GENERALIZATIONS OF STRESS-BASED GRAPH LAYOUT TO NON-EUCLIDEAN GEOMETRIES

by

Jacob MIller

A Dissertation Submitted to the Faculty of the

DEPARTMENT OF COMPUTER SCIENCE

In Partial Fulfillment of the Requirements
For the Degree of

DOCTOR OF PHILOSOPHY IN COMPUTER SCIENCE

In the Graduate College

THE UNIVERSITY OF ARIZONA

2024

THE UNIVERSITY OF ARIZONA
GRADUATE COLLEGE

As members of the Dissertation Committee, we certify that we have read the dissertation prepared by Jacob MIller entitled "Generalizations of stress-based graph layout to non-Euclidean geometries" and recommend that it be accepted as fulfilling the dissertation requirement for the Degree of Doctor of Philosophy in Computer Science.

_____ Date: April 9, 2024
Stephen Kobourov
(Chair)

_____ Date: April 9, 2024
Alon Efrat
(Member)

_____ Date: April 9, 2024
Joshua Levine
(Member)

_____ Date: April 9, 2024
Karl Glasner
(Member)

Final approval and acceptance of this dissertation is contingent upon the candidate's submission of the final copies of the dissertation to the Graduate College.

I hereby certify that I have read this dissertation prepared under my direction and recommend that it be accepted as fulfilling the dissertation requirement.

_____ Date: April 9, 2024
Dissertation Director: Stephen Kobourov

ACKNOWLEDGEMENTS

I would like to thank Alon Efrat and Josh Levine for the many discussions and suggestions on several of the works that appear in this dissertation, as well as general life and career guidance.

Helen Purchase for the advice and guidance on conducting human subject studies.

Patrick Mackey for perspective and guidance with network visualization in practice.

Dhruv Bhatia for his contributions to our literature survey and human subjects study.

Vahan Huroyan coauthored several of the publications featured in this dissertation and served as a mentor to me in the early years of my PhD studies.

My committee for the thoughtful suggestions on the contents of this dissertation.

And of course, a big thanks to my advisor, Stephen Kobourov, for the many, many hours of meetings, discussions, and editing without which none of this would be possible.

Contents

List of Figures

List of Tables

ABSTRACT

There exists many algorithms to compute node-link diagrams for graph visualization. Almost all of these algorithms aim to draw the graph in the flat Euclidean plane, but there are potentially good reasons to draw graphs in non-Euclidean geometries. Spherical geometry has no absolute center in any drawing, allowing any point to be moved centrally and inherits user familiarity with map and globe visualizations. Hyperbolic geometry creates a focus+context effect, with high detail in the center of the drawing and lower detail on the periphery. Additionally, it is possible to achieve better quality metrics for some graphs in non-Euclidean spaces. In this dissertation, we first introduce non-Euclidean geometry and survey it's history in graph visualization. We then investigate how one can efficiently draw graphs in Euclidean and non-Euclidean spaces. Finally, we conduct a human subjects study to observe how these non-Euclidean visualizations perform with respect to task support.

Chapter 1 Introduction

Graphs (networks) are powerful models for real-world relationships, but are often very large or complicated and so visualizing them becomes an 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 and the edges between vertices drawn as curves. To compute an embedding of a graph using this abstraction, there exist many embedding algorithms. However, the target embedding space is often assumed to be Euclidean since this is the space of the computer screen or sheet of paper. Non-Euclidean geomerties offer novel visualization benefits for graphs through focus+context lower embedding errors. Here, we will explore Euclidean, spherical, and hyperbolic graph embedding (layout) algorithms.

Euclidean geometry needs little introduction, it is the standard geometry one thinks of as a computer screen. This makes it a perfectly natural embedding space for graph visualization. Spherical and hyperbolic space behave quite differently but offer potential benefits.

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. Further real-world examples can be found in Chapter 4. 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 [75,83,97]. Some real world complex networks such as social networks [75] and gene expression networks [148] 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 [73], performing a constrained 3-dimensional Euclidean computation [108], or transforming the distances to use Euclidean methods [117].

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) [33, 77] and t-distributed stochastic neighbor embedding (t-SNE) [132]. 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 [46].

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 [54, 147], and t-SNE has been adapted to graph layout in an algorithm known as tsNET [76] and later DRGraph [149]. 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.

Finally, there exists little research on how the choice of geometry effects task completion time and accuracy, especially on data more suited to a particular geometry. Du et al. [35] conduct a preliminary study to compare Euclidean, spherical, and hyperbolic graph drawings but focus on a single task and on small screen devices. A more thorough human subjects study is needed to compare these geometries to establish whether or not a lower embedding error for a graph corresponds to greater task support.

1.1 Overview

Below is a short summary of the remaining chapters of this dissertation, which address the problems above.

Non-Euclidean Graph Visualization Survey: In chapter 2 is a literature survey of the current state-of-the-art of non-Euclidean graph embedding techniques for visualization. The body of literature is analyzed along two primary dimensions; first by geometry and second by contribution. This chapter is taken from a publication that has been conditionally accepted in EuroVis 2024 [88].

Graph Embedding at Intermediate Scales: We begin by introducing a Euclidean graph embedding algorithm in chapter 3. This work sees the introduction of a new graph (data) embedding algorithm, LGS, 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 LGS, for some parameter values, outperforms competing state-of-the-art methods for preserving this intermediate structure via CD. This chapter is taken from a publication that appeared in the International Symposium on Graph Drawing 2023 [89].

Spherical Graph Embedding: A method for spherical graph embedding appears in chapter 4. 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 [90].

Hyperbolic Graph Embedding: In chapter 5, we explore hyperbolic graph embeddings for visualization. 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 [91] and is the source of section 5.1.

Task Support of Non-Euclidean Graph Embeddings: We take the lessons learned from the previous chapters, and conduct a human subjects experiment in chapter 6. We describe how we design, implement, conduct, and analyze a study for task support in Euclidean, spherical, and hyperbolic graph visualization styles. This work is yet unpublished.

Conclusions and Future Work: Finally, in chapter 7 we summarize the above chapters, reflect on lessons learned, and leave the reader with some open problems.

Chapter 2 Related Work Survey

This chapter is taken from a publication conditionally accepted to appear in EuroVis 2024 [88].

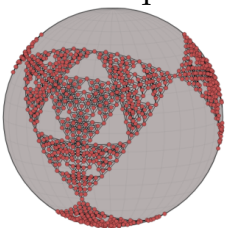
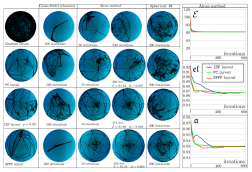
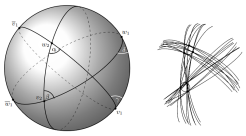
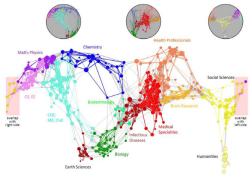
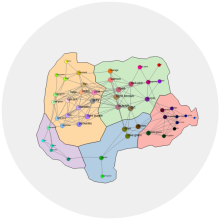
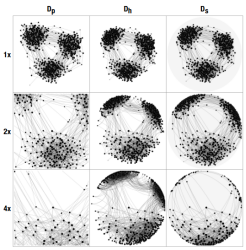
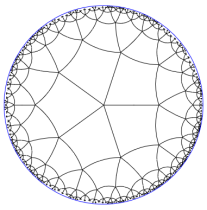
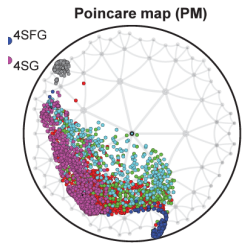
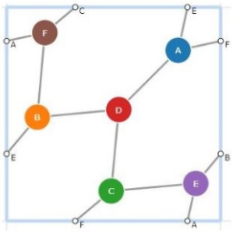
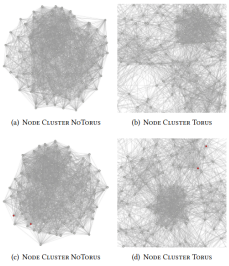
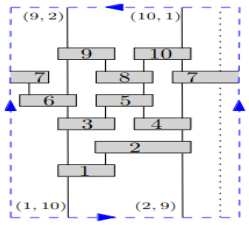
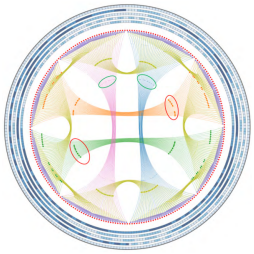
	Technique	Evaluation	Proof	Application
Spherical	 [90]	 [1]	 [13]	 [19]
Hyperbolic	 [91]	 [36]	 [40]	 [148]
Toroidal	 [24]	 [23]	 [9]	 [28]

TABLE 2.1: Example works in a given geometry (row) and type of contribution (column).

2.1 Survey description

When one begins to visualize a graph or network using a node-link diagram, one typically imagines placing the nodes and routing the edges in 2 dimensional Euclidean space; the space of the computer screen or a sheet of paper. However, representations in different geometries have distinct visualization benefits, such as natural focus+context, or the possibility to obtain a more faithful representation. There have been several works in applying graph visualization techniques beyond the standard 2 or 3 dimensional Euclidean space. Spherical [59, 144], hyperbolic [83, 94], and toroidal [24] geometries have been considered in many visualization papers, dating back to the 1990s. Some user-studies have also been performed [23, 36] and there are many application-specific implementations [19, 56, 62, 95]. In this paper we review relevant papers and analyze by geometry, (spherical, hyperbolic, torus), by contribution (technique, evaluation, proof, application), and by graph class (e.g., tree, small-world, large).

Several series of papers on non-Euclidean graph visualization appeared near the turn of the last century and recently this subject is of interest again to visualization researchers. While there are many surveys on graph visualization in general, none consider visualization in non-Euclidean spaces, leaving a notable gap in the literature. There are several graph visualization surveys for specific types of graph data, such as dynamic networks [7], multi-layer networks [86], and multi-variate networks [100]. There are also more general surveys on graph visualization with topics on group structures [133], and scientific visualization [139]. Von Landesberger et al. [135] survey visualizations of large networks, in which they point out that hyperbolic layouts scale well to large trees, though the survey itself does not overlap in content with ours.

We are guided by similar surveys on more narrow topics. Kale et al. review dynamic multivariate networks [67] and use a similar multi-dimensional tagging system to our methodology. Possibly the most related survey to our own is a survey paper analyzing the state of the art in geospatial network visualization [121]. While geospatial networks are (implicitly or explicitly) embedded on a sphere, these positions are typically given and have a very specific geographic interpretation. We also include graph visualization techniques that put arbitrary graphs on the sphere (not necessarily geographic), as well as in other geometric spaces.

We focus our attention on node-link diagram representations in non-Euclidean (or Riemannian) spaces. We initially generated a candidate set of papers by DBLP API, which includes papers from the proceedings of *IEEE VIS*, *IEEE PacificVis*, *EuroVis*, and the *Symposium on Graph Drawing*, along with the journals *TVCG* and *CGF*. Our search query was "graph | network+visual | draw | layout+hyperbol | spher | riemman | torus", and returned an initial corpus of 43. We supplemented this with prior knowledge of the field, of which there were 13 additional papers that did not come up in the DBLP query. While reading and re-reading these papers, we paid close attention to the related work cited in them. This yielded an additional 8 papers, bringing the final total to 64.

2.2 Methodology

We consider three dimensions in our analysis of the related work, described below along with our technique for collecting papers.

What is the geometry? We make note of the specific geometry studied for the graph visualization in the paper. We identify three non-standard geometries that are used for

graph visualization:

- **Spherical geometry** is the surface of a 3-dimensional sphere, used to depict graphs with geographic or geographic-like information.
- **Hyperbolic geometry** is analogous to the sphere with 'opposite' curvature. Most notable for its exponentially expanding space allowing for 'perfect' drawings of trees and hierarchies.
- **Toroidal geometry** is the surface of a 3-dimensional ring or 'doughnut'. Admits drawings of some non-planar graphs without crossings and can be embedded in the plane with no distortion.

What is the contribution? A cursory glance at the related work shows that there are many more theoretical and technique papers on this subject than human subject studies to show how such visualizations benefit us. We hypothesize that the contribution dimension of our analysis will be the most insightful in revealing what remains to be done in this subject area. We identify four type of contributions:

- **Technique** denotes papers whose main contribution is layout algorithm or drawing style.
- **Evaluation** are papers which include some experiment involving human participants to demonstrate a technique's effectiveness.
- **Proof** style papers are theoretical papers that present results about embedding graphs in non-Euclidean space.
- **Application** papers provide a specific application in non-Euclidean geometry, such as showing how a given dataset can be analyzed.

What types of graphs? The third dimension of our analysis is the type (or class) of graphs a paper considers in its scope. These include strict definitions such as trees and planar graphs, as well as more fuzzy definitions such as complex networks. To be consistent, we include a paper in a graph type category if the paper's authors use that word to describe their graphs.

In summary, we review work on graph visualization in non-Euclidean spaces and provide a taxonomy and analysis. We believe this survey is timely, with recent interest on the subject in the visualization and human-computer interaction communities, and will provide interesting directions for future research where much has yet to be done.

2.3 Background on non-Euclidean and Riemmanian Geometry

Geometry is among the oldest and most studied areas of mathematics, with roots in ancient times. Around 300 BC, Euclid wrote his *Elements* which employed the notion of an axiomatic system: a mathematical system in which all statements should be proved from a small number of indisputably true axioms (postulates). Euclid gave five axioms which make up what we now call Euclidean geometry.

1. Any two points describe a (unique) line.
2. Any line segment can be extended to a line.
3. A circle is described by a center and radius.
4. All right angles are equivalent.

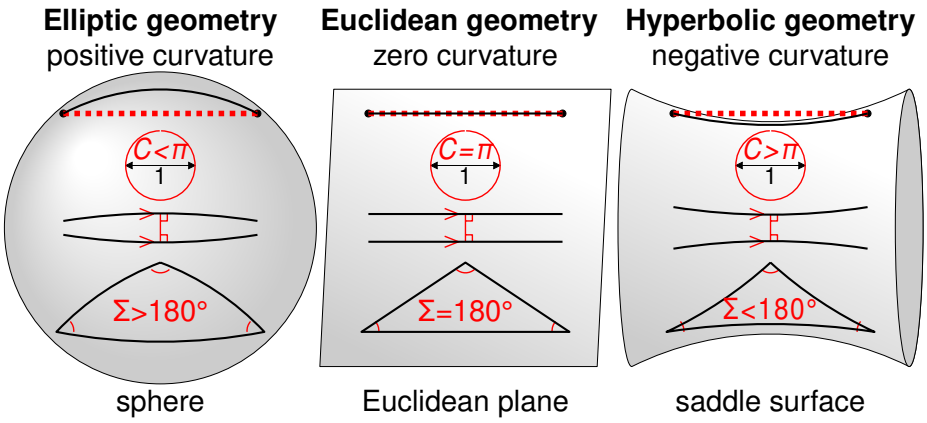


FIGURE 2.1: Illustration of non-Euclidean geometries and Playfair's axiom (middle two lines). Image credit `https://commons.wikimedia.org/w/index.php?curid=94781281`

5. "That, if a straight line falling on two straight lines make the interior angles on the same side less than two right angles, the two straight lines, if produced indefinitely, meet on that side on which the angles are less than two right angles."

The fifth postulate (which states that any two non-parallel lines will intersect exactly once) is noticeably more involved than the previous four. Mathematicians tried for centuries to prove the fifth postulate using the first four, but invariably failed. Despite its perceived clunky-ness, the fifth postulate seemed necessary to prove even simple theorems such as the Pythagorean theorem about right angle triangles.

2.3.1 Hyperbolic geometry

As it would turn out, the Euclidean notion of parallel lines is not necessary for a consistent geometric system. It was independently discovered by mathematicians Nikolai Lobachevsky and János Bolyai around 1830, that by allowing arbitrarily many

parallel lines to exist, a distinct geometry arises. This non-Euclidean geometry came to be known as *hyperbolic geometry*.

The fifth postulate is equivalent to the following statement, known as Playfair's axiom: "Given a line, L, and a point, p, not on L, ***exactly one*** line parallel to L can be drawn through p"; see Fig 2.1 for an illustration. Then, hyperbolic space is obtained by replacing the fifth postulate by the statement: "Given a line, L, and a point, p, not on L, ***infinitely many*** lines parallel to L can be drawn through p."

Most properties of Euclidean geometry hold in hyperbolic geometry; for instance, any two lines can intersect at most once. Things differ when parallel lines or more than two lines are involved. Hyperbolic triangles have many properties distinct from traditional trigonometry, notably that the sum of their internal angles is strictly less than 180.

Another notable difference is the absence of relative scale. In Euclidean geometry, affine transformations are linear maps of the plane to itself that can be nicely represented as a function with real valued matrix A and vector b, $f(x) = Ax + b$. This includes translations, reflections, rotations, and resizing (scale). Once we have a drawing, we can shrink it to a tablet screen or expand it to a billboard and shapes (i.e. angles) are unaltered. In hyperbolic space however, there is a relationship between distance and angles. Translations, reflections, and rotations can all preserve angles, but resizing will not.

2.3.2 Spherical (elliptical) geometry

Although the geometry of the surface of a sphere had been studied for many centuries (e.g., work by Theodosius around 200 B.C. [112]), it was considered a separate system with its own rules. Even today, the sphere is often thought of as a surface embedded in

3 dimensions of Euclidean space, but its geometry can be studied intrinsically only using the 2 dimensions of its surface.

It wasn't until 1859 that the classical understanding of spherical geometry was unified with the new ideas produced from the study of non-Euclidean geometry [21]. Cayley introduced *elliptic* geometry, a sphere with antipodal points identified to satisfy the result (present in both Euclidean and hyperbolic) that lines can intersect at most once (note that on a full sphere, lines always intersect twice). Then elliptic geometry is obtained by the following modification of Playfair's axiom: "Given a line, L, and a point, p, not on L, ***no*** lines parallel to L can be drawn through p." Or in other words, all lines must intersect.

Although elliptic geometry removes the trouble of antipodal points, we generally refer to spherical geometry (also called doubly elliptic geometry) throughout this paper, as it is more widely used for visualization.

While the first four of Euclid's postulates hold in spherical geometry, the third must be slightly modified; a circle of arbitrary radius cannot exist. Since the sphere is finite, a circle can only have radius equal to the radius of the sphere it exists on.

Many of the unique properties of hyperbolic geometry have the opposite implication for spherical geometry. For instance, the sum of internal angles of a spherical triangle is strictly greater than 180 as seen in Fig. 2.2. Spherical space also has an absolute scale though since the surface is finite there is a strict upper limit on how much things can be stretched.

2.3.3 Riemannian and Toroidal geometry

Riemannian geometry is the study of manifolds (spaces which locally resemble Euclidean space) together with a Riemannian metric (an inner product on the tangent space at each

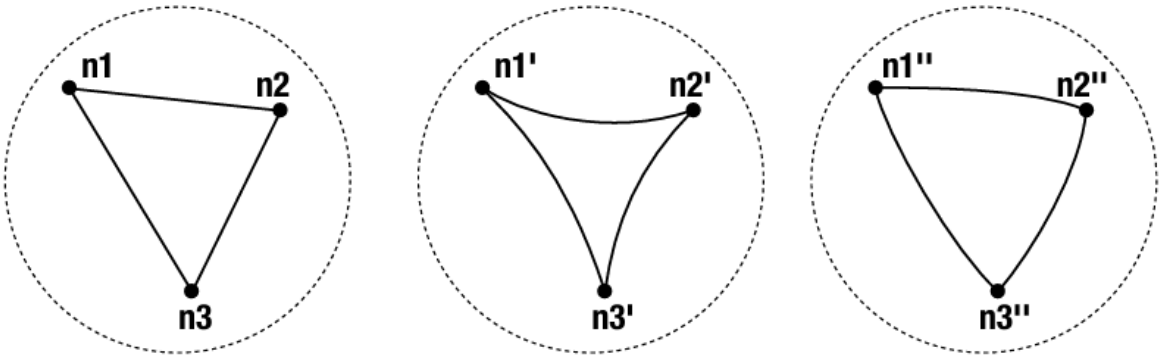


FIGURE 2.2: Comparison of visualization of a triangle in Euclidean plane, hyperbolic plane, and sphere [36]. Note how the hyperbolic triangle has smaller angles while the spherical triangle has larger.

point). This includes the Euclidean and non-Euclidean geometries with the standard dot product, along with many other surfaces. The torus has been used for graph visualization and is worth exploring.

A torus is a product of two circles that is topologically closed, which produces the typically doughnut-shape. The torus can be described as the surface of an axis of revolution of a circle in 3 dimensional space. It is an example of a surface with genus 1, where genus is the maximum number of closed, circular cuts that can be made and the surface remain connected. Note that the spaces discussed before all have genus 0.

A property of interest in graph drawing is the crossing number of the graph [119], which is the minimum number of crossings required to draw a graph on a 2 dimensional surface of genus 0. A surface with higher genus admits smaller crossing numbers [64]. Notably, the two cannonical non-planar graphs (K_5 and $K_{3,3}$) can be realized without crossings on the torus.

Although the surface of a torus is often depicted in 3 dimensions with curvature, a flat torus actually has curvature of 0, just like the Euclidean plane. Similar to how one

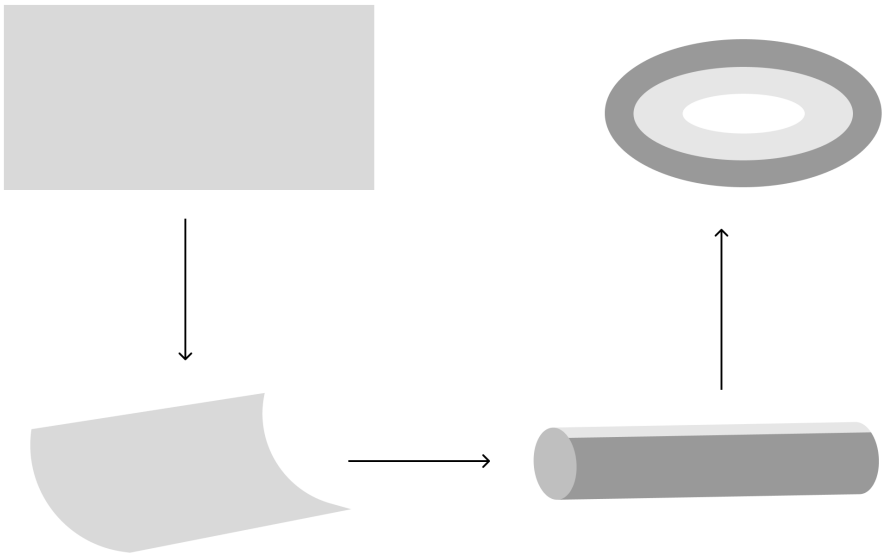


FIGURE 2.3: Creating a torus from a piece of paper. Note how first two opposite ends of the rectangle are connected, then the top/bottom of the cylinder is connected to create the torus. This type of torus is generally called a flat torus.

can bend a sheet of paper into a cylinder in 3 dimensions with no distortion, the torus could be created from a sheet of paper with no distortions if we had a 4th dimension to work with. While no perfect 3 dimensional embedding is possible, a nearly perfect embedding of the flat torus is obtained in 2 dimensions by a rectangle with the left/right and top/bottom edges identified as displayed in Fig. 2.3.

2.3.4 Terminology

A ***graph*** (network) is a finite set of objects V (called *vertices*) together with a set of relationships on those objects $E \subseteq V \times V$ (called *edges*). Edges may be directed or

undirected, and potentially weighted. Graphs arise in many application areas, making their visualization of particular research interest. While some fields make a distinction between graphs and networks, we treat them interchangeably in this survey.

A common visualization idiom for graphs is the ***node-link diagram***, with vertices drawn as nodes (simple shapes) and edges drawn as links (curves between the corresponding vertices). Node-link diagrams are constructed from an ***embedding*** (layout) of the graph, an assignment of positions to the vertices and a routing of edges. The resulting picture is called a ***drawing*** of the graph. In a typical straight-line drawing, the routing of the edges is totally determined by the placement of the incident vertices, as the line segment between them.

In the plane, one can draw a straight line segment between two points and measure its length – this is known as the Euclidean distance or L_2 norm. This intuitive concept can be generalized to arbitrary surfaces with the ***geodesic***: the length of the shortest curve between two points. In non-Euclidean geometries (such as 2D sphere and 2D hyperbolic space), this is a circular arc defined by a closed-form function. In Euclidean and hyperbolic space, there is a unique 'straight line' between any two points. For the sphere there may be many possible straight lines if the points are antipodal, but the length is the same for each. On the torus, however, there are several possible lines so it is required to measure the length of each of them to compute the geodesic length

Non-Eulcidean geometries cannot be perfectly embedded into the 2 dimensional plane. The study of cartography is a field dedicated to this problem [126] as it has long been known for the sphere. A ***projection*** is a linear map that preserves some combination of angles, areas, geodesics, or distances but a projection (from spherical or hyperbolic space) that preserves all these aspects simultaneously is not possible.

There are many classes of graphs that are subject of particular interest. ***Trees*** are graphs that contain no cycles. If a root of the tree is specified, this encodes a hierarchy, where each non-root vertex has exactly one parent it descends from. ***Planar graphs*** are graphs which can be drawn as a node-link diagram without any two links intersecting. ***Complex networks*** are graphs that arise naturally in real-world networks, defined by phenomena such as small average path length and high clustering coefficient.

Many dimension reduction (DR) techniques can be applied to graph embedding. These techniques take as input a distance matrix, D, on a set of high dimensional Euclidean coordinates where $D_{i,j}$ is the distance between objects i and j. Defining distance with the graph theoretic (i.e. shortest path) distance allows one to apply these DR techniques to graph embedding. A particular technique used in graph embedding is ***multi-dimensional scaling (MDS)*** (stress minimization), which tries to match the pairwise embedding distances to the original input distances.

2.4 Geometry

Our first dimension of analysis is geometry. We give a brief overview of the chronology of how each geometry has been used in graph visualization.

2.4.1 Sphere

Spherical geometry has been used in visualization since antiquity. Much of the visualization work on the sphere has been inspired from the Earth's globe, with orthographic projections and map metaphors being common.

Gross et al. [59] describe an approach for graph layout on the sphere. This is a force-directed method which places nodes initially with an initialization scheme that attempts to keep nodes close to the surface of the sphere. Edges are not routed along geodesics, but as straight-line segments in 3D Euclidean space.

Sangole and Knopf [118] apply a self-organizing map (SOM) that creates closed spherical surfaces from the input data, transforming the discrete input into a continuous distribution on the sphere. For a globe-like feel, they doubly encode the density of the input data with a rainbow color map and the height of the point on the globe. Low density areas are blue and the lowest points to resemble oceans, while high density areas are red and high, resembling mountains.

Wu and Takatsuka [144] adapt a SOM to the sphere for multivariate network visualization. This requires adapting the grid used in a SOM to a triangulation of the sphere. They also implement a visual interface to view the embedding with either an orthographic or cylindracal projection. The spherical SOM has also been used in the field of networking [129], to visualize file sharing activities on a computer network with the help of an orthographic projection.

Traditional force-directed algorithms were generalized to the sphere [73] and the tree visualization scheme SphereTree was created in [17]. MDS has also been generalized to the sphere with a few different approaches. Osinska et al. [103] perform 3D Euclidean MDS and normalize the vectors to lie on a unit sphere. Another approach is to perform Euclidean MDS in 3D, but constrain the optimization so that the points never leave the surface of a sphere during the algorithm [34, 108]. Finally, [38, 90] describe methods to perform MDS using latitude and longitude coordinates, constraining the computations and movements to the surface of the sphere.

Spherical layouts have been investigated in an immersive setting such as virtual reality [80, 145], discussed further in section 2.5.4. Several other examples of spherical graph visualization include the Map of Science [19], as well as the “Places and Spaces" [18] and “Worldprocessor" [62] exhibitions.

2.4.2 Hyperbolic

Hyperbolic space first received attention in the visualization community due to its exponentially expanding structure, and eye catching representations such as Fig. 2.5. This particular Escher work is inspired from the Poincaré disk, a model of hyperbolic space which maps lines to circular arcs in Euclidean space and admits many aesthetically pleasing tessellations. Hyperbolic embeddings are also of interest in the field of machine learning, as a desirable latent space for embeddings [106]. Here we cover works which deal primarily with hyperbolic geometry in graph visualization.

