Force-Directed Embedding of Scale-Free Networks in the Hyperbolic Plane

Thomas Bläsius
Karlsruhe Institute of Technology, Germany

Tobias Friedrich
Hasso Plattner Institute, University of Potsdam, Germany

Maximilian Katzmann
Hasso Plattner Institute, University of Potsdam, Germany

Abstract

Force-directed drawing algorithms are the most commonly used approach to visualize networks. While they are usually very robust, the performance of Euclidean spring embedders decreases if the graph exhibits the high level of heterogeneity that typically occurs in scale-free real-world networks. As heterogeneity naturally emerges from hyperbolic geometry (in fact, scale-free networks are often perceived to have an underlying hyperbolic geometry), it is natural to embed them into the hyperbolic plane instead. Previous techniques that produce hyperbolic embeddings usually make assumptions about the given network, which (if not met) impairs the quality of the embedding. It is still an open problem to adapt force-directed embedding algorithms to make use of the heterogeneity of the hyperbolic plane, while also preserving their robustness.

We identify fundamental differences between the behavior of spring embedders in Euclidean and hyperbolic space, and adapt the technique to take advantage of the heterogeneity of the hyperbolic plane.

2012 ACM Subject Classification Theory of computation → Random projections and metric embeddings

Keywords and phrases force-directed drawing algorithms, spring embedding, hyperbolic space

Digital Object Identifier 10.4230/LIPIcs.SEA.2021.22

Supplementary Material *Software (Source Code)*: `https://github.com/maxkatzmann/hyperbolic-spring-embedder.git`
archived at `swh:1:dir:7b9445f64fae3be4bbe3a692c2f94ded0bc600d1`

Funding This research was funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – Grant No. 390859508.

1 Introduction

Network science is an increasingly popular field that ties in with many different research areas such as biology or social science, where researchers examine real-world networks in order to explain observed phenomena. While the goal is typically a mathematical analysis of these graphs, more often than not the first step to understanding the structure of a network, is to gain an intuition by *looking* at it, using a suitable visualization. The most natural way to visualize a graph is to draw its vertices as points and edges as lines between them. In such a drawing, it is typically desirable to have short edges while non-adjacent vertices should be farther apart. On the one hand, this reduces visual clutter. On the other hand, it preserves the typical interpretation of edges as a representation of similarity.

In Euclidean space, the approach that is most commonly used to embed graphs in such a way are *spring embedders* or *force-directed drawing algorithms* [11]. Starting with a random position for each vertex, they simulate physical forces between vertices. *Attractive forces* pull adjacent vertices together and *repulsive forces* push non-adjacent vertices apart.

19th International Symposium on Experimental Algorithms (SEA 2021).
Editors: David Coudert and Emanuele Natale; Article No. 22; pp. 22:1–22:18
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Due to their basic nature, spring embedders can be applied to all types of graphs. However, they typically struggle if a network contains high-degree vertices that tie together otherwise loosely connected parts of the graph. In the Euclidean plane, there is not enough space close to the high-degree vertices, to make all their edges short while keeping non-connected parts away from each other, often leading to a visualization that resembles a ball of wool. This can be resolved by embedding a network in the hyperbolic plane instead. There, space expands exponentially, i.e., the area and circumference of a disk grows exponentially with its radius. This makes it possible to have many vertices with pairwise large distance being close to a single high-degree vertex.

In fact, a heterogeneous degree distribution (few vertices of high degree and many low-degree vertices) emerges naturally from a hyperbolic geometry which is therefore perceived to be underlying these so-called *scale-free* graphs [3, 13]. In particular, choosing an origin in the hyperbolic plane, one can imagine a vertex's distance to that origin (the vertex's *radius*) to be a measure of popularity: high-degree vertices are placed near the origin and low-degree vertices are farther away. Additionally, the angular distance (around the origin) between two vertices measures their similarity: the *angular coordinates* of adjacent vertices are close. If we now distribute the vertices of a network uniformly within a hyperbolic disk, we obtain few very popular vertices (near the disk center) that are connected to many unpopular vertices (near the disk's boundary) as a result of the exponential expansion of space.

To actually present a hyperbolic drawing to a user, one has to project the hyperbolic plane to the Euclidean plane. This naturally results in a nice fish-eye view that highlights what is currently in the center of the projection [14]. With these advantages of the hyperbolic plane and the popularity of spring embedders, it is not surprising that a spring embedder has been adapted to work for the hyperbolic plane [12]. This approach already produces good results when the embedding is constrained to a small portion of the hyperbolic plane and it showcases the nice fish-eye view effect obtained by the projection. Our goal is to extend their work by considering larger portions of the plane in order to utilize the natural heterogeneity of hyperbolic space to embed scale-free networks. Unfortunately, spring embedders encounter some fundamental problems when confronted with this heterogeneity.

Due to the exponential expansion of space, geodesic lines between pairs of points are bend towards the origin. Thus, moving towards another vertex almost always means moving towards the origin first. Therefore, a less popular vertex (one with low degree) has to be moved closer to the origin to get to where it actually belongs. However, this brings it closer to every other vertex (the smaller the radius, the higher the popularity), which is prevented by the repulsive forces. Thus, even bad embeddings with very long edges are rather stable.

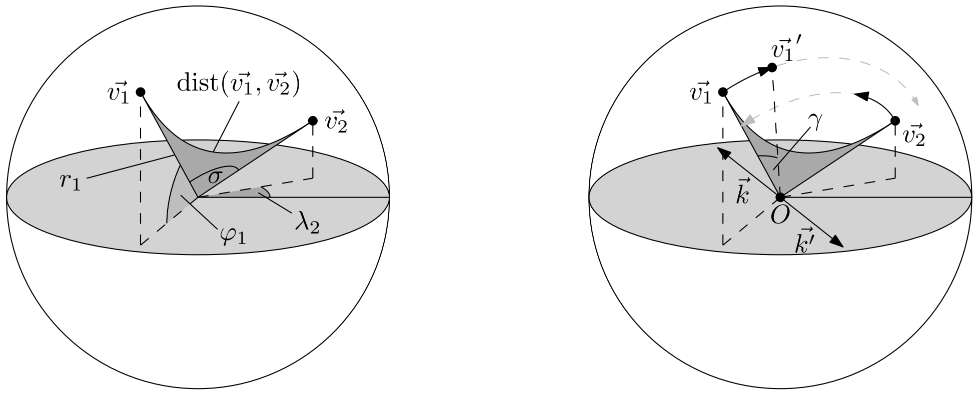
Beyond spring embedders, other approaches have been proposed to generate hyperbolic embeddings. Some of them determine hyperbolic coordinates for the vertices using a spanning tree of the graph, for example to perform greedy routing [5, 10] or to visualize hierarchical data in three-dimensional hyperbolic space [15]. In order to embed graphs with an underlying hyperbolic geometry, the following techniques have been proposed.

An often used approach are maximum likelihood estimation embedders, which try to find the coordinates for the vertices that maximize the probability of the network being generated by an underlying hyperbolic model [18]. In the *step model*, such a hyperbolic random graph is generated by placing n vertices in a hyperbolic disk of radius R and connecting any two vertices with hyperbolic distance at most R. Thus, the probability of a graph being produced by the model is 0, if it has edges longer than R or non-edges shorter than R. Essentially, the goal of the embedder is to find an embedding such that adjacent vertices are close to each other and non-adjacent vertices are farther apart, which is exactly, what spring embedders pursue as well. *HyperMap* [18] tries to solve this problem, by replaying the network's geometric growth in the hyperbolic plane: starting with high-degree vertices near the origin, each vertex is placed close to its neighbors such that the probability is maximized that the currently embedded graph emerged from the model. An improved version called *HyperMap-CN* was later obtained by additionally taking the common neighborhood of two vertices into account [17]. A further adaption yielded an algorithm with quasilinear running time [2]. Additionally, a novel approach to embedding networks into the hyperbolic plane are *coalescent* embeddings [16]. There, non-linear dimensionality reduction is applied to a matrix representing distances between vertices in the graph. The result is a Euclidean embedding of the network, in which metric distances between vertices match the corresponding distances in the input matrix. The hyperbolic embedding is then obtained by deriving a circular order of the vertices from the Euclidean embedding and combining it with information about the degree of a vertex to approximate its position in the hyperbolic disk.

