclidean embedding despite the distortion being greater? I'm aware of a single study on this subject [27], and I think there are several areas of this study that can be improved. Firstly, they use the tangent plane hyperbolic

embeddings from [62], which we show in [73] that HMDS can do much better in embedding error. Secondly, moving about in a non-Euclidean space (translating a projection for instance) incurs rotation. Many spherical interaction systems address this by anchoring the projection on one axis, you'll notice north is always up on a globe. The iSphere study reported this rotation as a reason hyperbolic graph embeddings were difficult to navigate, but not for the sphere so I am left to assume they anchored the spherical system but not the hyperbolic system in their study (this rotation problem was not mentioned). Finally, the graphs used in the study were a single class, 3 cluster block graphs with varying modularity (between cluster edge density). At least three classes of graphs should be used for a thorough comparison, one that achieves lower embedding error for each geometry that we show in [72].

By the end of June, I want to have a preliminary experiment design in place to be able to submit an IRB so that I can conduct the study in the fall. The high level design would be as follows: Take a set of (say 5) graphs that have lowest distortion in Euclidean, hyperbolic, and spherical spaces (totaling 15). A participant would be presented with a graph embedding in a particular geometry, and be asked to perform a simple task (e.g. Given node A, find vertex adjacent to A with highest degree). A between-subjects study should be employed, so a participant only sees Euclidean, hyperbolic, or spherical embeddings. This not only keeps experiment time to a reasonable level, but prevents bias in the order in which a participant experiences the embeddings. Basic interaction for each system should include a zoom and pan, and highlighting incident links on node hover/click. These tools exist from the papers described in chapter 3.

I hypothesize that a lower embedding error will correspond to better accuracy, as the graph 'lives' more naturally in that space than other spaces. A potential confounding factor is unfamiliarity of participants with hyperbolic visualizations. I think this can be overcome with a thorough introduction and several training questions that are not factored into the analysis to allow a participant to 'get over' the initial learning curve, but it is something to heavily consider. In August I want to put together these tools and implement the experiment in an online tool (I've recently become familiar with Qualtrics) and conduct the study in September/October. From there, all that is left is to analyze results and add them to the paper and finish up by December of this year.

4.5 Dagstuhl Graph Visualization System

A colleague of mine collected a large Dagstuhl seminar dataset which includes both seminar events and their participants. We've extracted several graphs from the data, a seminar-seminar graph (edge between events if k shared participants), a participant-participant graph (edge between participants if k co-attended events), a bipartite graph with both node types, and a DAG of related seminars (seminars report previous related seminars).

We are particularly interested in what seminars and researchers bring together different fields or where there is potential for cross discipline collaboration. We are building a web

application to facilitate these high level questions. My contributions will be to help design and build the application, and more importantly designing a graph embedding to facilitate our research questions.

Each seminar field contains a list of keywords and phrases, so we have much more information than *just* the connectivity of the graph. I took these keywords and performed a sentence-level embedding in high dimensional space (leveraging the existing tool in sentence transformers). Then, each seminar can be placed in this space by taking the geometric mean of their keywords. Perform a dimension reduction on this data to produce a 2-dimensional embedding of the graph that is based only on the keywords, the edges were not 'seen' by the projection algorithm. Now when the edges are drawn on the node-link diagram, very long links indicate two seminars who have very different keywords but had many participants in common, and these long links can be emphasized with a proper link coloring scheme. We can perform the opposite as well, embed the graph based on the connectivity of the edges, but draw links based on the keyword embedding (by say, building a knn graph in HD space). Now, long links indicate seminars who have very similar keywords but who had few participants in common. These two embeddings along with the node-link drawing scheme highlight the two interesting aspects of the data we are looking for. Evaluation for this type of project will look like a case study, where we show that we can find interesting results hiding in the data.

The timeline of this project is to build up the web system over the summer. We will submit a poster presentation to GD in September, collect feedback from the community, and plan to submit this to the PacificVis conference in October.