One of the earliest approaches of graph drawing in the hyperbolic plane is by Lamping et al [83], dubbed the hyperbolic browser; see Fig. 2.6(a). They embed trees in hyperbolic space using a linear time algorithm and construct the embedding top-down from parent to children, evenly spacing children on the half-plane opposite the parent all at unit length. This is always possible thanks to the exponentially expanding hyperbolic geometry. To represent this embedding, they use the Poincaré disk which maps the entirety of hyperbolic space into a Euclidean disk. Since the entire tree is visible (up to pixel resolution) this provides a powerful focus+context effect, allowing one to see the parent node and its children in high detail, but also get a sense of how deep or wide each of the subtrees are. Clicking on a node places it in the center of the projection, via a smooth transition in the drawing. This approach was generalized to arbitrary graphs

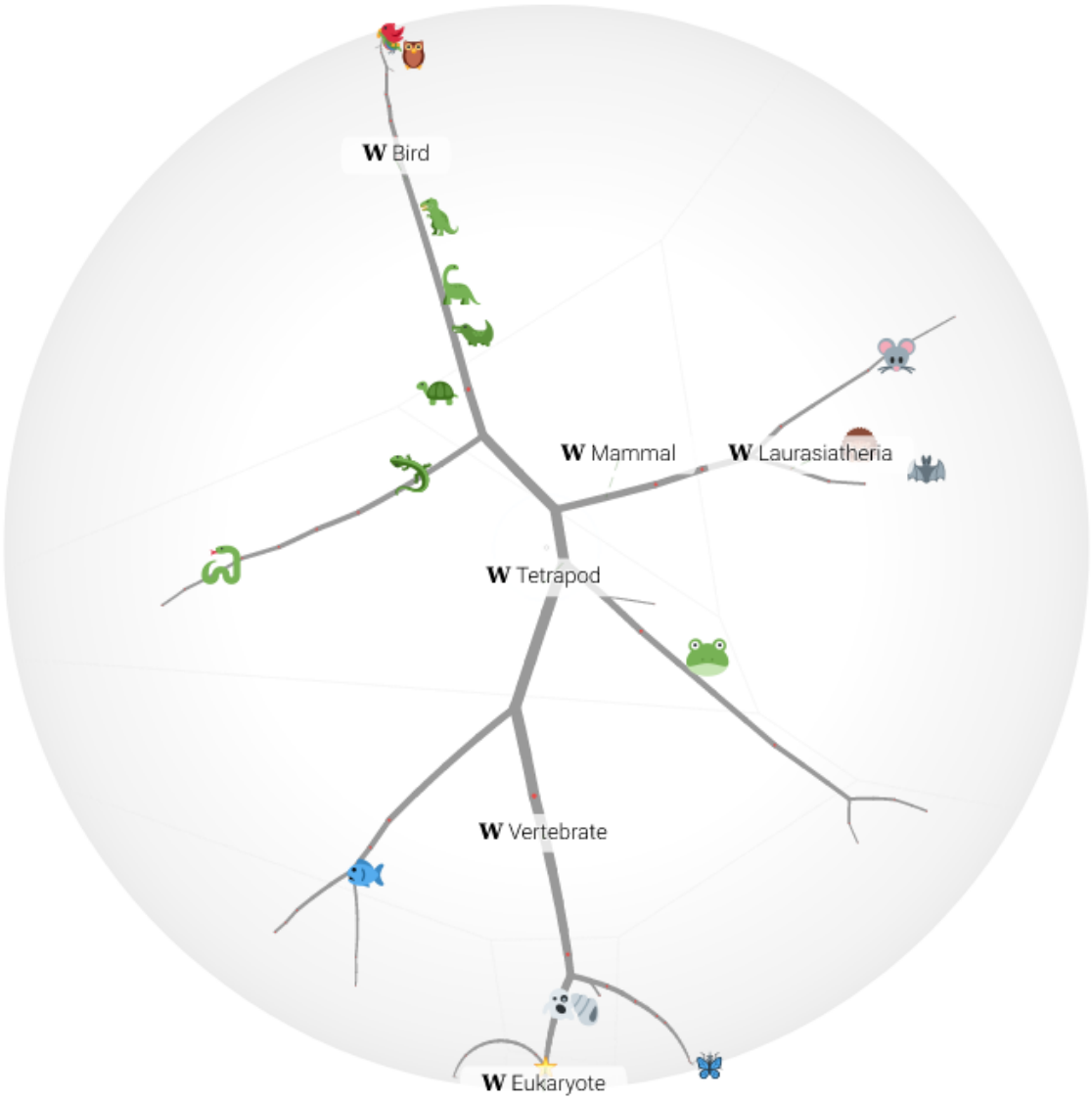


FIGURE 2.4: Hyperbolic Tree of Life for browser based hyperbolic visualization. Source code and demo found from [56]

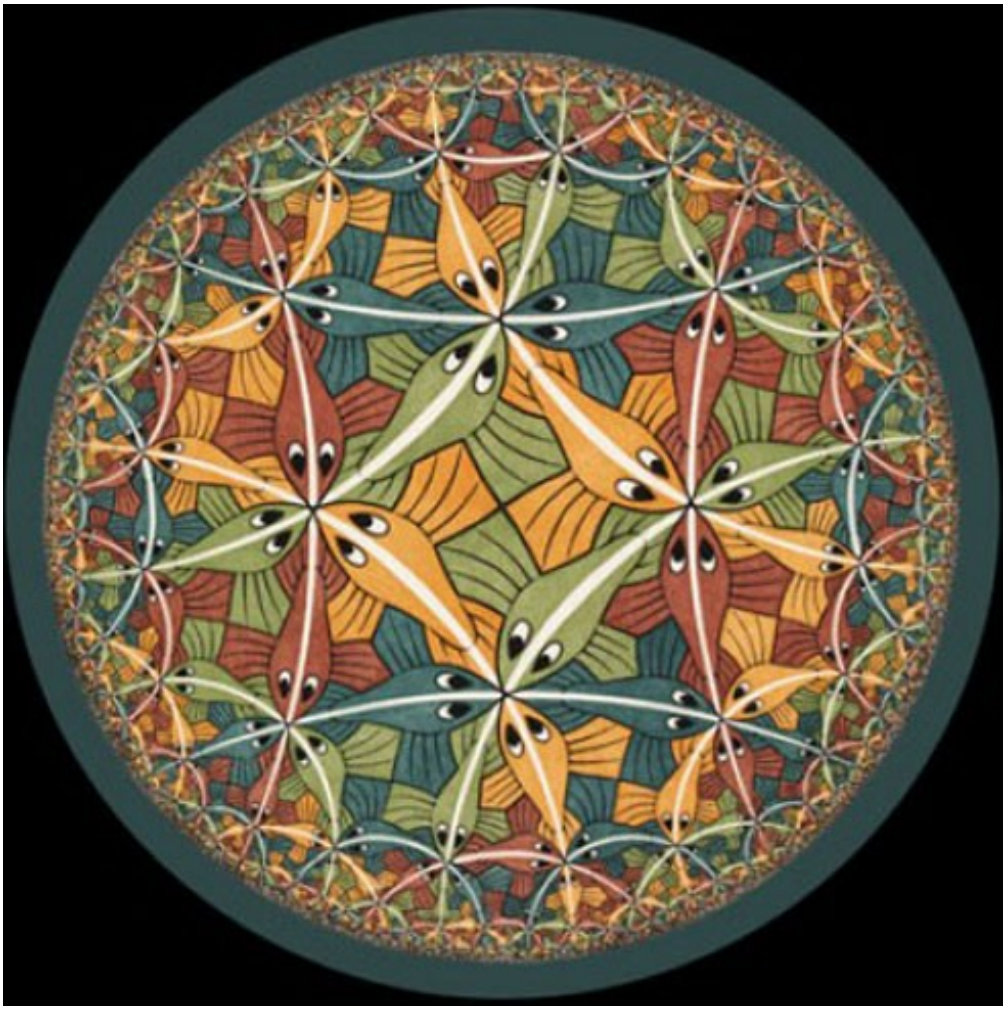
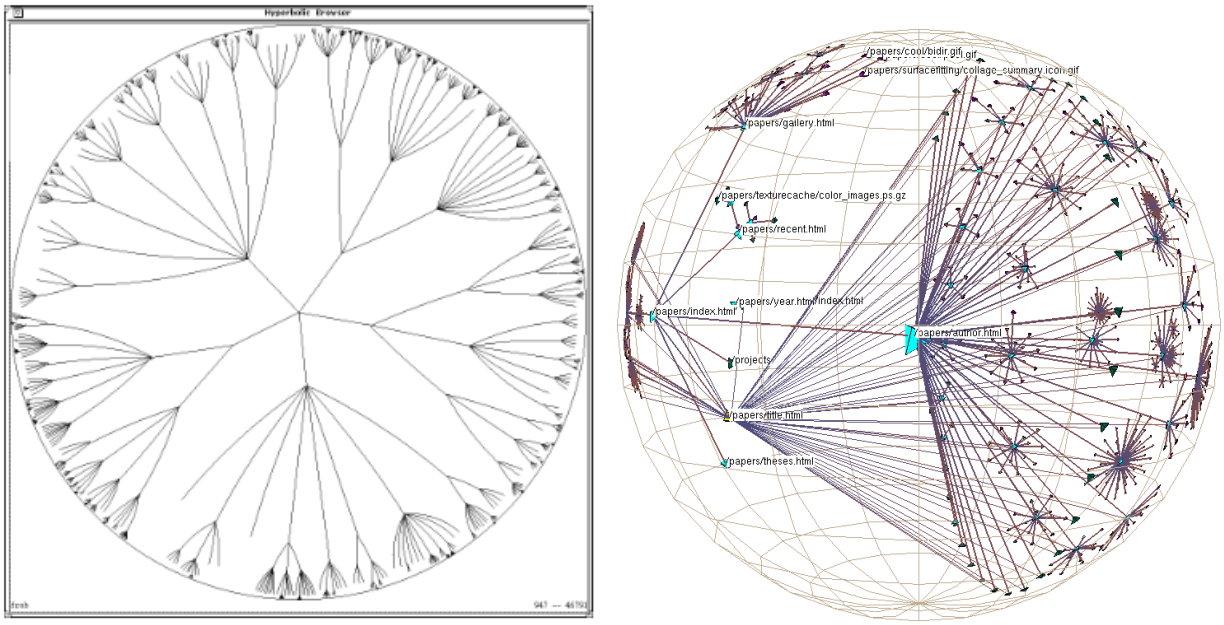


FIGURE 2.5: *Circle Limit III*; M.C. Escher [45]

by [63] by first computing a spanning tree and filling in the removed edges after layout (or keeping them hidden).

Several application papers made use of the on the Lamping et al. [83] technique. Phylogenetic trees [10], word trees [6], the tree of life [56] all made natural candidates, as large trees with interest to domain experts. The hyperbolic browsing system also made its way into several systems and tools, such as the Hierarchy Visualization System (HVS) [4], the Java InfoVis Toolkit [8], and TreeBolic [16].



(a) Hyperbolic Browser from [83] (b) H3 from [97]

FIGURE 2.6: Hyperbolic tree visualizations either took the 2 dimensional (a) or 3 dimensional (b) approach.

More recently, Lamping et al.'s [83] hyperbolic browser has been re-implemented using d3.js [57, 58] for web-based visualization of large hierarchies such as the tree of life [56] as shown in Fig. 2.4.

General graphs can be drawn in hyperbolic space as well, [91] provide several different techniques to do so, building on earlier approaches from [73, 108, 137].

While most work considers the 2-dimensional hyperbolic plane, Munzner used 3D hyperbolic space to visualize trees [94, 95, 97], one of the examples shown in Fig. 2.6(b). Just like 2 dimensional hyperbolic space can be projected into the 2 dimensional plane, 3 dimensional hyperbolic space can be projected into the 3 dimensinal Euclidean space. While the Poincaré projection is preferred in two dimensions due to its simple construction and conformal properties, it maps geodesics to curved arcs which are more

difficult for a reader to follow in 3 dimensions. For this reason, Munzner chose to use the Beltrami-Klein projection in spite of the larger distortion, as it maps geodesics to Euclidean lines. The ball itself can be rotated when viewed from outside, or entered by zooming to get a closer look. The node in the center of the ball can be changed by navigating the tree. This technique was later extended to general graphs by first computing a spanning tree [96], laying out with the above technique, and filling in the cycles afterwards. Munzner's work has been re-implemented in two subsequent systems: Walrus [66] and h3py [146].

In the early 2000s, hyperbolic space received attention as a potential target space for self-organizing maps (SOMs) with the hyperbolic SOM (HSOM) [138]. Although restricted to a grid, the HSOM algorithm is only linear in complexity, requires only input distances, and can be used as a visualization. This algorithm was integrated into a visualization tool [136] where the SOM lattice was drawn on a Poincaré disk and the data points drawn as 3 dimensional objects coming up out of the plane; see Fig 2.7.

2.4.3 Torus

Torus-based graph visualizations are relatively recent in comparison to the other non-Euclidean geometries above. Early on, torus embeddings of graphs was primarily of theoretical interest. Since the torus has genus 1, it admits crossing-free drawings of many non-planar graphs. An example of a non-planar graph (K_5) drawn without crossings is shown in Fig. 2.9 [64].

Kocay et al. [74] first describe an algorithm to embed any 2-connected toroidal graph on the torus, where a toroidal graph is one that can be embedded without crossings on the torus. They do this by generalizing Read's algorithm for Euclidean planar graphs,

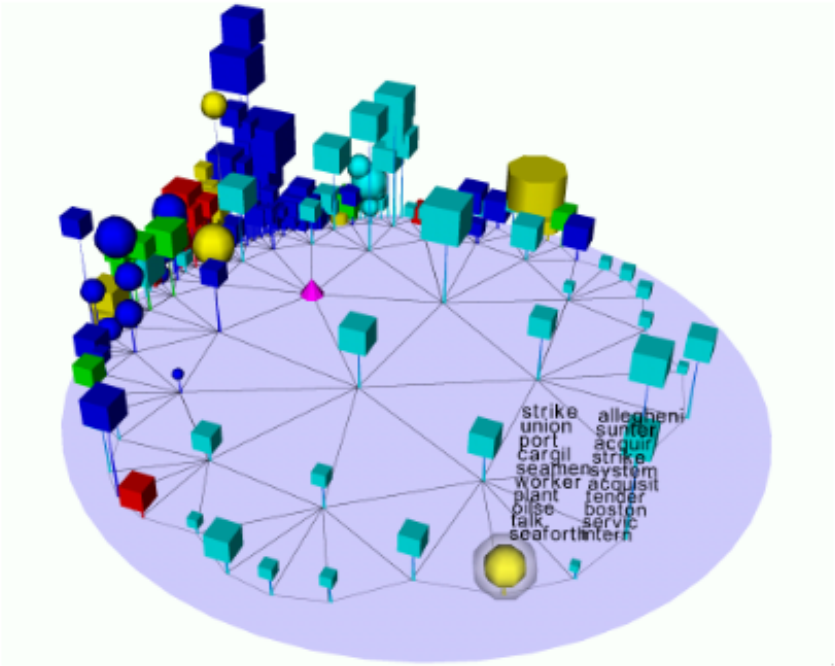


FIGURE 2.7: Hyperbolic self-organizing map visualization from [136]. The shown flat disk is a Poincaré projection, while the data are drawn as 3 dimensional shapes coming out of the plane.

and go on to show that this will always produce a crossing free drawing on the torus. These drawings are depicted as rectangles, with the surrounding area duplicated to show the 'wrapping around' of the torus; see Fig. 2.8.

This rectangular embedding of the flat torus has been used to create planar drawings of non-planar graphs, by duplicating vertices and edges like the sides of the square in Fig. 2.8. Biedl [9] shows that any toroidal graph has a visibility representation on a rectangular flat torus, where a visibility representation is an embedding in which vertices are mapped to horizontal line segments and edges are mapped to vertical line segments.

More recently, a series of papers has introduced a practical technique for graph layout and interactive visualization of graphs on the torus. Starting with [24], Chen et al. identify

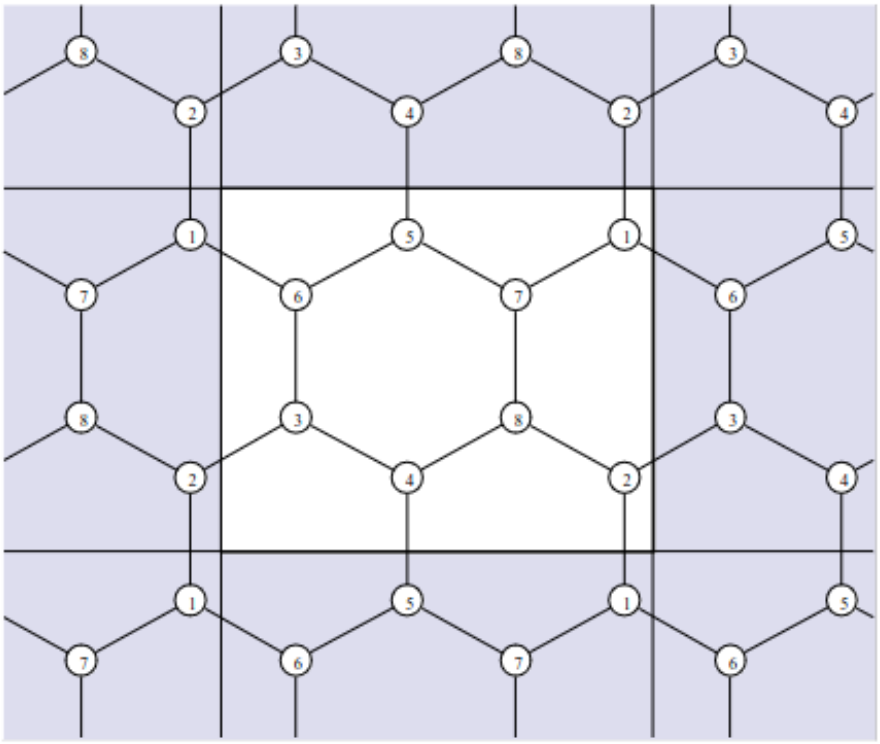


FIGURE 2.8: Torus graph embedding depiction from [74]. This tiling approach is often taken for small graphs on the torus.

3 research questions that they aim to address:

- **RQ1**: develop layout algorithms that take advantage of the extra flexibility of graph layout on the torus;
- **RQ2**: determine how we can best visualize the layout of a node-link diagram on the surface of a torus on a piece of paper or 2D computer monitor; and
- **RQ3**: determine what, if any, perceptual benefits graph layout on a torus has over standard layout on a 2D plane.

This initial paper begins to address these challenges by proposing a stress-based algorithm for torus layout, providing some interaction styles, and a small but thorough

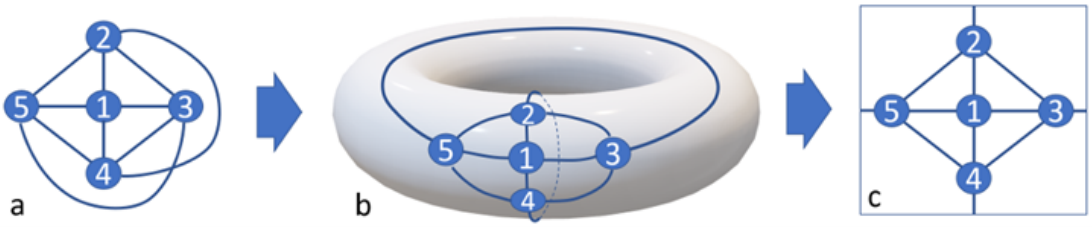


FIGURE 2.9: The canonical non-planar graph K_5 can be drawn without crossings on the torus, as shown in this example from [24].

user study to evaluate the different interactions and how they compare to traditional drawings. A follow up paper [23] improved the previous optimization, and took a closer look at cluster based tasks on the torus, performing a user study and found some advantages over traditional drawings (e.g., when identifying clusters).

Most recently, Chen et al. [25] use eye tracking data to further understand how readers use the toroidal graph drawings to make inferences. They use the tracking data to identify patterns and compare different representations of the torus. They found that duplicating the torus at each of the 8 neighboring grid squares was the most effective, but poorly utilizes space. This leaves a space-efficient full-context torus visualization as future work.

2.4.4 Other

Focus+context effects can also be achieved with lens effects [130, 131]. In particular, at first glance the Poincaré disk appears to resemble a fisheye lens, which in some cases is preferable to a flat drawing [50]. 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.

2.5 Contribution

Our second dimension of analysis is contribution. We discuss papers based on what we believe is their primary contribution. The four broad categories are *techniques, evaluation, theory, and application*. It is common for any visualization research paper to have some mix of all four of these, in this case we select which contribution we believe to be the most novel.

2.5.1 Technique

The classical straight-line graph layout problem is to assign 2 dimensional Euclidean coordinates to the vertices of the graph so that the resulting drawing captures well the underlying graph properties (e.g. preserving distances, minimizing crossings, preserving symmetries, etc.). This assignment is known as an *embedding*, and it totally determines a straight line drawing. Nodes are drawn as circles centered at the embedded position, and edges are straight-line segments connecting the corresponding endpoints. There are many different approaches for producing Euclidean graph embeddings; the survey by Gibson et al. includes a good selection [55].

The straight-line graph layout problem naturally generalizes to Riemannian spaces, by instead finding an embedding in the desired space with the straight line requirement becoming geodesic curves. We identify four distinct approaches to solving this problem algorithmically.

Projection-Reprojection

The first approach is to have the algorithm take as input a Euclidean drawing of the graph, and project the positions into the desired space. This is particularly intuitive in spherical space, where geographic map projections from the sphere to the Euclidean plane been studied since classical times. This is one of the approaches presented by Perry et al. [108], referred to as the “Projection–Reprojection" method. Here, any Euclidean visualization (e.g. node-link diagram) can be drawn on the sphere, by treating the input as a sheet of paper, and ‘folding’ it around the curved sphere surface; see Fig. 2.10. A similar technique for the sphere is presented in [36] and another is adapted to hyperbolic space in [91]. Any conceivable node-link based graph visualization idiom can be adapted to a non-Euclidean space using this technique, including map-style drawing [51] or BubbleSets [30], as it only requires a set of coordinates and lines to draw between them. Additionally, this method requires only linear time (given a pre-computed input) and is straightforward to implement. However, this method does not use any unique aspect of the target geometry (such as the ‘wrap-around’ of the sphere, or exponential expansion of hyperbolic space), as shown in [108]. This method works for any space which admits a projection to Euclidean (since a projection is linear, it has an inverse), but we are unaware of any work which attempts such a method for the torus.

Tangent planes

The second technique is to exploit the fact that Riemannian geometries are locally flat, and perform a force-directed style layout algorithm in the tangent spaces about the vertices. This is the approach in [73] and generalizes layout algorithms driven by forces. When computing the forces on a vertex, first compute the tangent plane about that point (a

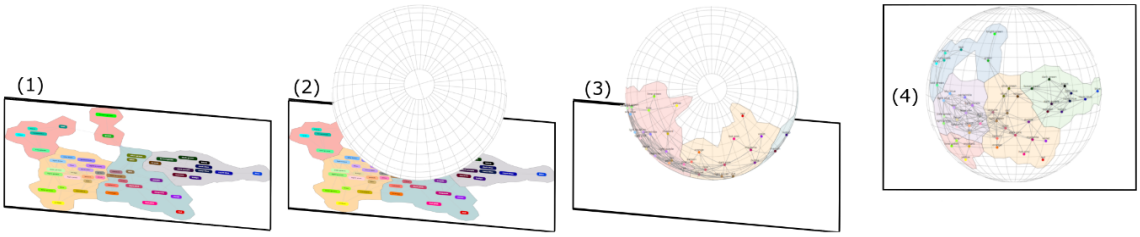


FIGURE 2.10: Example from [108], illustrating the Projection-Reprojection drawing method for the sphere. By treating the input Euclidean drawing as a sheet of paper, the spherical drawing can be obtained by 'wrapping' the input around the ball.

plane which, in 3 dimensions, intersects the surface at exactly one point). Project the other vertices into that plane, and compute forces and update the central vertex position as normal. Finally, project the central vertex back into the target space. Repeatedly apply until the quality of the drawing is sufficient. The tangent plane technique has been reapplied in [36, 91]. While this approach makes use of some unique properties of the geometry, this method is computationally quite expensive and does not achieve as high of quality as the later approaches [91]. While the formulation is for a general Riemannian geometry, we are unaware of any implementation for the torus.

Constrained optimization

The third technique requires us to view our target space as the 2 dimensional surface of a 3 dimensional object. Then, this surface is defined by a set of 3 dimensional Euclidean coordinates that satisfy some property or equation (e.g. a unit sphere is all points with magnitude exactly one). One can perform a force-based graph layout technique in 3 dimensions, requiring that any movement of the vertices still satisfies the properties of the desired space.

This particular technique has been applied only to the sphere, first described by [34] for dimension reduction and adapted to graph layout by [108]. This is also the approach taken by [114], who show this is related to information diffusion models, such as independent cascade. Rodighiero [111] describes this constraint as 'gravity' attracting the nodes to a spherical surface. The Fruchterman-Reingold algorithm was also adapted in this fashion [60]. Although this is easy to describe at a high level, the technical details are quite difficult. One must ensure the step size is sufficiently small, errors will quickly accumulate in the 3 dimensional representation, and the underlying mathematics is quite complicated.

This approach could be extended to hyperbolic space and the torus, but has not been done so. For instance, just like the equation $x^2 + y^2 + z^2 = 1$ describes a sphere in 3 dimensions with components x, y, z the equation $x^2 + y^2 - z^2 = 1$ describes a hyperbolic surface.

Native formulation

Something common to the previous three methods is that they all require the use of the friendly properties of Euclidean space. It is desirable to solve the problem entirely within the target geometry, for fewer computations, simplicity, and in order to fully benefit from the advantages of the particular geometry (e.g., more space in hyperbolic geometry).

The tree embedding algorithms discussed for hyperbolic geometry [83, 97, 117] are native formulations but they are limited to trees or spanning trees of graphs. Similarly, there are algorithms to generate crossing free embeddings of toroidal graphs on the torus [74], but again these are limited to toroidal graphs.

For general graphs, a native formulation typically requires generalizing some Euclidean graph layout approach. The most popular choice for this has been stress minimization, also known as Multi-dimensional Scaling (MDS) [77]. Stress-based approaches aim to match the graph theoretic distance between all pairs of vertices to the geodesic distance between the corresponding pairs of nodes in the embedding.

This was first done in the sphere for dimension reduction by [38], then adapted to graph embedding with an improved optimization by [90]. The basic idea is straightforward: replace the L_2 norm computation in Euclidean MDS with the spherical geodesic distance function. Since positions on the sphere are completely defined by a pair of angles, and this function is differentiable, one can find a minimum without considering cumulative error, or points ending up outside the sphere.

Classical MDS has been explored in hyperbolic space, by replacing the conversion to similarities with an appropriate hyperbolic scaling function [29,117]. Using a similar idea to the spherical MDS, metric and non-metric MDS have been generalized to hyperbolic space by incorporating hyperbolic geodesic distance into the cost function [91, 136–138, 148].

While [24] use a stress minimization scheme to compute their toroidal layouts, the torus is missing a closed-form distance function. Chen et al. overcome this by checking which of the straight-line distances is closest to the ideal for each pair of vertices, but a toroidal layout that uses closed form functions is an interesting direction for future work.

2.5.2 Evaluation

Here we focus on papers whose primary contribution is an evaluation of non-Euclidean graph layouts, and also, more generally, typical ways in which such layouts are evaluated.

Quality Measures

The evaluation in most technique and application papers are quantitative metrics: metrics that evaluate a layout or embedding with a single number. This may be the time in seconds to produce the drawing, or a faithfulness or aesthetic metric which captures how well the drawing reflects the data or how readable the drawing is. Such quality metrics were originally defined and studied in the Euclidean setting.

Aesthetic metrics: Properties such as number of crossings, crossing angle, angular resolution, vertex resolution, symmetry, and average edge length are all desirable to optimize for readability, with number of crossings being particularly impactful [109]. These concepts can extend to non-Euclidean geometries, but there is not much work in this area. Eppstein shows that the vertex and angular resolution is bounded in hyperbolic space, meaning one cannot achieve a good resolution in general [40]. The torus admits smaller crossing numbers in theory [74], and this has been shown to be achievable for some real world graphs [23]. However, much remains to be explored in this area. For example, hyperbolic spaces can have symmetries not possible in Euclidean space (see Fig. 2.5), but measuring hyperbolic symmetry cannot yet be done numerically.

Faithfulness metrics: Faithfulness metrics instead measure how well the embedding captures the information present in the data [98]. These include metrics such as stress [53], neighborhood preservation [76], and distortion [75].

In the graph drawing literature, the normalized stress of a layout is a standard quality measure [52,76,149], defined as:

$$\sum_{i<j} \frac{(||X_i - X_j|| - d_{i,j})^2}{(d_{i,j})^2} \tag{2.1}$$

where X_i is the embedded position of vertex i, and $d_{i,j}$ is the desired distance between vertex i and vertex j.

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 the size of a layout can have drastic effects since they have an absolute scale [90]. For this reason, *distortion* [91, 117] should be used as a quality metric in place of the normalized stress, defined as

$$\binom{|V|}{2}^{-1} \sum_{i<j} \frac{|\,||X_i - X_j|| - d_{i,j}|}{d_{i,j}} \tag{2.2}$$

Distortion is less sensitive to the dilation of the drawing, but note that selecting the correct size is still important to finding a good embedding. Stress has also been generalized to the torus in [24]. Distortion can be generalized to the torus, but no work so far makes use of it for toroidal embeddings.

Other quality metrics such as neighborhood preservation have yet to be studied in non-Euclidean spaces.

User studies

While less common, an important form of evaluation for non-Euclidean graph visualization is the user study. Several papers incorporate some form of user study

as part of the evaluation/justification for a technique or application [24,123] but there are fewer papers dedicated to user evaluation.

Du et al. [36] compare Euclidean, spherical, and hyperbolic graph drawings by performing a user study. They first present their iSphere system for large graph visualization, which takes any Euclidean, straight line drawing, and performs an inverse Stereographic (conformal) projection on it to map it to the sphere. They present participants with layouts of stochastic block model random graphs, and ask them questions about some nodes or groups of nodes. The experiment design was within-subjects, each participant saw examples of Euclidean, spherical, and hyperbolic displays, supporting zooming and panning. The layout was driven by MDS for all 3 displays: standard for Euclidean, inverse stereographic projection for sphere, and [73] for hyperbolic. All the tasks are about node degrees in the drawings. In particular, they ask for: (1) the neighbor of highest degree for a highlighted node; (2) the common neighbor of highest degree for two highlighted nodes; and (3) the highest degree node along a highlighted path. The sizes of graphs were also controlled: $|V|$ ranged from {128, 512, 2048} and $|E|$ from {1024, 4096, 16384}. The number of clusters generated was 3 for each graph. For each size of graph, they used two different modularities, low and high (a graph statistic which expresses how dense the clusters in a graph are). They conclude with statistical analysis on their collected participant data. The results seemed to indicate that the sphere and Euclidean visualizations had comparable time and accuracy, while hyperbolic performed worse (see Fig. 2.11).

