Browser-based Hyperbolic Visualization of Graphs

Jacob Miller* University of Arizona
Stephen Kobourov† University of Arizona
Vahan Huroyan‡ University of Arizona

Figure 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).***

Abstract

Hyperbolic geometry offers a natural 'focus+context' for data visualization and has been shown to underlie real-world complex networks. However, current hyperbolic network visualization approaches are limited to special types of networks and do not scale to large datasets. With this in mind, we designed, implemented, and analyzed three methods for hyperbolic visualization of networks in the browser based on inverse projections, generalized force-directed algorithms, and hyperbolic multi-dimensional scaling (H-MDS). A comparison with Euclidean MDS shows that H-MDS produces embeddings with lower distortion for several types of networks. All three methods can handle node-link representations and are available in fully functional web-based systems.

Index Terms: Graph drawing; Hyperbolic geometry; Non-Euclidean embedding; Stochastic gradient descent

1 Introduction

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 [35] 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 [25], there are no tools that support browser-based hyperbolic visualization of general graphs.

We describe three methods for laying out graphs in the 2-dimensional hyperbolic space, H^2. The first method relies on taking a pre-computed Euclidean layout of a graph and projecting it into hyperbolic space, providing standard map interactions, such as pan, zoom, re-center, click and drag. We implement this method in a web based system that provides several layout algorithms for node-link and map-based visualization. This allows us to view and interact with GMaps, MapSets, BubbleSets, and LineSets in hyperbolic space. The second method makes use of a generalization of force-directed algorithms to Riemannian geometries [34]. 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) [16, 61]. 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 [64]. 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.

2 Related Work

The graph layout problem typically involves placing nodes and routing edges in 2-dimensional Euclidean space. Force-directed algorithms model the system as a set of springs and attempt to balance the forces on nodes. Both their conceptual simplicity and their generally aesthetically pleasing results have made this class of algorithms particularly useful for computing graph layouts [33]. Force-directed algorithms have been generalized to Riemannian geometries, (e.g., spherical and hyperbolic) by computing tangent planes at each node [34].

To the best of our knowledge, there are no browser-based tools for visualizing general graphs in hyperbolic space. Table 1 gives an

*e-mail: jacobmiller1@email.arizona.edu
†e-mail: kobourov@cs.arizona.edu
‡e-mail: vahanhuroyan@math.arizona.edu

2022 IEEE 15th Pacific Visualization Symposium (PacificVis) | 978-1-6654-2335-9/22/$31.00 ©2022 IEEE | DOI: 10.1109/PACIFICVIS53943.2022.00016

2165-8773/22/$31.00 ©2022 IEEE
DOI 10.1109/PacificVis53943.2022.00016

Authorized licensed use limited to: The University of Arizona. Downloaded on April 02,2024 at 04:20:58 UTC from IEEE Xplore. Restrictions apply.

overview of previous work in hyperbolic network visualization. One of the earliest approaches by Lamping *et al.* [38] embeds hierarchies into the hyperbolic plane by recursively placing each node's children evenly spaced around the arc of a circle. This is possible thanks to the exponential expansion intrinsic to the geometry. They make use of the Poincaré projection to display the graph on the computer monitor, which also provides the now well known 'focus+context' effect. Navigating the hierarchy is done by re-centering the projection at a new node in the hyperbolic plane. The embedding can be computed in linear time and arbitrary graphs can also be visualized using this approach by utilizing a spanning tree of the graph and 'filling in' the rest of the edges later.

Table 1: ***Hyperbolic browsing systems***

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

A bioinformatics-motivated java application by Bingham and Sudarsanam [6] uses a similar approach to visualize phylogenetic trees. Andrews *et al.* [2] also rely on Lamping *et al.*'s work in their Hierarchy Visualization System as do Baumgartner and Waugh [4] who visualize Roget's thesaurus. The Java InfoVis Toolkit also implements a hyperbolic hierarchy browser [5] and TreeBolic implements the hyperbolic tree layout [9]. More recently, Glatzhofer developed a hyperbolic hierarchy browser utilizing d3.js, a javascript graphics library which works in the browser, and can display large hierarchies smoothly with different layout algorithms [24–26].