4.6 Conclusion

We have developed methods to 'naturally' apply dimension reduction (and thus graph layout) into the spaces of spherical and hyperbolic geometry, and a layout method to preserve local, global, or in-between structure. There are many avenues to continue to adapt layout algorithms for different purposes.

REFERENCES

[1] Alper, B., Riche, N.H., Ramos, G.A., Czerwinski, M.: Design study of LineSets, a novel set visualization technique. IEEE Trans. Vis. Comput. Graph. **17**(12), 2259–2267 (2011)

[2] Andrews, K., Putz, W., Nussbaumer, A.: The hierarchical visualisation system (HVS). In: 11th International Conference on Information Visualisation. pp. 257–262. IEEE Computer Society (2007)

[3] Anguita, D., Ghio, A., Oneto, L., Parra, X., Reyes-Ortiz, J.L., et al.: A public domain dataset for human activity recognition using smartphones. In: Esann. vol. 3, p. 3 (2013)

[4] Barry, N.: Hyperbolic Canvas Github Page. `https://github.com/ItsNickBarry/hyperbolic-canvas`, accessed: 2021-06-06

[5] Baumgartner, J., Waugh, T.A.: Roget2000: a 2d hyperbolic tree visualization of Roget's thesaurus. In: Visualization and Data Analysis. SPIE Proceedings, vol. 4665, pp. 339–346 (2002)

[6] Belmonte, N.G.: Javascript InfoVis Toolkit. `https://philogb.github.io/jit/demos.html`, accessed: 2021-06-06

[7] Bingham, J., Sudarsanam, S.: Visualizing large hierarchical clusters in hyperbolic space. Bioinform. **16**(7), 660–661 (2000)

[8] Bläsius, T., Friedrich, T., Krohmer, A., Laue, S.: Efficient embedding of scale-free graphs in the hyperbolic plane. IEEE/ACM Trans. Netw. **26**(2), 920–933 (2018)

[9] Börsig, K., Brandes, U., Pásztor, B.: Stochastic gradient descent works really well for stress minimization. In: Auber, D., Valtr, P. (eds.) Graph Drawing and Network Visualization - 28th International Symposium, GD 2020. Revised Selected Papers. Lecture Notes in Computer Science, vol. 12590, pp. 18–25. Springer (2020). https://doi.org/10.1007/978-3-030-68766-3_2, `https://doi.org/10.1007/978-3-030-68766-3\_2`

[10] Bottou, L.: Large-scale machine learning with stochastic gradient descent. In: Proceedings of COMPSTAT 2010, pp. 177–186. Springer (2010)

[11] Bottou, L.: Stochastic gradient descent tricks. In: Neural networks: Tricks of the trade, pp. 421–436. Springer (2012)

[12] Bottou, L.: Stochastic gradient descent tricks. In: Montavon, G., Orr, G.B., Müller, K. (eds.) Neural Networks: Tricks of the Trade - Second Edition, Lecture Notes in Computer Science, vol. 7700, pp. 421–436. Springer (2012). https://doi.org/10.1007/978-3-642-35289-8_25, https://doi.org/10.1007/978-3-642-35289-8_25

[13] Bou, B.: Treebolic2 Webpage. http://treebolic.sourceforge.net/treebolic2/en/index.html, accessed: 2021-06-06

[14] Börner, K.: (2012), http://scimaps.org/home.html

[15] Börner, K., Klavans, R., Patek, M., Zoss, A.M., Biberstine, J.R., Light, R.P., Larivière, V., Boyack, K.W.: Design and update of a classification system: The ucsd map of science. PLOS ONE **7**(7), 1–10 (07 2012)

[16] Cai, H., Zheng, V.W., Chang, K.C.: A comprehensive survey of graph embedding: Problems, techniques, and applications. IEEE Trans. Knowl. Data Eng. **30**(9), 1616–1637 (2018). https://doi.org/10.1109/TKDE.2018.2807452, https://doi.org/10.1109/TKDE.2018.2807452

[17] Celinska, D., Kopczynski, E.: Programming languages in GitHub: A visualization in hyperbolic plane. In: Proceedings of the Eleventh International Conference on Web and Social Media. pp. 727–728. AAAI Press (2017)

