Hyperbolic Embeddings for Near-Optimal Greedy Routing

Thomas Bläsius* Tobias Friedrich* Maximilian Katzmann* Anton Krohmer*

Abstract

Greedy routing computes paths between nodes in a network by successively moving to the neighbor closest to the target with respect to coordinates given by an embedding into some metric space. Its advantage is that only local information is used for routing decisions. We present different algorithms for generating graph embeddings into the hyperbolic plane that are well suited for greedy routing. In particular our embeddings guarantee that greedy routing always succeeds in reaching the target and we try to minimize the lengths of the resulting greedy paths. We evaluate our algorithm on multiple generated and real wold networks. For networks that are generally assumed to have a hidden underlying hyperbolic geometry, such as the Internet graph [2], we achieve near-optimal results, i.e., the resulting greedy paths are only slightly longer than the corresponding shortest paths. In the case of the Internet graph, they are only 6% longer when using our best algorithm, which greatly improves upon the previous best known embedding, whose creation required substantial manual intervention.

1 Introduction.

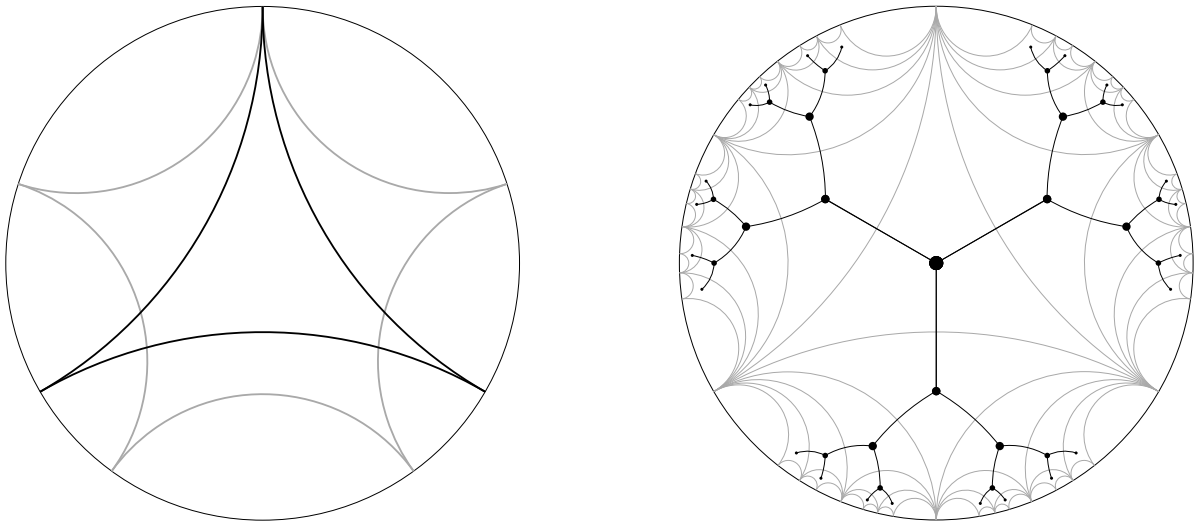
The Internet is the largest computer network in the world. At its core, it relies on one simple process: forwarding information from one participant of the network to another. This is done by storing a part of the network topology in each node, for example in the form of forwarding tables. On its way to the destination, data is then passed from node to node using this information. With increasing network size this method becomes infeasible due to the vast amounts of information each node needs to successfully route messages. In their seminal paper, Boguná, Papadopoulos, and Krioukov [2] propose to use greedy routing in the hyperbolic plane to solve this issue: They embedded the Internet into the hyperbolic plane by assigning a hyperbolic coordinate to every autonomous system. A message is then routed by always sending it to the neighbor of the current node closest to the destination (with respect to the hyperbolic distance). They achieve a *success ratio* of 97% (i.e., for 97% of all vertex pairs, greedy routing finds a path without getting stuck in a dead end) and a *stretch* of 10% (i.e., the resulting paths are on average 10% longer than the shortest paths).

As the method used by Boguná et al [2] to create their hyperbolic embedding "require[d] substantial manual intervention and do[es] not scale to large networks" [9], there have been multiple attempts to recreate comparable results purely algorithmically [11, 10, 1]. These approaches are based on the assumption that the input graph has a hidden underlying hyperbolic geometry that has to be rediscovered. Though these algorithms achieve this task fairly well, the success ratio for greedy routing on the Internet graph does not come close to the 97% of Boguná et al [2]. In fact, recent experiments indicate that even a perfect recovery of the "lost" hyperbolic coordinates would only lead to success ratios of 80% [1].

Instead of trying to rediscover hidden information, we propose to use algorithms tailored towards greedy routing. As shown by Kleinberg [8], every graph has an embedding in the hyperbolic plane with 100% success ratio (but potentially high stretch). The idea of Kleinberg's algorithm is to embed a spanning tree such that greedy routing on this tree is always successful. This property then extends to the whole graph. Kleinberg's approach has been extended in two ways. Eppstein and Goodrich [5] address the issues of coordinates becoming too large. Cvetkovski and Crovella [4] show how to compute greedy embeddings that allow for changes in a dynamic network without having to recompute the whole embedding. Though Kleinberg [8] as well as Cvetkovski and Crovella [4] ran experiments on small test instances, the algorithms have neither been evaluated with the focus on achieving a good stretch nor have they been applied to large networks like the Internet graph.