While the above techniques mostly produce good embeddings, they are not very robust. For example, maximum likelihood embedders rely on a good initial embedding of the core (the high-degree vertices) and place vertices with larger distance to the center near their higher-degree neighbors. If the initial embedding of the core is bad (which can happen if there are not enough high-degree vertices), the overall embedding will be bad as well. The coalescent embedder encounters a similar issue if the initial Euclidean embedding is not good.

In the Euclidean plane, spring embedders have proven to be very robust and to quickly produce good embeddings for non-complex graphs. It is still an open problem to adapt this approach to work in the hyperbolic plane in a way that exploits the geometry to better visualize heterogeneous networks. To answer this question, we provide a proof of concept which shows that good hyperbolic embeddings of heterogeneous networks can be obtained using a spring embedder. Our experiments indicate that the quality of the resulting embeddings is on par with the one obtained using the previously mentioned embedding techniques. As a consequence, we believe that our proof of concept lays the groundwork for transferring the ensemble of techniques that have been developed to improve Euclidean spring embedders into the hyperbolic setting.

Outline and Contribution. After a brief introduction into the hyperbolic space in Section 2, we describe our embedding process in Section 3. First, we identify a fundamental difference between Euclidean and hyperbolic spring embedders. Basically, the forces simulate a *region of influence* around each vertex: attractive forces pull neighbors into this region and repulsive forces push non-adjacent vertices out of it. In the hyperbolic plane this region of influence has a very different impact on the embedding than in the Euclidean plane. In order to adapt to this difference, we split the forces into two types: one that affects a vertex's popularity and one that tunes the similarity aspect. This, however, reduces the dimensionality of the spring embedder, which makes it harder to escape local optima. We propose to overcome this issue by embedding a network in the three-dimensional hyperbolic space first (Sections 3.1 to 3.4), and transitioning the resulting embedding to the plane afterwards (Section 3.5). In order to evaluate this technique, we conducted experiments on different kinds of networks. In Section 4 our results are compared to the existing embedding techniques mentioned above.



(a) The central angle σ is used to determine the hyperbolic distance between the two vertices $\vec{v_1}$ and $\vec{v_2}$.

(b) Vertex $\vec{v_1}$ is rotated towards $\vec{v_2}$ around $\vec{k}$, which goes through the origin O and is perpendicular to the plane defined by $\vec{v_1}$, $\vec{v_2}$, and O.

Figure 1 Distances and vertex movement in three-dimensional hyperbolic space.

2 Preliminaries

Hyperbolic Plane. While space expands polynomially in the Euclidean geometry, the expansion is exponential in the hyperbolic geometry. In the hyperbolic plane $\mathbb{H}_2$ a circle with radius r has area $2\pi(\cosh(r)-1)$ and circumference $2\pi\sinh(r)$, with $\cosh(x)=(e^x+e^{-x})/2$ and $\sinh(x)=(e^x-e^{-x})/2$, both growing as $e^x/2\pm o(1)$.

A point $p\in\mathbb{H}_2$ is identified using polar coordinates $p=(r,\varphi)$, where r is the *radius* and defines the distance to a designated origin O and $\varphi\in[0,2\pi)$ is the *angular coordinate* and denotes the angular distance to a reference ray starting at O. Given two points $p_1=(r_1,\varphi_1)$ and $p_2=(r_2,\varphi_2)$, the hyperbolic distance between them is given by

$$\cosh(\text{dist}(p_1,p_2))=\cosh(r_1)\cosh(r_2)-\sinh(r_1)\sinh(r_2)\cos(\Delta(\varphi_1,\varphi_2)),$$

where $\Delta(\varphi_1,\varphi_2)=\pi-|\pi-|\varphi_1-\varphi_2||$ is the *angular distance* between p_1 and p_2. Finally, given two points with radii $r_1,r_2\le R$, respectively, the maximum angular distance such that their hyperbolic distance is still at most R [8, Lemma 3.1], is given by

$$\theta(r_1,r_2)=\arccos\left(\frac{\cosh(r_1)\cosh(r_2)-\cosh(R)}{\sinh(r_1)\sinh(r_2)}\right)=2e^{\frac{R-r_1-r_2}{2}}(1+\Theta(e^{R-r_1-r_2})). \tag{1}$$

Three-Dimensional Hyperbolic Space. In $\mathbb{H}_3$ the coordinates of a vertex v are represented by a tuple $\vec{v}=(r,\lambda,\varphi)$, which describes its radius, latitude, and longitude, respectively. As can be seen in Figure 1a, the hyperbolic distance between two vertices $\vec{v_1}$ and $\vec{v_2}$ is obtained by first determining the central angle σ between them, which is given by

$$\cos(\sigma)=\sin(\varphi_1)\sin(\varphi_2)+\cos(\varphi_1)\cos(\varphi_2)\cos(\Delta\lambda),$$

for $\Delta\lambda=|\lambda_1-\lambda_2|$. Afterwards, as in the two-dimensional case, the distance is obtained as

$$\cosh(\text{dist}(\vec{v_1},\vec{v_2}))=\cosh(r_1)\cosh(r_2)-\sinh(r_1)\sinh(r_2)\cos(\sigma). \tag{2}$$

The rotation of a vertex $\vec{v_1}$ towards another vertex $\vec{v_2}$ around the origin works just as it does in Euclidean space. We therefore convert our polar coordinates to Cartesian coordinates and perform the rotation as if it was in Euclidean space. This rotation is defined by a single

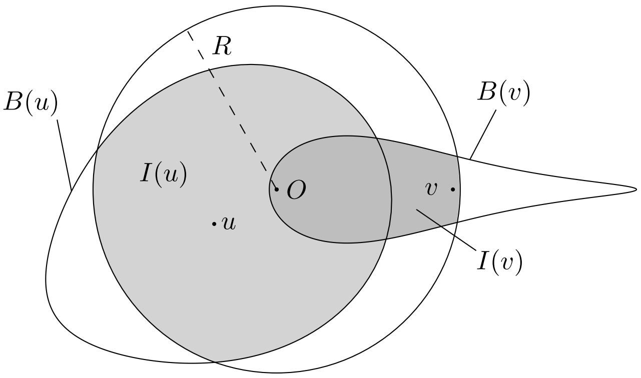


Figure 2 Two vertices and their regions of influence are shown in a hyperbolic disk of radius R. As the distance between v and the origin is larger than the one between u and the origin, v's region of influence $I(v)$ is smaller than $I(u)$, although $B(u)$ and $B(v)$ have the same radius R.

vector $\vec{k}$. The direction of $\vec{k}$ denotes the axis that the vertex rotates around. This axis goes through the origin and is perpendicular to the plane defined by the origin O and the points $\vec{v}_1$ and $\vec{v}_2$ (the right-hand rule applies). We obtain $\vec{k} = \vec{v}_1 \times \vec{v}_2$, where $\times$ denotes the cross-product. The length of the vector determines the rotation angle. In Figure 1b one can see how vertex $\vec{v}_1$ is rotated towards $\vec{v}_2$. It is rotated by $|\vec{k}| = \gamma$ around the axis denoted by $\vec{k}$. Note that inverting the rotation axis from $\vec{k}$ to $\vec{k}'$ inverts the direction of the rotation.

Given the Cartesian coordinates of $\vec{v}$ and a rotation vector $\vec{k}$, the rotation $R(\vec{v}, \vec{k})$ is applied using *Rodrigues' formula*, yielding the coordinates of the rotated vector $\vec{v}'$ as

$$\vec{v}' = R(\vec{v}, \vec{k}) = \vec{v} \cos\left(|\vec{k}|\right) + \left(\vec{k} \times \vec{v}\right) \sin\left(|\vec{k}|\right) + \vec{k}\left(\vec{k} \cdot \vec{v}\right)\left(1 - \cos\left(|\vec{k}|\right)\right), \tag{3}$$

where $\times$ and $\cdot$ denote the cross product and dot product, respectively.