[18] Chen, K., Dwyer, T., Bach, B., Marriott, K.: It's a wrap: Toroidal wrapping of network visualisations supports cluster understanding tasks. In: CHI '21: CHI Conference on Human Factors in Computing Systems. ACM (2021), https://doi.org/10.1145/3411764.3445439

[19] Chen, K., Dwyer, T., Marriott, K., Bach, B.: Doughnets: Visualising networks using torus wrapping. In: CHI '20: CHI Conference on Human Factors in Computing Systems. ACM (2020)

[20] Chen, L., Buja, A.: Local multidimensional scaling for nonlinear dimension reduction, graph drawing, and proximity analysis. Journal of the American Statistical Association **104**(485), 209–219 (2009)

[21] Clough, J.R., Evans, T.S.: Embedding graphs in lorentzian spacetime. PloS one **12**(11), e0187301 (2017)

[22] Collins, C., Penn, G., Carpendale, S.: BubbleSets: Revealing set relations with isocontours over existing visualizations. IEEE Trans. Vis. Comput. Graph. **15**(6), 1009–1016 (2009)

[23] Cvetkovski, A., Crovella, M.: Multidimensional scaling in the poincare disk (2016), https://open.bu.edu/handle/2144/26684

[24] Davis, T.A., Hu, Y.: The University of Florida sparse matrix collection. ACM Trans. Math. Softw. **38**(1), 1:1–1:25 (2011), https://doi.org/10.1145/2049662.2049663

[25] De Leeuw, J.: Applications of convex analysis to multidimensional scaling (2005)

[26] De Leeuw, J., Mair, P.: Multidimensional scaling using majorization: SMACOF in R. Journal of statistical software **31**, 1–30 (2009)

[27] Du, F., Cao, N., Lin, Y.R., Xu, P., Tong, H.: isphere: Focus+ context sphere visualization for interactive large graph exploration. In: Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems. pp. 2916–2927 (2017)

[28] Du, F., Cao, N., Lin, Y., Xu, P., Tong, H.: iSphere: Focus+context sphere visualization for interactive large graph exploration. In: Proceedings of the 2017 CHI Conference on Human Factors in Computing. pp. 2916–2927. ACM (2017)

[29] Efrat, A., Hu, Y., Kobourov, S., Pupyrev, S.: MapSets: Visualizing embedded and clustered graphs. J. Graph Algorithms Appl. **19**(2), 571–593 (2015)

[30] Elbaz, A.E., Keller, Y., Kimmel, R.: Texture mapping via spherical multidimensional scaling. In: Scale Space and PDE Methods in Computer Vision. Lecture Notes in Computer Science, vol. 3459, pp. 443–455. Springer (2005)

[31] Eppstein, D.: Limitations on realistic hyperbolic graph drawing. In: Graph Drawing and Network Visualization - 29th International Symposium. Lecture Notes in Computer Science, Springer (2021)

[32] Eppstein, D.: Limitations on realistic hyperbolic graph drawing. In: Purchase, H.C., Rutter, I. (eds.) Graph Drawing and Network Visualization - 29th International Symposium, GD 2021, Tübingen, Germany, September 14-17, 2021, Revised Selected Papers. Lecture Notes in Computer Science, vol. 12868, pp. 343–357. Springer (2021)

[33] Eppstein, D., Goodrich, M.T.: Succinct greedy geometric routing using hyperbolic geometry. IEEE Trans. Computers **60**(11), 1571–1580 (2011)

[34] Ertoz, L., Steinbach, M., Kumar, V.: A new shared nearest neighbor clustering algorithm and its applications. In: Workshop on clustering high dimensional data and its applications at 2nd SIAM international conference on data mining. vol. 8 (2002)

[35] Espadoto, M., Martins, R.M., Kerren, A., Hirata, N.S.T., Telea, A.C.: Toward a quantitative survey of dimension reduction techniques. IEEE Trans. Vis. Comput. Graph. **27**(3), 2153–2173 (2021). https://doi.org/10.1109/TVCG.2019.2944182, https://doi.org/10.1109/TVCG.2019.2944182