Meanwhile, for the torus a thorough user study was conducted in [23]. This study was primarily aimed at asking whether toroidal visualizations were more beneficial for cluster identification. The graph sizes used in the study ranged from $|V| \in$ (68,134) with

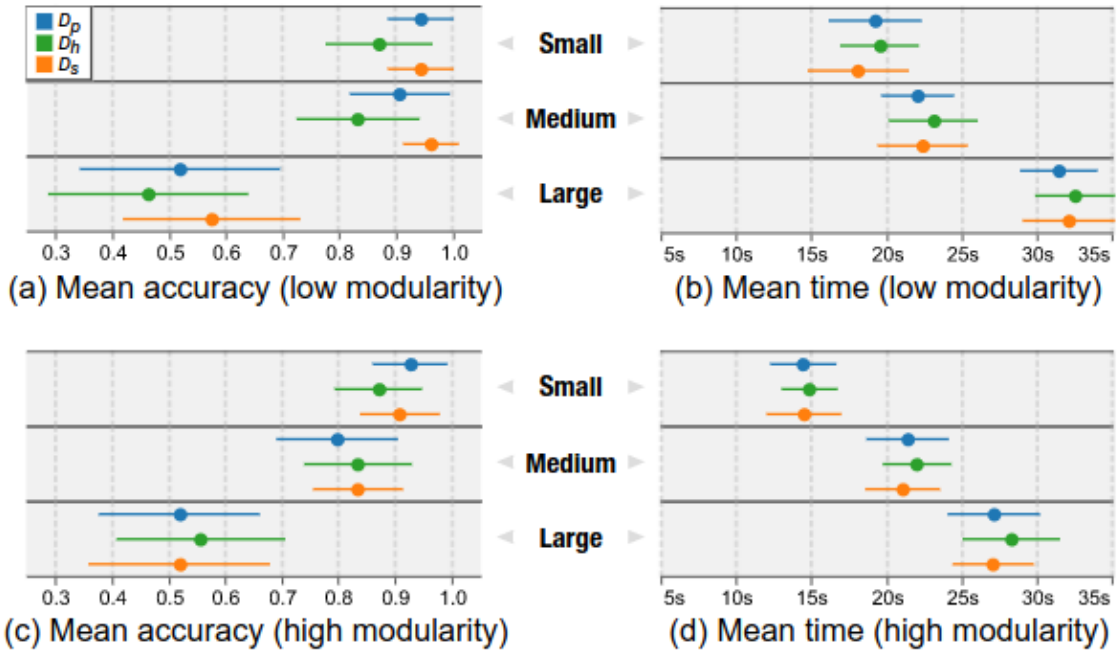
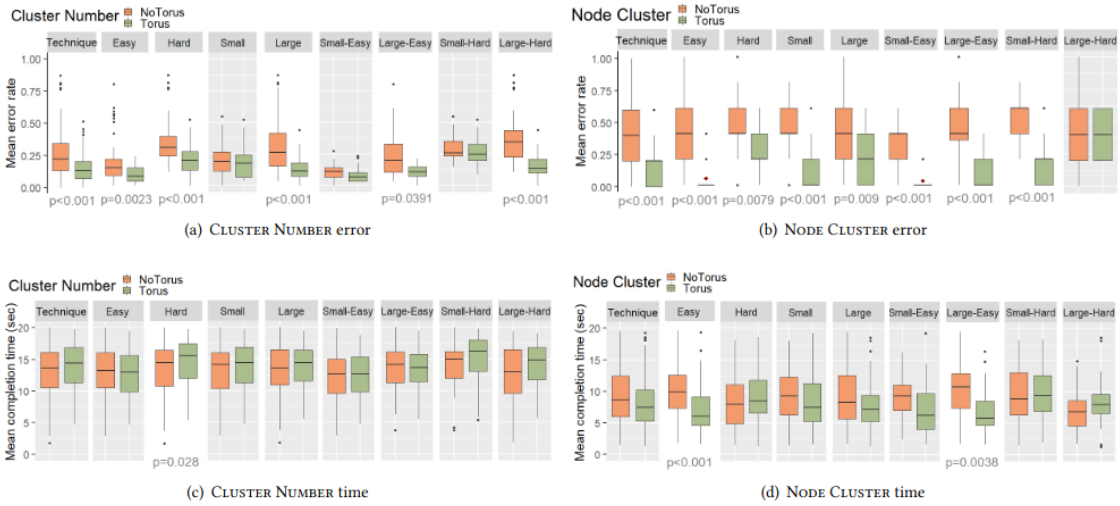


FIGURE 2.11: Plots from [36], time (right) and error (left) of participant results. The top row are results for low modularity graphs (weak clustering) and the bottom for high modularity graphs (strong clustering). The colors indicate the display geometry: blue for Euclidean, green for hyperbolic, orange for spherical.

$|E| \in (710, 2590)$. Similar to [36], they divide their graphs into small and large classes, and into low and high modularity. Graphs are randomly generated from stochastic block model type generators.

The authors registered several hypotheses on the open science forum, such as that the torus-based visualization would be beneficial over Euclidean drawings in identifying the number of clusters (in time and error) when the graph modularity is low (i.e. when clusters are not very dense). They performed a within-subjects study so that all participants see all variations; namely both the torus and Euclidean layouts with small/large, low/high modularity, and 2 tasks. The two tasks were: (1) What is the



(a) Cluster Number error (b) Node Cluster error

(c) Cluster Number time (d) Node Cluster time

FIGURE 2.12: Box plots of results from [23], showing that Torus based visualization was often more accurate than NoTorus (Euclidean) visualization of graphs for cluster counting task.

number of clusters; and (2) Do the two highlighted nodes belong to the same cluster?

The statistical analysis of the participant data provides several insights. They found statistically significant results that torus-based layouts increased participant accuracy over Euclidean layout in some cases; we include a figure showing the mean time and accuracy from their study in Fig. 2.12.

2.5.3 Proof/theory

Planar graphs are of particular interest in spherical graph visualization. Aleardi et al. [1] develop an algorithm that embeds a planar graph on the sphere without crossings. The algorithm is efficient, but can generate very short edges, resulting in occasionally undesirable layouts. The authors present several use cases, including as an initialization for a Euclidean layout.

Kryven et al. introduce a measure of visual complexity, *spherical cover number* of a drawing, which is the minimum number of spheres needed to cover all edges of a 3 dimensional graph drawing (sim. circles for 2 dimensions) [78].

Kang and Lin [69] show that planar Cayley graphs (graphs that describe a finite group and operations on that group, and is also planar) admit symmetrical spherical drawings. These drawings reside in $\mathbb{S}^2$, are rotationally symmetric, and have uniform edge lengths. They provide a construction for each drawing and enumerate all planar Cayley graphs.

A generalization of the straight line drawing for the Euclidean plane is the geodesic drawing for the sphere, where all edges must be geodesics of the endpoints. This type of drawing has been studied by [13].

Mohar first showed how one can draw general graphs in the hyperbolic plane [93]. 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 [40, 134]. 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.

Although embedding of arbitrary graphs have theoretical limitations, there are still many graph structures that cannot be well captured in Euclidean space but can be well captured in hyperbolic space. Ouyang et al. [104] construct arbitrary hyperbolic tesselations, and provide an algorithm to color and distort edges in aesthetically pleasing and interesting patterns.

It has been shown that some graphs can be embedded with lower error in hyperbolic

space than in Euclidean space [12]. Zhou and Sharpee [148] show that hyperbolic MDS (H-MDS) can be used to detect the underlying geometry of a dataset, when comparing its embedding error to Euclidean metric MDS. They go further to show that the underlying space of genomes is hyperbolic. Krioukov et al's [75] work indicates that hyperbolic geometry may underlie complex networks and hierarchical networks, such as phylogenetic trees and the internet.

Hyperbolic geometry is of interest in networking and routing, in the form of greedy embeddings. It has been shown that any connected, finite graph admits a greedy embedding in hyperbolic space, which is not generally true in Euclidean geometry [70]. 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 [42, 43].

Cabello et al. [20] study toroidal embeddings: given a disconnected graph $G \cup G'$, does there exist a toroidal embedding of the graph with optimal crossings such that no edge of G intersects an edge of G'. Some results are presented, but the problem appears to remain open. Norine [101] shows that any 4-Pfaffian graph can be drawn on the torus such that that every perfect matching intersects itself an even number of times.

2.5.4 Application

Applications of network visualization on the sphere are primarily of the form of a map metaphor, or for geographic data.

Shelley et al. [123] present a software titled GerbilSphere, which places a user in the center of a sphere with a network drawn on the sphere surface. The target is dynamic, large-scale networks so the tasks are primarily high level movement of groups. The paper

FIGURE 2.13: Using 3D edge routing around the globe to visualize international air interconnections network from [82].

includes a small user study to test the effectiveness, reporting that navigation is more efficient with GerbilSphere than traditional network visualization.

Lambert et al. [82] use a 3D edge bundling technique based on edge routing to support visualization of geographical networks as shown in Fig. 2.13 where they visualize international air traffic network. The width of the bundles is based on the number of edges within it, and the data is overlayed on the globe since the data has geographic coordinates.

Efforts to visualize the difficult to conceptualize processes in non-Euclidean geometry has had some interest. Francis and Sullivan [47] create a visualization algorithm and

rendering to visualize a sphere eversion (turning it inside-out). Further, hyperbolic tessellations have been the subject of study for education and aesthetics [104]. An open-source hyperbolic visualization tool, RogueViz [22], includes different projections and educational tools, although its restriction to tessellations of the hyperbolic plane makes it less than ideal for general graphs.

Non-Euclidean geometry is of particular interest in virtual reality. This has been explored for the sphere by Kwon et al. [79, 80], who present a technique for edge routing on the sphere with VR in mind, running the edges outside of the sphere. A user is placed inside the sphere with a head-mounted display, and is free to look around to gain understanding of the network data. In [72], a user is placed in the center of a spherical drawing dubbed an 'egocentric' view. This also allows areas to be 'retracted' and brought closer to the user.

A class of graph known as a 'torus graph' is common in networking and HPC, in fact some years ago high-dimensional torus networks were used in four of the top ten supercomputers and eight of the top ten on the Graph500 list [85]. These are graphs which describe the topology of the torus in n dimensions. They are created by starting with an n-dimensional rectangular lattice, then connecting the outside vertices in the same dimension; left to right and top to bottom for a two dimensional torus graph. There has been interest in visualizing these types of networks, and we have classified these works as application papers since they are a method of visualizing data from a torus.

TorusVis [27] is a visualization design study attempting to visualize the topology of torus networks from super computing clusters. They use a circular layout for the graph visualization part of their dashboard, with an ordering algorithm based on Hilbert curves. They use edge bundling to decrease clutter within the circle. This circular layout was later

improved and extended into a technique and analytics tool [28]. Also [85] look specifically at the IBM Blue Gene machine's torus network using four connected views depicting the network at different levels of detail. The approach is demonstrated by analyzing network traffic for a simulation running on the IBM Blue architecture. Small multiples with links between them are used by [128] to visualize torus networks.

2.6 Types of Graphs

Our final dimension for analysis is the types of graphs under consideration, as the choice of graphs for visualization in different geometries can affect the visualization methods and goals.

2.6.1 Trees and Planar graphs

Hyperbolic tree browsers had a large effect on the research landscape of tree and hierarchy visualization. Schneiderman et al. [125] perform a thorough investigation of early tree visualization techniques, including hyperbolic trees as well as treemaps and cone trees. They show that at the time of writing, the Lamping et al. [83] paper for the hyperbolic browser was one of the most cited tree visualization papers.

Hyperbolic tree visualizations are typically in 2D or 3D. Most 2D approaches employ node-link diagrams drawn in the Poincaré disk [10, 63, 83]. Meanwhile, most 3D approaches [66, 94–97, 146] make use of the Beltrami-Klein projection.

`Treevis.net` provides several examples of hyperbolic and sphere-based tree visualizations [122]. For example, SphereTree [17] puts a treemap onto the surface of a sphere, encoding vertices as nested rectangles. The sphere appears hollow, allowing

a reader to see the 'backside' of the surface. The hyperbolic wheel [81] scheme places a sunburst chart in the Poincaré disk, so that areas of vertices further from the center decrease exponentially.

Planar and non-planar graphs remain planar and non-planar respectively in spherical and hyperbolic geometries. Spherical planar embeddings of graphs are studied by both [13] who aim to eliminate geodesic crossings and [69] who exhaustively show a small class of graphs (Planar Cayley graphs) have symmetrical planar spherical embeddings. Planarity can be generalized to higher-genus surfaces. A graph which can be drawn without crossings on the torus is called a toroidal graph. Algorithms to construct crossing-free embeddings of toroidal graphs are known [74], but there are no known efforts to use these algorithms for visualization.

2.6.2 Complex networks

Krioukov et al. [75] show that complex networks have an underlying hyperbolic geometry. What this means is that the graph theoretic distance between vertices in a complex network is roughly equivalent to the hyperbolic geodesic distance of a hyperbolic embedding of that graph. One of the ways they show this is by defining a random graph model, a hyperbolic random graph, which uniformly distributes points in the hyperbolic plane and connects points with an edge if they are within some radius. They show this random graph model exhibits complex network properties where many other random graph models (e.g. Euclidean geometric random graphs) do not.

This result lead to a series of papers in hyperbolic graph drawing. The main question is how to generate a hyperbolic graph embedding which realizes the properties above. Bläsius et al. [12] propose a maximum likelihood estimation formulation for hyperbolic

graph embedding, and show computationally that they achieve lower distortion than the force-directed approach of [73]. Meanwhile, Sala et al. [117] develop a hyperbolic classical MDS approach that achieves lower distortion than Euclidean embeddings on example datasets. For metric MDS, Miller et al. [91] apply and stochastic gradient descent optimization to also achieve lower distortion that Euclidean embeddings. Bläsius et al. [11] recently adapted a force-directed approach with two types of forces: popularity force, which operates on the radial component (i.e. its distance from the origin) and similarity force, which operates on the angle of a node. By treating these forces independently, they achieve comparable distortion error to competing methods.

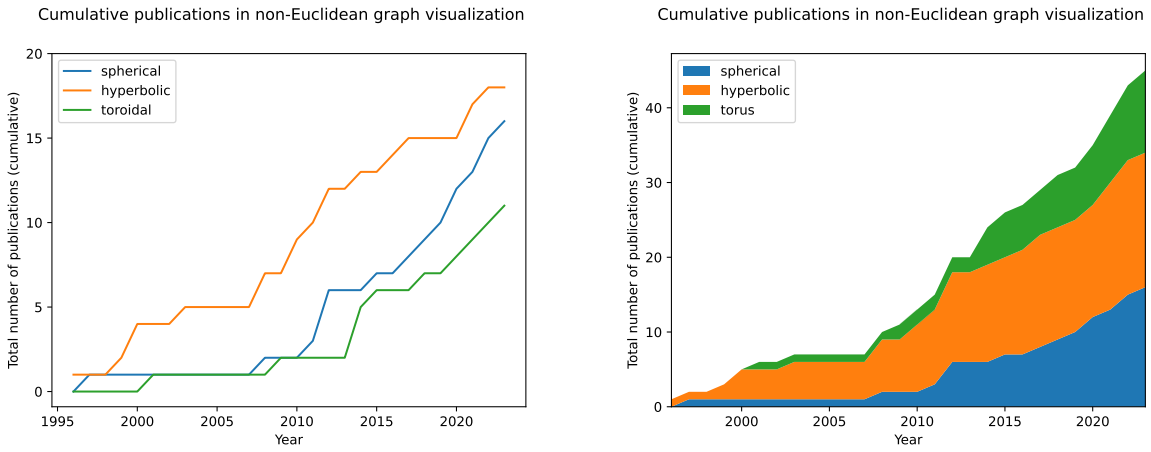
2.7 Discussion

In this survey we attempt to summarize and analyze work on visualizing graphs in non-Euclidean spaces. We have tagged all papers under consideration and categorized them in a json file with paper titles, authors, DOIs, and other attributes, which we provide as supplemental material. Plots and tables referred to in this section are based on this data.

Geometry	Papers
spherical	16
hyperbolic	18
torus	11

TABLE 2.2: Count of papers in survey that fall into each geometry.

Looking at the total number of papers over time, grouped by geometry, we see that the three categories have roughly an equal number of papers currently. Most of the papers in hyperbolic geometry are from before 2010, when spherical papers become more popular. Torus papers have been more frequent in recent years and seem to be catching up.



(a) (b)

FIGURE 2.14: Number of publications in each geometry over time (cumulative). (a) depicts a simple line chart with total number of publications increasing cumulatively each year. (b) depicts the same data as a stacked area chart, with the height of the chart being the total number of publications of all categories and the colors within the distribution of spherical, hyperbolic, and toroidal geometries.

The count of papers by geometry can be found in Table 2.2. As to be expected, the number of spherical and hyperbolic papers is roughly equal, while there fewer torus papers. We investigate how this trend evolved over time in Fig. 2.14 (a). From 1995-2009, hyperbolic publications were more numerous with their applications to tree visualization, and received another round of papers in the early 2010s for their relationship to complex networks. Spherical visualization papers became more prevalent in the 2010s as well, perhaps due to the development of map-based network drawings and the release of open source web tools like d3.geo and THREE.js. We see a similar spike in torus papers, when visualization of torus networks for HPC systems became of interest.

A count of papers instead by contribution is found in Table 2.3. Technique, theory, and application papers are roughly equally common. Notably, there are far fewer evaluation

Contribution	Papers
technique	15
evaluation	4
theory	13
application	19

TABLE 2.3: Number of Papers based on contribution.

papers than one might expect for a field of research originating 30 years ago.

	technique	evaluation	proof	application
spherical	6	2	3	5
hyperbolic	4	1	6	6
torus	1	2	4	4

TABLE 2.4: Papers in each geometry based on contribution.

We can observe the intersection of two of the dimensions of our categorization; geometry and contribution. A table that counts the occurrences of these intersections is found in Table 2.4. A more striking visual is the matrix visualization Fig. 2.15. We can clearly see where there is potential for future research. Toroidal techniques and evaluations for all geometries have much room to explore (the white spaces in the table).

2.8 Conclusions

We have reviewed over 60 papers in the non-Euclidean and Riemannian graph visualization area, as well as summarized and categorized the literature. This subject has been of interest to the visualization community in the last three decades and we hope that visualization researchers might use this survey to get an overview of what is known and what remains to be explored.

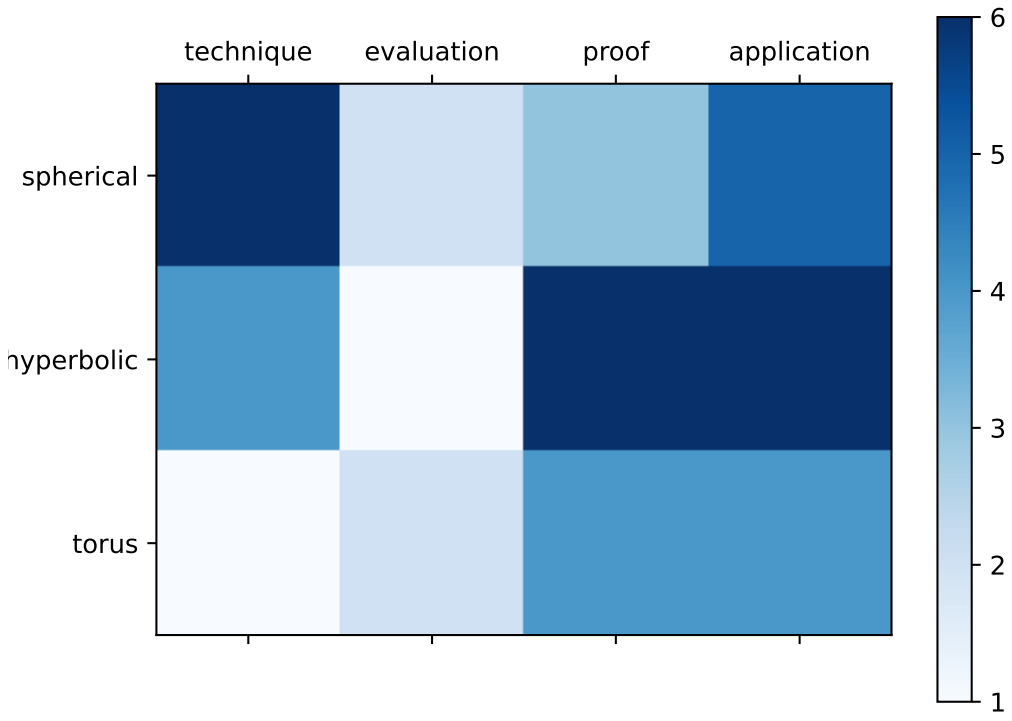


FIGURE 2.15: A matrix based visualization of Table 2.4. Darker squares indicate more papers published in that category, while lighter squares indicate fewer papers.

Chapter 3 LGS: Graph drawing from Local to Global Scale

This chapter is taken from a publication that appeared in the International Symposium on Graph Drawing 2023 [89].

3.1 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 [127], 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. Further, local DR algorithms attempt to preserve the local neighborhoods, while global DR algorithms attempt to retain all pairwise distances.

Two popular techniques that are adapted in graph visualization are (metric) Multi-Dimensional Scaling (MDS) [33, 77] and t-distributed stochastic neighbor embedding (t-SNE) [132]. The goals of these two algorithms are somewhat orthogonal: MDS focuses on

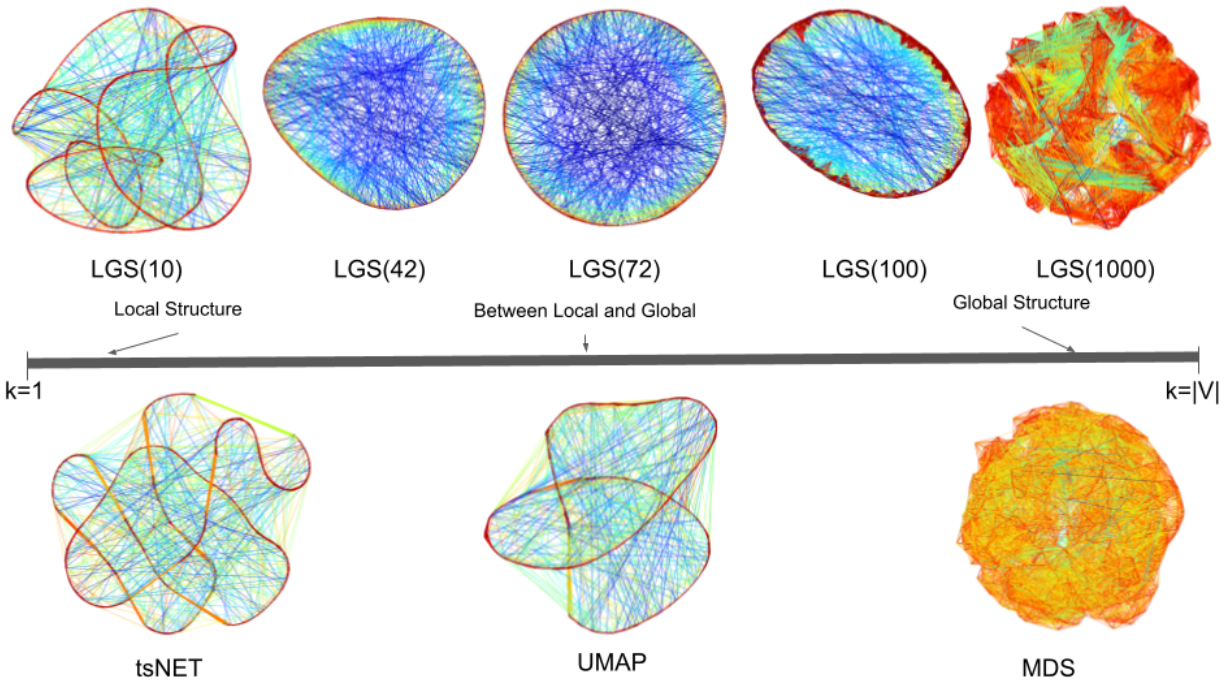
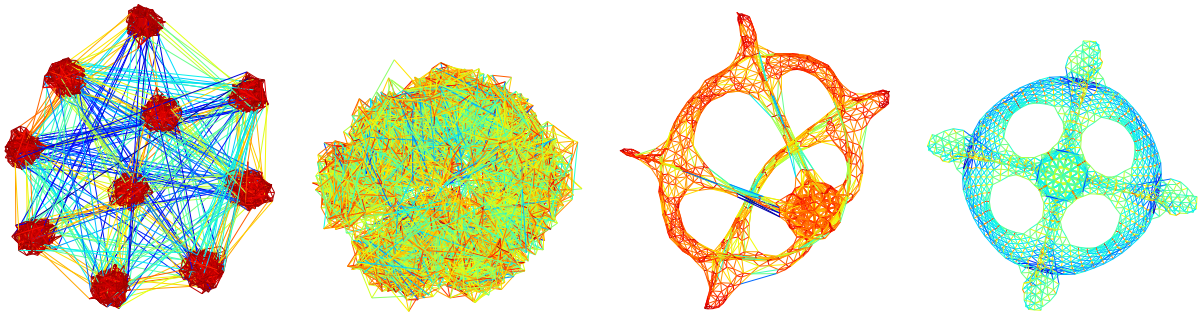


FIGURE 3.1: Example embeddings of the connected_watts_1000 graph, a small-world network generated by the Watts-Strogatz random graph model, where vertices are placed evenly around one cycle, connected by an edge to their nearest neighbors in space, then some edges are rewired as 'chords' in the cycle. 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 [76], UMAP [87], and MDS [147], 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.

preserving all pairwise distances, while t-SNE aims to preserve the likelihood of points being close in the embedding if they were close in the original space. MDS is said to preserve *global* structure, while t-SNE is said to preserve *local* neighborhoods [46]. These ideas are directly applicable to graph visualization, where we can define the distances as the graph theoretic distances, e.g., via all-pairs shortest paths (APSP) computation. In the graph layout literature, MDS is often referred to as stress minimization [54, 147], and t-SNE has been adapted to graph layout in an algorithm known as tsNET [76] and later DRGraph [149]. Choosing the "best" graph embedding algorithm depends on the graph structure and the task. MDS is effective for structured/mesh-like graphs, while t-SNE works better for clustered/dense graphs. This phenomenon also applies to local and global force-directed layouts as well [76].

Automating the selection of the "best" embedding algorithm is challenging due to its dependency on graph structure. We introduce the Local-to-Global Structures (LGS) algorithm which provides a parameter-tuneable framework that can produces embeddings that span the spectrum from local optimization to global optimization.

Smaller values of the LGS parameter prioritize local structure, while larger values emphasize global structure. LGS enables exploration of the trade-off, revealing meaningful middle ground solutions. 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://github.com/Mickey253/L2G`. We provide a video and additional layouts and analysis in supplemental material.



(a) (b) (c) (d)

FIGURE 3.2: block_2000 drawn by tsNET (a) and MDS (b); dwt_1005 drawn by tsNET (c) and MDS (d). While local methods such as tsNET perform well on graphs with distinct local structure (a), they can distort the global shape of the graph (c). Similarly, global methods such as MDS can capture the shape well (d), but it may miss the more interesting local structures when they exist (b).

3.2 The Local-to-Global Structures (LGS) Algorithm

Local methods (e.g., t-SNE) preserve local neighborhoods, while global methods (e.g., MDS) capture all pair-wise distances. We propose the Local-to-Global Structures (LGS) algorithm that achieves the following 3 goals:

G1 A single parameter controlling local-global embedding balance

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

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

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 an embedding match well with the nearest neighbors in the actual graph, then the embedding accurately preserves the local structures. By "global structure" we refer to the preservation of all pairwise

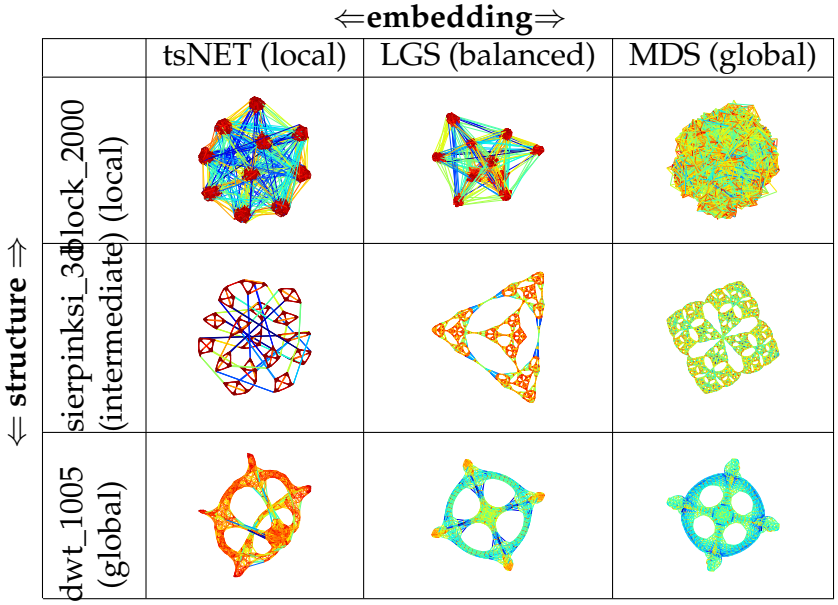


FIGURE 3.3: Local embedding methods perform well on graphs with distinct local structure (block_2000), but they can distort the global shape of the graph (dwt_1005). Global methods capture the overall shape (e.g., dwt_1005), but may miss important local structures (block_2000). LGS(100) performs well for graphs with both local and global structure, such as sierpinski_3d, allowing us to see its fractal nature.

graph distances (including long ones) in the embedding. Finally, "intermediate structure" refers to capturing both local neighbors and global structure. Fig. 3.3 shows graphs exemplifying local, intermediate, and global structures and Sec. 3.2.2 defines formal embedding measures: neighborhood error, cluster distance, and stress. In Sec. 3.2.1 we explain the selection process for the balance parameter k and the objective function to ensure that the solution aligns with the stated goals. For **G1**, we modify MDS to preserve distances in a neighborhood defined by a parameter k). Thus, preserving distances for large neighborhoods satisfies **G3**. This leaves a question for **G2**: Does applying distance preservation to a subset of pairs result in locally faithful embeddings?

3.2.1 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 [26]). 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. 3.3. Although the within-cluster density is high, there are many edges between different clusters. 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 out-of-cluster edges.