3 Embedding Process

In a force-directed embedding, the forces simulate a *region of influence* around a vertex v, denoted by $B(v)$ (a ball around v). Attractive forces pull v's neighbors into $B(v)$ and repulsive forces push non-adjacent vertices out of it.

In the Euclidean plane the size of the region of influence is an input parameter that determines the preferred length of the edges, essentially scaling the embedding. In hyperbolic space the region of influence has a very different impact on the embedding. Recall that a scale-free network emerges naturally, if we assume that its vertices are distributed uniformly within a hyperbolic disk. Since there are no vertices outside of the disk, the region of influence $I(v)$ of a vertex v can be seen as a ball around v that is *constrained* to the disk. This is depicted in Figure 2, where we interpret the polar coordinates in hyperbolic space as polar coordinates in Euclidean space. The size of the region of influence $I(v)$ changes with v's distance to the origin, even though the radius of the ball denoting $I(v)$ is fixed. The closer v is to the origin, the larger is the portion of the disk that is covered by $I(v)$. Consequently, the popularity of v is high. With increasing distance between v and the origin, the size of $I(v)$ decreases, i.e., v becomes less popular (again see Figure 2). The exponential expansion of space now leads to an interesting phenomenon: the heterogeneous degree distribution of scale-free networks emerges naturally by using *the same radius* for the hyperbolic disk (that the vertices are distributed in) and the ball that denotes the region of influence [13].

Unfortunately, this property of hyperbolic geometry impedes the successful application of force-directed embedding algorithms. As mentioned in the introduction, moving a vertex towards another vertex decreases its distance to (almost) all other vertices. Consequently, the resulting repulsive forces prevent this movement, leading to bad stable embeddings. We overcome this problem by dividing the forces into two types, one effecting the popularity (i.e., the radii), and the other only the similarity (i.e., the angular coordinates). That way, vertices no longer move on their geodesic lines. This division, however, reduces the movement of the vertices to one dimension, which decreases the chances of escaping local optima. We circumvent this issue by first embedding the graph in three-dimensional hyperbolic space. In this way, the forces affecting the similarity move the vertices on a two-dimensional surface of a sphere, leading to a similar behavior as in the Euclidean plane.

The general process can now be described as follows. Starting with a random initial embedding of the graph in three-dimensional hyperbolic space, we iteratively apply forces to move adjacent vertices close to each other and non-adjacent vertices farther apart, and in the process adapt the radii of the vertices to tune their region of influence. Once this embedding is stable, a plane is identified, that minimizes the distance to all vertices. Finally, forces are applied to pull the vertices towards this designated plane, resulting in a two-dimensional embedding of the network. In the following sections, we explain each step in greater detail.

3.1 Initial Embedding

Recall that we imagine the vertices of our network to be evenly distributed in a disk lying in the hyperbolic plane that has the same radius as the region of influence around each vertex. This radius R can be estimated such that it best fits the size of the given network [3].

We start with an initial embedding $\mathcal{E}$ that assigns each vertex a point in a three-dimensional hyperbolic sphere of radius R. To this end, we draw a point uniformly distributed on the surface of the three-dimensional unit sphere, and a radius uniformly at random from $[0, R]$.

3.2 Forces

Recall that we have two types of movements. Changing the radial coordinate only affects the size of the region of influence. Rotating a vertex around the origin moves the region of influence without changing its size. To accommodate for these two different movements, we apply two types of forces separately: *popularity forces* affect the radius of a vertex, and *similarity forces* affect the latitude and longitude of a vertex within the sphere.

Popularity Forces. A single vertex v with radius r_v is adjusted by comparing its degree $\deg(v)$ with the expected number of vertices in its region of influence, assuming that the current radii $r_1, \dots, r_n$ of all vertices are fixed. Consider the following random experiment: n vertices with radii $r_1, \dots, r_n \leq R$ are placed in a hyperbolic disk by choosing their angular coordinates uniformly at random. Without loss of generality we can assume that $\varphi_v = 0$. Now we observe the random variable X which denotes the number of vertices in $I(v)$. Recall that a vertex u is contained in $I(v)$ if its distance to v is at most R. Moreover, $\theta(r_u, r_v)$ denotes the maximum angular distance between u and v such that this is true (Equation (1)). Since the probability for this to happen is $\Pr[u \in I(v)] = 2\theta(r_u, r_v)/2\pi$, we can compute $\mathbb{E}[X]$ as

$$\mathbb{E}[X] = \frac{1}{\pi} \sum_{u \in V \setminus \{v\}} \theta(r_v, r_u).$$

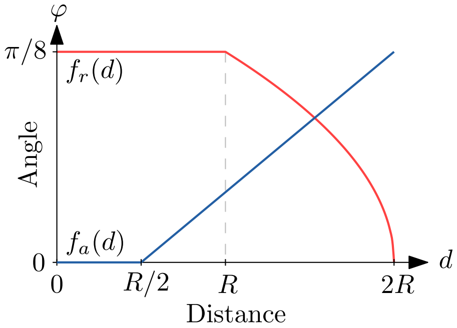


Figure 3 The functions $f_a(d)$ and $f_r(d)$ determine the magnitude of the attractive and repulsive forces, respectively.

When applying the popularity force on a vertex v, we compare $\mathbb{E}[X]$ with $\deg(v)$. If $\mathbb{E}[X] > \deg(v)$, the region of influence of v is too large and r_v is increased. If $\deg(v) = \mathbb{E}[X]$, r_v is not changed. Otherwise it is decreased. In preliminary experiments we determined that adjusting the radii using a fixed, small step size delivered the best results.

We note, that the radius of a vertex is not bounded from above, i.e., the radius R of our disk (and therefore the size of the region of influence) can change during the embedding process and is set to be the maximum radius of a vertex in the current embedding. That way we can accommodate for potential errors that were made during the initial estimation of R.

Similarity Forces. The similarity forces move vertices without changing their radii. This allows us to move vertices close to each other, without getting closer to all other vertices.

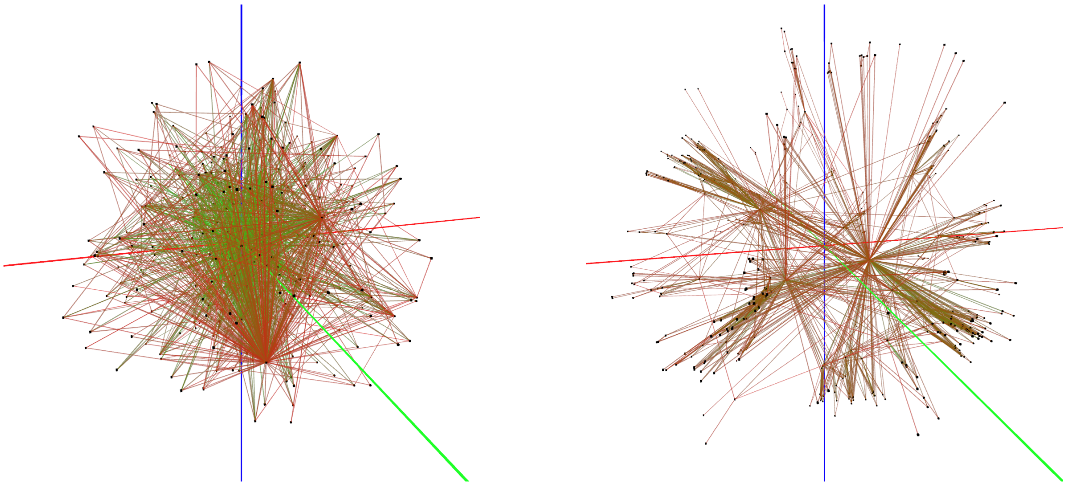
Let $u \neq v$ be two vertices with coordinates $\vec{u}$ and $\vec{v}$, respectively. In the following, we observe the force that is caused by u and acts on v. Formally, the force is determined using a function $f\colon \mathbb{H}_3 \times \mathbb{H}_3 \to \mathbb{R}^3$. The resulting vector describes the axis around which the vertex, that the force acts on, is rotated. The direction of the force $f(\vec{u}, \vec{v})$ is perpendicular to the plane containing u, v and the origin, and is given by $\vec{k} = (\vec{v} \times \vec{u})/(|\vec{v} \times \vec{u}|)$. If the force is repulsive, we invert the direction of the rotation by rotating around $-\vec{k}$. The length of $f(\vec{u}, \vec{v})$ describes the magnitude of the force and was chosen to match the angle φ that v is rotated by. It depends on the hyperbolic distance $\mathrm{dist}(\vec{u}, \vec{v})$ between the two vertices (Equation (2)). We define two functions f_a (for attractive forces) and f_r (for repulsive forces) that map the distance $d \in [0, 2R]$ to the magnitude $\varphi \in [0, \pi/8]$ of the force. In preliminary experiments, the upper bound of $\pi/8$ for φ proved to be useful in avoiding very large jumps. As they delivered the best results in preliminary experiments, we chose the two functions to be