In this paper, we identify the main degrees of freedom in Kleinberg's embedding method (actually we use a slight variation that is more space efficient), provide strategies of how to fill them to achieve a good stretch, and evaluate the different methods on generated graphs and on real-world networks from different areas, such as biological networks, social networks, and infrastructural networks. For the above-mentioned Internet graph, our algorithm generates hyperbolic coordinates that enable greedy routing with 100% success ratio and 6% stretch.

*Hasso Plattner Institute, Potsdam, Germany
firstname.lastname@hpi.de



(a) (b)

Figure 1: **(a)** An ideal triangle (black) and an ideal 5-gon (gray) centered at the origin of the Poincaré Disk. All sides have equal distance to the origin. **(b)** The first five levels of an $\{\infty, 3\}$-tiling of the hyperbolic plane (gray) and its dual (black).

Outline. Section 2 provides a brief overview of the hyperbolic plane and the tools that are necessary to create greedy embeddings. Afterwards, Section 3 explains how these tools are applied to obtain greedy embeddings algorithmically, while Section 4 discusses how the degrees of freedom of this algorithmic approach can be filled in order to improve the quality of the embedding. Thereafter, an experimental evaluation of our algorithm is given in Section 5, followed by a discussion on potential numerical difficulties in Section 6.

2 Preliminaries.

The *Poincaré disk* is a model that represents the hyperbolic plane by mapping it to the interior of a Euclidean unit disk. As all illustrations in this paper use the Poincaré disk, we introduce its basic properties in the following.

The center of the Poincaré disk represents an arbitrarily chosen origin of the hyperbolic plane. The circle bounding it is called *boundary circle*. The closer a point lies to the boundary circle, the larger is its distance from the origin. The points on the boundary circle (not actually belonging to the hyperbolic plane) can be interpreted as having infinite distance to the origin. They are called *ideal points*. Straight lines in the hyperbolic plane map to circular arcs perpendicular to the boundary circle. Thus, each straight line connects two ideal points.

Let $\iota_0, \dots, \iota_{k-1}$ be k ideal points appearing in this cyclic order around the boundary circle. The *ideal k-gon* for these points consists of k lines, each connecting two consecutive ideal points ι_i and ι_{i+1} (mod k). These lines are the *sides* of the ideal k-gon. Figure 1a shows an ideal triangle. The ideal k-gons we consider are typically *regular*, which means that they have a *center* that has equal distance to all sides of the k-gon. This is equivalent to requiring the ideal points to be evenly distributed on the boundary circle of the Poincaré disk when using the center of the k-gon as origin.

Note that the sides of an ideal k-gon are parallel lines as they do not actually intersect (the ideal points are not part of the hyperbolic plane). Thus, the ideal triangle in Figure 1a separates the hyperbolic plane into an interior part (which has actually constant area π) and three non-intersecting half-planes (note that it is impossible to have three non-intersecting half-planes in the Euclidean plane). For each of the three half-planes, we can use its boundary line as one side of another ideal triangle that separates the half-plane into another interior and two more non-intersecting half planes. Applying this step recursively to all newly appearing half planes leads to a so-called $\{\infty, 3\}$-*tiling*, which divides the hyperbolic plane uniformly into ideal triangles. Every triangle shares each of its sides with another triangle and every corner is part of infinitely many ideal triangles. To get a symmetric tiling, the ideal triangles are chosen in such a way that the line separating two triangles is the perpendicular bisector of

their centers. Figure 1b shows a $\{\infty, 3\}$-tiling in the Poincaré Disk. Connecting the centers of every pair of adjacent ideal triangles yields the dual of the $\{\infty, 3\}$-tiling: an embedding of the infinite complete binary tree in which each edge has the same length. These statements directly extend to k-gons and $\{\infty, k\}$-tilings of higher order than 3.

Though the Poincaré disk is well suited to get an intuition, our computations are typically done in the *native representation*. In the native representation, we use an arbitrarily chosen origin together with a ray starting in the origin as reference. Then every point p is identified by its radial coordinates (r, φ), where r is the distance between p and the origin and φ is the angle between the reference ray and the ray from the origin through p.