Instead of considering distances directly, we find the top k most connected vertices for each vertex based on the hypothesis that more possible walks between vertices indicate greater similarity; see Fig. 3.4. Despite v_A and v_B not sharing an edge, they have the same set of neighbors. When v_A and v_B are both neighbors to a set of vertices, we can confidently state their similarity, confirmed by their shared proximity to vc, v_d, etc. [44]. 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. To find the top k "most connected" vertices for each vertex, follow this procedure: Given an adjacency matrix of an undirected graph, A_G, raise it to the c-th power, take the sum of all powers $\mathbf{A}^* = \sum_{\mathbf{1} \leq \mathbf{i} \leq \mathbf{c}} \mathbf{A_G^i}$, to obtain a matrix whose (i, j)-th element shows 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, we find the k largest values in row i (by sorting). We define these top k vertices to be the "most connected" neighborhood $N_k(v_a)$ of vertex v_a; see Fig. 3.4. We further 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 range of values for s, and set

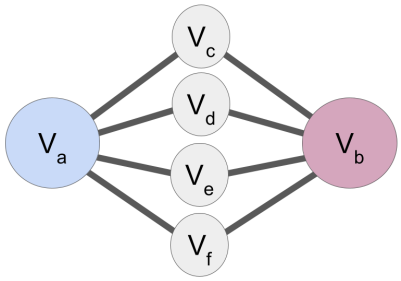


FIGURE 3.4: An example of how we may skip over immediate neighbors when selecting neighborhoods to preserve. In this case, $c = 2$. There is only one unique walk of length ≤ 2 from v_a to v_c, v_d, v_e, v_f, but there are 4 such walks from v_a to v_b. In this case, v_b would be the first vertex added to v_a's most connected neighborhood.

$s = 0.1$; see the supplemental material. We propose a procedure to reduce the number of matrix multiplications which we used in our experiments.

Objective Function We remark, 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 [26, 52], to force pairs of vertices away from each other. For a given pairwise distance matrix $\left[d_{ij}\right]_{i,j=1}^{n}$ we define the following generalized stress function as an objective function:

$$\sigma(X) = \sum_{(i,j)\in N_k} (\|X_i - X_j\| - d_{ij})^2 - \alpha \sum_{(i,j)\notin N_k} \log \|X_i - X_j\|, \tag{3.1}$$

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 [26] and MaxEnt [52], the proposed objective function in Eq. 3.1 differs in (1) how the set N_k is selected (LMDS uses a kNN 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 [14, 147]. The parameter space of the algorithm is discussed in the supplemental material [89].

3.2.2 Evaluation Metrics

We discuss the evaluation metrics for embedding algorithms: local neighborhood error (NE) score, intermediate structure (CD), and global distances (Stress).

NE Metric:

Neighborhood hits (NH) measures how well an embedding preserves local structures [26, 46]. 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. NH is defined as:

$$NH(Y, X, r) = \frac{1}{n} \sum_{i=1}^{n} \frac{|N_Y(p_i, r) \cap N_X(p_i, r)|}{|N_Y(p_i, r) \cup N_X(p_i, r)|} \tag{3.2}$$

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

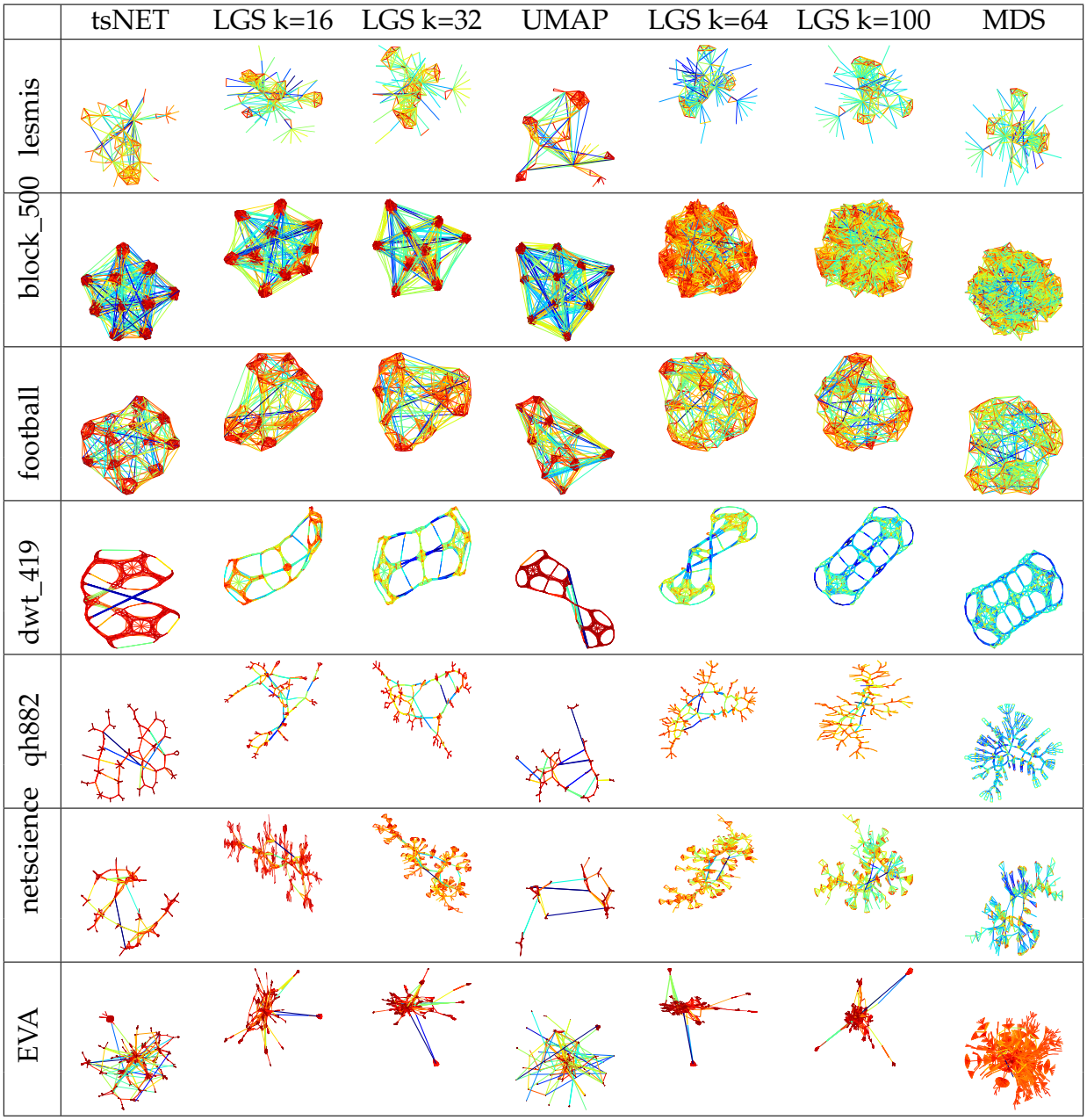


FIGURE 3.5: Example embeddings. The first and last columns show the two extremes tsNET (local) and MDS (global); the middle column shows UMAP. The remaining columns show a gradual increase of LGS's k parameter, moving from local to global distance preservation (left to right). Note LGS outputs are vertically higher in each row.

preservation (NP) [52, 76, 149], 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 introduce a new metric to measure how well intermediate structures are captured in an embedding. Since the distances between clusters in t-SNE cannot be interpreted as actual distances [142], 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 cluster labels are 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 use k-means clustering in the high-dimensional data case, and modularity clustering in the graph case. The distances between clusters in the high-dimensional case is given by the Euclidean distance between the cluster centers. For graphs, we measure the distance between clusters by first taking the normalized count of edges between them, then subtracting the normalized count to convert similarity into dissimilarity. This produces a cluster-distance matrix, δ. Let $C_1, \ldots, C_n$ be the set of vertices belonging to

cluster $1, \ldots, n$, then

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

where $\mathbb{1}$ is the indicator function (1 if (u, v) is an edge and 0 otherwise). Once δ is computed, we compute the geometric center of each embedded cluster and compute the cluster-level stress between the graph-level-cluster and realized-cluster distances. This measure is small when similar clusters are placed closer and dissimilar clusters are placed far apart. The cluster distance (CD) is:

$$CD(\delta, \chi) = \sum_{i,j} \left(\frac{\delta_{i,j} - ||\chi_i - \chi_j||}{\delta_{i,j}} \right)^2 \tag{3.3}$$

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. Although there are several existing metrics to measure cluster accuracy, such as silhouette distance and between/within-cluster sum of squares, they are not well suited to measure the quality of intermediate embeddings. Ideally, we would need a measure that checks how well the clusters are preserved and also verifies that the relative placement of the clusters is meaningful. The CD metric verifies meaningful cluster placements by measuring all pairwise distances between cluster centers. We remark that the CD metric works best when the clusters have convex shapes (or shapes similar to spheres). For arbitrary non-convex shapes, such as half-moons or donuts, the CD metric might not provide meaningful insights.

Stress Metric:

Stress has been used in many graph embedding evaluations.

$$\text{stress}(d, X) = \sum_{i,j} \left(\frac{d_{i,j} - ||X_i - X_j||}{d_{i,j}} \right)^2 \tag{3.4}$$

where d is the given distance matrix and X is the embedding. Embeddings are scaled to ensure fair comparisons in computing stress [52,76,149].

3.3 LGS Embedding of Graphs

We start with a visual analysis and discussion of layouts produced by LGS. 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 LGS 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. 3.1. Higher k values make LGS similar to MDS, with more uniform edge lengths. This reveals global structures (e.g., mesh, grid, lattice) but may overlook clusters.

Grid_cluster is a synthetic example with 900 vertices (9 clusters of size 100 each) and 10108 edges, created by stochastic block model (SBM) to illustrate the notion of cluster distance preservation. 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. 3.6. Note the visual similarities between LGS(32) and tsNET; both seperate each cluster into dense sub-regions and place them seemingly randomly in the plane. Also note the similarities between LGS(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. LGS(100) accurately places each cluster in the appropriate position, making it the only one that clearly shows the 3x3 underlying lattice. Although less dense and separable the clusters are more faithful in terms of 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 (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 seen in real-world examples, such as social networks. The embeddings of connected_watts_1000, obtained by LGS, tsNET, UMAP, and MDS are in Fig. 3.1. We observe that 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 LGS, 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 LGS algorithm at work, as small local structures build upon each other to create a global shape; see Fig. 3.9. 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. LGS can be used to balance these extremes.

We demonstrate more examples in Table 3.5, with additional embeddings available in the supplemental material. Note that for lower values of k the embedding obtained by LGS 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. 3.7 and Table 3.1. For intermediate values of k, LGS often outperforms tsNET, UMap and MDS with respect to the intermediate structure preservation, measured by cluster distance (CD).

3.4 Evaluation

We test LGS on a selection of real-world and synthetic graphs from [32,76,149]; A full list can be found in the appendix.

We compare LGS against state-of-the-art techniques for local and global embeddings: tsNET from the repository linked in [76], UMAP from the umap python library written by the authors of [87], and MDS via the python bindings from [147] with default parameters.

Our implementation of LGS is 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.

3.4.1 NE, CD and Stress Values and Trends

To evaluate how well LGS preserves local neighborhoods, we compare the average NE scores over several runs and present our results in Table 3.1. We can see a general trend: although, tsNET performs better with respect to NE values, LGS 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, LGS 'transitions' from tsNET to UMAP then finally to MDS as one goes from left to right, increasing k.

Next, we report the average CD scores for the graphs in our benchmark in Table 3.2. 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.

We compute and report the averaged stress scores in Table 3.3. MDS is consistently good at minimizing the stress, but we see a salient trade-off between the stress scores of LGS's tsNET-like embeddings with low k values and LGS'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.

Effect of k on Evaluation Metrics:

To visually explain LGS behavior, we plot examples NE, CD, and stress with respect to k. In Fig. 3.7, 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. These plots show what we expect to see: as k increases NE increases and stress decreases. In Fig. 3.8(a-b), we plot the CD values of our layout with tsNET, UMAP, MDS for comparison. In many layouts LGS indeed has the lowest CD score. Values of k were chosen to be representative of the local-global tradeoff.

3.5 Discussion and Limitations

We described LGS: an adaptable algorithmic framework for embeddings that can prioritize local neighborhoods, global structure, or a balance between the two. LGS provides flexible structure preservation choices with comparable embedding quality to previous single-purpose methods (local or global), while also outperforming state-of-the-art methods in preserving intermediate structures.

There are several limitations: Our results are based on a small number of graphs. Additional systematic experimentation would further support the usefulness of LGS Some experiments for high-dimensional datasets are available in supplementary material [89]. LGS modifies MDS's objective function to accommodate varying neighborhood sizes, similarly, one could adapt the KL divergence cost function of t-SNE.

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 [142].

LGS algorithm has several hyperparameters, including c, α, and k. We provide default values for c and α based on experiments, and leave k as a true hyperparameter. 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. An interactive LGS version is not yet available. While LGS 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 APSP. While LGS's runtime is comparable with those of tsNET and MDS (see Fig. 3.8(c)) both can be sped up through the use of approximations [49, 102, 149], Speeding up LGS is a potential future work.

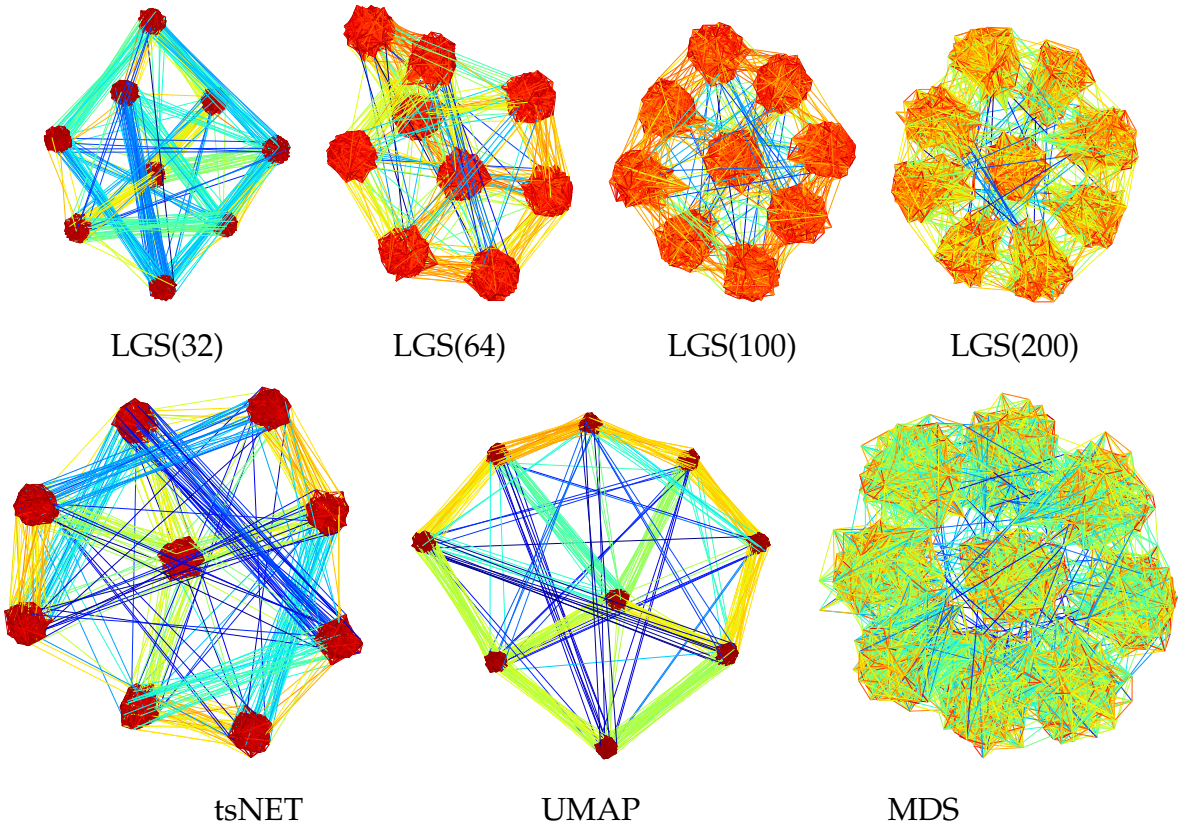


FIGURE 3.6: The grid_cluster graph is generated so that each cluster has many out-of-cluster edges to its neighbors in a 3×3 lattice, providing a recognizable intermediate structure. tsNET and UMAP do not place clusters on a grid, MDS mixes the clusters; LGS(100) captures the 3×3 grid and shows distinct clusters.

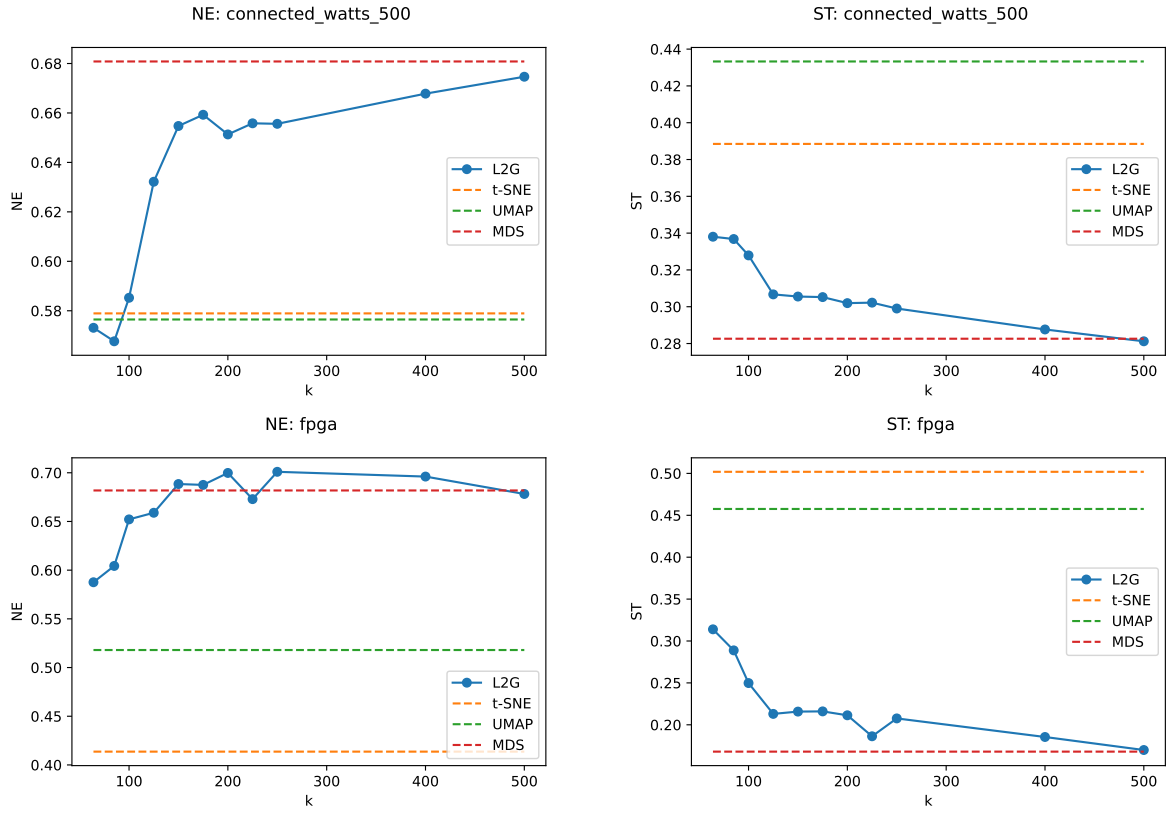
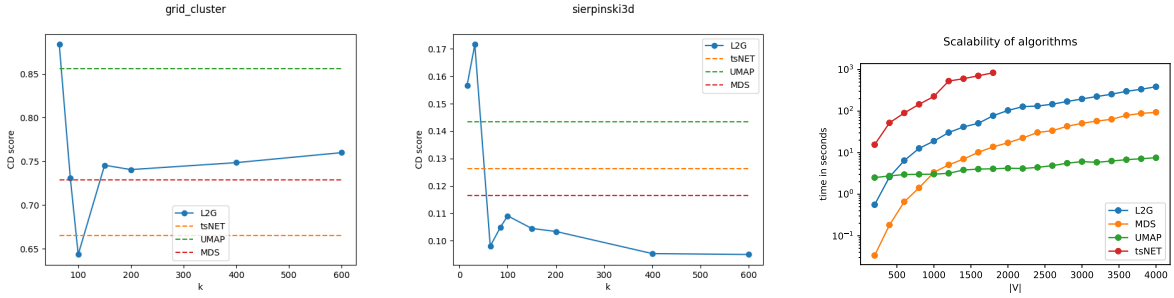


FIGURE 3.7: Behavior of NE and stress: as k increases NE gets worse and stress gets better (LGS transitions from preserving local to global structure); tsNET, UMAP, and MDS values are shown as dotted lines for comparison. Note that in general, we expect to see an upward trend in NE, a downward trend for stress, and a parabola shape for CD.



(a) (b) (c)

FIGURE 3.8: (a-b) CD metric on the grid_cluster and sierpinkski3d graphs. Note that in these examples there are values of k which outperform competing algorithms. (d) Running time of each tested algorithm.

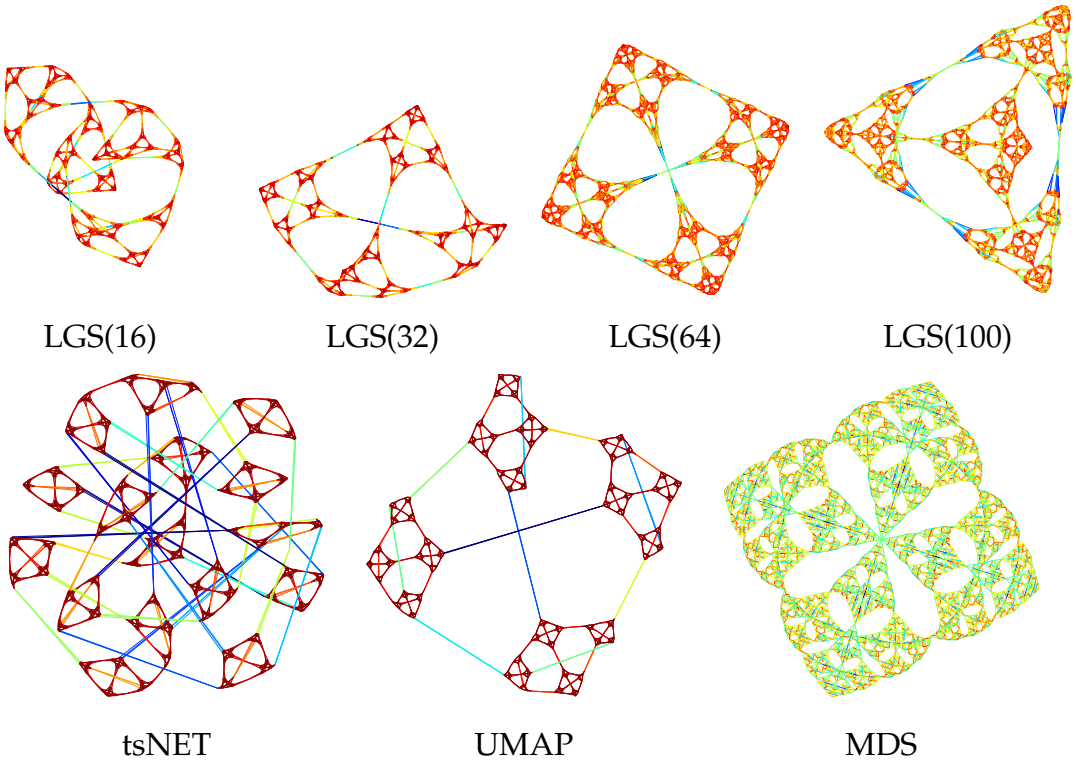


FIGURE 3.9: Sierpinksi3d graph is a fractal with regular local and global structure. LGS manages to capture the recursive nature of the underlying structure. tsNET, UMAP miss the global placement of pyramids and MDS stretches them.

	k=32	k=64	k=85	k=100	k=150	k=200		tsnet	umap	mds
lesmis	0.3761	0.3169	0.3126	0.3174	0.3166	0.3125		**0.3058**	0.3952	0.3141
can_96	0.4393	0.3980	0.4605	0.4650	0.4661	0.4658		0.3523	**0.3417**	0.4653
football	0.4390	0.4469	0.4452	0.4377	0.4371	**0.4369**		0.4636	0.4693	0.4380
rajat11	0.4347	0.4374	0.4377	0.3868	0.3641	0.3615		0.3942	**0.3331**	0.3608
mesh3e1	0.1841	0.1415	0.0933	0.0971	0.1444	**0.0**		0.1210	0.1241	0.0003
connected_watts_300	0.1995	0.3990	0.5311	0.5386	0.5302	0.5365		0.1805	**0.1778**	0.5453
block_model_300	0.6474	0.6767	0.6748	0.6794	0.6874	0.6741		**0.5538**	0.5675	0.6785
powerlaw300	0.5070	0.5075	0.5126	0.5044	0.5233	0.5062		**0.3563**	0.3849	0.5075
netscience	0.4556	0.4597	0.4751	0.4860	0.5017	0.5118		0.4319	**0.3207**	0.5123
dwt_419	0.3016	0.3151	**0.2404**	0.4085	0.2666	0.3203		0.2796	0.2892	0.2421
powerlaw500	0.5942	0.5630	0.5433	0.5712	0.5593	0.5432		**0.4237**	0.4683	0.5604
block_model_500	0.5396	0.6551	0.6832	0.6945	0.7024	0.7115		**0.4377**	0.4430	0.7151
connected_watts_500	0.5788	0.5731	**0.5676**	0.5852	0.6547	0.6513		0.5789	0.5764	0.6808
grid_cluster	0.3767	0.3831	0.3767	**0.3239**	0.3456	0.3863		0.8487	0.3491	0.4742
price_1000	0.7103	0.7108	0.7228	0.7313	0.7400	0.7468		0.6015	**0.5297**	0.7943
connected_watts_1000	0.6217	0.6712	0.7870	0.8199	0.8563	0.8472		0.6283	**0.6090**	0.8390
powerlaw1000	0.6555	0.6362	0.6409	0.6382	0.6241	0.6532		**0.3956**	0.4206	0.6384
block_model_1000	0.5753	0.5552	0.5593	0.6179	0.7549	0.7777		**0.4703**	0.4880	0.7822
dwt_1005	0.4999	0.4517	0.4417	0.4586	0.4499	0.4826		**0.3692**	0.4581	0.4372
btree9	0.7337	0.8082	0.8321	0.8625	0.8812	0.8811		0.6960	**0.5876**	0.9241
CSphd	0.5816	0.5797	0.5711	0.5783	0.6015	0.6128		0.6510	**0.4378**	0.6051
fpga	0.5199	0.5875	0.6043	0.6521	0.6884	0.6998		**0.4136**	0.5179	0.6818
sierpinski3d	0.6225	0.5729	0.5313	0.5439	0.5192	**0.4252**		0.4504	0.4554	0.4600
EVA	0.4544	0.4690	0.4822	0.4797	0.5070	0.5123		0.8763	**0.2295**	0.5629

TABLE 3.1: NE scores on LGS for varying values of k (left) and on competing algorithms (right). The colormap is normalized by row with dark orange representing the lowest score (best) and dark purple representing the highest (worst). Bold text indicates the lowest score in that row.

	k=32	k=64	k=85	k=100	k=150	k=200		tsnet	umap	mds
lesmis	0.7308	**0.6745**	0.7116	0.7210	0.7135	0.7260		0.6949	0.6904	0.7195
can_96	0.7088	0.6701	0.6669	0.6666	0.6666	0.6666		**0.5174**	0.6747	0.6666
football	0.7747	0.7580	0.7911	0.7933	0.7983	0.8002		0.7804	**0.7090**	0.7979
rajat11	0.4092	0.4045	**0.3672**	0.3862	0.4083	0.4075		0.4607	0.5093	0.4080
mesh3e1	0.4087	0.3951	0.3824	0.3879	0.3865	0.3674		0.3948	0.3739	**0.3673**
connected_watts_300	0.3044	0.2445	0.2098	0.1989	0.1966	0.1968		0.2853	0.3028	**0.1929**
block_model_300	0.7454	0.7726	0.7994	0.7646	0.7727	0.7980		0.7767	**0.7307**	0.7901
powerlaw300	N/A	N/A	N/A	N/A	N/A	N/A		N/A	N/A	N/A
netscience	0.2716	0.2720	0.2717	0.2698	0.2649	0.2689		0.3222	0.3775	**0.2439**
dwt_419	0.1938	0.1899	0.1898	0.1962	0.1925	0.1943		**0.1657**	0.2173	0.1880
powerlaw500	0.4923	0.4332	0.4174	0.4336	0.4066	0.4249		**0.3607**	0.4261	0.4271
block_model_500	**0.6414**	0.7065	0.6626	0.6837	0.6819	0.6855		0.6551	0.7569	0.6975
connected_watts_500	0.2480	0.2268	0.2321	0.2378	0.2235	0.2191		0.2417	0.2294	**0.2120**
grid_cluster	0.9131	0.7905	0.8347	0.8040	0.7488	0.7415		0.7786	0.7454	**0.7335**
price_1000	N/A	N/A	N/A	N/A	N/A	N/A		N/A	N/A	N/A
connected_watts_1000	0.2734	0.2304	0.2946	0.2802	0.3648	0.2894		0.2333	0.2323	**0.2184**
powerlaw1000	0.3917	0.3671	0.3754	0.3640	**0.3590**	0.3775		0.4362	0.4210	0.3683
block_model_1000	0.6308	**0.5358**	0.5447	0.5435	0.5853	0.5419		0.5632	0.5977	0.5991
dwt_1005	0.1597	0.1414	0.1413	0.1466	0.1414	0.1415		0.1438	0.1456	**0.1386**
btree9	N/A	N/A	N/A	N/A	N/A	N/A		N/A	N/A	N/A
CSphd	N/A	N/A	N/A	N/A	N/A	N/A		N/A	N/A	N/A
fpga	0.1692	0.1511	0.1493	0.1601	0.1497	0.1506		0.1770	0.1966	**0.1305**
sierpinski3d	0.1252	0.1156	0.1050	0.1038	0.1021	0.1016		0.1244	0.1321	**0.0974**
EVA	N/A	N/A	N/A	N/A	N/A	N/A		N/A	N/A	N/A

TABLE 3.2: CD scores following the same scheme as Table 3.1. NA indicates no clusters present in the data.

	k=32	k=64	k=85	k=100	k=150	k=200		tsnet	umap	mds
lesmis	0.1988	0.1702	0.1673	0.1670	0.1665	**0.1656**		0.2290	0.4058	0.1661
can_96	0.1782	0.1526	0.1400	**0.1390**	0.1391	0.1391		0.2315	0.2237	0.1393
football	0.2719	0.2622	0.2578	0.2549	0.2544	**0.2544**		0.3521	0.3888	0.2548
rajat11	0.1636	0.1685	0.1774	0.1474	0.1250	**0.1246**		0.3167	0.4076	0.1251
mesh3e1	0.1125	0.0655	0.0328	0.0482	0.0613	0.0050		0.0858	0.0251	**0.0049**
connected_watts_300	0.4984	0.2359	0.2353	0.2267	0.2008	0.2081		0.4218	0.4702	**0.1896**
block_model_300	0.3592	0.3132	0.3087	0.3051	0.3052	0.2929		0.3889	0.4437	**0.2863**
powerlaw300	0.1975	0.1848	0.1741	0.1668	0.1644	0.1532		0.2619	0.3731	**0.1505**
netscience	0.1914	0.1539	0.1604	0.1627	0.1493	0.1472		0.3296	0.5198	**0.1131**
dwt_419	0.0688	0.0638	0.0360	0.1206	0.0521	0.0759		0.1417	0.1646	**0.0312**
powerlaw500	0.5465	0.2172	0.2023	0.2133	0.2022	0.1893		0.2643	0.3413	**0.1772**
block_model_500	0.3360	0.3224	0.3160	0.3108	0.3047	0.3016		0.4047	0.4510	**0.2855**
connected_watts_500	0.4816	0.3380	0.3367	0.3279	0.3055	0.3019		0.3884	0.4332	**0.2826**
grid_cluster	0.5494	0.5335	0.5814	0.3049	0.2553	0.2478		0.4195	0.3760	**0.2371**
price_1000	0.3543	0.2428	0.2320	0.2347	0.2127	0.2011		0.3654	0.5721	**0.1848**
connected_watts_1000	0.5578	0.3867	1.0963	0.9221	1.6998	0.8549		0.4169	0.4460	**0.3166**
powerlaw1000	0.3012	0.2418	0.2344	0.2410	0.2129	0.2196		0.2907	0.3615	**0.1815**
block_model_1000	0.4389	0.3456	0.3444	0.3409	0.3292	0.3212		0.4078	0.4379	**0.2921**
dwt_1005	0.1858	0.1206	0.0939	0.1165	0.0891	0.1355		0.3556	0.0669	**0.0424**
btree9	0.4973	0.3090	0.3510	0.2740	0.2818	0.2563		0.3539	0.3603	**0.2307**
CSphd	0.2642	0.2483	0.2218	0.1925	0.2051	0.1883		0.2943	0.4243	**0.1446**
fpga	0.4414	0.3139	0.2888	0.2498	0.2157	0.2113		0.5018	0.4575	**0.1679**
sierpinski3d	0.5063	0.3015	0.2097	0.2304	0.1769	0.1298		0.5194	0.2525	**0.1252**
EVA	0.6843	1.0466	0.7862	0.6387	0.4662	0.6394		0.3535	0.4913	**0.1907**

TABLE 3.3: Stress scores following the same scheme as Table 3.1

Chapter 4 Spherical Multi-dimensional Scaling

This chapter is taken from a publication that appeared in the International Symposium on Graph Drawing 2022 [90].