While most prior work considers the 2-dimensional hyperbolic plane, Munzner has also used 3D hyperbolic space to visualize hierarchies with the help of the Beltrami-Klein projection [40–43]. Here geodesics are mapped to straight lines rather than the circular arcs of the Poincaré projection. Munzner's work has been re-implemented in two subsequent systems: Walrus [29] and h3py [63].

Hyperbolic space has been explored in the context of non-linear dimensionality reduction, specifically multi-dimensional scaling (MDS). The idea is to match pairwise similarities with distances in an embedding: the more similar two elements are, the closer they are in the embedding. The Euclidean distance is traditionally used as a closeness metric. Computing a graph layout can be interpreted as an MDS problem by treating the graph theoretic distance between pairs of nodes as their pairwise distance metric.

There are three different types of MDS: classical, metric, and non-metric (although these labels are not used consistently in different fields) Here, we refer to Torgerson's MDS as classical MDS, where the input distances are converted to similarities, and principal component analysis (PCA) is used to obtain the embedding [57]. Metric MDS minimizes a loss function, commonly known as *stress* [53]. Finally, in non-metric MDS the input distances are not necessarily distances, but can be ranks [36].

Classical MDS has been explored in hyperbolic space, by replacing the conversion to similarities with an appropriate hyperbolic scaling function [14,52]. Using a similar idea, metric and non-metric MDS have been generalized to hyperbolic space by incorporating hyperbolic geodesic distance into the cost function [59–61, 65].

It has been shown that some graphs can be embedded with lower error in hyperbolic space than in Euclidean space [7]. Zhou and Sharpee [65] show that hyperbolic MDS (H-MDS) can be used to detect the underlying geometry of a dataset, when comparing its

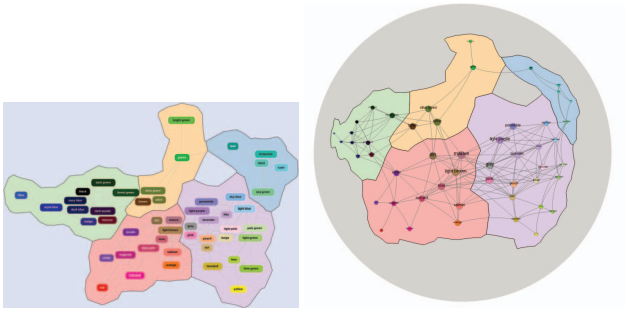

Figure 2: ***An example of a GMap Euclidean layout (left) and its hyperbolic realization (right) via inverse projection.***

embedding error to Euclidean non-metric MDS. They go further to show that the underlying space of genomes is hyperbolic. Krioukov *et al.*'s [35] work indicates that hyperbolic geometry may underlie complex networks and hierarchical networks, such as phylogenetic trees and the internet.

Greedy embeddings also appear to have a close relationship with hyperbolic geometry. Indeed, any connected, finite graph admits a greedy embedding in hyperbolic space, which is not generally true in Euclidean geometry [31]. 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 [22].

An open-source hyperbolic visualization tool, RogueViz [11], includes different projections and educational tools, although its restriction to tessellations of the hyperbolic plane makes it less than ideal for general graphs. Self-organizing maps have been generalized to hyperbolic space, but are restricted to lattices [45].

Stress-based approaches have been explored in other Riemannian spaces such as the sphere [18, 46] and the torus [12, 13], as different spaces provide different visualization advantages. Unlike in the plane, on the sphere one can avoid issues such as central/peripheral placement, and on the torus larger classes of graphs can be drawn without crossings. Human subject studies show that these spaces are no worse than Euclidean space for common navigation tasks [13, 18].

One can achieve a similar focus+context effect by using lens effects [55,56]. In particular, at first glance the Poincaré disk appears to resemble a fisheye lens. However, a lens effect generally applies only to a subset of the visible data, scaling or warping it to bring it into focus. The focus+context view is applied across all of the data in the Poincaré disk, with an exponential decrease in data size away from the center, but the entirety of the object remaining in view.