[36] Frey, D., Pimentel, R.: Principal component analysis and factor analysis (1978)

[37] Fruchterman, T.M.J., Reingold, E.M.: Graph drawing by force-directed placement. Softw. Pract. Exp. **21**(11), 1129–1164 (1991). https://doi.org/10.1002/spe.4380211102, https://doi.org/10.1002/spe.4380211102

[38] Fu, C., Zhang, Y., Cai, D., Ren, X.: Atsne: Efficient and robust visualization on gpu through hierarchical optimization. In: Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining. pp. 176–186 (2019)

[39] Gansner, E.R., Hu, Y., Kobourov, S.: GMap: Visualizing graphs and clusters as maps. In: IEEE Pacific Visualization Symposium PacificVis. pp. 201–208. IEEE Computer Society (2010)

[40] Gansner, E.R., Hu, Y., North, S.C.: A maxent-stress model for graph layout. IEEE Trans. Vis. Comput. Graph. **19**(6), 927–940 (2013). https://doi.org/10.1109/TVCG.2012.299, https://doi.org/10.1109/TVCG.2012.299

[41] Gansner, E.R., Koren, Y., North, S.: Graph drawing by stress majorization. In: International Symposium on Graph Drawing. pp. 239–250. Springer (2004)

[42] Gansner, E.R., Koren, Y., North, S.C.: Graph drawing by stress majorization. In: Pach, J. (ed.) Graph Drawing, 12th International Symposium, GD 2004. Revised Selected Papers. Lecture Notes in Computer Science, vol. 3383, pp. 239–250. Springer (2004), https://doi.org/10.1007/978-3-540-31843-9_25

[43] Ghojogh, B., Ghodsi, A., Karray, F., Crowley, M.: Uniform manifold approximation and projection (UMAP) and its variants: Tutorial and survey. CoRR **abs/2109.02508** (2021), https://arxiv.org/abs/2109.02508

[44] Gibson, H., Faith, J., Vickers, P.: A survey of two-dimensional graph layout techniques for information visualisation. Information visualization **12**(3-4), 324–357 (2013)

[45] Glatzhofer, M.: Hyperbolic tree of life. https://hyperbolic-tree-of-life.github.io/

[46] Glatzhofer, M.: Hyperbolic Browsing. Master's thesis, Institute of Interactive Systems and Data Science (ISDS),Graz University of Technology, Austria (2018)

[47] Glatzofer, M.: d3-hypertree Github page. https://github.com/glouwa/d3-hypertree, accessed: 2021-06-06

[48] Gürbüzbalaban, M., Ozdaglar, A., Parrilo, P.A.: Why random reshuffling beats stochastic gradient descent. Mathematical Programming **186**(1), 49–84 (2021)

[49] Günther, I.: (2007), https://world-processor.com

[50] Harel, D., Koren, Y.: Graph drawing by high-dimensional embedding. J. Graph Algorithms Appl. **8**(2), 195–214 (2004). https://doi.org/10.7155/jgaa.00089, https://doi.org/10.7155/jgaa.00089

[51] Holland, P.W., Laskey, K.B., Leinhardt, S.: Stochastic blockmodels: First steps. Social networks **5**(2), 109–137 (1983)

[52] Hossain, M.I., Huroyan, V., Kobourov, S.G., Navarrete, R.: Multi-perspective, simultaneous embedding. IEEE Trans. Vis. Comput. Graph. **27**(2), 1569–1579 (2021). https://doi.org/10.1109/TVCG.2020.3030373, https://doi.org/10.1109/TVCG.2020.3030373

[53] Hu, Y.: Efficient, high-quality force-directed graph drawing. Mathematica journal **10**(1), 37–71 (2005)

[54] Huroyan, V., Navarrete, R., Hossain, M.I., Kobourov, S.: Embedding neighborhoods simultaneously t-sne (ens-t-sne). arXiv preprint arXiv:2205.11720 (2022)