3 Greedy Embedding.

Let $G = (V, E)$ be a graph together with an *embedding* $\mathcal{E}$ into a metric space (i.e., $\mathcal{E}$ is a function that maps V into the metric space). Given a source $s \in V$ and a target $t \in V$, greedy routing aims to find a path from s to t with only local knowledge. Starting at s, in each step the next vertex on the path is chosen to be the neighbor of the current vertex that is closest to t, where the embedding into the metric space is used to determine the distance between two vertices. Greedy routing is *successful* if the target t is reached without getting stuck in a dead end. A necessary and sufficient requirement for successful greedy routing is that every vertex v has a neighbor that is closer to t than v itself (with respect to the metric). Thus, greedy routing *fails* at vertex v if v has no neighbor that is closer to the target than v. We say that the embedding $\mathcal{E}$ is *suitable for greedy routing*, if greedy routing (with respect to $\mathcal{E}$) is successful for every pair of vertices. Assume we have a spanning tree T of G together with an embedding that is suitable for greedy routing in T. Then this property extends to the whole graph, i.e., the embedding is also suitable for greedy routing in G [8, Observation II.1]. In the remainder of this section, we thus only consider greedy embeddings of trees.

3.1 Characterizing all Greedy Embeddings of a Tree.

In the following theorem we characterize what makes an embedding of a tree into a (hyperbolic or Euclidean) space suitable for greedy routing. It is a generalization of Kleinberg's proof of correctness [8, Section II-C] and extends the sufficient condition presented by Cvetkovski and Crovella [4, Lemma 2] to a complete characterization. Given a tree $T = (V, E)$, we use T_u and T_v to denote the two connected components in $T - \{u, v\}$, obtained by removing the edge $\{u, v\}$ from T, where T_u contains u and T_v contains v.

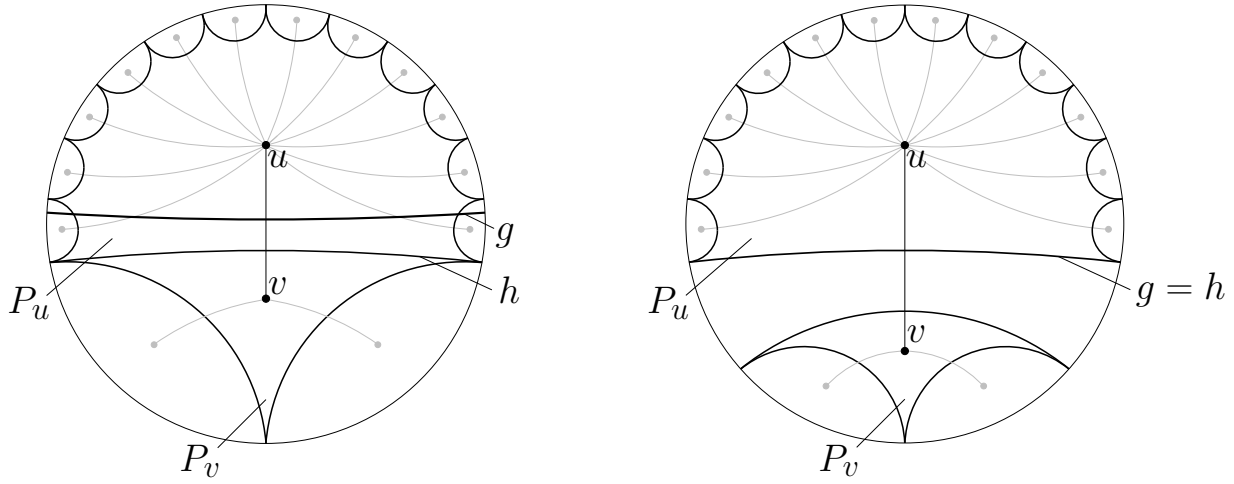
THEOREM 3.1. *Let $\mathcal{E}$ be an embedding of a tree $T = (V, E)$ into an Euclidean or hyperbolic space. Then $\mathcal{E}$ is suitable for greedy routing if and only if for every edge $\{u, v\} \in E$, the perpendicular bisector of $\overline{uv}$ separates T_u from T_v.*

Proof. To prove that $\mathcal{E}$ is suitable for greedy routing, we need to show the necessary and sufficient requirement that, given a target vertex $t \in V$, at every vertex $s \in V$ there is a vertex $v \in V$ that is closer to t than s. Therefore, let $s \neq t$ be any two vertices in V. Furthermore, let $\pi_{st} = s, v, \ldots, t$ be the unique path from s to t in T. It suffices to show that $d(v, t) < d(s, t)$. We know that the perpendicular bisector g of the geodesic $\overline{sv}$ separates T into the two components T_s and T_v of $T - \{s, v\}$, such that s is on one side of g, and v and t are on the other. Using the triangle inequality we can conclude that all points that are on v's side of g are closer to v than to s. It follows that $d(v, t) < d(s, t)$.

Conversely, let $\{s, v\} \in E$ be the edge whose perpendicular bisector g does not separate T into the two connected components T_s and T_v. Assume without loss of generality that there exists a node $t \in T_v$ that is on the same side of g as s. Then t is closer to s than v, and thus $d(s, t) < d(v, t)$ holds. When navigating from s to t, the distance to the target is increased by going to v. Therefore, greedy routing will fail at s.