It is worth mentioning that there are also 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 [21, 58]. 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.

Zheng *et al.* show that stochastic gradient descent (SGD) can be used effectively to solve MDS for graph layout in Euclidean space [64]. In this paper we show that SGD can also be deployed in hyperbolic space and produces good embeddings.

3 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

Authorized licensed use limited to: The University of Arizona. Downloaded on April 02,2024 at 04:20:58 UTC from IEEE Xplore. Restrictions apply.

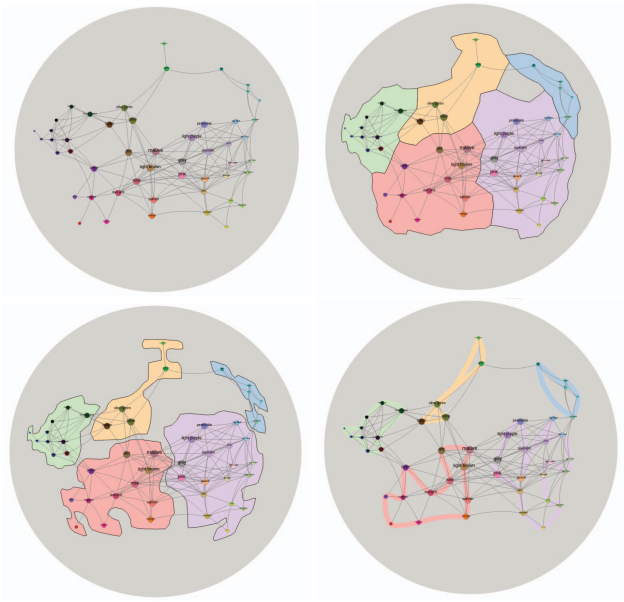

Figure 3: ***Different GMap drawing options for the same graph using inverse projection from Euclidean to hyperbolic space.***

graph visualization system. The system offers several layout algorithms, clustering algorithms and visualization styles, with a focus on map-like representations such as GMaps [23], MapSets [19], BubbleSets [15], and LineSets [1]. Several human-subject studies suggest that such map-like visualizations are at least as good as traditional node-link diagrams when it comes to task performance, memorization, and recall of the data [49,51].

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 [20]. 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 [46].

The system relies on two different layout algorithms for computing a Euclidean layout: *sdfp* is a multi-level force-directed algorithm [28] and *neato* is a implementation of the Kamada-Kawai algorithm [30]. We show examples of the *colors graph* drawn as a node-link, GMap, BubbleSet, and LineSet diagram; see Fig. 3. 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 [3]. 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 [46].

3.1 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).

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

3.1.1 Hyperbolic Projections:

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

Knowing that a perfect embedding in the plane is impossible, useful maps can still be created by choosing which information to preserve. Three well-known projections of the sphere are gnomonic, orthographic, and stereographic projections. The gnomonic projection preserves straight lines; geodesics of the sphere are shown as straight lines in the projection. This is particularly useful in flight planning, and is said to be the oldest map projection. The orthographic projection resembles the view of the Earth from space, and preserves scale at the center of the projection, making it useful in visualization. Finally, the stereographic projection preserves angles and has its roots in star charts used in sailing [54].

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.

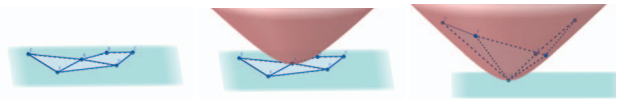

Figure 4: ***Illustration of an inverse projection: wrapping a plane drawing on a hyperboloid.***

3.1.2 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

Authorized licensed use limited to: The University of Arizona. Downloaded on April 02,2024 at 04:20:58 UTC from IEEE Xplore. Restrictions apply.

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. 4.

Figure 5: ***An example of the default (center), increased zoom (left), and increased coverage (right) for the same graph.***

3.1.3 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 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.

3.2 Visualization Considerations

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

3.2.1 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. 6, but interacting with the actual visualization in GMap better conveys the idea. Fig. 7 additionally shows the difference between panning to the edge of a map in Euclidean space and hyperbolic space.