4.1 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

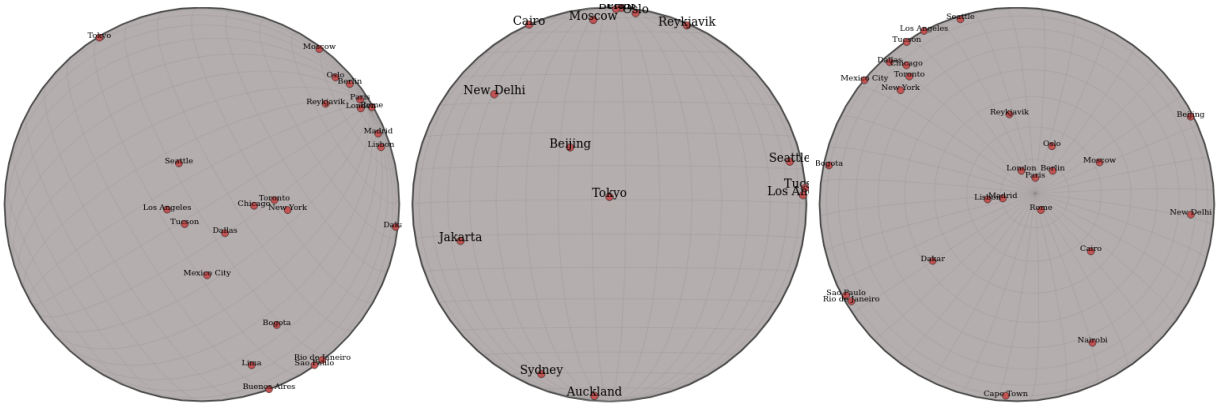


FIGURE 4.1: Applying spherical MDS to embed 30 cities from around the Earth (given pairwise distances between the cities). The spherical MDS recovers the underlying geometry.

include the Multi-Dimensional Scaling (MDS) [124], Principal Component Analysis (PCA) [48], t-distributed Stochastic Neighbor Embedding (t-SNE) [132], and Uniform Manifold Approximation Projection (UMAP) [87]. 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 [38, 108] and hyperbolic geometry [75, 91, 117]. 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 [38] and applying stochastic gradient descent [110, 147]. 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.

4.2 Multidimensional Scaling in Spherical Space

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

$$\sigma_S(X) = \sum_{i<j} w_{ij}(\delta(X_i, X_j) - d_{ij})^2 \tag{4.1}$$

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.

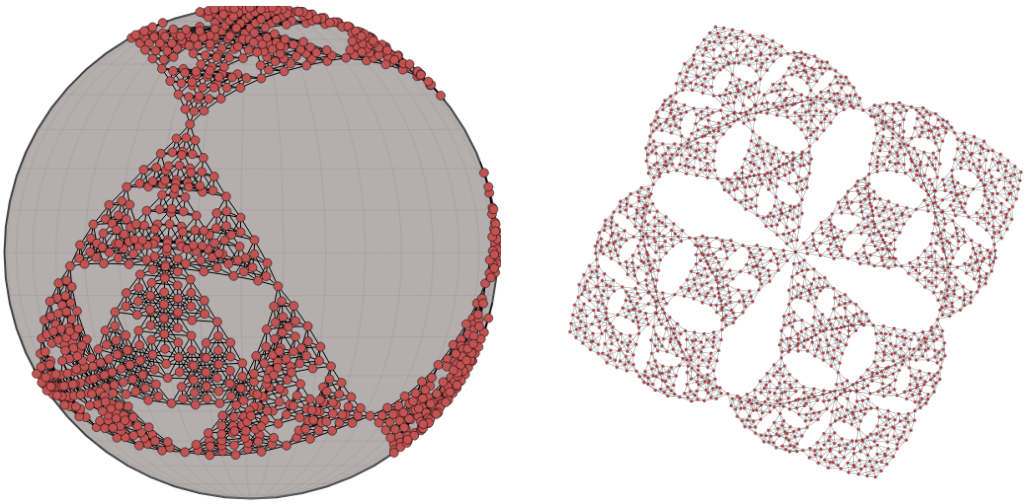


FIGURE 4.2: The Sierpinski3d [76] graph on the sphere (left) and in the plane (right). While the Euclidean drawing on the right is aesthetically pleasing, it looks deceptively like a 2D structure and implies a center. The sphere more accurately captures the structure.

We minimize equation (4.1) 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 [113].

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 (4.1) 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:

$$\frac{\partial f_{i,j}}{\partial X_k} = \begin{cases} 2w_{i,j}(\delta(X_i, X_j) - d_{ij}) \left(\frac{\partial \delta(X_i,X_j)}{\partial \phi_i}, \frac{\partial \delta(X_i,X_j)}{\partial \lambda_i}\right) & k = i \\ 2w_{i,j}(\delta(X_i, X_j) - d_{ij}) \left(\frac{\partial \delta(X_i,X_j)}{\partial \phi_j}, \frac{\partial \delta(X_i,X_j)}{\partial \lambda_j}\right) & k = j \\ 0 & \text{otherwise} \end{cases} \tag{4.2}$$

We can apply SGD to equation (4.1) 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 [14, 91, 147].

Algorithm 1 Stochastic gradient descent algorithm for spherical MDS

```
procedure STOCHASTIC GRADIENT DESCENT(d)
    X ←random initialization
    while Δ(stress(X)) > ϵ or max iterations is reached do
        for (i,j) in random order do
            X_i ← X_i − η df_{i,j}/dX_i, according to (4.2)
            X_j ← X_j − η df_{j,i}/dX_j, according to (4.2)
        end for
    end while
    return X
end procedure
```

4.3 Evaluation

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

4.3.1 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 $\sum g(t)^2 < \infty$ [15].

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. [147], 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. [147] 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.

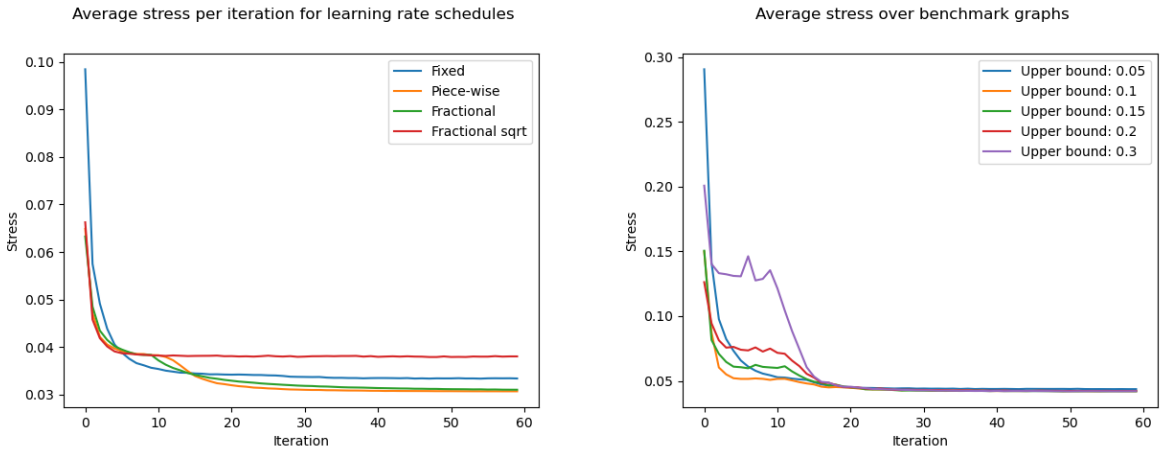


FIGURE 4.3: (Left) Effect of the learning rate schedule on the optimization. The piece-wise schedule adapted from [147] arrives at a minimum faster on average. (Right) Effect of the upper bound on the learning rate on the optimization. An upper bound of 0.1 behaves predictably. Values for both are averaged over all graphs in our benchmark.

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. 4.3.

Randomization is a to select pairs i, j in the stochastic optimization function. While SGD was originally done using sampling with replacement, random reshuffling has been shown to converge in fewer total updates [61]. 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.

4.3.2 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* [32] 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 4.1. 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 convergence and stress as the size of the data becomes large as we expect; see Fig. 4.4.

4.4 Geometry Comparison

Here we discuss some possible drawbacks of graph embedding in spherical space and compare graph embeddings between Euclidean, spherical and hyperbolic spaces.

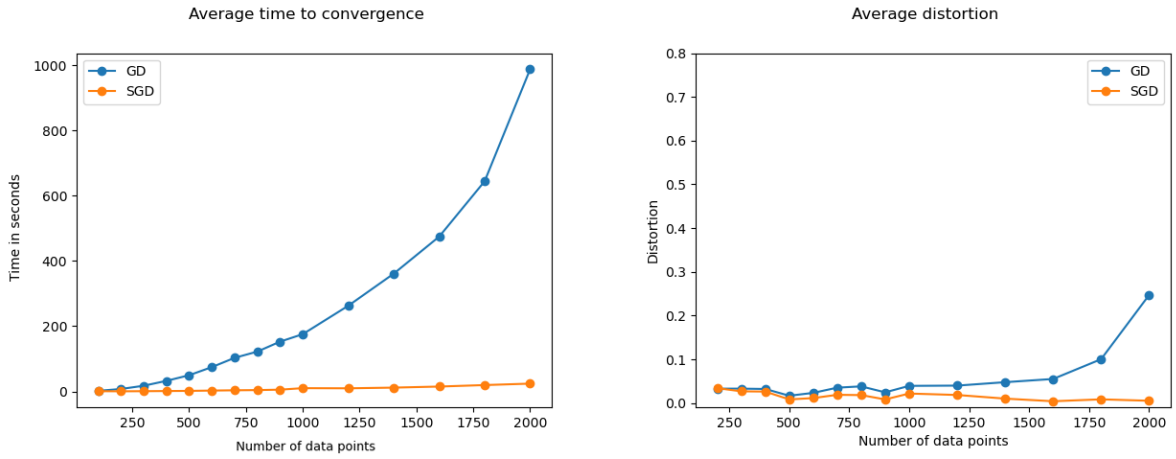


FIGURE 4.4: How the SGD optimization scheme fairs compares to the exact GD in terms of time (left) and error (right). The larger the size of the graph, the more benefit is seen from the use of SGD.

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 [52, 76]) which works fine in Euclidean space, but becomes a problem in spherical and hyperbolic spaces. In order to more fairly compare embedding error across geometries, we use the *distortion* [91, 117] metric, defined in equation 2.2.

4.5 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

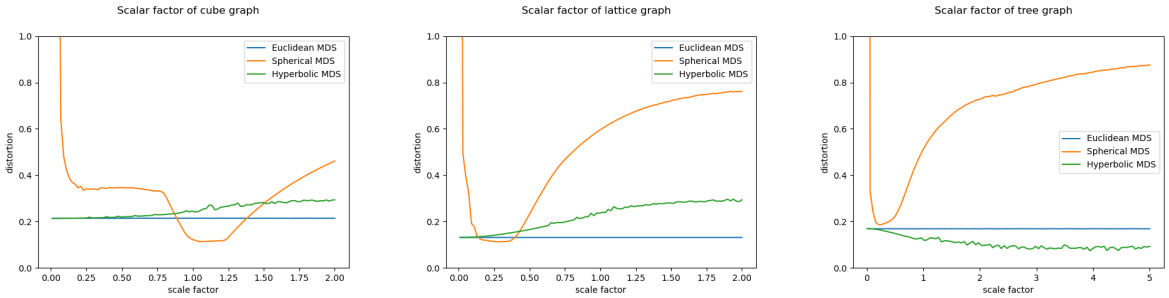


FIGURE 4.5: Behavior of distortion on selected graphs with respect to dilation factor in each geometry.

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. 4.6. 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. (4.1) to also optimize the radius:

$$\arg \min_{R, X_1, \ldots X_n \in S^2_R} \sum_{i,j=1}^{N} \left(\delta_R(X_i, X_j) - d_{ij}\right)^2. \tag{4.3}$$

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.

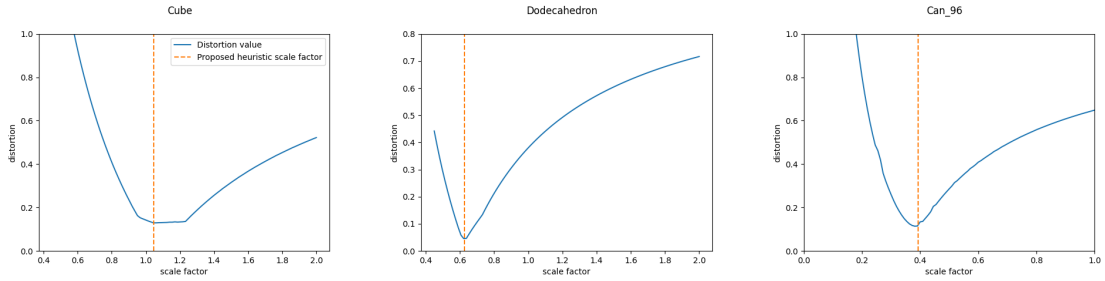


FIGURE 4.6: Effect of dilation on distortion. Our proposed heuristic (orange line) is often very close to the minimum (of the blue curve).

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.

4.5.1 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 [147] for the Euclidean MDS implementation and [91] 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 [134], regular 3D polytopes can be thought of as tesselations

	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

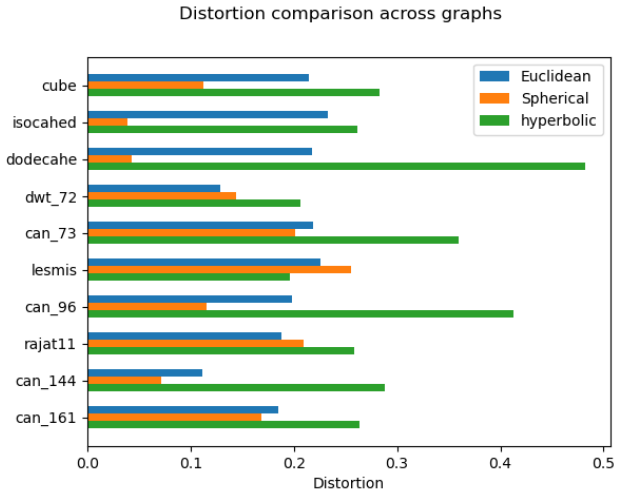


FIGURE 4.7: The left subfigure shows a subset of results from the direct comparison for distortion in Euclidean, spherical and hyperbolic space. The right subfigure plots the first 10 rows. We note that 3D polytopes and meshes (the can graphs) are particularly well suited to the sphere, the LesMis graph is a complex network which is best embedded into hyperbolic space, and Euclidean space is better for the remaining ones.

of the sphere, and trees have been described as “discrete hyperbolic spaces" [75]. The results are summarized in Fig. 4.7 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. 4.8 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.

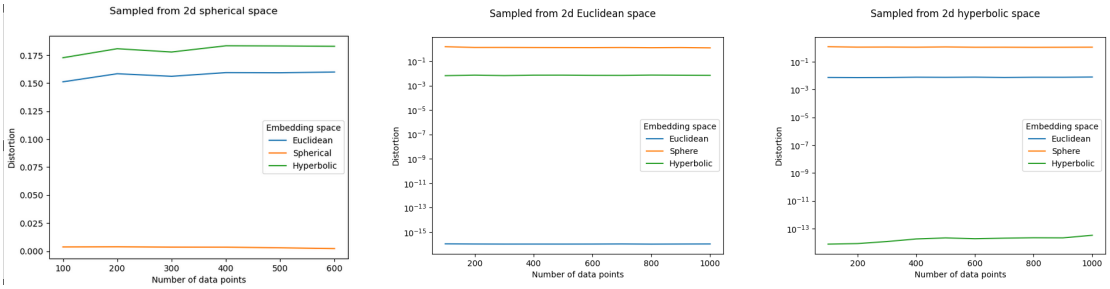


FIGURE 4.8: Results from sampling data uniformly at random from each consistent geometry: as expected SMDS, MDS and HMDS perform dramatically better on data that comes from the geometry it embeds in.

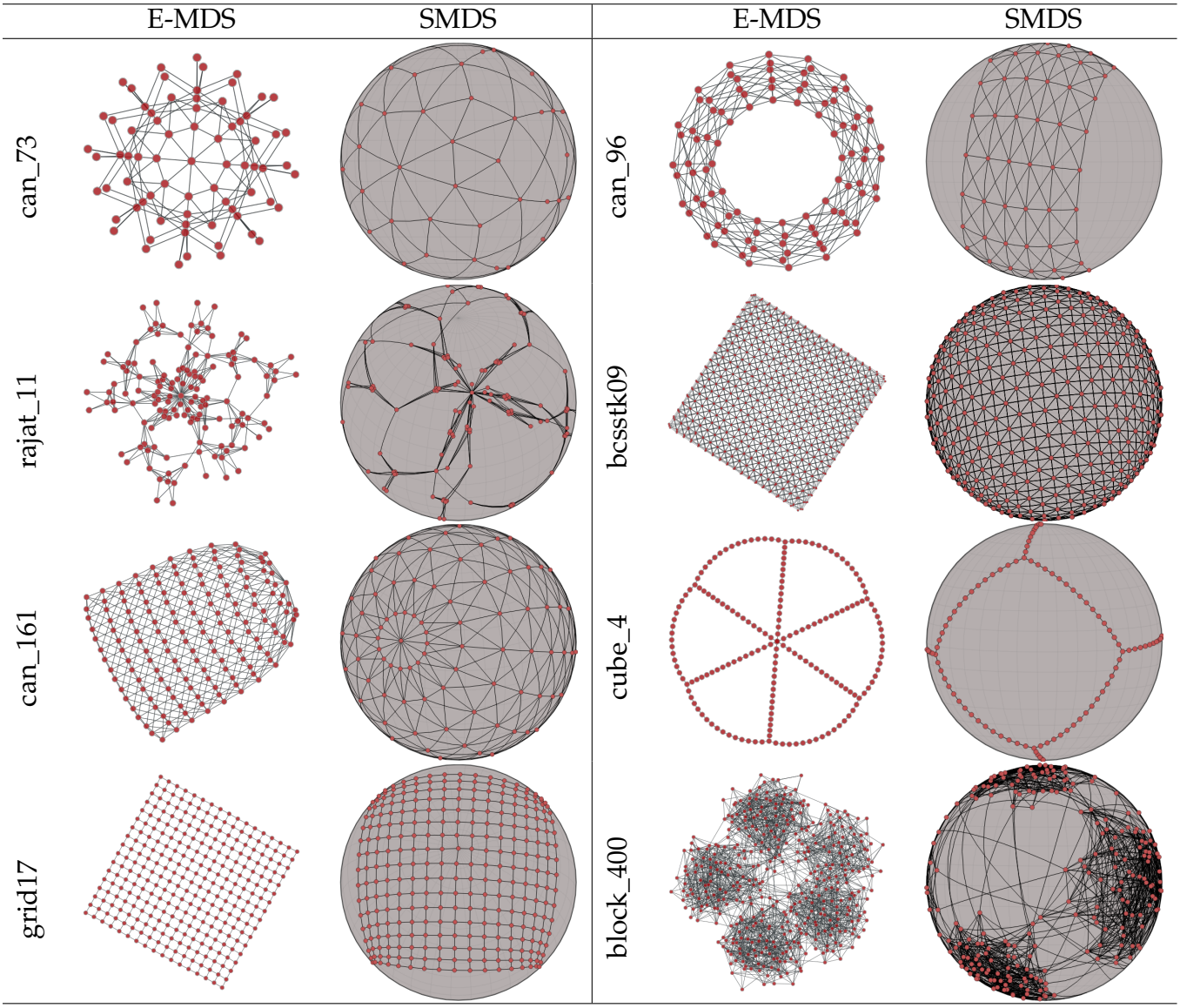
4.6 Conclusions and Future Work

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 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.

TABLE 4.1: Layouts



Chapter 5 Hyperbolic Multidimensional Scaling

This chapter is taken from a publication that appeared in PacificVis 2022 [91].

5.1 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 [75] 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 [57], 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,

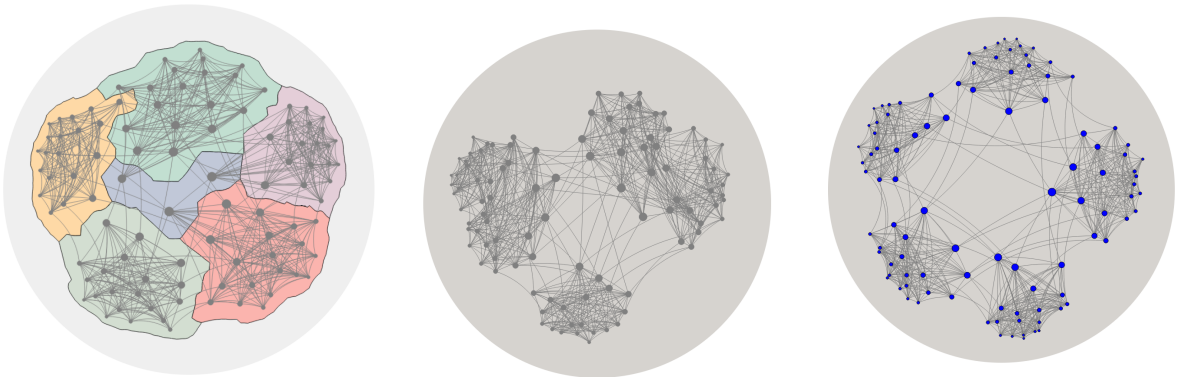


FIGURE 5.1: Example layouts of the same graph, generated by the three hyperbolic graph embedding algorithms discussed in this paper: inverse projection (left), force-directed (center) and hyperbolic-MDS (right).

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 [73]. 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) [31, 138]. 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 [147]. 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`.

5.2 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 [51], MapSets [37], BubbleSets [30], and LineSets [2]. 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 [115, 116].

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 [39]. 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 [108].

The system relies on two different layout algorithms for computing a Euclidean layout: *sdfp* is a multi-level force-directed algorithm [65] and *neato* is a implementation of the Kamada-Kawai algorithm [68]. We show examples of the *colors graph* drawn as a node-link, GMap, BubbleSet, and LineSet diagram; see Fig. 5.2. 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 [5]. 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 [108].

5.2.1 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

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

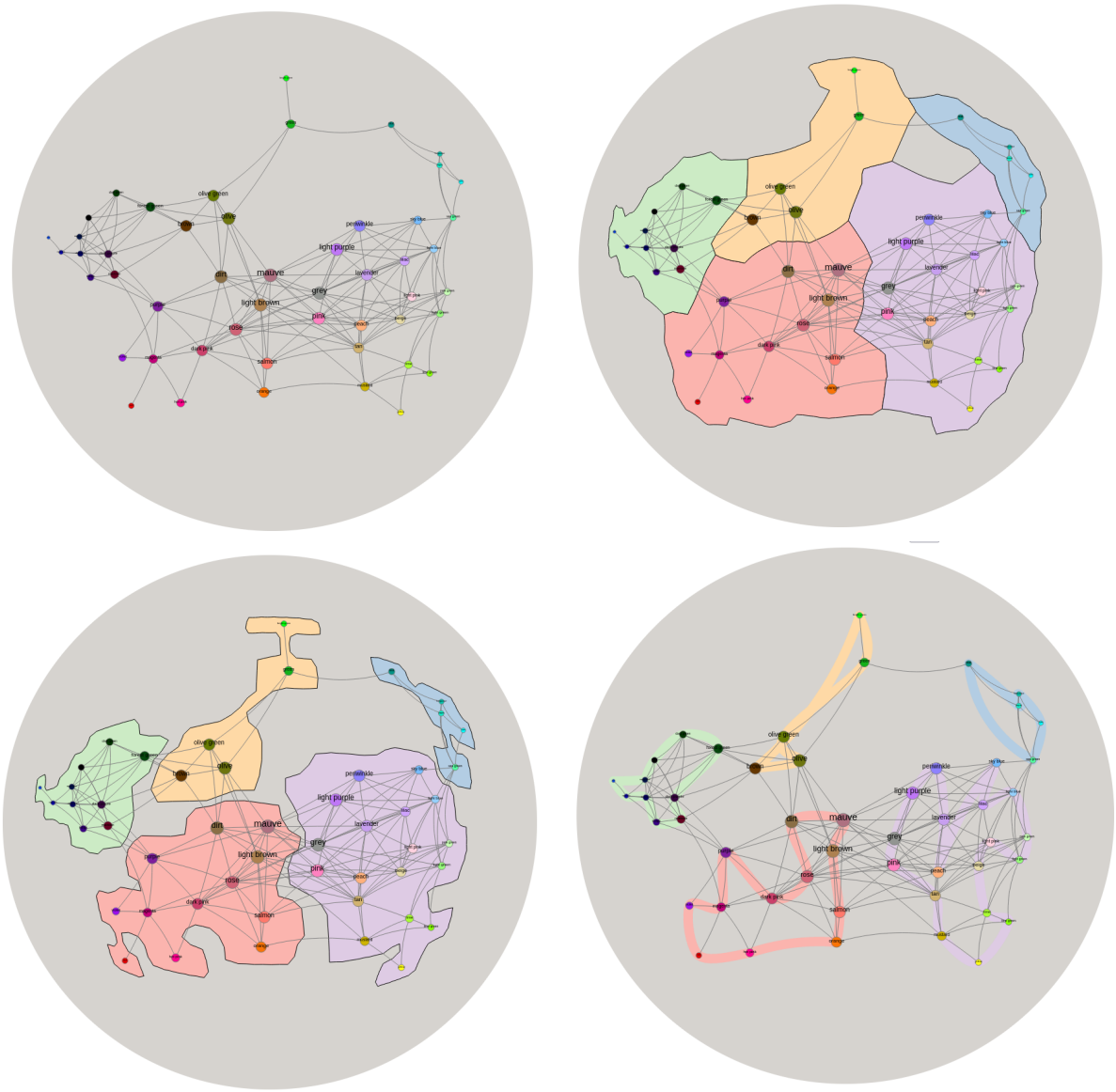


FIGURE 5.2: Different GMap drawing options for the same graph using inverse projection from Euclidean to hyperbolic space.

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 [126].

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 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.