[55] Hyun, Y.: https://www.caida.org/catalog/software/walrus/#H2540 (2000), accessed: 2021-06-06

[56] Jolliffe, I.T.: Principal Component Analysis. Springer Series in Statistics, Springer (1986). https://doi.org/10.1007/978-1-4757-1904-8, https://doi.org/10.1007/978-1-4757-1904-8

[57] Kamada, T., Kawai, S.: An algorithm for drawing general undirected graphs. Inf. Process. Lett. **31**(1), 7–15 (1989)

[58] Kamada, T., Kawai, S.: An algorithm for drawing general undirected graphs. Inf. Process. Lett. **31**(1), 7–15 (1989). https://doi.org/10.1016/0020-0190(89)90102-6, https://doi.org/10.1016/0020-0190(89)90102-6

[59] Kleinberg, R.: Geographic routing using hyperbolic space. In: INFOCOM 2007. 26th IEEE International Conference on Computer Communications. pp. 1902–1909. IEEE (2007)

[60] Knuth, D.E.: The Stanford GraphBase: a platform for combinatorial computing, vol. 1. AcM Press New York (1993)

[61] Kobourov, S.: Force-directed drawing algorithms. In: Handbook on Graph Drawing and Visualization, pp. 383–408. Chapman and Hall/CRC (2013)

[62] Kobourov, S., Wampler, K.: Non-Euclidean spring embedders. IEEE Trans. Vis. Comput. Graph. **11**(6), 757–767 (2005)

[63] Koren, Y.: Graph drawing by subspace optimization. In: Deussen, O., Hansen, C.D., Keim, D.A., Saupe, D. (eds.) 6th Joint Eurographics - IEEE TCVG Symposium on Visualization, VisSym 2004, Konstanz, Germany, May 19-21, 2004. pp. 65–74. Eurographics Association (2004). https://doi.org/10.2312/VisSym/VisSym04/065-074, https://doi.org/10.2312/VisSym/VisSym04/065-074

[64] Krioukov, D., Papadopoulos, F., Kitsak, M., Vahdat, A., Boguná, M.: Hyperbolic geometry of complex networks. Physical Review E **82**(3), 036106 (2010)

[65] Kruiger, J.F., Rauber, P.E., Martins, R.M., Kerren, A., Kobourov, S.G., Telea, A.C.: Graph layouts by t-sne. Comput. Graph. Forum **36**(3), 283–294 (2017). https://doi.org/10.1111/cgf.13187, https://doi.org/10.1111/cgf.13187

[66] Kruskal, J.B.: Multidimensional scaling by optimizing goodness of fit to a nonmetric hypothesis. Psychometrika **29**(1), 1–27 (1964)

[67] Kwon, O., Muelder, C., Lee, K., Ma, K.: A study of layout, rendering, and interaction methods for immersive graph visualization. IEEE Trans. Vis. Comput. Graph. **22**(7), 1802–1815 (2016)

[68] Lamping, J., Rao, R., Pirolli, P.: A focus+context technique based on hyperbolic geometry for visualizing large hierarchies. In: Human Factors in Computing Systems, CHI '95 Conference Proceedings. pp. 401–408. ACM/Addison-Wesley (1995)

[69] Van der Maaten, L., Hinton, G.: Visualizing data using t-sne. Journal of machine learning research **9**(11) (2008)

[70] McInnes, L., Healy, J.: UMAP: uniform manifold approximation and projection for dimension reduction. CoRR **abs/1802.03426** (2018), http://arxiv.org/abs/1802.03426

[71] Miller, J., Huroyan, V., Kobourov, S.: L2g: From local to global embeddings and back. In: under review at IEEE VIS (2023)

[72] Miller, J., Huroyan, V., Kobourov, S.: Spherical graph drawing by multi-dimensional scaling. In: Graph Drawing and Network Visualization: 30th International Symposium, GD 2022, Tokyo, Japan, September 13–16, 2022, Revised Selected Papers. pp. 77–92. Springer (2023)