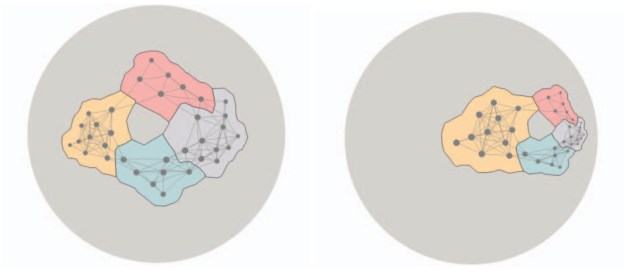

Figure 6: ***The same graph centered about two different origins.***

3.2.2 Parameters

There are a handful of settings in the parameter space, all of which can be hand-tuned within given bounds.

Edge opacity: controls the opacity of links. This is useful as the number of edges near the border in large graph can be so large that the layout is difficult to discern. An edge *opacity slider* helps mitigate this problem. The slider adjusts edge opacity from 0 (fully transparent) to 1 (fully opaque), which is the default setting.

Label size: controls the relative size of labels in the view. The natural focus+context view of a graph in hyperbolic space is also reflected in the node label sizes, which decrease when farther away from the center. This process is automated by adjusting the font size of the labels in an inversely proportional fashion with respect to their Euclidean distance to the center of the disk. A *label size slider* allows the (central) font size to be adjusted from 0 (no labels) up to 40 px Arial. The initial setting is 15px, which ensures readability.

Zoom: controls the size of the Poincaré disk in the browser window. Intuitively, the zoom slider brings the disk closer or further away from the point of perspective; see Fig 5. The slider begins at 100% of the browser window size and the slider scales from 50% to 150% of the window size.

Coverage: controls the total area the layout occupies in the Poincaré disk. As a consequence of Euclid's parallel postulate not holding in hyperbolic space, the hyperbolic plane is not invariant to scale [52]. However, we can re-scale the layout while it is still in the Euclidean plane, before we project it to H^2. By default we use an initial scaling factor proportional to the diameter (longest shortest path) of the graph. The scale can be adjusted from 50% to 150% of the default layout size.

3.2.3 Tasks Considerations

Brehmer and Munzner [10] define an abstract task taxonomy based on a large body of related research. For domain specific tasks, Lee *et al.* [39] identify task abstractions for network data and Saket *et al.* [50,51] identify task abstractions for grouped node-link diagrams (e.g., map-like drawings).

At a high level, we provide support for discover-type tasks by navigation. Node-link diagrams make adjacency easily apparent, and map-like drawings provide for straight-forward cluster identification. Panning aids in search tasks, such as location or exploration. A node/cluster can also be selected by double clicking, which smoothly recenters the layout around the selected node/cluster. Given the infinite space and 'focus+context' nature of the Poincaré projection, it is possible that a viewer may get lost. To alleviate this potential problem, we provide a *reset button*, that restores the layout to the original output of the underlying algorithm.

4 Force-directed Method

Our projection-based hyperbolic visualization method uses a pre-computed 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

Authorized licensed use limited to: The University of Arizona. Downloaded on April 02,2024 at 04:20:58 UTC from IEEE Xplore. Restrictions apply.

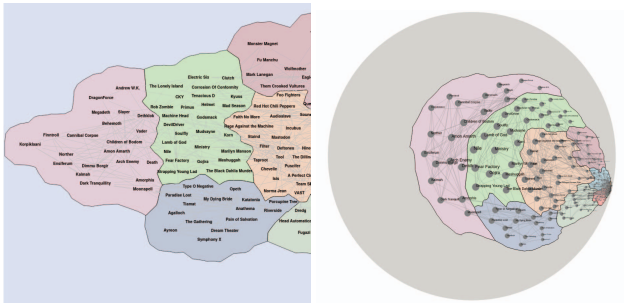

Figure 7: ***A 2D Euclidean GMap instance of the MusicLand graph (left) and its hyperbolic realization (right).***