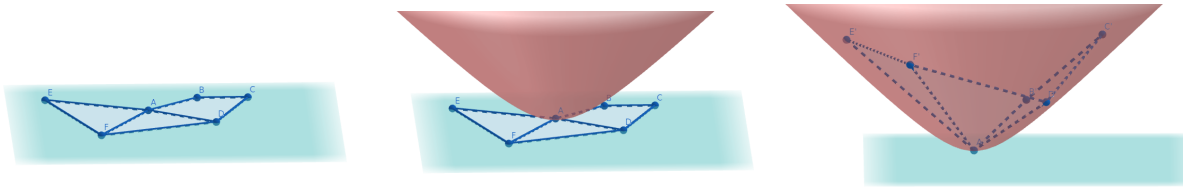


FIGURE 5.3: Illustration of an inverse projection: wrapping a plane drawing on a hyperboloid.

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. 5.3.

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

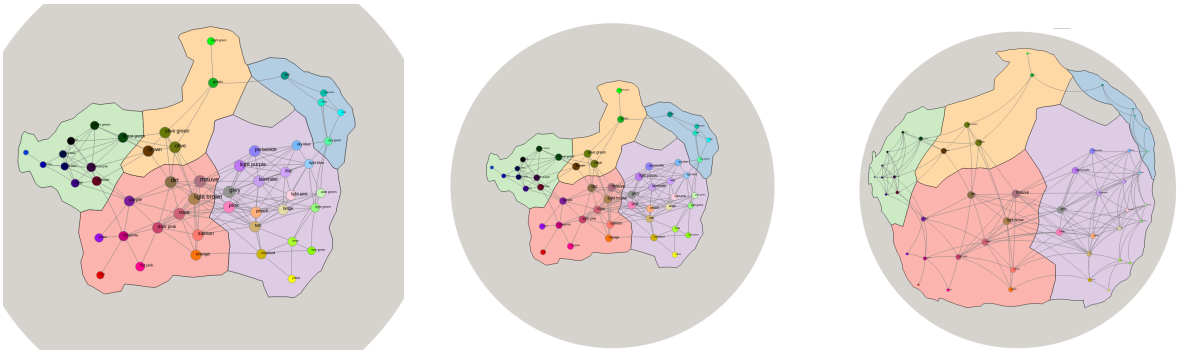


FIGURE 5.4: An example of the default (center), increased zoom (left), and increased coverage (right) for the same graph.

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.

5.2.2 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. 5.5, but interacting with the actual visualization in GMap better conveys the idea. Fig. 5.6 additionally shows the difference between panning to the edge of a map in Euclidean space and hyperbolic space.

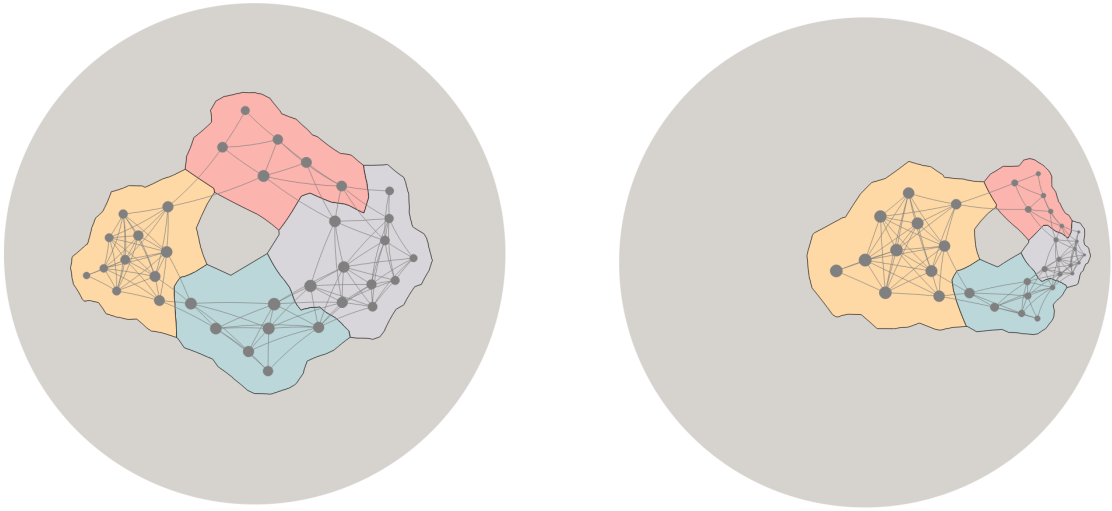


FIGURE 5.5: The same graph centered about two different origins.

5.3 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 [73]. 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

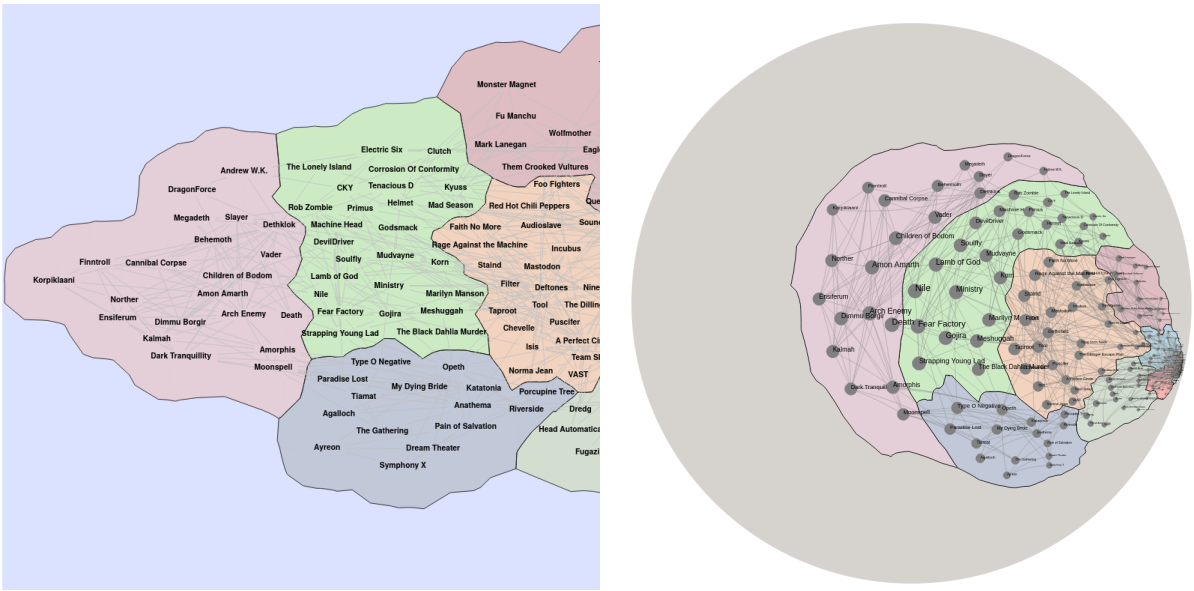


FIGURE 5.6: A 2D Euclidean GMap instance of the MusicLand graph (left) and its hyperbolic realization (right).

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 [108].

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 [68]. 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.

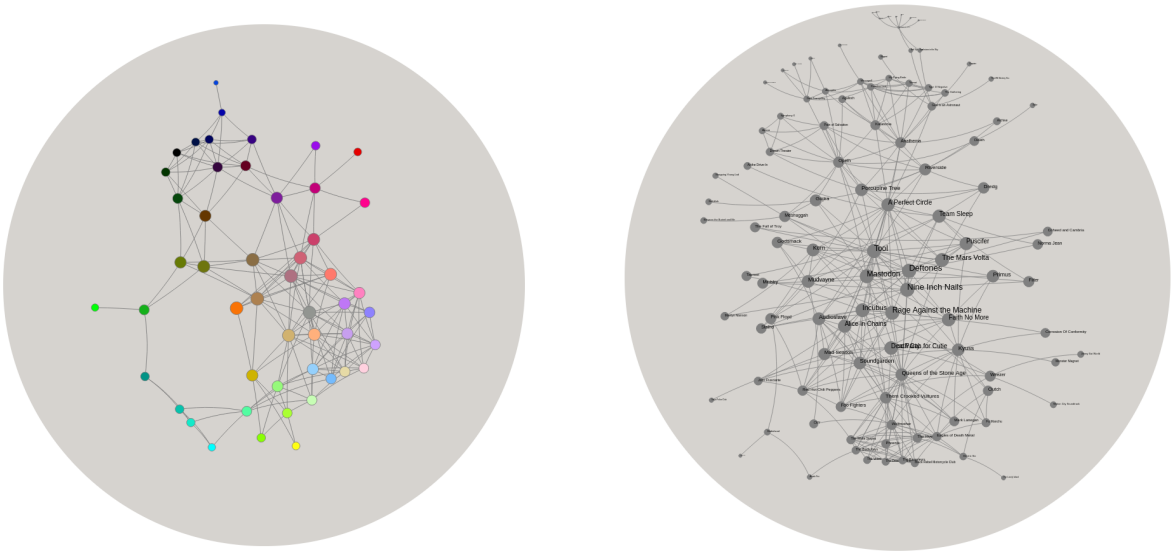


FIGURE 5.7: Force-directed colors (left) and MusicLand (right) graphs.

5.3.1 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 [73]. 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

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

5.3.2 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.

5.3.3 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/LineSets 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 [99].

Similarly, LineSets requires Bezier curves between nodes in a cluster, which should also be computable in hyperbolic space.

5.4 Multidimensional Scaling in hyperbolic space

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

$$Stress = \sum_{i<j} w_{ij}(\|X_i - X_j\| - d_{ij})^2 \tag{5.1}$$

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 [137]. 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.

5.4.1 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 (5.1). 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 [110]. 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 [61] and has been shown to work well for Euclidean SGD [147]. 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. 5.8. We use random reshuffling and define an *iteration* as a full pass through all pairs and show, in Section 5.4.1, that we only need a constant number of iterations.

Initialization As the stress function (5.1) 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 [14]. 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 [32]) 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

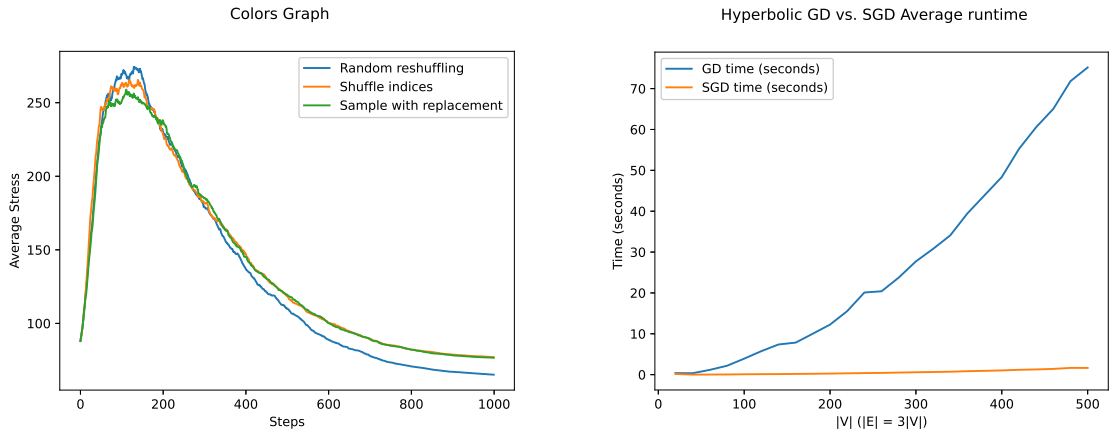


FIGURE 5.8: Effect of randomization techniques (left) described in 5.4.1, showing average stress over 30 runs at each step on the Colors graph and average runtime (right) of HMDS using GD and SGD over 30 runs (preprocessing omitted). Similar results were found on other graphs.

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 [14].

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 [14, 113]. We upper bound the product

$\eta w_{ij} \leq 1$ as in [147]. 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. $s(t_0) = \eta_{max} = d^2_{max}$ and $s(t_{max} = \eta_{min} = \epsilon d^2_{min}$ 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 [147]. 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 [147] 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 $a = d^2_{min}$ 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. 5.9. 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. 5.11

5.4.2 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 5.4.1, while the overall complexity of SGD is no different than GD, the run time is significantly faster; see Fig 5.8. 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. 5.11. We show a selection of 4 graphs from the sparse matrix collection [32] 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.

5.5 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. 5.12. 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.

5.6 Discussion, Limitations and Future Work

We described three methods for visualizing graphs in hyperbolic space, which are illustrated in Fig. 5.1. We present a small-scale comparison of the three approaches by comparing time and distortion values; see Fig. 5.14. 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. 5.6, 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 [36, 41], but it is also known that some graphs can be embedded in hyperbolic space with lower error [75] Determining whether a lower distortion in a geometry corresponds to better task support remains a promising direction for future work.

As discussed in Section 5.5, 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 [80].

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 [140]. Perhaps optimizing over additional aesthetic criteria could improve the readability of hyperbolic graph layouts.

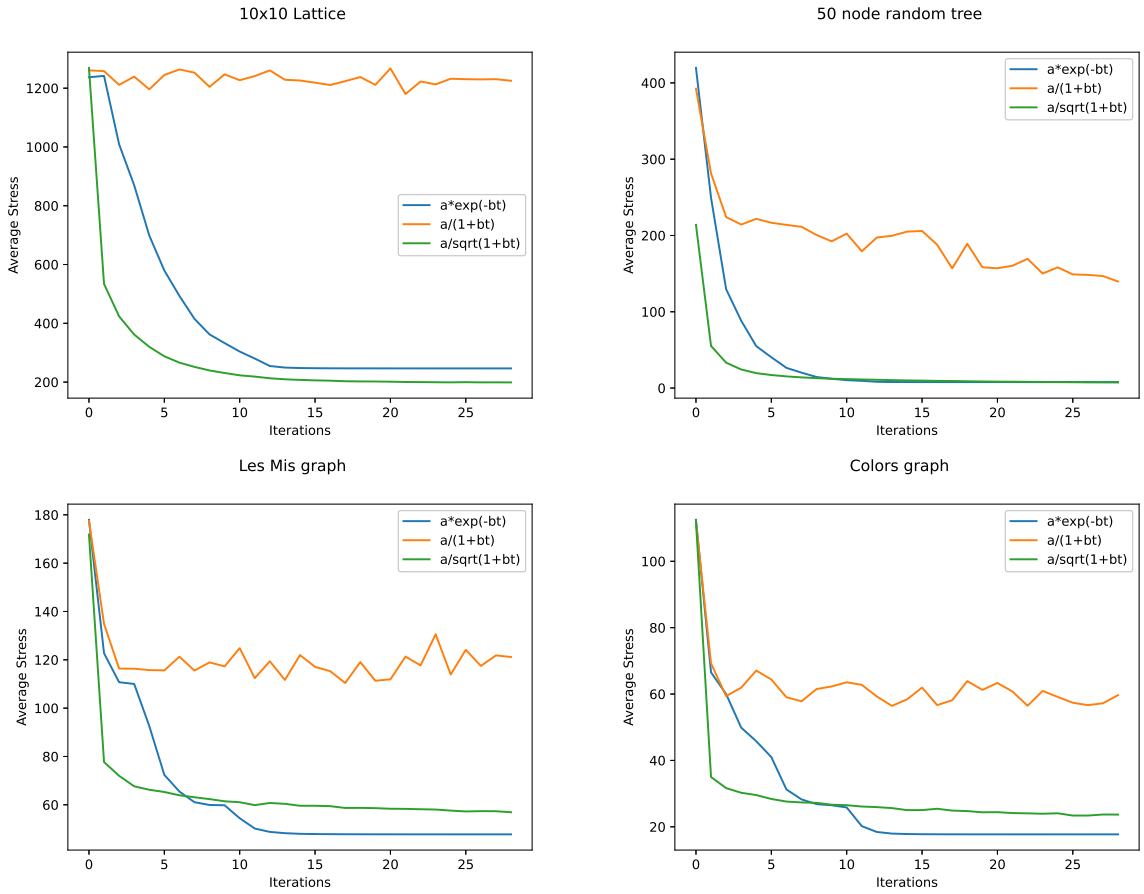


FIGURE 5.9: Effect of learning rate on various classes of graphs (average over 15 runs each). Graphs used are a 10x10 grid (top left), 50 node random trees (top right), the Les Mis graph [71] (bottom left), and the colors graph (bottom right).

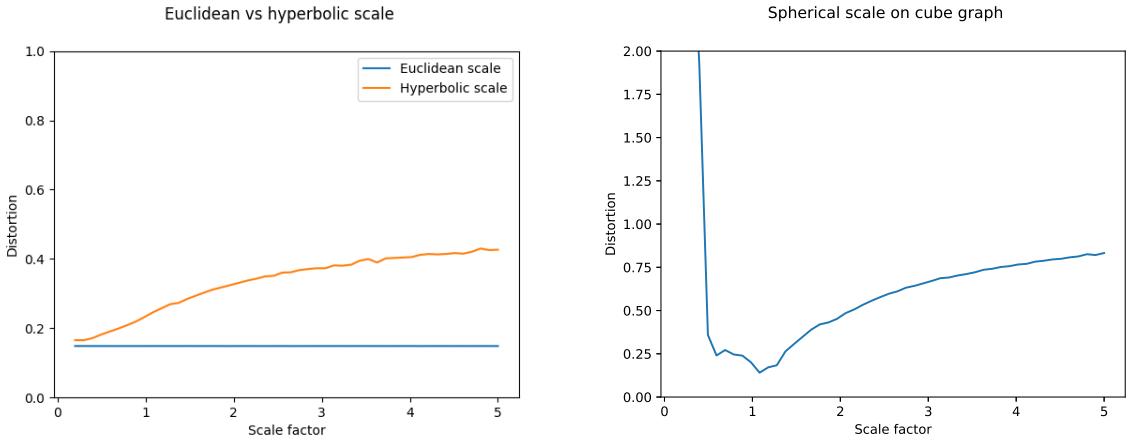


FIGURE 5.10: Left: Distortion on triangular lattice graph shown in Fig. 5.12. Hyperbolic space gets worse as the scale increases, but Euclidean can embed the graph with constant error. Right: The effect of scale on the sphere on a cube graph. For this example, there is a noticeable optimum at around $\pi/3$ (note that the diameter of a cube graph is 3).

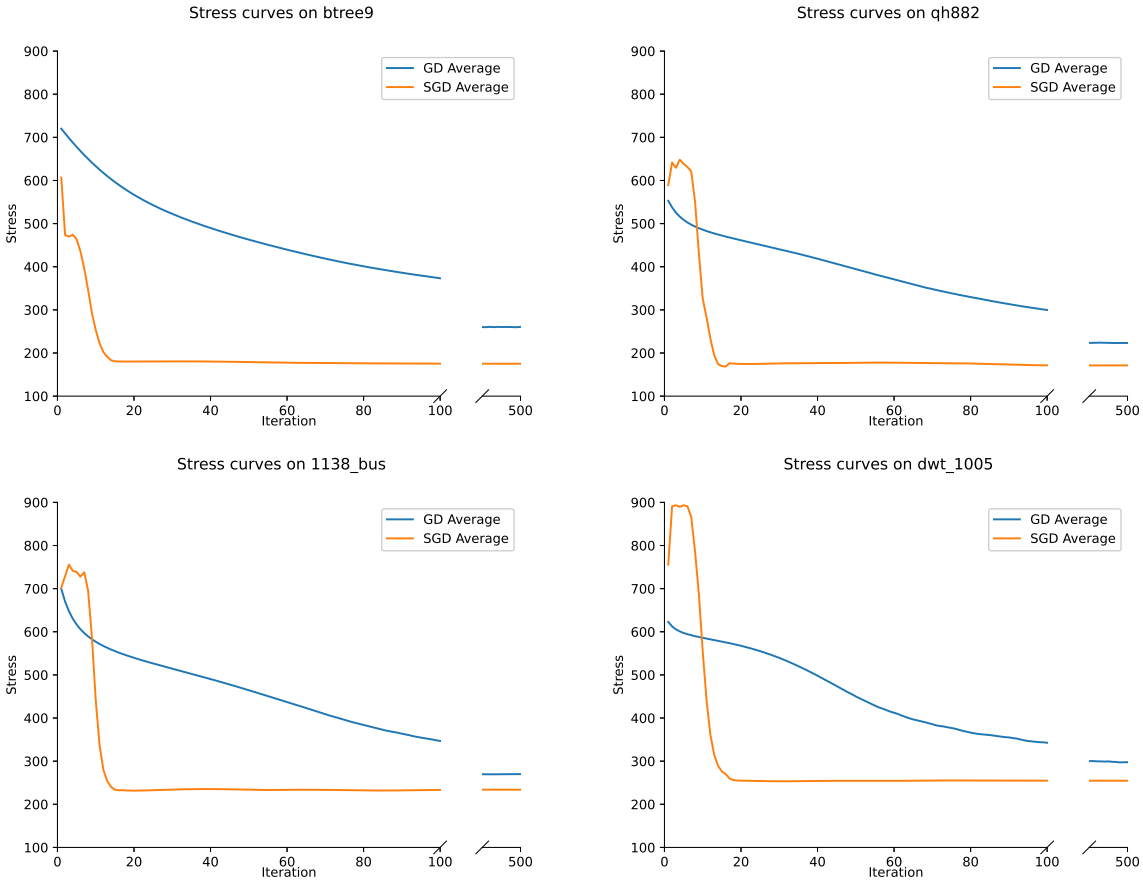


FIGURE 5.11: Average stress plots of GD and SGD. Initial stress values are omitted.

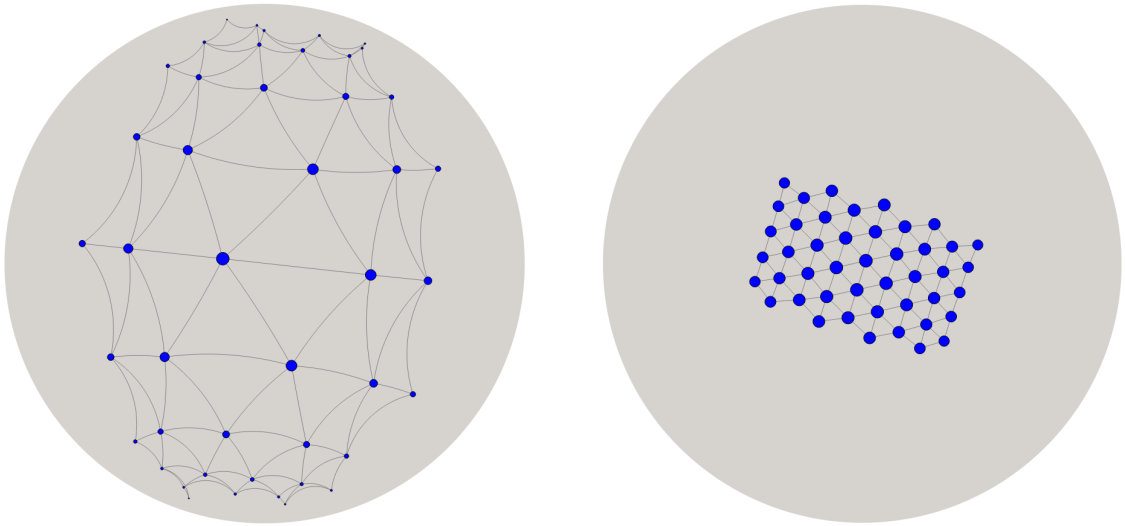


FIGURE 5.12: Triangular lattice with scaling factor $\alpha = 1$ (left) and optimized $\alpha = 0.22$ (right).

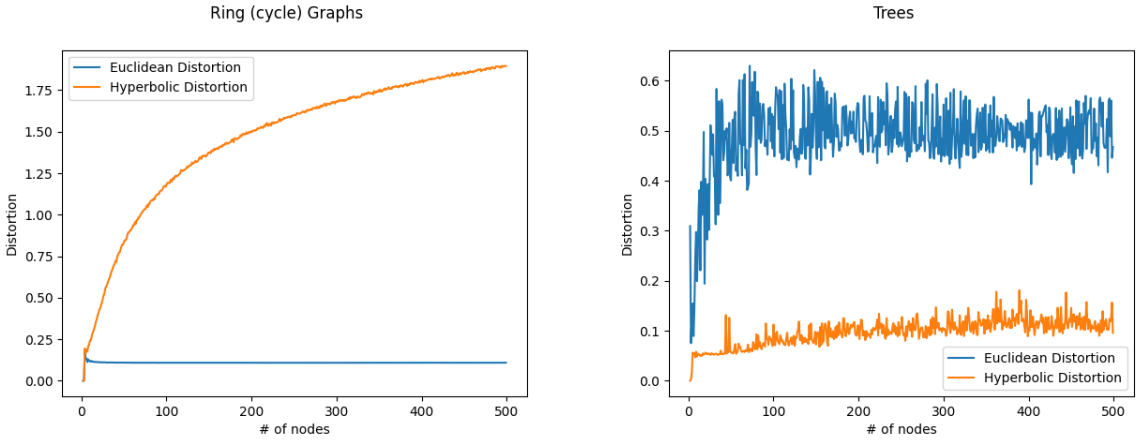


FIGURE 5.13: Euclidean and hyperbolic embedding distortion on rings (left) and trees (right). It can be seen that the number of nodes in a ring in Euclidean space does not matter, but distortion gets worse with size of the ring in hyperbolic space. The inverse is true for trees, they can be embedded with constant distortion in hyperbolic space but not Euclidean.

	Projection	FDA	SGD		Projection	FDA	SGD
Colors	1.1062	1.1555	0.2514		0.5864	0.662	0.2227
Music	1.7201	13.9767	0.435		1.844	0.6182	0.1957
btree8	2.2095	29.9494	1.9346		1.3885	0.8873	0.1379
1138_bus	7.4513	210.5452	10.625		2.3551	0.9352	0.1576
		Time (s)				Error (distortion)	

FIGURE 5.14: Average time in seconds (left) and average distortion value (right) on the listed graphs for each of three methods presented in the paper.

Chapter 6 Non-Euclidean Human Subjects Study

6.1 Study Introduction

When one begins to visualize a graph or network using a node-link diagram, one typically imagines placing the nodes and routing the edges in 2 dimensional Euclidean space; the space of the computer screen or a sheet of paper. However, representations in different geometries have proposed visualization benefits such as focus+context or achieving a more faithful representation. There have been several works in applying graph visualization techniques beyond the standard 2 or 3 dimensional Euclidean space. Spherical [59, 73, 118], hyperbolic [83, 94, 97], and toroidal [23, 24] geometries have been considered in many visualization papers, dating back to the 1990s.

Although these visualization styles have proposed benefits not seen in typical Euclidean drawings, there has been little work in evaluating their performance with human subjects studies. Thus, there is little one can say about when it is appropriate to make use of such non-Euclidean graph visualizations. Preliminary results seem to indicate some limited benefits, but are limited to a single type of task and single type of graph [35]. More specifically, the effectiveness of the specific geometry could depend on what graph is used, since some graphs "live" more naturally (i.e. are embedded with lower error) in spherical/hyperbolic space [90].

We designed, implemented, ran, and analyzed a human subjects study that covers more of the possible graph task space and includes graphs representative of Euclidean,

	Adjacency	Accessibility	Common Connection	Connectivity	Overview
Retrieve Value					
Filter					
Compute Derived Value					
Find Extremum					
Sort					
Determine Range					
Characterize Distribution					
Find Anomalies					
Cluster					
Correlate					

TABLE 6.1: Task Taxonomy

spherical, and hyperbolic space. We show that while the use of such geometries may be detrimental for some graph types, they may also be beneficial with others.

We would like to more conclusively show whether or not there are measurable benefits to spherical or hyperbolic network visualization. Specifically, we would like to answer the following research questions:

- **RQ1**: Does a lower embedding error of a graph in a geometry correspond to better task support?
- **RQ2**: Are there some tasks better suited to certain visualization geometries?
- **RQ3**: Is there an overall benefit to using spherical/hyperbolic geometry for network visualization over Euclidean?

6.2 Methodology and Design

6.2.1 Conditions

The conditions of interest are the geometries of the visualization: Euclidean, spherical, and hyperbolic. As much as feasible, the conditions differ only in the behaviour of the geometry with great care to ensure everything else remained the same.

Euclidean (E): The Euclidean condition represents a standard node-link diagram, drawn in the flat, 2D space of the computer screen. Nodes are drawn as circles and edges drawn as straight line segments between their incident nodes. Node radius is set to 10 pixels, with edge width set to 1.5 pixels. The drawing is scaled so that it just fits the viewing window upon start. The border of the viewing window is drawn as 1 pixel wide black line, indicating the boundary. An example is shown in Fig. **??**(a).

Hyperbolic (H): The hyperbolic condition is where the graph is first embedded in hyperbolic space, and positions on the computer screen are found from a Poincarè projection. Nodes still drawn as circles, with edges becoming circular arc segments which represent geodesic segments in hyperbolic space. In the Poincarè projection, there is an exponential shrinkage of depicted area the further from the center. For this reason, while nodes in the center of the projection have a radius of 10 pixels, they get smaller the further from the center they are. Edge width is 1.5 pixels, scaling appropriately as the nodes. The boundary of the disk is drawn with a solid 1 pixel black circle. An example is shown in Fig. **??**(b).

Spherical (S): Much like the hyperbolic condition, the spherical condition is where the graph is first embedded on a sphere, but positions are found from an orthographic (view-from-space) projection. Nodes are drawn as circles with 10 pixel radius at the center, but due to the projection become ellipses the closer they get to the boundary. Edges are again circular arc segments that represent geodesic segments in spherical space. The width of the segments is 1.5 pixels. The boundary of the projection is drawn with a solid 1 pixel black circle. Unique to the sphere, due to the orthographic projection, half of the drawing space is always obscured at one time. An example is shown in Fig. **??**(c).

6.2.2 Interactions

Although introducing interactions also introduces complexity, we have opted to use interactive tools for each geometry. In particular, the **H** and **S** conditions must be interactive to be useful, due to the focus+context. We record all interactions used by a participant, and analyze them in section **??**. Specific interaction implementations are discussed below.