3.2 Adaptive Tree Embedding Recall from Section 2 that a line separating two adjacent ideal k-gons in a regular $\{\infty, k\}$-tiling is the perpendicular bisector of the line segment connecting the centers of these k-gons. Thus, the corresponding embedding of the infinite k-regular tree (obtained by taking the dual of the regular $\{\infty, k\}$-tiling) satisfies Theorem 3.1 and is thus suited for greedy routing. This observation forms the basis of Kleinberg's embedding algorithm, which works as follows: The spanning tree T is mapped into the infinite k-regular tree, where k is the maximum degree of T. The dual of the uniform $\{\infty, k\}$-tiling then gives an embedding of the infinite k-regular tree, inducing an embedding of T, which is greedy due to the above observation.

Note that a large maximum degree k has two effects on the coordinates in this embedding procedure. In each layer of the tiling, the currently available angle is separated into $k - 1$ sectors. Thus, for larger k, the differences between angular coordinates of distinct vertices decrease quickly, making it necessary to use coordinates with high precision. Similarly, the distance between the center and the sides of a regular k-gon increases with growing k, which leads to large radial



(a) (b)

Figure 2: **(a)** An ideal 13-gon P_u and an ideal triangle P_v are sharing the side h. Their perpendicular bisector g does not seperate the tree into two connected components. **(b)** The distance between P_u and P_v is increased such that the perpendicular bisector g lies on the side h of the larger polygon P_u.

coordinates. Thus, setting k to the maximum degree of T seems to be a waste of space, in particular if most vertices of T have much smaller degree. In order to decrease the size of the coordinates we propose the *adaptive embedding*, a variation of Kleinberg's method that adapts the size of the ideal polygons to the actual vertex degrees.

To explain the adaptive embedding, consider the tree $T = (V, E)$ with two adjacent vertices $u, v \in V$. The vertex u is placed at the center of an ideal $\deg(u)$-gon P_u while v is placed at the center of an ideal $\deg(v)$-gon P_v and (for now) assume that both polygons share a side. This situation is depicted in Figure 2a, where h is the side shared by P_u and P_v. Observe that there is an important difference compared to a regular $\{\infty, k\}$-tiling: as the distance between the center and the sides of an ideal k-gon increases with growing k, the line h is not the perpendicular bisector of the edge $\{u, v\}$. Figure 2a also shows the actual perpendicular bisector g. Thus, if the difference between $\deg(u)$ and $\deg(v)$ is large enough, the perpendicular bisector of $\{u, v\}$ does not separate T into T_u and T_v (again, see Figure 2a). Thus, the embedding does not satisfy Theorem 3.1 and is not suitable for greedy routing. However, this issue can be easily fixed by introducing a gap between P_u and P_v such that the side of the larger polygon actually is the perpendicular bisector; in Figure 2b, the gap ensures that the side h of the polygon P_u equals the perpendicular bisector g of $\{u, v\}$.

To actually implement the adaptive embedding, we need to know how large the gap between two ideal polygons of different size has to be chosen. Thus, we need to know the distance $\ell(k)$ between the center and the sides of an ideal k-gon, which we determine in the following lemma. We note that for $k \to \infty$ the distance $\ell(k)$ behaves like $\log(k)$.

LEMMA 3.1. *The distance $\ell(k)$ between the center and the sides of an ideal k-gon is given by*

$$\ell(k) = 2 \cdot \operatorname{arctanh}\left(\sec(\pi/k) - \tan(\pi/k)\right).$$

Proof. By taking advantage of the representation of the Poincaré disk as the unit circle in the Euclidean plane, we can determine the desired distance using the Euclidean geometry and convert it into the hyperbolic geometry afterwards. Let $\iota_0, \ldots, \iota_{k-1}$ be the ideal points of an ideal k-gon that, without loss of generality, is centered at the origin of the disk. Recall from Section 2 that the ideal points are therefore evenly distributed on the boundary circle, meaning the angle between the rays from the origin O through two consecutive points is $2\pi/k$. Additionally, it suffices to consider the geodesic of any two consecutive ideal points ι_i and ι_{i+1} (mod k). Again, without loss of generality we can assume that ι_i is placed at $(1, 0)$ and that ι_{i+1} is its counterclockwise successor; see Figure 3. Since the geodesic connecting ι_i and ι_{i+1} (mod k) is a circular arc that meets the boundary perpendicularly, it remains to determine the distance between the circular arc and the origin O of the disk. Now the center C of the corresponding circle lies on the angle bisector of the rays through ι_i and ι_{i+1} (mod k) and thus its distance to

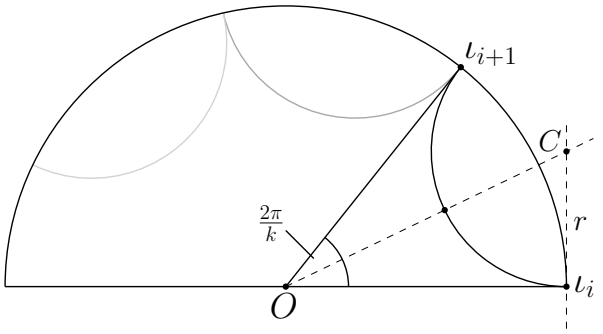


Figure 3: Determining the distance between a side of an ideal k-gon and its center. An excerpt of the Poincaré disk is shown.

the origin is given by $\sec(\pi/k)$; see Figure 3. Additionally, the radius r of the circle is $\tan(\pi/k)$. The distance between the origin O and the geodesic is thus given by $\sec(\pi/k) - \tan(\pi/k)$. Finally, we convert this Euclidean distance in the Poincaré disk to the hyperbolic distance and obtain the above equation.