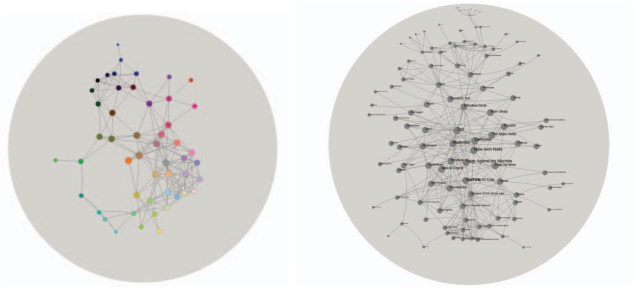

Figure 8: ***Force-directed colors (left) and MusicLand (right) graphs.***

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 [34]. The implementation, which provides visualization in the browser, and is made available in a browser based system through GitHub.

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

For instance, on the sphere, layout methods that correctly make use of the geometry allow 3D polytopes to wrap 'around' the sphere. Thus, compared to the plane, a more accurate realization of their structure is possible; see method two of [46].

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 [30]. 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.

4.1 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 [34]. 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

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

4.2 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.

4.3 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

Authorized licensed use limited to: The University of Arizona. Downloaded on April 02,2024 at 04:20:58 UTC from IEEE Xplore. Restrictions apply.

been shown that Voronoi diagrams for 2-dimensional points generalize to hyperbolic space and can be computed in $O(n \log n)$ time [44]. Similarly, LineSets requires Bezier curves between nodes in a cluster, which should also be computable in hyperbolic space.

5 Multidimensional Scaling in H^2

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

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

$$Stress = \sum_{i<j} w_{ij}(\|X_i - X_j\| - d_{ij})^2 \quad (1)$$

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 [60]. 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.

5.1 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.

5.1.1 Randomization

Computing the stress function for a given embedding requires $O(|V|^2)$ time by definition; see (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 [47]. 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 [27] and has been shown to work well for Euclidean SGD [64]. A third method known as index shuffling randomizes the indices in place, and pairs are chosen from this new ordering.

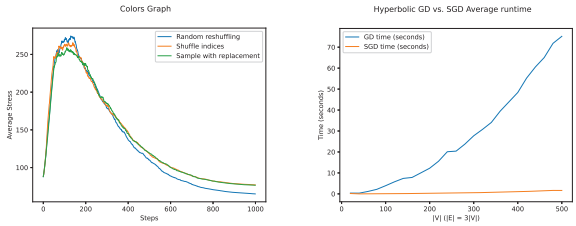

Figure 9: ***Effect of randomization techniques (left) described in 5.1.1, showing average stress over 30 runs at each step on the Colors graph and average runtime (right) of HMDS using GD and SGD over 30 runs (pre-processing omitted). Similar results were found on other graphs.***

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. 9. We use random reshuffling and define an *iteration* as a full pass through all pairs and show, in Section 5.1.4, that we only need a constant number of iterations.

5.1.2 Initialization

As the stress function (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 [8]. 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 [17]) and compared our smart initialization to random initialization.

Knowing the Euclidean algorithm is good at escaping local minima, we run Euclidean SGD on the graph for 5 iterations, then project this layout into hyperbolic space to obtain our smart initialization. Random initialization is obtained by placing each node uniformly at random in a circle of hyperbolic radius 1. While we initially saw small improvements, there was no statistically significant benefit of smart initialization over using a random initialization, confirming the results of [8].

5.1.3 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 [8, 48]. We upper bound the product $\eta w_{ij} \leq 1$ as in [64]. This allows us to use a larger initial rate to 'jump' out of bad neighborhoods and possible local

Authorized licensed use limited to: The University of Arizona. Downloaded on April 02,2024 at 04:20:58 UTC from IEEE Xplore. Restrictions apply.