Pan

The geometric pan interaction allows users to look closely at the details of some of the more dense graphs we have included in our study. It allows users to grab and drag the graph to investigate subsections of graphs closely. This has often been used in user studies for graph visualizations and can be found in studies like [36, 105, 120].

Recenter around click

A interaction typically present in hyperbolic browsing systems [92] is the recenter interaction. When the user double clicks on a point in the drawn space, the system will automatically pan so that the clicked point is now in the center of the drawing. So any point of interest can be easily clicked and navigated to without manually dragging the mouse. An example is shown in Fig. 6.1.

Hover

Early feedback indicated that participants were unsure exactly where they were clicking to perform the recenter interaction. Taking this into account, we implemented a simple hover interaction on the nodes of our drawings. When the mouse cursor enters the boundary of a node, the color changes, the boundary is bolded, and the incident links are bolded. The benefits of static and motion highlighting of nodes in a node-link diagram are mentioned in [141] with reference to solving visual tasks such as the ones we include in our test. This interaction functions identically in all 3 geometry systems, and an example is found in Fig. 6.2.

Reset Visualization

We included the reset visualization to reload the default state of the visualization in case the user gets lost while navigating the graph or faces some kind of visual error. This interaction is a part of our study as a fail-safe method to allow users to not get stuck and continue interacting with the graph if they face any problems.

6.2.3 Fixed parameters

We discuss the fixed parameters of the experiment including the data (graphs and layout) and aesthetic choices (sizes and colors).

Graphs

The selection of graphs was a careful choice in order to be able to effectively answer **RQ1**. We initially intended to use only real-world graphs taken from [90] but ultimately decided controlling for size and density was desirable for most graphs to reduce confounding factors. For each geometry, we generated two random graphs in a scheme nearly identical to the Random Geometric Graph generator for Euclidean space [107], which has already been generalized to hyperbolic space [75] and generalizes naturally to the sphere. These models first uniformly distribute $|V|$ points in the desired geometry, then connect two points with an edge if they are within a geodesic radius r.

We slightly modify this procedure by instead taking the *first* $|E|$ pairs to be edges, sorted by smallest distance. In this way, we can specify both $|V|$ and $|E|$ in the graph to obtain the desired size and density while still capturing the target space. We additionally ensure we only accept connected graphs from the generator to be included in the study. These graphs are named R_rand_small and R_rand_large (where R is replaced by the type of geometry) and are summarized in Table 6.2.

We included three additional graphs of interest that would not appear from the generator but are representative of the types of graphs that obtain low distortion in their geometry. *dwt_162* for Euclidean space is a wire mesh, *tree_150* is tree on 150 nodes, and *dodecahedron_3* is the graph representation of the planar solid dodecahedron, subdivided three times. These are also summarized in Table 6.2.

	|V|	|E|	avg_degree	**E**_distortion	**H**_distortion	**S**_distortion
E_rand_small	50	150	6.0000	**0.1033**	0.1362	0.1217
E_rand_large	100	400	8.0000	**0.1011**	0.1312	0.1495
dwt_162	162	510	6.2963	**0.0928**	0.2217	0.1191
H_rand_small	50	150	6.0000	0.2465	**0.0913**	0.1143
H_rand_large	100	400	8.0000	0.2459	**0.1234**	0.1829
tree_150	150	149	1.9867	0.2815	**0.1514**	0.2068
S_rand_small	50	150	6.0000	0.2443	0.1121	**0.0897**
S_rand_large	100	400	8.0000	0.2450	0.1439	**0.1060**
dodecahedron_3	230	240	2.0870	0.3014	0.3615	**0.0544**

TABLE 6.2: Summary of the graphs that appeared in our experiment along with their embedding scores in each geometry. Accompanying images of each graph and layout can found in supplementary material.

Graph Layout

The layout algorithms used were chosen carefully to be as similar as possible in order to reduce noise resulting from layouts in our results. We opted for stress-based approaches for each geometry, as they are readily available. We use a Euclidean MDS layout from [147], spherical MDS from [90], and hyperbolic MDS from [92]. Each of the 9 graphs $\times$ 3 geometries $=$ 27 graph-layout pairs were visually inspected to ensure they produced reasonable results. A representative example is shown in Fig. **??**, and all layouts can be found in supplementary material.

Style

We attempt to keep node sizes as consistent as possible across each condition, despite the differing behaviour of the geometries. At the exact geometric center of each condition, the node size is precisely 10 pixels. In Euclidean geometry, the size and shape is unchanging. Hyperbolic geometry has a strong focus+context where the size of nodes rapidly decrease as they approach the boundary. In spherical geometry, the node circles are stretched to

become ellipses, eventually becoming infinitely thin when they disappear "behind" the globe.

For some tasks, we ask questions related to "highlighted" nodes, which appear 50% larger in radius than they would otherwise.

Each condition has a default white background, with a 1 pixel black border to indicate the visualization space. This border is a rectangle in **E**, but a circle in both **H** and **S** conditions due to the chosen projections.

For the nodes, we opted for color blind safe colors as mentioned by [143] and used light blue for visualizing nodes by default, yellow for nodes hovered by the user, magenta for neighbors of the nodes hovered by the user and finally orange for "highlighted" nodes related to the question; see Fig. 6.2. The exact color scheme can be found in the appendix.

Edges are 1.5 pixels in width and are grey. When an incident node has been hovered over by the participant, the edge will increase to 3 pixels in width and turn black.

Display Size

As the interactions and feel of the tools were developed with desktop PCs in mind, we decided to restrict the availability of the test on mobile, tablet and other smaller devices. We set the minimum width of a device to be 801 pixels and any device under that measurement would not be able to access any page of the tool. Controlling the use of such small devices removes a potentially confounding variable in the study.

6.2.4 Tasks

In an effort to cover as broad a spectrum of tasks as possible, we created a table that represents the intersection between the graph task taxonomy of Lee et al. [84] and the

more general visualization taxonomy of Amar et al. [3]; See Table 6.1.

First, we identified where Du et al.'s [35] three tasks fell in this table, shown in red (**D1, D2, D3**). From these tasks, we looked to how we might expand on this study to cover more rows and columns of the table. We narrowed down and concluded with a set of 6 tasks, shown in blue (**T1, T2, T3, T4, T5, T6**) and detailed below:

- **T1:** Count neighbors of node with given degree
- **T2:** Count common neighbors of two nodes
- **T3:** Determine the length of the shortest path between two nodes
- **T4:** Estimate the size (nodes or edges) of the graph
- **T5:** Determine the most frequent degree of a group of nodes
- **T6:** Count how many additional nodes are reachable from a given node in given number of steps

Between the six of them, these tasks span the columns of Table 6.1, which covers the graph specific taxonomy while being somewhat spread amongst the rows giving us a good mix for general data visualization tasks. Each of these tasks are also reasonably able to be completed within 30 seconds, an important facet to keep the completion time of the study at realizable lengths.

An instance of one of the above tasks is the graph in which the task occurs together with a node or nodes which are being asked about and a collection of 4 possible answers (of which one is correct).

The instances of tasks that appeared in our study were generated programatically, but all instances were human verified to ensure 1) unique correct answers, 2) the instance is

able to be completed within 30 seconds (this would not be the case for instance, if one had to count several nodes of degree 20+ to find the correct answer). Possible answers are always shown in sorted order.

For each graph, we create 3 instances of each of our 6 tasks, giving us $9 \times 6 \times 3 = 162$ total unique instances. An example of one instance of each task is shown in Fig. 6.3 and we have included all 162 instances as a json file in supplementary material.

6.2.5 Experimental design

Clearly, there are too many graph/task/geometry triples ($9 \times 6 \times 3 = 162$) to be able to reasonably perform a fully within-subjects study. However, we felt it important to be able to directly compare participant results on geometry so we opted for a within subjects design with respect to geometries and graphs (every participant will see all three geometries and all 9 graphs), but broke our tasks into 3 groups of 2 (each participant will only see two of the six tasks).

This gives us a design where each participant answers 54 questions (3 geometries x 9 graphs x 2 tasks). The order in which participants used each geometry as well as the order of the questions was selected from a balanced Latin square to reduce learning effects.

Experiment outline

After consenting to participating but before starting the study, participants are first shown an introduction page. This page introduces definitions, descriptions of interactions, and an example visualization of the condition for the first part of the study. Participants are encouraged to spend time familiarizing themselves with the current condition,

interacting with a small sample graph displayed in either **E**, **H**, or **S** conditions; See Fig **??**.

Participants would then be asked to complete 18 questions in the starting condition, one for each graph-task pair. At the end of these, participants were asked to answer a few subjective questions about the condition (such as how useful or aesthetically pleasing the condition was).

After performing a block of 18 questions, participants end up back on a modified version of the landing page, indicating that they may now take a break and allowing them to familiarize themselves with the next condition. This repeats until all three conditions are exhausted.

Finally, the final page of the experiment collects general demographics such as age group and gender, and plaintext feedback.

6.3 Results and analysis

We collected experiment data through crowd sourcing, making our webpage available through the social media website Reddit; specifically the r/takemysurvey and r/samplesize subreddits. We additionally did further recruiting through friends and colleagues. No incentives were offered for taking the experiment. In total, we had 26 participants complete the study. Of these, about 80% were men, 12% were woman, and the remaining 8% preferred not to say. 85% of participants were between the ages of 18-35 and the remaining in the ages of 36-55. Familiarity with network visualizations ranged from 15% being either completely new or have seen in passing, 45% had discussed in coursework, 20% use them for their work, and 20% were experts.

6.3.1 Hypotheses

To answer the research questions posed in Section 6.1, we developed the following hypotheses:

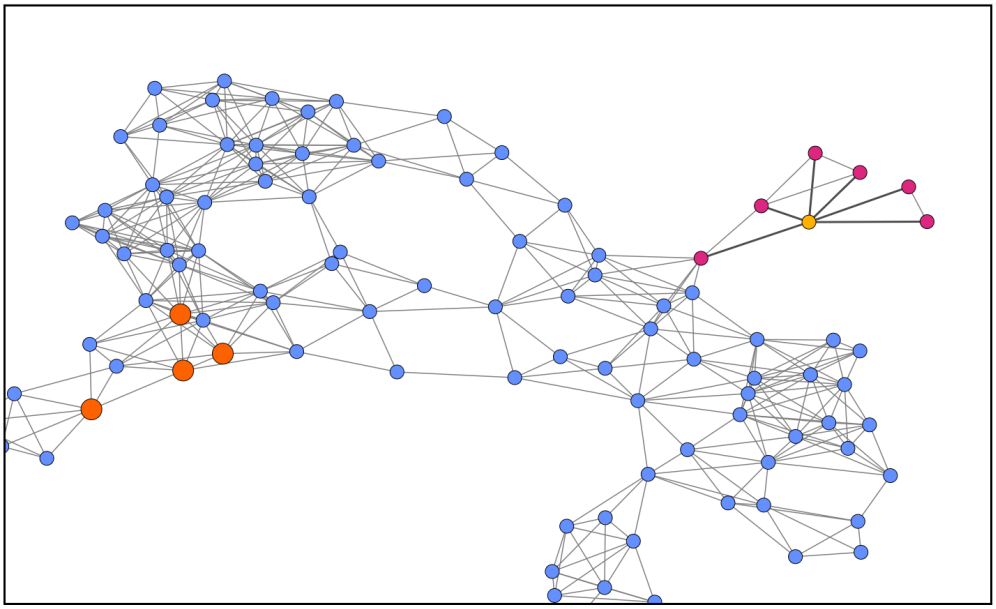
- **H1a** Graphs which have lower distortion in Euclidean space will be significantly more accurate in the Euclidean visualization condition
- **H1b** Graphs which have lower distortion in hyperbolic space will be significantly more accurate in the hyperbolic visualization condition
- **H1c** Graphs which have lower distortion in spherical space will be significantly more accurate in the spherical visualization condition
- **H2** The Euclidean visualization condition will be overall faster than the other two

6.3.2 Quantitative analysis

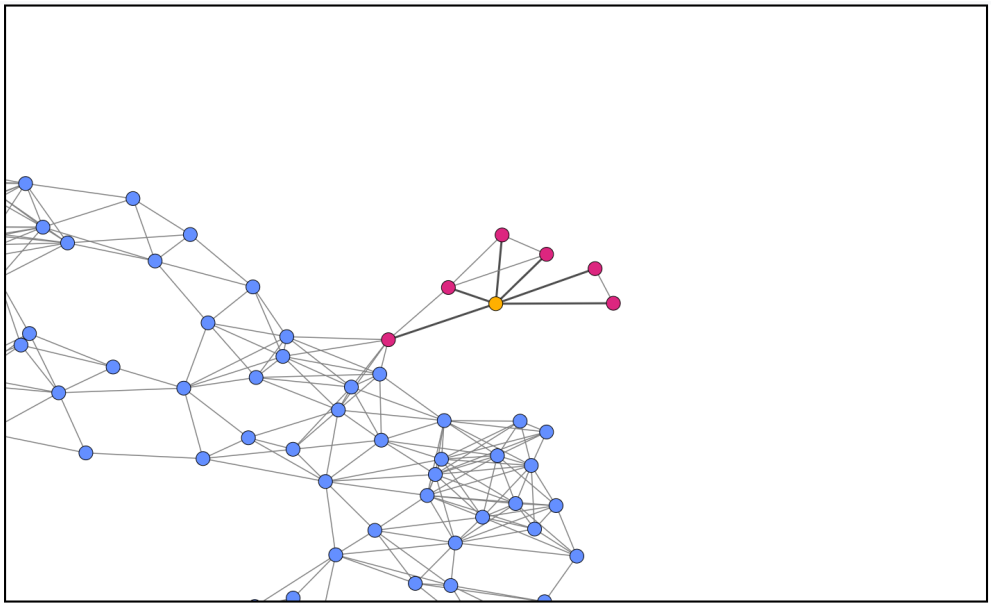
6.3.3 Qualitative Feedback

6.4 Discussion

6.5 Conclusion



(a) The hovered node (yellow with magenta neighbors) is the point of interest which would be centered.



(b) The hovered node after double clicking on it moves to the center of the viewing window

FIGURE 6.1: Re-centering a node by double clicking on it. (a) shows a node which is to be re-centered and (b) shows the node in (a) re-centered to the center of viewing window.

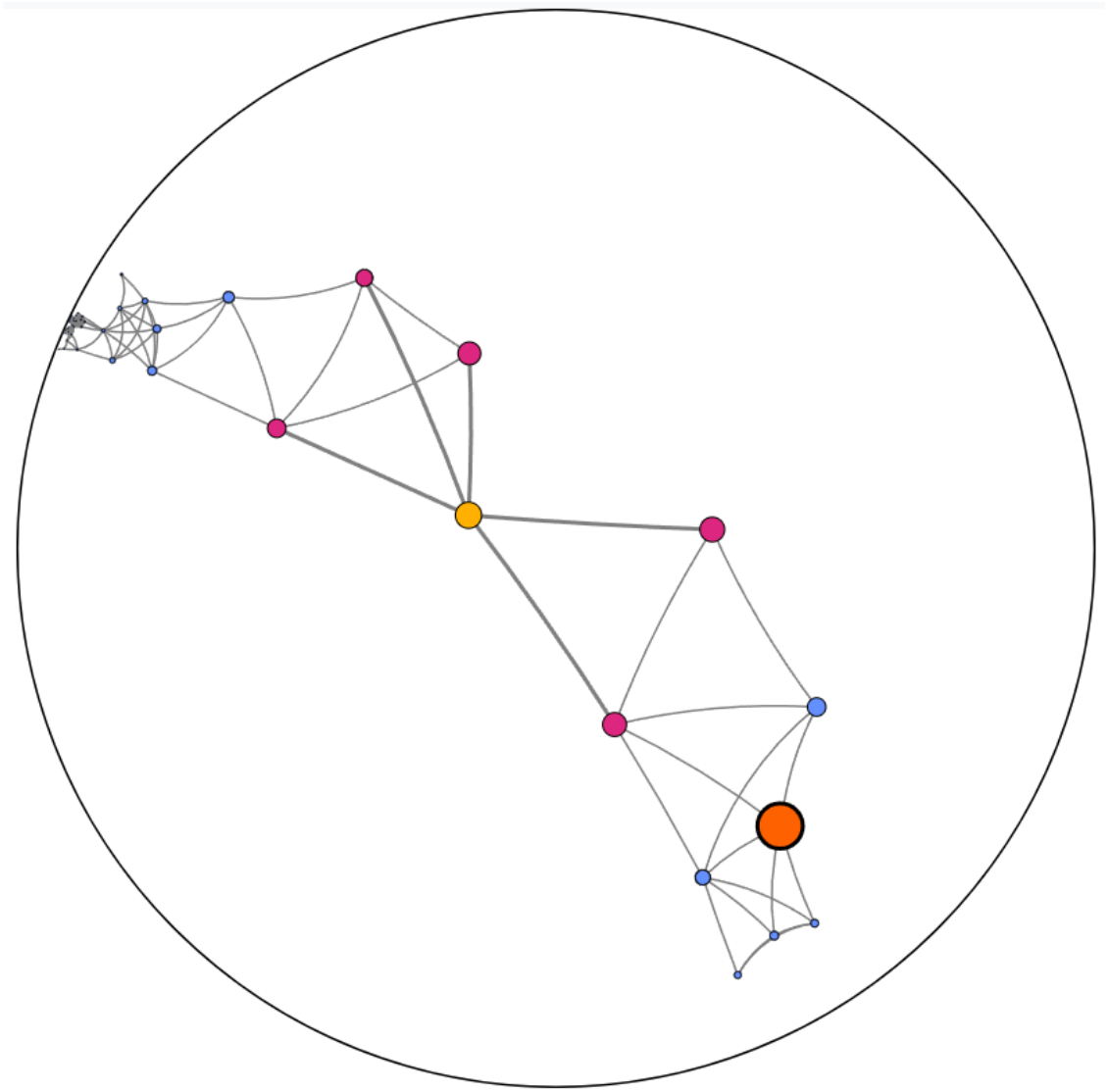


FIGURE 6.2: On hovering a node, it changes color to yellow from a default blue and all direct neighbors of hovered node also change their color to magenta. The edges between the hovered node and its neighbors are emphasized by increasing width and opacity. The orange node is the node in reference to the task asked to complete by the user.

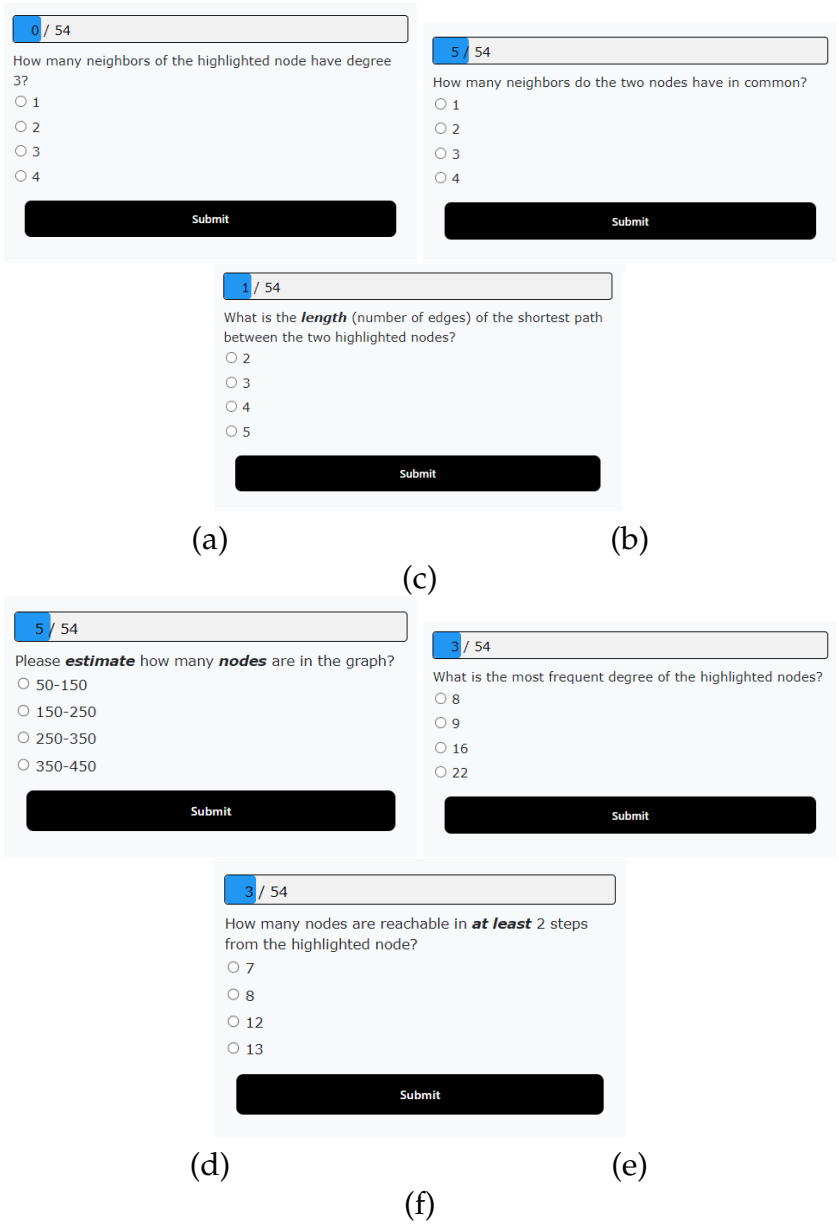


FIGURE 6.3: All tasks displayed to users (a) T1 (b) T2 (c) T3 (d) T4 (e) T5 (f) T6

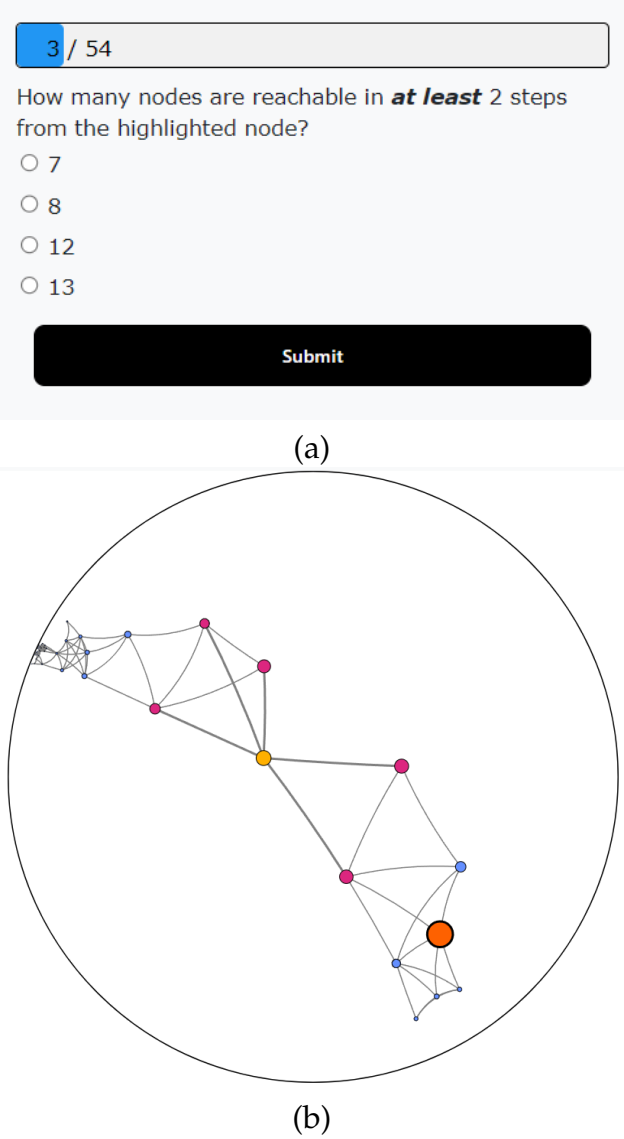


FIGURE 6.4: An example of T6 with (a) question asked to the user and (b) graph displayed along with task to answer question.

Chapter 7 Future Work and Conclusions

7.1 Future Work

While there exists a large body of interesting research in this subject area, there is still much to be done. We give an overview of potential future work discovered in 2 but not addressed by the previous chapters.

We are unaware of any further human-subject studies evaluating graph embedding techniques in spherical and hyperbolic space. For example, is there qualitative difference between the Projection-Reprojection methods (which are very fast, but do not take advantage of the underlying geometry) and the more computationally expensive native formulations?

A large advantage of the sphere as pointed out by [60, 90] is that there is no defined 'center' or 'periphery'. Any node can be placed in the center of the drawing, something not possible in Euclidean space. Can this be used to prevent anchoring or similar biases?

There are many potentially interesting questions in quantitative measures for non-Euclidean geometries. Are there bounds for other aesthetic criteria such as crossing angle like there are for angular resolution in hyperbolic space [40]? Do similar bounds exist for the sphere and torus? Can we characterize the graphs that 'live naturally' (i.e. have low distortion) in hyperbolic/spherical/toroidal space? Do other faithfulness metrics like Neighborhood Preservation need modifying to make sense in non-Euclidean space?

Like the spherical and hyperbolic space, the torus has an absolute scale. However, the effect of dilation/resizing of a drawing on the torus has not yet been studied.

We do not know how people read, understand, and utilize non-Euclidean embeddings. More human-subject studies can provide guidelines for the design of other techniques and applications.

Rodighiero [111] posits that the choice of projection of the sphere effects the reader perception of the network data and argue that a projection that preserves the continuity of the network is desirable. Can we: (1) show that this is true; and (2) select projections more intelligently?

Although there have been some attempts to utilize non-Euclidean geometry in the virtual reality [79,80], this field seems ripe for exploration of immersive visualization. Of course, data can be put on a globe, but the focus+context effect of hyperbolic space is exaggerated in 3D.

7.2 Conclusions

We began in chapter 2 by introducing the concept of non-Euclidean geometry and reviewed the history and current state-of-the-art of its use in graph visualization. Then, an example Euclidean graph layout algorithm was developed in chapter 3. In 4, we took our first look at non-Euclidean graph layout through spherical multi-dimensional scaling, followed by several methods to create hyperbolic graph drawings in 5. We took the observations and lessons learned from the previous 3 chapters and conducted a human subjects study in chapter 6, showing that there are cases where non-Euclidean

graph visualization has real benefit. Finally, we concluded with some open problems and future work in chapter 7.

Bibliography

[1] Luca Castelli Aleardi, Gaspard Denis, and Éric Fusy. Fast spherical drawing of triangulations: An experimental study of graph drawing tools. In Gianlorenzo D'Angelo, editor, *17th International Symposium on Experimental Algorithms, SEA 2018, June 27-29, 2018, L'Aquila, Italy*, volume 103 of *LIPIcs*, pages 24:1–24:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018.

[2] Basak Alper, Nathalie Henry Riche, Gonzalo A. Ramos, and Mary Czerwinski. Design study of LineSets, a novel set visualization technique. *IEEE Trans. Vis. Comput. Graph.*, 17(12):2259–2267, 2011.

[3] Robert A. Amar, James Eagan, and John T. Stasko. Low-level components of analytic activity in information visualization. In John T. Stasko and Matthew O. Ward, editors, *IEEE Symposium on Information Visualization (InfoVis 2005), 23-25 October 2005, Minneapolis, MN, USA*, pages 111–117. IEEE Computer Society, 2005.

[4] Keith Andrews, Werner Putz, and Alexander Nussbaumer. The hierarchical visualisation system (HVS). In *11th International Conference on Information Visualisation*, pages 257–262. IEEE Computer Society, 2007.

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

[6] Jason Baumgartner and Tim A. Waugh. Roget2000: a 2d hyperbolic tree visualization of Roget's thesaurus. In *Visualization and Data Analysis*, volume 4665 of *SPIE Proceedings*, pages 339–346, 2002.

[7] Fabian Beck, Michael Burch, Stephan Diehl, and Daniel Weiskopf. A taxonomy and survey of dynamic graph visualization. *Comput. Graph. Forum*, 36(1):133–159, 2017.

[8] Nicolas Garcia Belmonte. Javascript InfoVis Toolkit. `https://philogb.github.io/jit/demos.html`. accessed: 2021-06-06.

[9] Therese Biedl. Visibility representations of toroidal and klein-bottle graphs. In Patrizio Angelini and Reinhard von Hanxleden, editors, *Graph Drawing and Network Visualization - 30th International Symposium, GD 2022, Tokyo, Japan, September 13-16, 2022, Revised Selected Papers*, volume 13764 of *Lecture Notes in Computer Science*, pages 404–417. Springer, 2022.

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

[11] Thomas Bläsius, Tobias Friedrich, and Maximilian Katzmann. Force-directed embedding of scale-free networks in the hyperbolic plane. In David Coudert and Emanuele Natale, editors, *19th International Symposium on Experimental Algorithms, SEA 2021, June 7-9, 2021, Nice, France*, volume 190 of *LIPIcs*, pages 22:1–22:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.

[12] Thomas Bläsius, Tobias Friedrich, Anton Krohmer, and Sören Laue. Efficient embedding of scale-free graphs in the hyperbolic plane. *IEEE/ACM Trans. Netw.*, 26(2):920–933, 2018.

[13] Marthe Bonamy, Bojan Mohar, and Alexandra Wesolek. Limiting crossing numbers for geodesic drawings on the sphere. In David Auber and Pavel Valtr, editors, *Graph Drawing and Network Visualization - 28th International Symposium, GD 2020, Vancouver, BC, Canada, September 16-18, 2020, Revised Selected Papers*, volume 12590 of *Lecture Notes in Computer Science*, pages 341–355. Springer, 2020.

[14] Katharina Börsig, Ulrik Brandes, and Barna Pásztor. Stochastic gradient descent works really well for stress minimization. In David Auber and Pavel Valtr, editors, *Graph Drawing and Network Visualization - 28th International Symposium, GD 2020. Revised Selected Papers*, volume 12590 of *Lecture Notes in Computer Science*, pages 18–25. Springer, 2020.

[15] Léon Bottou. Stochastic gradient descent tricks. In *Neural networks: Tricks of the trade*, pages 421–436. Springer, 2012.