[73] Miller, J., Kobourov, S., Huroyan, V.: Browser based hyperbolic visualization of graphs. In: IEEE Pacific Visualization Symposium PacificVis. IEEE Computer Society (2022)

[74] Munzner, T.: H3: laying out large directed graphs in 3d hyperbolic space. In: IEEE Symposium on Information Visualization. pp. 2–10. IEEE Computer Society (1997)

[75] Munzner, T.: Exploring large graphs in 3d hyperbolic space. IEEE Computer Graphics and Applications **18**(4), 18–23 (1998)

[76] Munzner, T.: Interactive visualization of large graphs and networks. Ph.D. thesis, Stanford University (2000)

[77] Munzner, T.: Visualization analysis and design. CRC press (2014)

[78] Munzner, T., Burchard, P.: Visualizing the structure of the world wide web in 3d hyperbolic space. In: Procedings of the 1995 Symposium on Virtual Reality Modeling Language. pp. 33–38. ACM (1995)

[79] Nguyen, Q.H., Eades, P., Hong, S.: On the faithfulness of graph visualizations. In: Carpendale, S., Chen, W., Hong, S. (eds.) IEEE Pacific Visualization Symposium, PacificVis 2013, February 27 2013-March 1, 2013, Sydney, NSW, Australia. pp. 209–216. IEEE Computer Society (2013). https://doi.org/10.1109/PacificVis.2013.6596147, https://doi.org/10.1109/PacificVis.2013.6596147

[80] Nielsen, F., Nock, R.: Hyperbolic voronoi diagrams made easy. In: Prodeedings of the 2010 International Conference on Computational Science and Its Applications, ICCSA 2010. pp. 74–80. IEEE Computer Society (2010)

[81] Noack, A.: Energy models for graph clustering. J. Graph Algorithms Appl. **11**(2), 453–480 (2007). https://doi.org/10.7155/jgaa.00154, https://doi.org/10.7155/jgaa.00154

[82] Nocaj, A., Ortmann, M., Brandes, U.: Untangling the hairballs of multi-centered, small-world online social media networks. J. Graph Algorithms Appl. **19**(2), 595–618 (2015). https://doi.org/10.7155/jgaa.00370, https://doi.org/10.7155/jgaa.00370

[83] Ontrup, J., Ritter, H.J.: Hyperbolic self-organizing maps for semantic navigation. In: Advances in Neural Information Processing Systems 14. pp. 1417–1424. MIT Press (2001)

[84] Ortmann, M., Klimenta, M., Brandes, U.: A sparse stress model. J. Graph Algorithms Appl. **21**(5), 791–821 (2017). https://doi.org/10.7155/jgaa.00440, `https://doi.org/10.7155/jgaa.00440`

[85] Perry, S., Yin, M.S., Gray, K., Kobourov, S.: Drawing graphs on the sphere. In: AVI '20: International Conference on Advanced Visual Interfaces. pp. 17:1–17:9. ACM (2020)

[86] Purchase, H.C.: Metrics for graph drawing aesthetics. J. Vis. Lang. Comput. **13**(5), 501–516 (2002). https://doi.org/10.1006/jvlc.2002.0232, `https://doi.org/10.1006/jvlc.2002.0232`

[87] Robbins, H., Monro, S.: A stochastic approximation method. The annals of mathematical statistics pp. 400–407 (1951)

[88] Roweis, S.T., Saul, L.K.: Nonlinear dimensionality reduction by locally linear embedding. Science **290**(5500), 2323–2326 (2000)

[89] Ruder, S.: An overview of gradient descent optimization algorithms. CoRR **abs/1609.04747** (2016)

[90] Saket, B., Scheidegger, C., Kobourov, S., Börner, K.: Map-based visualizations increase recall accuracy of data. Comput. Graph. Forum **34**(3), 441–450 (2015)

[91] Saket, B., Simonetto, P., Kobourov, S., Börner, K.: Node, node-link, and node-link-group diagrams: An evaluation. IEEE Trans. Vis. Comput. Graph. **20**(12), 2231–2240 (2014)