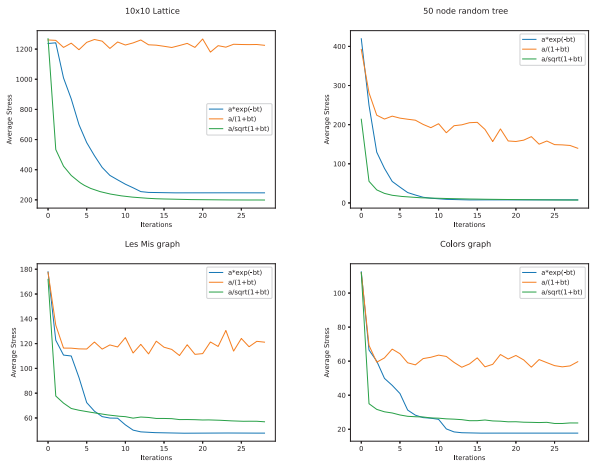

Figure 10: ***Effect of learning rate on various classes of graphs (average over 15 runs each). Graphs used are a 10x10 grid (top left), 50 node random trees (top right), the Les Mis graph [32] (bottom left), and the colors graph (bottom right).***

optima, but still converge as η goes to 0. We set a maximum and a minimum learning rate, a function $s(t)$ that outputs a learning rate η at time step t_i. $s(t_0) = \eta_{max} = d_{max}^2$ and $s(t_{max} = \eta_{min} = \varepsilon d_{min}^2$ where d_{max} and d_{min} correspond to the longest and shortest shortest paths of the input graph, respectively.

Euclidean SGD works particularly well with an exponential decay learning rate [64]. 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 [64] using $\eta_{max}e^{-bt}$, the traditional $\Theta(1/t)$ as $\frac{a}{1+bt}$ and the $\Theta(1/\sqrt{t})$ schedule as $\frac{a}{\sqrt{1+bt}}$. We set $a = d_{min}^2$ and $b = -(t_{max})log\frac{\eta_{min}}{\eta_{max}}$.

As expected, the $\Theta(1/t)$ schedule struggles to step out of local minima. It is somewhat surprising that the $\Theta(1/\sqrt{t})$ schedule appears to achieve lower minima for some classes of graphs; see Fig. 10. This could be due to the function's larger learning rates allowing the system to avoid local minima.

5.1.4 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. 12

5.2 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.

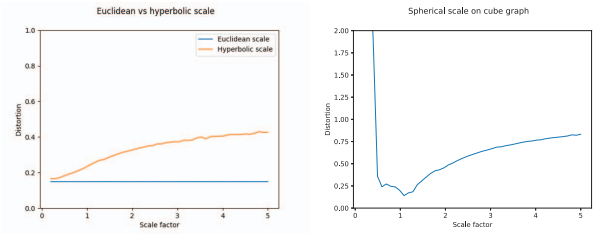

Figure 11: ***Left: Distortion on triangular lattice graph shown in Fig. 14. Hyperbolic space gets worse as the scale increases, but Euclidean can embed the graph with constant error. Right: The effect of scale on the sphere on a cube graph. For this example, there is a noticeable optimum at around $\pi/3$ (note that the diameter of a cube graph is 3).***

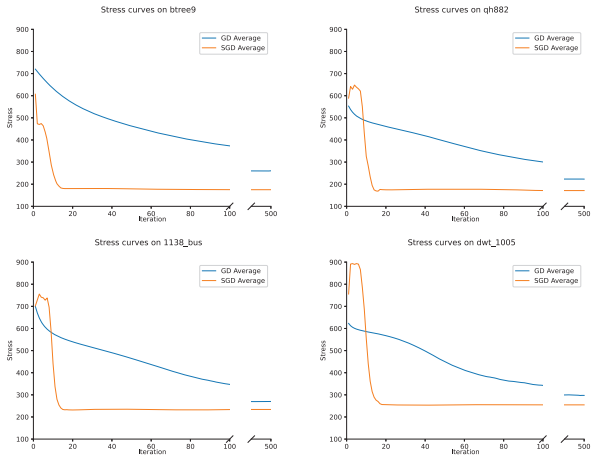

Figure 12: ***Average stress plots of GD and SGD. Initial stress values are omitted.***

As mentioned in section 5.1.4, while the overall complexity of SGD is no different than GD, the run time is significantly faster; see Fig 9. 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. 12. We show a selection of 4 graphs from the sparse matrix collection [17] 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.