$$f_a(d) = \begin{cases} 0, & d \leq R/2, \\ \pi/8 \cdot \left(\frac{2d-R}{3R}\right), & \text{otherwise} \end{cases} \quad \text{and} \quad f_r(d) = \begin{cases} \pi/8, & d \leq R, \\ \pi/8 \cdot \left(2 - \frac{d}{R}\right)^{1/2}, & \text{otherwise,} \end{cases}$$

which are depicted in Figure 3. Note that the repulsive force is as strong as possible if the distance between the two vertices is less than R, i.e., when v is in the region of influence of u.

3.3 Force Application

After the initial random embedding we alternatingly apply batches of popularity and similarity forces. In each iteration we compute the forces that act on the vertices and rotate them accordingly. Additionally, some precautions are taken that help stabilize the embedding. In a single iteration i, the total force that acts on a vertex v is determined, by first summing



(a) The initial embedding with random coordinates for all vertices.

(b) A stable embedding after applying popularity and similarity forces.

Figure 4 Two three-dimensional embeddings (before and after applying forces) of a hyperbolic random graph with 492 vertices.

the forces that are caused by all other vertices, which we denote with $\vec{k}_i(v)$. In order to stabilize the embedding, $\vec{k}_i(v)$ is then scaled using a *temperature* $\tau_i \in (0,1]$ that decreases as $\tau_i = 0.975 \cdot \tau_{i-1}$ in every iteration. Additionally, to prevent oscillations, a *velocity* is simulated by adding a portion ν (we use $\nu = 1/2$) of the forces that acted on v in the previous iteration. Taken together, we obtain the rotation vector $\vec{\kappa}_i(v)$, describing the total force that acts on v in iteration i as $\vec{\kappa}_i(v) = \nu\vec{\kappa}_{i-1}(v) + \tau_i\vec{k}_i(v)$ and rotate v accordingly.

3.4 Stability

After every iteration i we obtain a new embedding $\mathcal{E}_i$ describing the current positions of all vertices. From these positions we can derive a *potential* ρ_i which describes how unstable the embedding is after iteration i. The potential is defined as the sum of the strengths of all forces and can be computed as $\rho_i = \sum_{v \in V} |\vec{\kappa}_i(v)|$. After every iteration we compute the potential ρ_i and compare it with previous ρ_j for $j < i$ to detect whether the potential is decreasing, meaning the embedding is getting more stable. Figure 4 compares the initial embedding of a graph with a stable one, obtained after iteratively applying the forces.

After iteration i the process stops if the potential has decreased in the last couple of iterations, meaning $\rho_j > \rho_{j+1}$, for $i - 10 < j < i$ and the difference to the previous potential ρ_{i-1} drops below a given stability threshold. Additionally, the process stops if the number of iterations has exceeded a predefined threshold. Usually, the potential fluctuates in the beginning, since the temperature is high, but as the temperature decreases over time, so does the potential and the embedding becomes stable eventually.

3.5 Transition to the Plane

Once the three-dimensional embedding is stable, we convert it to a two-dimensional embedding, by first determining the plane that contains the origin and minimizes the distance to all vertices, using principal component analysis. Then, the embedding is rotated such that this plane aligns with the plane $P = \{(r, \lambda, \varphi) \mid \varphi = 0\}$. Afterwards, in addition to the forces

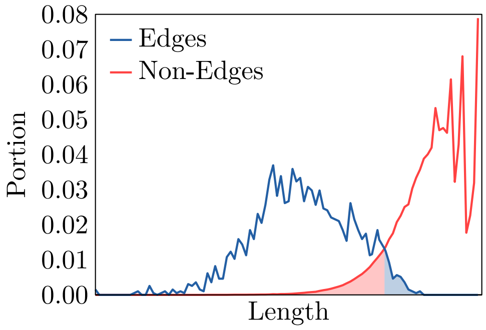


Figure 5 An edge-length histogram that is used to measure the embedding quality. It shows the portion of edges (blue) and non-edges (red) of a given length. The filled regions denote the errors.

that were used so far, we apply forces that pull the vertices towards P. To this end, for every vertex v with coordinate $\vec{v} = (r, \lambda, \varphi)$, we introduce a virtual vertex v' with coordinate $\vec{v'} = (r, \lambda, 0)$. The total force $\vec{\kappa}_i(v)$ that acts on v is now determined as before, with the addition of a force that attracts v towards v'. In particular, we obtain the new position of v by rotating it around $\vec{\kappa}_i(v) + \vec{\kappa}'_i(v)$. The additional force $\vec{\kappa}'_i(v) = \tau_i \cdot \pi/15 \cdot (\vec{v}' \times \vec{v})/(|\vec{v}' \times \vec{v}|)$ is independent of the distance between v and v', constantly pulling v towards the plane. By still applying the popularity and similarity forces, we try to preserve the quality of the existing embedding while transitioning it towards the plane. This process ends as soon as the average distance between the vertices and the plane drops below a certain threshold or a maximum number of iterations is reached. Then, all vertices are moved onto the plane, yielding the final two-dimensional embedding.

4 Experiments

In order to evaluate whether the adapted spring embedder works and how it compares to existing embedding techniques, we implemented it in C++[1], using *Eigen* [7] to perform the principal component analysis needed for the transition to the two-dimensional plane. Afterwards we ran experiments on the largest component of 16 real-world networks [19] and 18 hyperbolic random graphs with different parameter configurations. In addition to our spring embedder, we considered the Euclidean spring embedder *FMMM* [9] (using the implementation found in *OGDF* [4]), as well as the maximum likelihood estimation embedder HyperMap-CN [17], the previously mentioned quasilinear adaptation, which we abbreviate with *BFKL* [2], and the coalescent embedder [16].

Embedding Quality. Recall, that the goal of an embedding technique is to place adjacent vertices closely together and non-adjacent ones farther apart. In order to measure how well this goal is achieved we introduce the *edge-length histogram*, as can be seen in Figure 5. It is based only on a graph's adjacency information and an embedding. This allows us to evaluate embeddings of real-world networks, where ground truth data is usually not available. The *edge curve* (blue) denotes the relative number of edges of a given length. The *non-edge curve* (red) does the same for non-edges. We measure the error of the embedding by determining

[1] Our code is available at `https://github.com/maxkatzmann/hyperbolic-spring-embedder.git` (archived at `swh:1:dir:7b9445f64fae3be4bbe3a692c2f94ded0bc600d1`).