Now we describe the adaptive embedding algorithm. Given a tree $T = (V, E)$, we choose an arbitrary root r and place it at the origin, in the center of an ideal $\deg(r)$-gon P_r. Afterwards we proceed with each child independently. When processing a vertex v, we know the coordinates of its parent u and we know a side h of the $\deg(u)$-gon P_u with center u that should separate u from v; see Figure 2b. Then we place v onto the line perpendicular to h such that v has distance $\ell_{u,v} = 2\max\{\ell(\deg(u)), \ell(\deg(v))\}$ from u and lies on the other side of h than u. Recall that $\ell(k)$ denotes the distance between the center and the sides of an ideal k-gon; see Lemma 3.1. Now that we know the coordinates of v, we add the ideal $\deg(v)$-gon P_v with center v. Afterwards we recursively proceed with the children of v.

Note that the choice of the distance $\ell_{u,v}$ makes sure that the two ideal polygons P_u and P_v do not intersect and that in fact the perpendicular bisector of $\{u, v\}$ is a side of P_u or of P_v (depending on whether $\deg(u)$ or $\deg(v)$ is larger). Thus, the resulting embedding of the tree satisfies Theorem 3.1 and we obtain the following theorem.

THEOREM 3.2. *The adaptive embedding of a tree is suitable for greedy routing.*

4 Degrees of Freedom.

Our adaptive embedding leaves two degrees of freedom: the choice of the spanning tree that is embedded, and the ordering of the children around each node. In the following we discuss these degrees of freedom more closely and propose different strategies of how to fill them.

4.1 Choice of the Spanning Tree. The first degree of freedom is the choice of the spanning tree that is embedded. When choosing a spanning tree as the basis for the embedding, we aim for two desirable properties. First, the stretch of the resulting embedding should be as low as possible, and second the height of the spanning tree should be small in order to prevent numerical problems. The first type of trees we consider are trees obtained from a breadth first search starting at a random node in the graph. The resulting BFS-trees have a small height (achieving our second goal). They are, however, not optimized for achieving a good stretch. In the following we thus propose a different choice for the tree focusing on the stretch.

Since the stretch is the ratio between the length of the shortest paths and the routes obtained when routing the network greedily, using a spanning tree that contains edges that are part of many shortest paths should result in a good stretch. Therefore, we aim to find a tree that maximizes the *edge betweenness centrality*, which measures how many shortest paths go through an edge. Given a graph $G = (V, E)$ the betweenness centrality of an edge $e \in E$ is defined as

$$\mathrm{bc}(e) = \sum_{v \in V} \sum_{w \in V} \frac{\sigma_{vw}(e)}{\sigma_{vw}},$$

where σ_{vw} is the number of shortest paths between v and w and $\sigma_{vw}(e)$ is the number of shortest paths between the two vertices that also contains edge e. Good approximations for the betweenness centralities of the edges in a graph can be obtained quickly using the algorithm *KADABRA* [3]. We then use the betweenness centrality values as edge weights and compute a maximum spanning tree. That way, edges that are crucial for information flow in the network are more likely to be part of the tree than other edges.

4.2 Ordering Children. When embedding a tree the order of the children of a node defines the relative positions between their subtrees. Consequently, the node order influences the geometric length of non-tree edges in the embedded graph. Intuitively, having short edges (and long non-edges) leads to geometric distances more similar to the graph-theoretic distances, which decreases the chances of a detour on the greedy path. Thus, we want adjacent vertices to be close and non-adjacent vertices to be farther apart.

The first strategy we propose to achieve this is to reorder the subtrees at each vertex in the spanning tree, such that subtrees that are connected by more

edges are placed closer to each other. This problem can be modelled using a weighted conflict graph, in which each node represents a subtree and weighted edges represent how many edges (of the original graph) connect the corresponding subtrees. For this conflict graph, we then have to solve the problem Optimal Cyclic Arrangement (OCA). Formally, the input of OCA is a weighted, undirected graph $G = (V, E)$ with weight-function $w\colon E \to \mathbb{N}$. Given a cyclic ordering (i.e., a bijection that maps the vertices V to a cycle of length n), the cost of an edge $\{u, v\} \in E$ is the distance between u and v in the cyclic ordering multiplied with its weight $w(\{u, v\})$. The cost of the ordering is the sum of all edge costs. The goal of OCA is to find a cyclic ordering with minimum cost. Unfortunately, OCA is NP-hard, as the NP-hard problem Optimum Linear Arrangement [7] can be easily reduced to it. In our experiments, we therefore use a heuristic that iteratively improves the order greedily.