5.2.1 Comparison across geometries

The stress function (Eq. 1) offers a natural evaluation for the quality of a graph embedding, based on how well pairwise distances in the embedding match the corresponding graph distances. While the quadratic term in the stress equation makes it suitable for gradient descent, it also makes it somewhat disingenuous to compare across different graphs and different embedding spaces. With this in mind, we measure how well a graph embedding captures the underlying graph structure using the related distortion measure; see [52].

$$Distortion = \frac{1}{\binom{|V|}{2}} \sum_{i,j} \frac{\left| \|X_i - X_j\| - d_{ij} \right|}{d_{ij}} \qquad (2)$$

Authorized licensed use limited to: The University of Arizona. Downloaded on April 02,2024 at 04:20:58 UTC from IEEE Xplore. Restrictions apply.

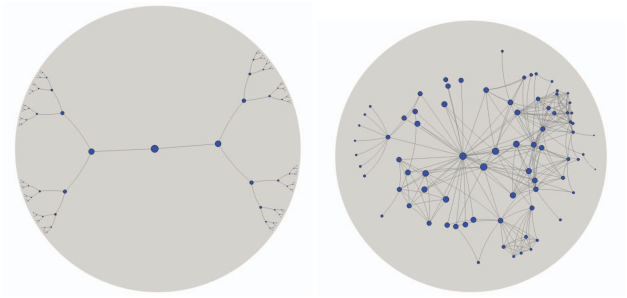

Figure 13: ***A full binary tree (left) and the Les Mis [32] graph (right), an example of a small complex (social) network.***

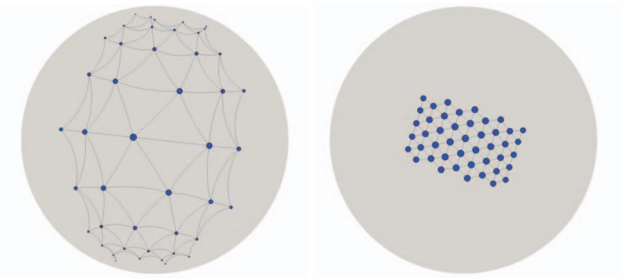

Figure 14: ***Triangular lattice with scaling factor $\alpha = 1$ (left) and optimized $\alpha = 0.22$ (right).***

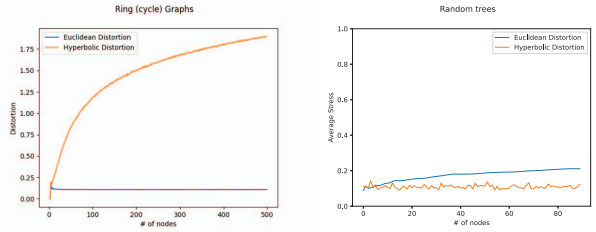

Figure 15: ***Euclidean and hyperbolic embedding distortion on rings (left) and trees (right). It can be seen that the number of nodes in a ring in Euclidean space does not matter, but distortion gets worse with size of the ring in hyperbolic space. The inverse is true for trees, they can be embedded with constant distortion in hyperbolic space but not Euclidean.***

Similar to stress, perfectly capturing all pairwise distances will result in distortion of 0. It has been shown that some classes of graphs, such as trees, can be embedded in the hyperbolic plane with lower distortion value than in Euclidean space [7, 35]. We numerically demonstrate that the SGD algorithm for H-MDS achieves lower distortion values for trees. It has also been shown that some classes of graphs, such as cycles, can be embedded in the Euclidean plane with constant distortion but cannot be embedded with constant distortion in the hyperbolic space. We demonstrate both of these properties for trees and cycles; see Fig. 15.

The ability to compare graph embeddings in various spaces (Euclidean, Spherical and Hyperbolic) creates an interesting application of MDS. Similar to how Zhou and Sharpee detect the geometry of a dataset [65], we can determine which of the three consistent geometries is best suited for a given graph, by performing Euclidean MDS, Spherical-MDS and H-MDS, and comparing their corresponding distortion values. Table 2 shows the distortion values for several classes of graphs for Euclidean space, Spherical space and Hyperbolic space. The cube graph (and other graphs that correspond to 3D platonic solids) embeds best in spherical space. Lattices (as well as paths and cycles) embed best in Euclidean space. Trees (and other hierarchies) embed best in hyperbolic space.