[92] Sala, F., Sa, C.D., Gu, A., Ré, C.: Representation tradeoffs for hyperbolic embeddings. In: Proceedings of the 35th International Conference on Machine Learning. Proceedings of Machine Learning Research, vol. 80, pp. 4457–4466. PMLR (2018)

[93] Schulz, H.J.: Treevis.net: A tree visualization reference. IEEE Computer Graphics and Applications **31**(6), 11–15 (2011)

[94] Shepard, R.N.: The analysis of proximities: multidimensional scaling with an unknown distance function. i. Psychometrika **27**(2), 125–140 (1962)

[95] Snyder, J.P.: Map projections: A working manual. U.S. Government Printing Office (1987)

[96] Tamassia, R.: Handbook of graph drawing and visualization. CRC press (2013)

[97] Tenenbaum, J.B., Silva, V.d., Langford, J.C.: A global geometric framework for nonlinear dimensionality reduction. Science **290**(5500), 2319–2323 (2000)

[98] Tominski, C., Gladisch, S., Kister, U., Dachselt, R., Schumann, H.: A Survey on Interactive Lenses in Visualization. In: EuroVis - STARs. The Eurographics Association (2014)

[99] Tominski, C., Gladisch, S., Kister, U., Dachselt, R., Schumann, H.: Interactive lenses for visualization: An extended survey. In: Computer Graphics Forum. vol. 36(6), pp. 173–200. Wiley Online Library (2017)

[100] Torgerson, W.S.: Multidimensional scaling: I. theory and method. Psychometrika **17**(4), 401–419 (1952)

[101] Verbeek, K., Suri, S.: Metric embedding, hyperbolic space, and social networks. Comput. Geom. **59**, 1–12 (2016)

[102] Walter, J., Ontrup, J., Wessling, D., Ritter, H.: Interactive visualization and navigation in large data collections using the hyperbolic space. In: Third IEEE International Conference on Data Mining. pp. 355–362. IEEE (2003)

[103] Walter, J.A.: H-MDS: a new approach for interactive visualization with multidimensional scaling in the hyperbolic space. Inf. Syst. **29**(4), 273–292 (2004)

[104] Walter, J.A., Ritter, H.J.: On interactive visualization of high-dimensional data using the hyperbolic plane. In: Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. pp. 123–132. ACM (2002)

[105] Wang, Y., Wang, Y., Zhang, H., Sun, Y., Fu, C., Sedlmair, M., Chen, B., Deussen, O.: Structure-aware fisheye views for efficient large graph exploration. IEEE Trans. Vis. Comput. Graph. **25**(1), 566–575 (2019), https://doi.org/10.1109/TVCG.2018.2864911

[106] Wattenberg, M., Viégas, F., Johnson, I.: How to use t-sne effectively. Distill (2016). https://doi.org/10.23915/distill.00002, http://distill.pub/2016/misread-tsne

[107] Yang, Y., Jenny, B., Dwyer, T., Marriott, K., Chen, H., Cordeil, M.: Maps and globes in virtual reality. Comput. Graph. Forum **37**(3), 427–438 (2018)

[108] Zhang, S., Kelleher, A.: H3py Github Page. https://github.com/buzzfeed/pyh3 (2016), accessed: 2021-06-06

[109] Zheng, J.X., Pawar, S., Goodman, D.F.M.: Graph drawing by stochastic gradient descent. IEEE Trans. Vis. Comput. Graph. **25**(9), 2738–2748 (2019). https://doi.org/10.1109/TVCG.2018.2859997, https://doi.org/10.1109/TVCG.2018.2859997

[110] Zhou, Y., Sharpee, T.O.: Hyperbolic geometry of gene expression. Iscience **24**(3), 102225 (2021)

[111] Zhu, M., Chen, W., Hu, Y., Hou, Y., Liu, L., Zhang, K.: Drgraph: An efficient graph layout algorithm for large-scale graphs by dimensionality reduction. IEEE Trans. Vis. Comput. Graph. **27**(2), 1666–1676 (2021). https://doi.org/10.1109/TVCG.2020.3030447, https://doi.org/10.1109/TVCG.2020.3030447