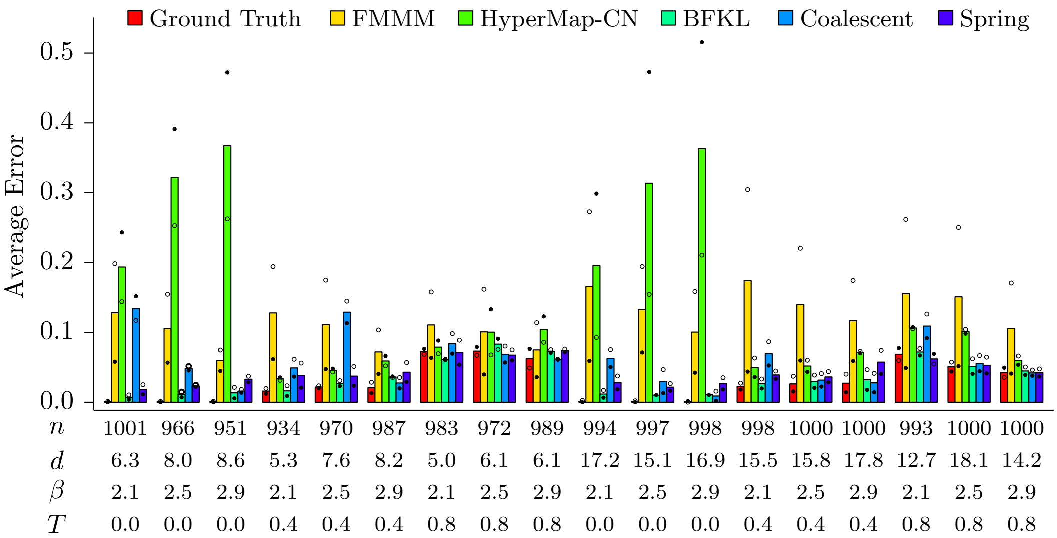


Figure 6 Errors obtained using different embedding techniques on hyperbolic random graphs. Filled circles indicate edge errors. Hollow circles indicate non-edge errors. Bars denote their average. The graphs differ in the number of vertices n, the average degree d, the power-law exponent of the degree distribution β, and the temperature T.

the area under the minimum of the two curves. The *edge error* is the corresponding area whenever the edge curve takes on the minimum (filled blue region). The *non-edge error* (denoted by the filled red region) is defined analogously. The *average error* is the average of the edge error and the non-edge error and the *balancing error* is the difference between them.

In a perfect embedding, the edge-length histogram would show a peak in the edge curve to the left of another peak in the non-edge curve without any overlap, meaning all edges are shorter than all non-edges. In other words, all vertices have their neighbors inside their region of influence and all non-neighbors outside of it. Consequently, the average error and the balancing error are both 0%. Whenever the two curves overlap, errors were made. The average error gives some general hints about the overall quality of the embedding. But assume an embedding has an average error of 50% (the worst error possible). This could mean, for example, that the edge error and the non-edge error are both close to 50%, or (in the extreme case where all vertices are placed on the same point) that the edge error is 0% and the non-edge error is 100%. In that case the latter embedding is arguably worse than the first one, which is revealed by looking at the balancing error.

Hyperbolic Random Graphs. Figure 6 shows the errors obtained in our experiments on hyperbolic random graphs. These graphs are sampled by placing n vertices uniformly at random in a hyperbolic disk of radius $R = 2 \log n + C$, where C controls the average degree d of the network. Furthermore, the power-law exponent $\beta \in (2, 3)$ impacts the degree distribution. The smaller β, the denser is the core of the network. In the step model, any two vertices are connected, if the distance between them is at most R. In a relaxed version, a temperature T (typically between 0 and 1) is introduced, which allows for long-range edges (and short non-edges) by smoothing the step function. This influences the clustering coefficient of the generated network: the smaller T, the higher the clustering. We note that this notion of temperature is independent of the one used to stabilize the embedding.

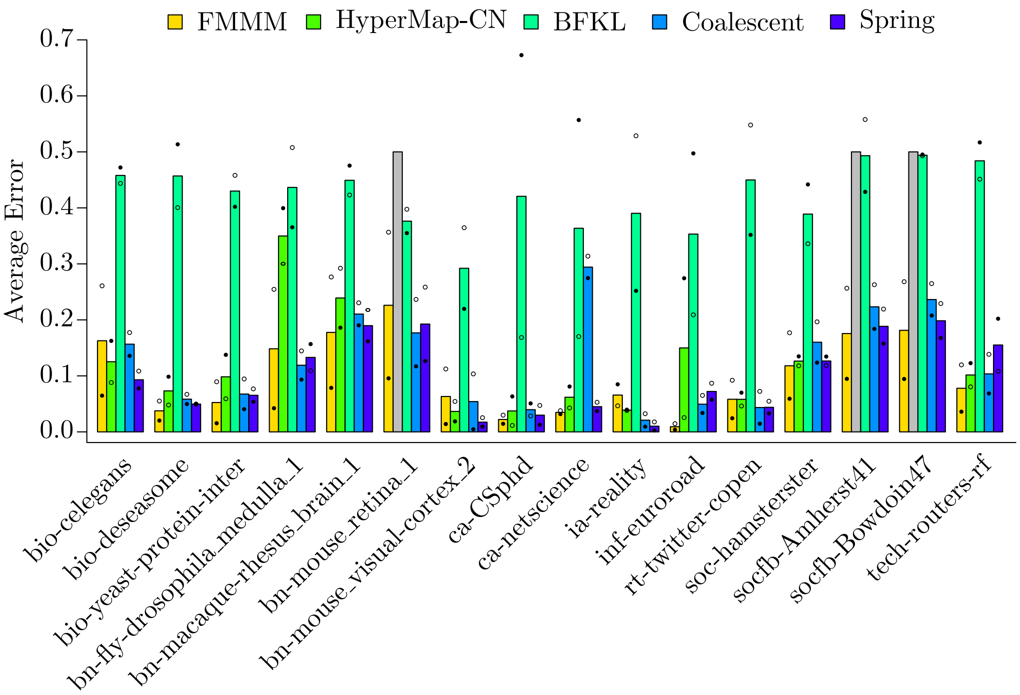


Figure 7 Errors obtained using different embedding techniques on the largest component of several real-world networks. Filled circles indicate edge errors. Hollow circles indicate non-edge errors. Bars denote their average. Grey bars indicate that the embedding process did not finish within 96 hours.

On these networks the considered techniques that generated embeddings in the hyperbolic plane delivered mostly good results. The mean errors of the different techniques over all generated graphs are:

	Ground Truth	FMMM	HyperMap-CN	BFKL	Coalescent	Spring
Average Error	2.8%	11.9%	14.5%	3.4%	5.9%	4.3%
Balancing Error	1.0%	13.4%	8.2%	1.0%	2.2%	1.8%

The BFKL embedder was the best one, delivering results that are very close to the ground truth (the coordinates sampled during the graph generation). With the exception of FMMM and HyperMap-CN, the remaining embedders also produced very good results.

As expected, FMMM had trouble fitting the hyperbolic random graphs into the Euclidean plane. Figure 6 shows that on most networks FMMM obtained a small edge error at the expense of a large non-edge error, yielding a balancing error of 13.4% on average. This does not come as a surprise as there is simply not enough space in the Euclidean plane to keep non-edges long while trying to obtain short edges. Unfortunately, HyperMap-CN seemed to have issues with low temperatures. Excluding graphs with $T = 0$, HyperMap-CN obtained an average error of 7.2%.

The hyperbolic spring embedder produced embeddings that are on par with the ground truth and sometimes even better. The average error and the balancing error are small on average, with 4.3% and 1.8%, respectively. This shows that spring embedders can be adapted to take advantage of the intrinsic heterogeneity of the hyperbolic plane to produce good embeddings of heterogeneous networks.