In our second strategy, to obtain short edges and distant non-edges, we make use of the fact that embeddings into the hyperbolic plane with these properties have been studied before, typically under the name *maximum likelihood embedding (MLE)* [11, 10, 1]. As already mentioned in the introduction, maximum likelihood embeddings do typically not lead to successful greedy routing. However, such an embedding determines for each vertex a cyclic order of all its neighbors, which we can copy to order the children of vertices in the tree. The intuition behind this is that these orderings led to short edges in the maximum likelihood embedding and thus probably also lead to short non-tree edges in our embedding. In our experiments, we use the current state-of-the-art maximum likelihood embedder [1].

5 Experimental Evaluation.

With our experimental evaluation, we want to answer the following main questions, where the last question refers to the fact that our implementation[1] prevents numerical difficulties by utilizing the multiple-precision library MPFR [6].

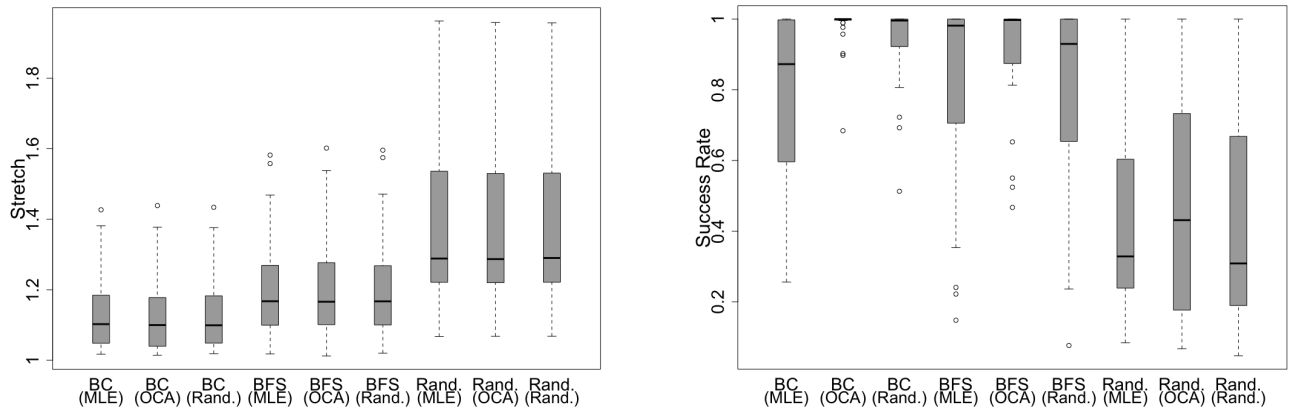
- How do different spanning trees and different child orders impact the stretch?
- How well does greedy routing perform on certain real-world networks, e.g., the Internet?
- Can we obtain good greedy embeddings using double coordinates (which represent less information stored in each vertex compared to multiple-precision coordinates)?

Experimental Setup. To evaluate the impact of the different degrees of freedom on the stretch, we implemented the adaptive embedding described in Section 3 as well as the different strategies of filling the degrees of freedom discussed in Section 4. Recall that these choices are as follows. For the spanning tree, we either choose a BFS-tree or a tree that maximizes the betweenness centrality, which we abbreviate with *BC-tree*. For the node order, we either heuristically solve an instance of Optimal Cyclic Arrangement (OCA) for each vertex of the tree to determine an order of its children or we derive the orders from a single maximum likelihood embedding (MLE). To have a baseline to which we can compare these strategies, we also ran experiments with random spanning trees and a random child order. As the spanning tree appears to have a high impact on the stretch, we generated 10 random spanning trees and used the mean of the resulting stretch values. Note that the strategies for choosing a spanning tree and node orders can be combined arbitrarily. Thus, with the three choices for the spanning tree (BFS, BC, Rand.), and the three choices for the node orders (OCA, MLE, Rand.), we obtain nine combinations in total.

The instances we consider are a combination of real-world networks of different types and generated hyperbolic random graphs [9]. More precisely, we chose 22 real-world networks from The Network Repository [12] coming from different domains such as social networks, biological networks, or infrastructural networks. We also included networks with a naturally underlying Euclidean geometry (such as road networks) for which we expected greedy routing to perform poorly. In addition to that, we used the Internet graph that was considered for greedy routing before [2]. Moreover, we ran our experiments on nine generated hyperbolic random graphs with varying power-law exponents (2.1, 2.5, and 2.9) and temperatures (0.1, 0.5, and 0.9). For all these graphs, we only embedded the largest connected component.

After embedding these networks, we computed the stretch values by sampling 10 000 vertex pairs uniformly at random and compared the shortest path between them to the route obtained by greedily navigating through the network. Moreover, we considered the embeddings obtained by rounding all coordinates to double values (we note that the computation still uses the multiple-precision library). For the resulting embeddings, we evaluated the stretch as well as the success rate (which is sometimes below 100% due to the rounding to double values).