[16] Bernard Bou. Treebolic2 Webpage. `http://treebolic.sourceforge.net/treebolic2/en/index.html`. accessed: 2021-06-06.

[17] Richard Brath and Peter MacMurchy. Sphere-based information visualization: Challenges and benefits. In Ebad Banissi, Stefan Bertschi, Camilla Forsell, Jimmy Johansson, Sarah Kenderdine, Francis T. Marchese, Muhammad Sarfraz, Liz J. Stuart, Anna Ursyn, Theodor G. Wyeld, Hanane Azzag, Mustapha Lebbah, and Gilles Venturini, editors, *16th International Conference on Information Visualisation, IV 2012, Montpellier, France, July 11-13, 2012*, pages 1–6. IEEE Computer Society, 2012.

[18] Katy Börner, 2012.

[19] Katy Börner, Richard Klavans, Michael Patek, Angela M. Zoss, Joseph R. Biberstine, Robert P. Light, Vincent Larivière, and Kevin W. Boyack. Design and update of a classification system: The UCSD map of science. *PLOS ONE*, 7(7):1–10, 07 2012.

[20] Sergio Cabello, Bojan Mohar, and Robert Sámal. Drawing a disconnected graph on the torus (extended abstract). *Electron. Notes Discret. Math.*, 49:779–786, 2015.

[21] Arthur Cayley. Iv. a sixth memoir upon quantics. *Philosophical Transactions of the Royal Society of London*, (149):61–90, 1859.

[22] Dorota Celinska and Eryk Kopczynski. Programming languages in GitHub: A visualization in hyperbolic plane. In *Proceedings of the Eleventh International Conference on Web and Social Media*, pages 727–728. AAAI Press, 2017.

[23] Kun-Ting Chen, Tim Dwyer, Benjamin Bach, and Kim Marriott. 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.

[24] Kun-Ting Chen, Tim Dwyer, Kim Marriott, and Benjamin Bach. Doughnets: Visualising networks using torus wrapping. In *CHI '20: CHI Conference on Human Factors in Computing Systems*. ACM, 2020.

[25] Kun-Ting Chen, Quynh Quang Ngo, Kuno Kurzhals, Kim Marriott, Tim Dwyer, Michael Sedlmair, and Daniel Weiskopf. Reading strategies for graph visualizations that wrap around in torus topology. In Enkelejda Kasneci, Frédérick Shic, and Mohamed Khamis, editors, *Proceedings of the 2023 Symposium on Eye Tracking Research and Applications, ETRA 2023, Tübingen, Germany, 30 May 2023 - 2 June 2023*, pages 67:1–67:7. ACM, 2023.

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

[27] Shenghui Cheng, Pradipta De, Shaofeng H.-C. Jiang, and Klaus Mueller. Torusvisnd: unraveling high-dimensional torus networks for network traffic visualizations. In Peer-Timo Bremer, Bernd Mohr, Martin Schulz, and Valerio Pascucci, editors, *Proceedings of the First Workshop on Visual Performance Analysis, VPA '14, New Orleans, Louisiana, USA, November 16-21, 2014*, pages 9–16. IEEE, 2014.

[28] Shenghui Cheng, Wen Zhong, Katherine E. Isaacs, and Klaus Mueller. Visualizing the topology and data traffic of multi-dimensional torus interconnect networks. *IEEE Access*, 6:57191–57204, 2018.

[29] James R Clough and Tim S Evans. Embedding graphs in lorentzian spacetime. *PloS one*, 12(11):e0187301, 2017.

[30] Christopher Collins, Gerald Penn, and Sheelagh Carpendale. BubbleSets: Revealing set relations with isocontours over existing visualizations. *IEEE Trans. Vis. Comput. Graph.*, 15(6):1009–1016, 2009.

[31] Andrej Cvetkovski and Mark Crovella. Multidimensional scaling in the poincare disk, 2016.

[32] Timothy A. Davis and Yifan Hu. The University of Florida sparse matrix collection. *ACM Trans. Math. Softw.*, 38(1):1:1–1:25, 2011.

[33] Jan De Leeuw. Applications of convex analysis to multidimensional scaling. 2005.

[34] Jan De Leeuw and Patrick Mair. Multidimensional scaling using majorization: SMACOF in R. *Journal of statistical software*, 31:1–30, 2009.

[35] Fan Du, Nan Cao, Yu-Ru Lin, Panpan Xu, and Hanghang Tong. isphere: Focus+ context sphere visualization for interactive large graph exploration. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, pages 2916–2927, 2017.

[36] Fan Du, Nan Cao, Yu-Ru Lin, Panpan Xu, and Hanghang Tong. isphere: Focus+context sphere visualization for interactive large graph exploration. In Gloria Mark, Susan R. Fussell, Cliff Lampe, m. c. schraefel, Juan Pablo Hourcade, Caroline Appert, and Daniel Wigdor, editors, *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems, Denver, CO, USA, May 06-11, 2017*, pages 2916–2927. ACM, 2017.

[37] Alon Efrat, Yifan Hu, Stephen Kobourov, and Sergey Pupyrev. MapSets: Visualizing embedded and clustered graphs. *J. Graph Algorithms Appl.*, 19(2):571–593, 2015.

[38] Asi Elad Elbaz, Yosi Keller, and Ron Kimmel. Texture mapping via spherical multi-dimensional scaling. In Ron Kimmel, Nir A. Sochen, and Joachim Weickert, editors, *Scale Space and PDE Methods in Computer Vision*, volume 3459 of *Lecture Notes in Computer Science*, pages 443–455. Springer, 2005.

[39] John Ellson, Emden R. Gansner, Eleftherios Koutsofios, Stephen C. North, and Gordon Woodhull. Graphviz - open source graph drawing tools. In Petra Mutzel, Michael Jünger, and Sebastian Leipert, editors, *Graph Drawing, 9th International*

Symposium, volume 2265 of *Lecture Notes in Computer Science*, pages 483–484. Springer, 2001.

[40] David Eppstein. Limitations on realistic hyperbolic graph drawing. In Helen C. Purchase and Ignaz Rutter, editors, *Graph Drawing and Network Visualization - 29th International Symposium, GD 2021, Tübingen, Germany, September 14-17, 2021, Revised Selected Papers*, volume 12868 of *Lecture Notes in Computer Science*, pages 343–357. Springer, 2021.

[41] David Eppstein. Limitations on realistic hyperbolic graph drawing. In *Graph Drawing and Network Visualization - 29th International Symposium*, Lecture Notes in Computer Science. Springer, 2021.

[42] David Eppstein and Michael T. Goodrich. Succinct greedy graph drawing in the hyperbolic plane. *CoRR*, abs/0806.0341, 2008.

[43] David Eppstein and Michael T. Goodrich. Succinct greedy geometric routing using hyperbolic geometry. *IEEE Trans. Computers*, 60(11):1571–1580, 2011.

[44] Levent Ertoz, Michael Steinbach, and Vipin Kumar. 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*, volume 8, 2002.

[45] Maurits Cornelis Escher and Jan Willen Vermeulen. Escher on escher: Exploring the infinite. 1989.

[46] Mateus Espadoto, Rafael Messias Martins, Andreas Kerren, Nina S. T. Hirata, and Alexandru C. Telea. Toward a quantitative survey of dimension reduction techniques. *IEEE Trans. Vis. Comput. Graph.*, 27(3):2153–2173, 2021.

[47] George K. Francis and John M. Sullivan. Visualizing a sphere eversion. *IEEE Trans. Vis. Comput. Graph.*, 10(5):509–515, 2004.

[48] DF Frey and RA Pimentel. Principal component analysis and factor analysis. 1978.

[49] Cong Fu, Yonghui Zhang, Deng Cai, and Xiang Ren. Atsne: Efficient and robust visualization on gpu through hierarchical optimization. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 176–186, 2019.

[50] George W. Furnas. Generalized fisheye views. In Marilyn M. Mantei and Peter Orbeton, editors, *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, CHI 1886, Boston, Massachusetts, USA, April 13-17, 1986*, pages 16–23. ACM, 1986.

[51] Emden R. Gansner, Yifan Hu, and Stephen Kobourov. GMap: Visualizing graphs and clusters as maps. In *IEEE Pacific Visualization Symposium PacificVis*, pages 201–208. IEEE Computer Society, 2010.

[52] Emden R. Gansner, Yifan Hu, and Stephen C. North. A maxent-stress model for graph layout. *IEEE Trans. Vis. Comput. Graph.*, 19(6):927–940, 2013.

[53] Emden R Gansner, Yehuda Koren, and Stephen North. Graph drawing by stress majorization. In *International Symposium on Graph Drawing*, pages 239–250. Springer, 2004.

[54] Emden R. Gansner, Yehuda Koren, and Stephen C. North. Graph drawing by stress majorization. In János Pach, editor, *Graph Drawing, 12th International Symposium, GD 2004. Revised Selected Papers*, volume 3383 of *Lecture Notes in Computer Science*, pages 239–250. Springer, 2004.

[55] Helen Gibson, Joe Faith, and Paul Vickers. A survey of two-dimensional graph layout techniques for information visualisation. *Information visualization*, 12(3-4):324–357, 2013.

[56] Michael Glatzhofer. Hyperbolic tree of life. `https://hyperbolic-tree-of-life.github.io/`.

[57] Michael Glatzhofer. Hyperbolic browsing. Master's thesis, Institute of Interactive Systems and Data Science (ISDS),Graz University of Technology, Austria, 2018.

[58] Michael Glatzofer. d3-hypertree Github page. `https://github.com/glouwa/d3-hypertree`. accessed: 2021-06-06.

[59] Markus H Gross, Thomas C Sprenger, and J Finger. Visualizing information on a sphere. In *Proceedings of VIZ'97: Visualization Conference, Information Visualization Symposium and Parallel Rendering Symposium*, pages 11–16. IEEE, 1997.

[60] Anshul Guha and Eliot Feibush. Layout and display of network graphs on a sphere. In Nadia Magnenat-Thalmann, Jian Zhang, Jinman Kim, George Papagiannakis, Bin Sheng, Daniel Thalmann, and Marina L. Gavrilova, editors, *Advances in Computer Graphics - 39th Computer Graphics International Conference, CGI 2022, Virtual Event, September 12-16, 2022, Proceedings*, volume 13443 of *Lecture Notes in Computer Science*, pages 67–78. Springer, 2022.

[61] Mert Gürbüzbalaban, Asu Ozdaglar, and Pablo A Parrilo. Why random reshuffling beats stochastic gradient descent. *Mathematical Programming*, 186(1):49–84, 2021.

[62] Ingo Günther, 2007.

[63] Ming C. Hao, Meichun Hsu, Umeshwar Dayal, and Adrian Krug. Web-based visualization of large hierarchical graphs using invisible links in a hyperbolic space. In Hiroshi Arisawa and Tiziana Catarci, editors, *Advances in Visual Information Management, proceedings of the Fifth Working Conference on Visual Database Systems (VDB5), Fukuoka, Japan, May 10-12, 2000*, volume 168 of *IFIP Conference Proceedings*, pages 83–94. Kluwer, 2000.

[64] Percy John Heawood. Map color theorems. *Quant. J. Math.*, 24:332–338, 1890.

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

[66] Young Hyun. `https://www.caida.org/catalog/software/walrus/#H2540`, 2000. Accessed: 2021-06-06.

[67] Bharat Kale, Maoyuan Sun, and Michael E Papka. The state of the art in visualizing dynamic multivariate networks. In *Computer Graphics Forum*, volume 42, pages 471–490. Wiley Online Library, 2023.

[68] Tomihisa Kamada and Satoru Kawai. An algorithm for drawing general undirected graphs. *Inf. Process. Lett.*, 31(1):7–15, 1989.

[69] Ming-Hsuan Kang and Wu-Hsiung Lin. Equilateral spherical drawings of planar cayley graphs. *J. Graph Algorithms Appl.*, 25(1):97–119, 2021.

[70] Robert Kleinberg. Geographic routing using hyperbolic space. In *INFOCOM 2007. 26th IEEE International Conference on Computer Communications*, pages 1902–1909. IEEE, 2007.

[71] Donald Ervin Knuth. *The Stanford GraphBase: a platform for combinatorial computing*, volume 1. AcM Press New York, 1993.

[72] Piriziwè Kobina, Thierry Duval, and Laurent Brisson. The spherical retractable bubble space: An egocentric graph visualization throughout a retractable visualization space. *Inf.*, 14(10):531, 2023.

[73] Stephen Kobourov and Kevin Wampler. Non-Euclidean spring embedders. *IEEE Trans. Vis. Comput. Graph.*, 11(6):757–767, 2005.

[74] William L. Kocay, Daniel Neilson, and Ryan Szypowski. Drawing graphs on the torus. *Ars Comb.*, 59, 2001.

[75] Dmitri Krioukov, Fragkiskos Papadopoulos, Maksim Kitsak, Amin Vahdat, and Marián Boguná. Hyperbolic geometry of complex networks. *Physical Review E*, 82(3):036106, 2010.

[76] Johannes F. Kruiger, Paulo E. Rauber, Rafael Messias Martins, Andreas Kerren, Stephen G. Kobourov, and Alexandru C. Telea. Graph layouts by t-sne. *Comput. Graph. Forum*, 36(3):283–294, 2017.

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

[78] Myroslav Kryven, Alexander Ravsky, and Alexander Wolff. Drawing graphs on few circles and few spheres. *J. Graph Algorithms Appl.*, 23(2):371–391, 2019.

[79] Oh-Hyun Kwon, Chris Muelder, Kyungwon Lee, and Kwan-Liu Ma. Spherical layout and rendering methods for immersive graph visualization. In Shixia Liu, Gerik Scheuermann, and Shigeo Takahashi, editors, *2015 IEEE Pacific Visualization Symposium, PacificVis 2015, Hangzhou, China, April 14-17, 2015*, pages 63–67. IEEE Computer Society, 2015.

[80] Oh-Hyun Kwon, Chris Muelder, Kyungwon Lee, and Kwan-Liu Ma. A study of layout, rendering, and interaction methods for immersive graph visualization. *IEEE Trans. Vis. Comput. Graph.*, 22(7):1802–1815, 2016.

[81] Ho-Ching Lam and Ivo D Dinov. Hyperbolic wheel: A novel hyperbolic space graph viewer for hierarchical information content. *International Scholarly Research Notices*, 2012, 2012.

[82] Antoine Lambert, Romain Bourqui, and David Auber. 3d edge bundling for geographical data visualization. In *2010 14th International Conference Information Visualisation*, pages 329–335, 2010.

[83] John Lamping, Ramana Rao, and Peter Pirolli. A focus+context technique based on hyperbolic geometry for visualizing large hierarchies. In Irvin R. Katz, Robert L. Mack, Linn Marks, Mary Beth Rosson, and Jakob Nielsen, editors, *Human Factors in Computing Systems, CHI '95 Conference Proceedings*, pages 401–408. ACM/Addison-Wesley, 1995.

[84] Bongshin Lee, Catherine Plaisant, Cynthia Sims Parr, Jean-Daniel Fekete, and Nathalie Henry. Task taxonomy for graph visualization. In *Proceedings of the 2006 AVI Workshop on BEyond Time and Errors: Novel Evaluation Methods for Information Visualization*, BELIV '06, page 1–5, New York, NY, USA, 2006. Association for Computing Machinery.

[85] Collin M. McCarthy, Katherine E. Isaacs, Abhinav Bhatele, Peer-Timo Bremer, and Bernd Hamann. Visualizing the five-dimensional torus network of the IBM blue gene/q. In Peer-Timo Bremer, Bernd Mohr, Martin Schulz, and Valerio Pascucci, editors, *Proceedings of the First Workshop on Visual Performance Analysis, VPA '14, New Orleans, Louisiana, USA, November 16-21, 2014*, pages 24–27. IEEE, 2014.

[86] Fintan McGee, Mohammad Ghoniem, Guy Melançon, Benoît Otjacques, and Bruno Pinaud. The state of the art in multilayer network visualization. *Comput. Graph. Forum*, 38(6):125–149, 2019.

[87] Leland McInnes and John Healy. UMAP: uniform manifold approximation and projection for dimension reduction. *CoRR*, abs/1802.03426, 2018.

[88] Jacob Miller, Dhruv Bhatia, and Stephen Kobourov. State of the Art of Graph Visualization in non-Euclidean Spaces. In *EuroVis - STARs*. The Eurographics Association, 2024.

[89] Jacob Miller, Vahan Huroyan, and Stephen Kobourov. L2g: From local to global embeddings and back. In *under review at IEEE VIS*, 2023.

[90] Jacob Miller, Vahan Huroyan, and Stephen Kobourov. 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, pages 77–92. Springer, 2023.

[91] Jacob Miller, Stephen Kobourov, and Vahan Huroyan. Browser based hyperbolic visualization of graphs. In *IEEE Pacific Visualization Symposium PacificVis*, Lecture Notes in Computer Science. IEEE Computer Society, 2022.

[92] Jacob Miller, Stephen G. Kobourov, and Vahan Huroyan. Browser-based hyperbolic visualization of graphs. In *15th IEEE Pacific Visualization Symposium, PacificVis 2022, Tsukuba, Japan, April 11-14, 2022*, pages 71–80. IEEE, 2022.

[93] Bojan Mohar. Drawing graphs in the hyperbolic plane. In Jan Kratochvíl, editor, *Graph Drawing, 7th International Symposium, GD'99, Stirín Castle, Czech Republic, September 1999, Proceedings*, volume 1731 of *Lecture Notes in Computer Science*, pages 127–136. Springer, 1999.

[94] Tamara Munzner. H3: laying out large directed graphs in 3d hyperbolic space. In *IEEE Symposium on Information Visualization*, pages 2–10. IEEE Computer Society, 1997.

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

[96] Tamara Munzner. *Interactive visualization of large graphs and networks*. PhD thesis, Stanford University, 2000.

[97] Tamara Munzner and Paul Burchard. Visualizing the structure of the world wide web in 3d hyperbolic space. In David R. Nadeau and John L. Moreland, editors,

Procedings of the 1995 Symposium on Virtual Reality Modeling Language, pages 33–38. ACM, 1995.

[98] Quan Hoang Nguyen, Peter Eades, and Seok-Hee Hong. On the faithfulness of graph visualizations. In Sheelagh Carpendale, Wei Chen, and Seok-Hee Hong, editors, *IEEE Pacific Visualization Symposium, PacificVis 2013, February 27 2013-March 1, 2013, Sydney, NSW, Australia*, pages 209–216. IEEE Computer Society, 2013.

[99] Frank Nielsen and Richard Nock. Hyperbolic voronoi diagrams made easy. In Bernady O. Apduhan, Osvaldo Gervasi, Andrés Iglesias, David Taniar, and Marina L. Gavrilova, editors, *Prodeedings of the 2010 International Conference on Computational Science and Its Applications, ICCSA 2010*, pages 74–80. IEEE Computer Society, 2010.

[100] Carolina Nobre, Miriah D. Meyer, Marc Streit, and Alexander Lex. The state of the art in visualizing multivariate networks. *Comput. Graph. Forum*, 38(3):807–832, 2019.

[101] Serguei Norine. Drawing 4-pfaffian graphs on the torus. *Comb.*, 29(1):109–119, 2009.

[102] Mark Ortmann, Mirza Klimenta, and Ulrik Brandes. A sparse stress model. *J. Graph Algorithms Appl.*, 21(5):791–821, 2017.

[103] Veslava Osinska and Piotr Bala. Classification visualization across mapping on a sphere. In Aleksander Zgrzywa, Kazimierz Choros, and Andrzej Sieminski, editors, *New Trends in Multimedia and Network Information Systems*, volume 181 of *Frontiers in Artificial Intelligence and Applications*, pages 95–106. IOS Press, 2008.

[104] Peichang Ouyang, Dongsheng Cheng, Yanhua Cao, and Xiaogen Zhan. The visualization of hyperbolic patterns from invariant mapping method. *Comput. Graph.*, 36(2):92–100, 2012.

[105] Jiacheng Pan, Wei Chen, Xiaodong Zhao, Shuyue Zhou, Wei Zeng, Minfeng Zhu, Jian Chen, Siwei Fu, and Yingcai Wu. Exemplar-based layout fine-tuning for node-link diagrams. *IEEE Trans. Vis. Comput. Graph.*, 27(2):1655–1665, 2021.

[106] Wei Peng, Tuomas Varanka, Abdelrahman Mostafa, Henglin Shi, and Guoying Zhao. Hyperbolic deep neural networks: A survey. *IEEE Transactions on pattern analysis and machine intelligence*, 44(12):10023–10044, 2021.

[107] Mathew Penrose. *Random Geometric Graphs*. Oxford University Press, 05 2003.

[108] Scott Perry, Mason Sun Yin, Kathryn Gray, and Stephen Kobourov. Drawing graphs on the sphere. In Genny Tortora, Giuliana Vitiello, and Marco Winckler, editors, *AVI '20: International Conference on Advanced Visual Interfaces*, pages 17:1–17:9. ACM, 2020.

[109] Helen C. Purchase. Metrics for graph drawing aesthetics. *J. Vis. Lang. Comput.*, 13(5):501–516, 2002.

[110] Herbert Robbins and Sutton Monro. A stochastic approximation method. *The annals of mathematical statistics*, pages 400–407, 1951.

[111] Dario Rodighiero. Drawing network visualizations on a continuous, spherical surface. In Ebad Banissi, Farzad Khosrow-shahi, Anna Ursyn, Mark W. McK. Bannatyne, João Moura Pires, Nuno Datia, Kawa Nazemi, Boris Kovalerchuk, John

Counsell, Andrew Agapiou, Zora Vrcelj, Hing-Wah Chau, Mengbi Li, Gehan Nagy, Richard Laing, Rita Francese, Muhammad Sarfraz, Fatma Bouali, Gilles Venturini, Marjan Trutschl, Urska Cvek, Heimo Müller, Minoru Nakayama, Marco Temperini, Tania Di Mascio, Filippo Sciarrone, Veronica Rossano, Ralf Dörner, Loredana Caruccio, Autilia Vitiello, Weidong Huang, Michele Risi, Ugo Erra, Razvan Andonie, Muhammad Aurangzeb Ahmad, Ana Figueiras, Alfredo Cuzzocrea, and Mabule Samuel Mabakane, editors, *24th International Conference on Information Visualisation, IV 2020, Melbourne, Australia, September 7-11, 2020*, pages 573–580. IEEE, 2020.

[112] Boris A Rosenfeld. *A history of non-Euclidean geometry: Evolution of the concept of a geometric space*, volume 12. Springer Science & Business Media, 2012.

[113] Sebastian Ruder. An overview of gradient descent optimization algorithms. *CoRR*, abs/1609.04747, 2016.

[114] Kazumi Saito, Masahiro Kimura, Kouzou Ohara, and Hiroshi Motoda. Graph embedding on spheres and its application to visualization of information diffusion data. In Alain Mille, Fabien Gandon, Jacques Misselis, Michael Rabinovich, and Steffen Staab, editors, *Proceedings of the 21st World Wide Web Conference, WWW 2012, Lyon, France, April 16-20, 2012 (Companion Volume)*, pages 1137–1144. ACM, 2012.

[115] Bahador Saket, Carlos Scheidegger, Stephen Kobourov, and Katy Börner. Map-based visualizations increase recall accuracy of data. *Comput. Graph. Forum*, 34(3):441–450, 2015.

[116] Bahador Saket, Paolo Simonetto, Stephen Kobourov, and Katy Börner. Node, node-link, and node-link-group diagrams: An evaluation. *IEEE Trans. Vis. Comput. Graph.*, 20(12):2231–2240, 2014.

[117] Frederic Sala, Christopher De Sa, Albert Gu, and Christopher Ré. Representation tradeoffs for hyperbolic embeddings. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 4457–4466. PMLR, 2018.

[118] Archana P. Sangole and George K. Knopf. Visualization of randomly ordered numeric data sets using spherical self-organizing feature maps. *Comput. Graph.*, 27(6):963–976, 2003.

[119] Marcus Schaefer. The graph crossing number and its variants: A survey. *The electronic journal of combinatorics*, pages DS21–Apr, 2012.

[120] Doug Schaffer, Zhengping Zuo, Saul Greenberg, Lyn Bartram, John Dill, Shelli Dubs, and Mark Roseman. Navigating hierarchically clustered networks through fisheye and full-zoom methods. *ACM Trans. Comput. Hum. Interact.*, 3(2):162–188, 1996.

[121] Sarah Schöttler, Yalong Yang, Hanspeter Pfister, and Benjamin Bach. Visualizing and interacting with geospatial networks: A survey and design space. *Comput. Graph. Forum*, 40(6):5–33, 2021.

[122] Hans-Jorg Schulz. Treevis.net: A tree visualization reference. *IEEE Computer Graphics and Applications*, 31(6):11–15, 2011.

[123] David S. Shelley and Mehmet Hadi Gunes. Gerbilsphere: Inner sphere network visualization. *Comput. Networks*, 56(3):1016–1028, 2012.

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

[125] Ben Shneiderman, Cody Dunne, Puneet Sharma, and Ping Wang. Innovation trajectories for information visualizations: Comparing treemaps, cone trees, and hyperbolic trees. *Information Visualization*, 11(2):87–105, 2012.

[126] John P. Snyder. *Map projections: A working manual*. U.S. Government Printing Office, 1987.

[127] Roberto Tamassia. *Handbook of graph drawing and visualization*. CRC press, 2013.

[128] Lucas Theisen, Aamer Shah, and Felix Wolf. Down to earth: how to visualize traffic on high-dimensional torus networks. In Peer-Timo Bremer, Bernd Mohr, Martin Schulz, and Valerio Pascucci, editors, *Proceedings of the First Workshop on Visual Performance Analysis, VPA '14, New Orleans, Louisiana, USA, November 16-21, 2014*, pages 17–23. IEEE, 2014.

[129] Satoshi Togawa, Kazuhide Kanenishi, and Yoneo Yano. Peer-to-peer file sharing communication detection using spherical SOM visualization for network management. In Michael J. Smith and Gavriel Salvendy, editors, *Human Interface and the Management of Information. Interacting with Information - Symposium on Human Interface 2011, Held as Part of HCI International 2011, Orlando, FL, USA, July 9-14, 2011, Proceedings, Part I*, volume 6771 of *Lecture Notes in Computer Science*, pages 259–267. Springer, 2011.

[130] Christian Tominski, Stefan Gladisch, Ulrike Kister, Raimund Dachselt, and Heidrun Schumann. A Survey on Interactive Lenses in Visualization. In *EuroVis - STARs*. The Eurographics Association, 2014.

[131] Christian Tominski, Stefan Gladisch, Ulrike Kister, Raimund Dachselt, and Heidrun Schumann. Interactive lenses for visualization: An extended survey. In *Computer Graphics Forum*, volume 36(6), pages 173–200. Wiley Online Library, 2017.

[132] Laurens Van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(11), 2008.

[133] Corinna Vehlow, Fabian Beck, and Daniel Weiskopf. Visualizing group structures in graphs: A survey. *Comput. Graph. Forum*, 36(6):201–225, 2017.

[134] Kevin Verbeek and Subhash Suri. Metric embedding, hyperbolic space, and social networks. *Comput. Geom.*, 59:1–12, 2016.

[135] Tatiana Von Landesberger, Arjan Kuijper, Tobias Schreck, Jörn Kohlhammer, Jarke J van Wijk, J-D Fekete, and Dieter W Fellner. Visual analysis of large graphs: state-of-the-art and future research challenges. In *Computer graphics forum*, volume 30, pages 1719–1749. Wiley Online Library, 2011.

[136] Jörg Walter, Jörg Ontrup, Daniel Wessling, and Helge Ritter. Interactive visualization and navigation in large data collections using the hyperbolic space. In *Third IEEE International Conference on Data Mining*, pages 355–362. IEEE, 2003.

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

[138] Jörg A. Walter and Helge J. Ritter. 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*, pages 123–132. ACM, 2002.

[139] Chaoli Wang and Jun Tao. Graphs in scientific visualization: A survey. *Comput. Graph. Forum*, 36(1):263–287, 2017.

[140] Yunhai Wang, Yanyan Wang, Haifeng Zhang, Yinqi Sun, Chi-Wing Fu, Michael Sedlmair, Baoquan Chen, and Oliver Deussen. Structure-aware fisheye views for efficient large graph exploration. *IEEE Trans. Vis. Comput. Graph.*, 25(1):566–575, 2019.

[141] Colin Ware and Robert J. Bobrow. Supporting visual queries on medium-sized node-link diagrams. *Inf. Vis.*, 4(1):49–58, 2005.

[142] Martin Wattenberg, Fernanda Viégas, and Ian Johnson. How to use t-sne effectively. *Distill*, 2016.

[143] Bang Wong. Points of view: Color blindness. *Nature methods*, 8:441, 06 2011.

[144] Yingxin Wu and Masahiro Takatsuka. Visualizing multivariate network on the surface of a sphere. In *Proceedings of the 2006 Asia-Pacific Symposium on Information Visualisation-Volume 60*, pages 77–83, 2006.

[145] Yalong Yang, Bernhard Jenny, Tim Dwyer, Kim Marriott, Haohui Chen, and Maxime Cordeil. Maps and globes in virtual reality. *Comput. Graph. Forum*, 37(3):427–438, 2018.

[146] Songxiao Zhang and Adam Kelleher. H3py Github Page. `https://github.com/buzzfeed/pyh3`, 2016. Accessed: 2021-06-06.

[147] Jonathan X. Zheng, Samraat Pawar, and Dan F. M. Goodman. Graph drawing by stochastic gradient descent. *IEEE Trans. Vis. Comput. Graph.*, 25(9):2738–2748, 2019.

[148] Yuansheng Zhou and Tatyana O Sharpee. Hyperbolic geometry of gene expression. *Iscience*, 24(3):102225, 2021.

[149] Minfeng Zhu, Wei Chen, Yuanzhe Hu, Yuxuan Hou, Liangjun Liu, and Kaiyuan Zhang. Drgraph: An efficient graph layout algorithm for large-scale graphs by dimensionality reduction. *IEEE Trans. Vis. Comput. Graph.*, 27(2):1666–1676, 2021.