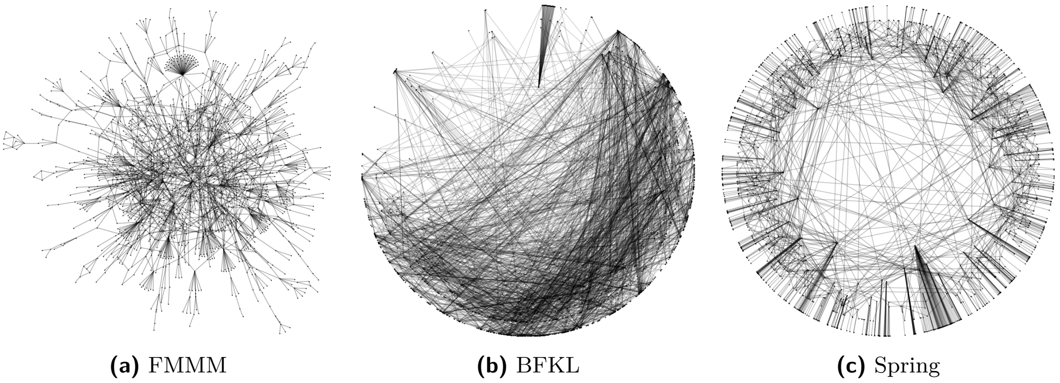

(a) FMMM **(b)** BFKL **(c)** Spring

Figure 8 Embeddings of the largest component (containing 1458 vertices) of the *bio-yeast-protein-inter* network, obtained using different techniques.

Real-World Networks. Figure 7 shows the errors of the embeddings obtained using the different techniques on real-world networks. The mean errors over all graphs are:

	FMMM	HyperMap-CN	BFKL	Coalescent	Spring
Average Error	10.1%	11.5%	42.1%	12.6%	10.1%
Balancing Error	10.9%	6.6%	15.8%	5.5%	4.1%

Overall the embedders produced good results, with the exception of BFKL with an average error of 42.1%. As noted by the authors, this technique excels on larger graphs [2]. Thus, one explanation for the bad performance is the constraint to smaller graph sizes ($<$ 7000 vertices) that was necessary to compensate for the running time of other techniques. In fact, while HyperMap-CN produced good results in general, it also encountered three instances where it did not finish the embedding process within 96 hours. The remaining techniques performed well with an average error of a little over 10%.

Figure 7 shows that FMMM encountered the same difficulties as on the hyperbolic random graphs and was often not able to maintain long non-edges while making the edges short (except for *inf-euroroad*, a road-network where the underlying geometry is arguably rather Euclidean). Figure 8 depicts embeddings of the *bio-yeast-protein-inter* network.[2] In the FMMM embedding (Figure 8a) one can see how higher degree vertices are placed near the center of the embedding and with them their low-degree neighbors. As a result, the center part is very cluttered and the structure of the network is unclear.

Likewise, the maximum likelihood estimation techniques were sometimes not able to produce embeddings that convey the structure of the network. They usually start the embedding process by first embedding the core (high-degree vertices that are supposed to be close to the origin) and determine the positions of later vertices based on the already placed ones. As can be seen in Figure 8b, this fails if there are not enough vertices in the core. In that case, the earlier stages of the embedding are rather arbitrary and later vertices can not compensate for the initial errors.

On the other hand, the natural approach of applying forces between the vertices to obtain embeddings that reflect the structure of the network well, proved to be very robust. In 11 of the 16 embeddings the hyperbolic spring embedder delivered the best results and was not far off on the remaining instances, yielding the best average and balancing errors

[2] Further embedding evaluations can be found in Appendix A.

of 10.1% and 4.1%, respectively. Figure 8c shows the hyperbolic spring embedding of the *bio-yeast-protein-inter* network. One can observe a good trade-off between edges being too long and non-edges being too short, resulting in a drawing where higher degree vertices are placed towards the center and their low-degree neighbors are close to them near the disk's boundary. This separation between high- and low-degree vertices helps in reducing the clutter, making it easier to see which vertices actually belong together.

5 Conclusion

It was previously observed that spring embedders encounter a fundamental problem when being subjected to the natural heterogeneity of hyperbolic space. After explaining the core difficulties we proposed a way to circumvent them: The application of the forces typically has close to no impact on the position of a vertex as movement along geodesic lines simultaneously affects its region of influence in two ways (popularity and similarity), which can be overcome by differentiating between two types of forces and increasing the dimensionality in order to better escape local optima. Our experiments indicate that the resulting approach produces embeddings that are on par with other commonly used hyperbolic embedding techniques.

Adapting the standard force-directed embedding approach to work in hyperbolic space paves the way for translating well known techniques that improve the quality of Euclidean spring embedders into the hyperbolic setting. These include the use of geometric data structures [1], the multilevel strategy of the previously mentioned FMMM embedder [9], and the application of random sampling in order to improve the running time [6]. We believe that these techniques can also be used to extend our proof of concept in order to further improve its performance.

References

1 Josh Barnes and Piet Hut. A hierarchical $O(NlogN)$ force-calculation algorithm. *Nature*, 324(6096):446–449, 1986. doi:10.1038/324446a0.

2 Thomas Bläsius, Tobias Friedrich, Anton Krohmer, and Sören Laue. Efficient embedding of scale-free graphs in the hyperbolic plane. In *24th Annual European Symposium on Algorithms, ESA 2016*, pages 16:1–16:18, 2016. doi:10.4230/LIPIcs.ESA.2016.16.

3 Marián Boguñá, Fragkiskos Papadopoulos, and Dmitri Krioukov. Sustaining the internet with hyperbolic mapping. *Nature Communications*, 1(62), 2010. doi:10.1038/ncomms1063.

4 Markus Chimani, Carsten Gutwenger, Michael Jünger, Gunnar W. Klau, Karsten Klein, and Petra Mutzel. The Open Graph Drawing Framework (OGDF). In *Handbook of Graph Drawing and Visualization*, chapter 17. CRC Press, 2014.

5 D. Eppstein and M. T. Goodrich. Succinct greedy geometric routing using hyperbolic geometry. *IEEE Transactions on Computers*, 60(11):1571–1580, 2011. doi:10.1109/TC.2010.257.

6 R. Gove. A Random Sampling $O(n)$ Force-calculation Algorithm for Graph Layouts. *Computer Graphics Forum*, 38(3):739–751, 2019. doi:10.1111/cgf.13724.

7 Gaël Guennebaud, Benoît Jacob, et al. Eigen v3. http://eigen.tuxfamily.org, 2010.

8 Luca Gugelmann, Konstantinos Panagiotou, and Ueli Peter. Random hyperbolic graphs: Degree sequence and clustering. In *Automata, Languages, and Programming*, pages 573–585, 2012. doi:10.1007/978-3-642-31585-5_51.

9 Stefan Hachul and Michael Jünger. Drawing large graphs with a potential-field-based multilevel algorithm. In *Graph Drawing, 12th International Symposium, GD 2004*, pages 285–295, 2004. doi:10.1007/978-3-540-31843-9_29.

10 Robert Kleinberg. Geographic routing using hyperbolic space. In *26th IEEE International Conference on Computer Communications*, pages 1902–1909, 2007. doi:10.1109/INFCOM.2007.221.

11 Stephen G. Kobourov. Force-directed drawing algorithms. In *Handbook of Graph Drawing and Visualization*, chapter 12. CRC Press, 2014.

12 Stephen G. Kobourov and Kevin Wampler. Non-euclidean spring embedders. *IEEE Transactions on Visualization and Computer Graphics*, 11(6):757–767, 2005. doi:10.1109/TVCG.2005.103.

13 Dmitri Krioukov, Fragkiskos Papadopoulos, Maksim Kitsak, Amin Vahdat, and Marián Boguñá. Hyperbolic geometry of complex networks. *Physical Review E*, 82(3):036106, 2010. doi:10.1103/PhysRevE.82.036106.

14 John Lamping, Ramana Rao, and Peter Pirolli. A focus+context technique based on hyperbolic geometry for visualizing large hierarchies. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 401–408, 1995. doi:10.1145/223904.223956.

15 Tamara Munzner. H3: Laying out large directed graphs in 3d hyperbolic space. In *Proceedings of the 1997 IEEE Symposium on Information Visualization (InfoVis '97)*, pages 2–10, 1997. doi:10.1109/INFVIS.1997.636718.