[1] The implementation was done in C++ and our code is available at `https://hpi.de/friedrich/research/hyperbolicgreedyroutings`



(a) (b)

Figure 4: **(a)** Using different kinds of spanning trees as the basis of an embedding has an impact on the stretch. Varying the node order influences the stretch only marginally. **(b)** In the embeddings rounded to double coordinates, the success rate potentially drops below 100%. In contrast to the stretch values, the success rate heavily depends on the child orders (with OCA being the best strategy).

Evaluation. Concerning the first question (how filling the different degrees of freedom impacts the stretch), Figure 4a shows for different combinations of strategies the resulting stretch values (using high precision coordinates). The x-axis shows the different kinds of spanning trees with the node orders in parentheses and the y-axis denotes the stretch. Boxes show the median value at the center and extend to the 25th and 75th percentile, while the whiskers denote the 10th and 90th percentile.

One can clearly see that the choice of the spanning tree has a significant impact while the child order appears to be irrelevant. On average using random spanning trees results in a stretch of about 37.7%. Using a BFS-tree improves this value to about 20.9%. This can be reduced even further by using spanning trees that maximize the betweenness centrality, resulting in an average stretch of 13.7%.

In Table 1, we list for each graph the technique that led to the best stretch values together with these stretch values. On the one hand, this shows that for 93.75% of the graphs, using the BC-tree leads to the best stretch. Although the child order does not make a huge difference, using the combination of BC-tree together with OCA-order was the most successful. On the other hand, the table shows for which networks greedy routing is well suited. Besides the Internet graph with a stretch of 7% it is worth noting that there are other networks with very low stretch values. With 1.4%, the smallest value is obtained for the biological network *bn-fly-drosophila-medulla-1*. We note that very low stretch values are not surprising for graphs that are almost trees. However, *bn-fly-drosophila-medulla-1* with an average degree of above 10 is far away from being a tree. On the other hand, there are graphs with small average degree that lead to a high stretch, e.g., *inf-euroroad* has an average degree of 2.5 and a stretch value of 20.7%. Note that this indicates that the metric of *inf-euroroad* is (non-surprisingly for a road network) rather different from the metric of the hyperbolic plane.

To obtain embeddings with small coordinates, we rounded each coordinate (computed with the multiple-precision library) to double values. With the resulting embeddings, greedy routing may lead to different paths, resulting in different stretch values. In fact, some of the resulting embeddings do not yield a success ratio of 100%. Table 1 thus includes for each graph the technique leading to the best success ratio, using the stretch values as tiebreakers. One can see that for most graphs, at least one of our embeddings resulted in 100% success rate (for all others the success rates are still very high). Moreover, the stretch values do not change much. For some graphs, they actually go down, e.g., for the internet graph, we obtain an embedding with double coordinates with 6.2% stretch (and 100% success rate).

To see whether some of our strategies are more susceptible to numerical difficulties than others, we plotted the success rates for the embeddings with double coordinates depending on the used strategy in Figure 4b. One can see that in this situation the different node orders actually matter, which was not the case for the stretch values; see Figure 4a. Especially the OCA ordering appears to be more robust than the other ordering strategies.

Table 1: Stretch values and success rates for all graphs. For embeddings using the multiple-precision library MPFR, the technique leading to the best stretch value together with the stretch value itself is shown. The success rates for these embeddings are 100%. For the embeddings rounded to double coordinates, the table shows the technique resulting in the best success rate, using the stretch values as tiebreakers, together with the corresponding success rates and stretch values. All values refer to the largest connected component in the graph.