5.2.2 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. 14. 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.

Table 2: ***Distortion on small graphs across geometries. Averaged over 10 runs.***

Graph	Spherical	Euclidean	Hyperbolic
Cube	0.1296	0.2437	0.2645
Lattice	0.2421	0.1486	0.2306
Tree	0.1944	0.1284	0.0682

6 Discussion, Limitations and Future Work

We described three methods for visualizing graphs in hyperbolic space, which are illustrated in Fig. 1. We present a small-scale

Authorized licensed use limited to: The University of Arizona. Downloaded on April 02,2024 at 04:20:58 UTC from IEEE Xplore. Restrictions apply.

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

comparison of the three approaches by comparing time and distortion values; see Fig. 16. 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. 7, 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 [18,21], but it is also known that some graphs can be embedded in hyperbolic space with lower error [35] Determining whether a lower distortion in a geometry corresponds to better task support remains a promising direction for future work.

As discussed in Section 5.2.2, 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 [37].

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

References

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

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

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

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

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

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

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

[8] K. Börsig, U. Brandes, and B. Pásztor. Stochastic gradient descent works really well for stress minimization. In *Graph Drawing and Network Visualization - 28th International Symposium*, vol. 12590, pp. 18–25. Springer, 2020.

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

[10] M. Brehmer and T. Munzner. A multi-level typology of abstract visualization tasks. *IEEE Trans. Vis. Comput. Graph.*, 19(12):2376–2385, 2013.

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

[12] K. Chen, T. Dwyer, B. Bach, and K. 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.

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

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

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

[16] A. Cvetkovski and M. Crovella. Multidimensional scaling in the poincare disk, 2016.

Authorized licensed use limited to: The University of Arizona. Downloaded on April 02,2024 at 04:20:58 UTC from IEEE Xplore. Restrictions apply.

[17] T. A. Davis and Y. Hu. The university of florida sparse matrix collection. *ACM Trans. Math. Softw.*, 38(1):1:1–1:25, 2011.

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

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

[20] J. Ellson, E. R. Gansner, E. Koutsofios, S. C. North, and G. Woodhull. Graphviz - open source graph drawing tools. In *Graph Drawing, 9th International Symposium*, vol. 2265 of *Lecture Notes in Computer Science*, pp. 483–484. Springer, 2001.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

[39] B. Lee, C. Plaisant, C. S. Parr, J.-D. Fekete, and N. Henry. Task taxonomy for graph visualization. In *Proceedings of the 2006 AVI workshop on BEyond time and errors: novel evaluation methods for information visualization*, pp. 1–5, 2006.

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

[41] T. Munzner. Exploring large graphs in 3d hyperbolic space. *IEEE Computer Graphics and Applications*, 18(4):18–23, 1998. doi: 10.1109/38.689657

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

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

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

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

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

[47] H. Robbins and S. Monro. A stochastic approximation method. *The annals of mathematical statistics*, pp. 400–407, 1951.

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

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

[50] B. Saket, P. Simonetto, and S. Kobourov. Group-level graph visualization taxonomy. In *16th Eurographics Conference on Visualization, EuroVis 2014*. Eurographics Association, 2014.

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

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

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

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

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

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

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

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

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

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

[61] J. A. Walter and H. 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*, pp. 123–132. ACM, 2002.

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

[63] S. Zhang and A. Kelleher. H3py Github Page. https://github.com/buzzfeed/pyh3, 2016. Accessed: 2021-06-06.

[64] J. X. Zheng, S. Pawar, and D. F. M. Goodman. Graph drawing by stochastic gradient descent. *IEEE Trans. Vis. Comput. Graph.*, 25(9):2738–2748, 2019.

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

Authorized licensed use limited to: The University of Arizona. Downloaded on April 02,2024 at 04:20:58 UTC from IEEE Xplore. Restrictions apply.