16 Alessandro Muscoloni, Josephine Maria Thomas, Sara Ciucci, Ginestra Bianconi, and Carlo Vittorio Cannistraci. Machine learning meets complex networks via coalescent embedding in the hyperbolic space. *Nature Communications*, 8(1):1615, 2017. doi:10.1038/s41467-017-01825-5.

17 Fragkiskos Papadopoulos, Rodrigo Aldecoa, and Dmitri Krioukov. Network geometry inference using common neighbors. *Physical Review E*, 92(2):022807, 2015. doi:10.1103/PhysRevE.92.022807.

18 Fragkiskos Papadopoulos, Constantinos Psomas, and Dmitri Krioukov. Network mapping by replaying hyperbolic growth. *IEEE/ACM Transactions on Networking*, 23(1):198–211, 2015. doi:10.1109/TNET.2013.2294052.

19 Ryan A. Rossi and Nesreen K. Ahmed. The network data repository with interactive graph analytics and visualization. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015. URL: http://networkrepository.com.

A Additional Embedding Comparisons

On the following pages we show a few more comparisons of embeddings produced using the various techniques as explained in Section 4.

Hyperbolic Random Graph (Figure 9)

Hyperbolic random graphs are sampled by distributing vertices uniformly at random in a hyperbolic disk and connecting any two that are sufficiently close. Figure 9a depicts such a graph using the sampled vertex positions. Since the underlying geometry of the network is hyperbolic and not Euclidean, FMMM has no chance in finding an embedding that represents the structure of the graph well (Figure 9b). The densely connected high-degree vertices as well as their lower-degree neighbors are all placed close to each other, yielding a cluttered embedding. Apart from the coalescent embedding (Figure 9e) the other techniques obtained embeddings that resemble the ground truth well. This is reflected by the corresponding error diagram in Figure 6 for the network with $n = 970$, $d = 7.6$, $\beta = 2.5$, and $T = 0.4$.

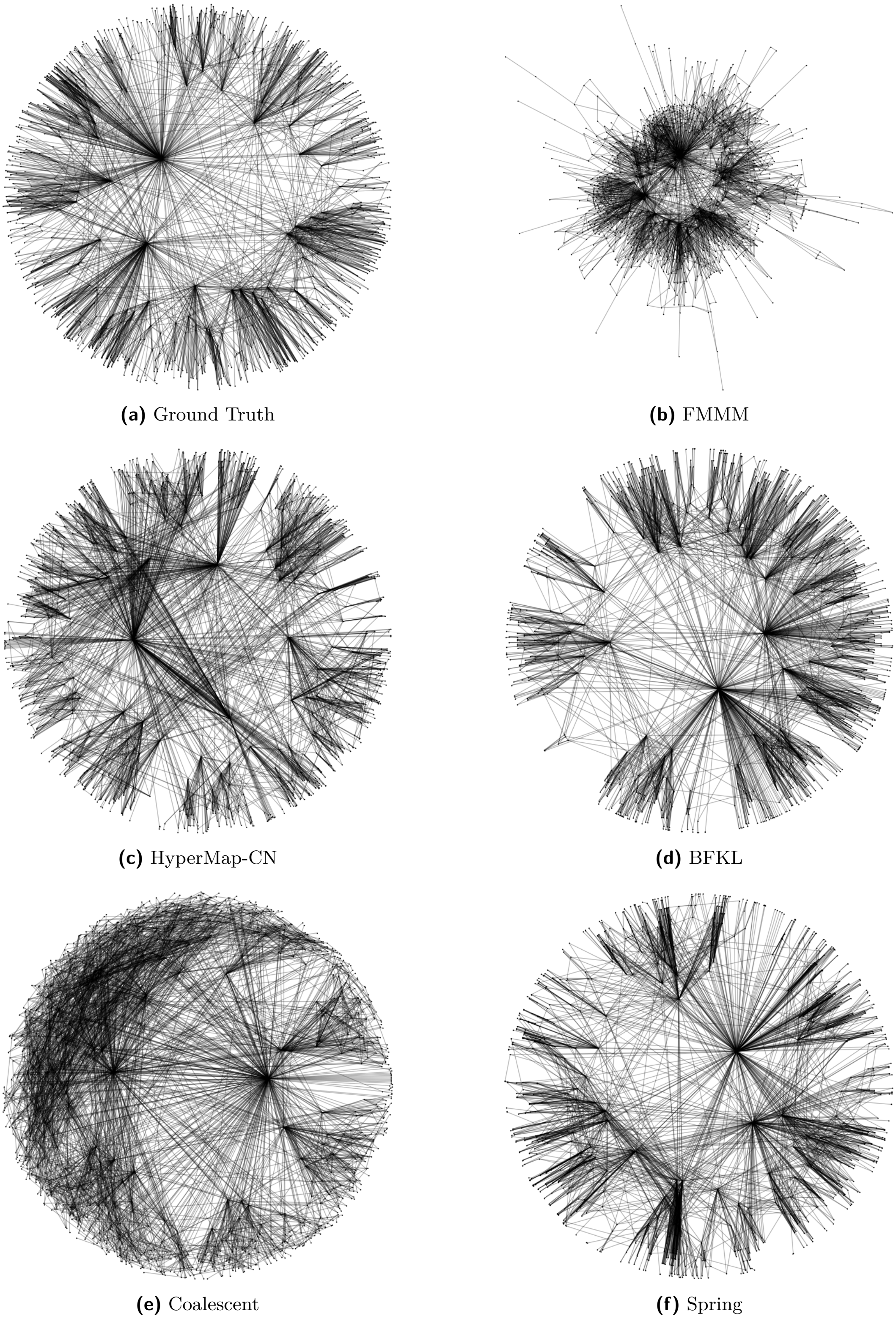
rt-twitter-copen (Figure 10)

Similarly to the embeddings in Figures 8a and 9a, FMMM succeeds in making the edges short, which leads to a small average error. However, in order to achieve this, many of the non-adjacent lower-degree vertices are placed close to the high-degree neighbors and, thus, close to each other. Consequently, the small edge-error is counteracted by a comparatively large non-edge error (denoted by the hollow-circle in Figure 7). While BFKL seemed to be unable to find a good initial placement of the higher-degree vertices, the remaining techniques produced reasonable results. Most notably, the spring embedder distributed the low-degree vertices near the boundary of the embedding and placed them close to their higher-degree neighbors. As a result, it obtained the best combination of average and balancing error.

inf-euroroad (Figure 11)

The underlying geometry of the road network *inf-euroroad* is arguably rather Euclidean than hyperbolic. Therefore, it is no surprise that FMMM, the only considered embedder that works with Euclidean geometry, obtained the best results here. The corresponding embedding in Figure 11a resembles that of a road network and is, therefore, a good representation of the structure of the graph. This is reflected by very small average and balancing errors, as shown in Figure 7.

In heterogeneous networks we interpret the degrees of the vertices as a measure of their popularity. In hyperbolic embeddings this is represented by the radial coordinates of the vertices: a vertex that is closer to the center has a larger popularity (higher degree). However, road networks are rather homogeneous, meaning most vertices have similar degree and, thus, similar popularity. The hyperbolic embedders represent this by placing all vertices in a ring close to the boundary of the embedding. The differences in the qualities of the embeddings is, thus, revealed by how well the techniques captured the similarities of the vertices, i.e., how close adjacent vertices are positioned in the ring. This is represented by the amount of edges that go through the center of the embedding. Clearly, the coalescent embedder captured this best, followed closely by the spring embedder, with HyperMap-CN and BFKL rather far off. The exact same ranking is reflected in the corresponding average errors in Figure 7.



(a) Ground Truth

(b) FMMM

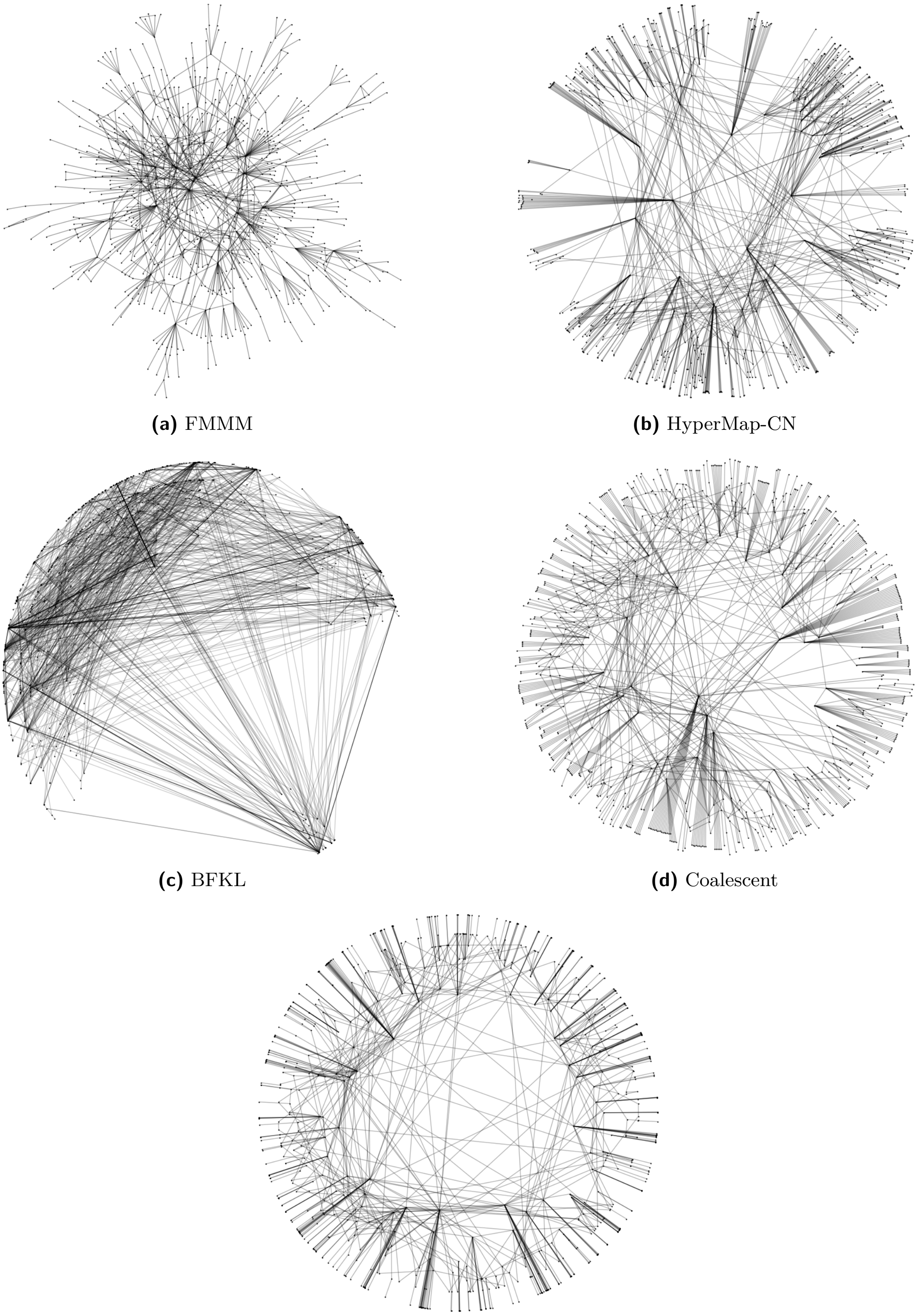
(c) HyperMap-CN

(d) BFKL

(e) Coalescent

(f) Spring

Figure 9 Embeddings of the largest component (containing 970 vertices) of a hyperbolic random graph (with average degree 7.6, power-law exponent 2.5, and temperature 0.4), obtained using different techniques.



(a) FMMM

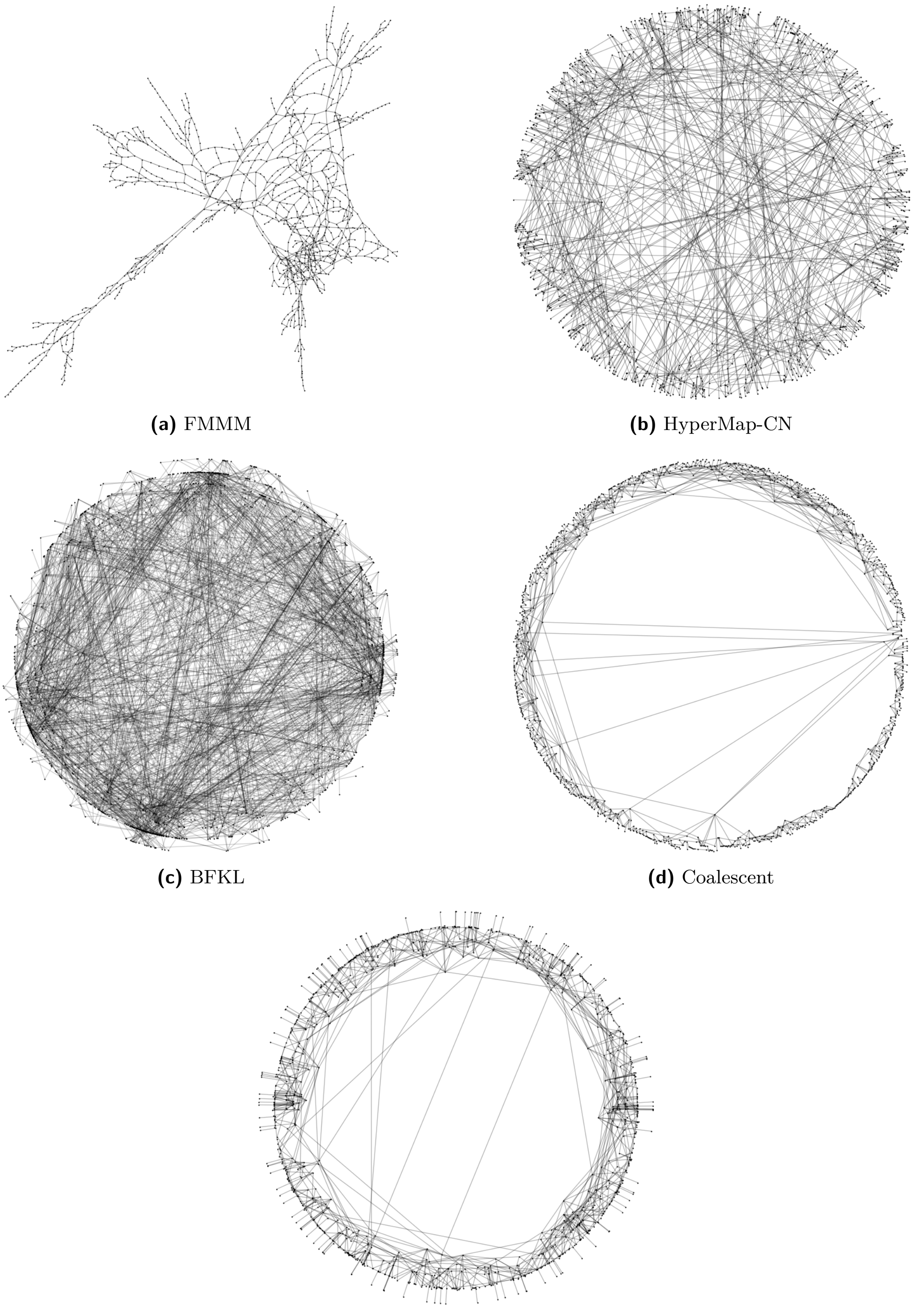
(b) HyperMap-CN

(c) BFKL

(d) Coalescent

(e) Spring

Figure 10 Embeddings of the largest component (containing 761 vertices) of the *rt-twitter-copen* network, obtained using different techniques.



(a) FMMM

(b) HyperMap-CN

(c) BFKL

(d) Coalescent

(e) Spring

Figure 11 Embeddings of the largest component (containing 1174 vertices) of the *inf-euroroad* network, obtained using different techniques. Here, FMMM obtained the best representation of the structure of the network.