Network Name	Nodes	Avg. Deg.	Stretch (MPFR)	Technique (MPFR)	Success (double)	Stretch (double)	Technique (double)
bio-celegans	453	8.9	4.4%	BC OCA	1.0000	4.7%	BC OCA
bio-celegans-dir	453	9.0	4.3%	BC OCA	1.0000	4.5%	BC OCA
bn-fly-drosophila-medulla-1	1770	10.1	1.4%	BC OCA	1.0000	1.5%	BC OCA
bn-mouse-retina-1	1076	168.8	14.7%	BC MLE	1.0000	16.6%	BFS MLE
chem-ENZYMES-g123	90	2.8	42.7%	BC MLE	1.0000	40.5%	BC MLE
chem-ENZYMES-g118	95	2.5	18.9%	BC OCA	1.0000	18.9%	BC Rand
ca-CSphd	1025	2.0	7.4%	BC MLE	1.0000	7.9%	BC OCA
ca-Erdos992	4991	3.0	26.0%	BC Rand	1.0000	25.5%	BC OCA
ca-GrQc	4158	6.5	36.5 %	BC MLE	0.9974	44.3%	BFS OCA
soc-advogato	5054	16.1	15.7 %	BC OCA	1.0000	16.3%	BC OCA
soc-anybeat	12645	7.8	3.7%	BC OCA	1.0000	4.3%	BC Rand
soc-gplus	23613	3.3	8.3%	BC Rand	0.9574	8.7%	BC OCA
soc-hamsterster	2000	16.1	21.6%	BC OCA	1.0000	25.5%	BFS MLE
soc-wiki-Vote	889	6.6	15.4%	BC OCA	1.0000	15.0%	BC Rand
ia-email-EU	32430	3.4	13.6%	BC OCA	0.9983	13.5%	BC OCA
ia-email-univ	1133	9.6	36.5 %	BC Rand	1.0000	36.2%	BFS OCA
ia-reality	6809	2.3	8.1%	BC OCA	1.0000	8.5%	BC OCA
inf-euroroad	1039	2.5	20.7%	BC OCA	1.0000	22.2%	BFS Rand
inf-power	4941	2.7	13.4%	BC Rand	0.9893	13.6%	BC OCA
infOpenFlights	2905	10.8	11.9%	BC OCA	1.0000	11.0%	BFS OCA
internet	13204	6.8	7.0%	BC OCA	1.0000	6.2%	BC OCA
as-caida	26475	4.0	8.0%	BC MLE	0.8974	7.9%	BC OCA
lp-afiro	51	4.0	27.0%	BFS MLE	1.0000	27.1%	BFS MLE
HRG$(0.1, 2.1)$	4932	4.5	1.2%	BFS OCA	1.0000	1.9%	BC OCA
HRG$(0.1, 2.5)$	9527	8.0	2.9%	BC OCA	1.0000	2.1%	BFS OCA
HRG$(0.1, 2.9)$	9747	8.5	2.5%	BC OCA	0.9998	2.6%	BC OCA
HRG$(0.5, 2.1)$	4567	3.9	2.2%	BC OCA	1.0000	2.1%	BC OCA
HRG$(0.5, 2.5)$	9712	7.8	3.3%	BC OCA	1.0000	3.2%	BC OCA
HRG$(0.5, 2.9)$	9863	6.7	10.0%	BC OCA	1.0000	10.2%	BC OCA
HRG$(0.9, 2.1)$	3592	3.0	3.2%	BC MLE	1.0000	3.2%	BC Rand
HRG$(0.9, 2.5)$	8254	3.7	9.8%	BC Rand	0.9998	10.5%	BC OCA
HRG$(0.9, 2.9)$	8456	3.4	16.7%	BC OCA	0.9767	17.0%	BC OCA

Overall, one can conclude that using spanning trees that maximize the betweenness centrality lead to very good stretch values. Moreover, combining it with the OCA heuristic makes it very robust when rounding to double coordinates.

6 Precision of the Coordinates

Recall from the previous section that we use a multiple-precision library to compute greedy embeddings. This is rather unsatisfying for the following reason. The benefit of using greedy routing instead of other routing techniques is that each vertex has to know only a small amount of information (namely the coordinates) to be able to route successfully. If, however, these coordinates need to be very precise, then the amount of information that is stored at each vertex is not that small after all. For this reason, we evaluated our greedy embeddings not only with the precise coordinates but also with coordinates rounded to double values.

Our experiments (see Table 1) showed that we were actually able to obtain embeddings with double coordinates, 100% success ratio, and good stretch values for most networks. Nonetheless, each of our algorithms had problems with some graphs (see Figure 4b), and thus it would be good to have guarantees for how precise the coordinates have to be to ensure successful greedy routing. To resolve this question, Eppstein and Goodrich [5] gave an algorithm that generates succinct greedy embeddings in which vertex positions are represented with only $O(\log n)$ bits per vertex. Unfortunately, our implementation of their algorithm failed to produce embeddings with 100% success rate and we in fact believe that the proof is flawed, as briefly argued in the following.

The embedding algorithm [5] first embeds a spanning tree T into a so-called dyadic tree metric which is then embedded into the hyperbolic plane. One main idea behind the dyadic tree metric is to contract certain paths in the tree T such that the height of the resulting tree is logarithmic. In the following, we call the resulting tree T'. Then the routing between vertices in T is done based on their positions in T'. To make this work, one needs to use tie-breakers to distinguish vertices of T that were contracted to the same node in T'. These tie-breakers are basically given by an in-order of T (think of ordering the vertices of T from left to right). Now consider the following situation. Let u, s, and v be vertices of T that lie on a path π that was contracted to a single node in T'. Assume that $u < s < v$ according to the left-to-right order. When routing from s to another node t that does not belong to π, then the tiebreakers decide in which direction to move on π: if $t < s$, the routing walks towards u, if $s < t$, the routing walks towards v. However, it is not hard to construct an example that includes vertices u' and v' with $u' < u < v < v'$ such that both u' and v' are successors of predecessors of all vertices on π. This means that one reaches u' and v' from s by walking upwards on π. However, when routing from s to u', the tie-breaker lets us walk towards u, and when routing from s to v', the tie-breaker lets us walk towards v, which cannot both be upwards on π. Thus, at least one of these two routing queries fails. Though it might be possible to resolve this issue by using a different tie-breaking method, we failed to come up with one that breaks all ties correctly.

7 Conclusion

We presented the first practical evaluation of hyperbolic embedders that are tailored towards greedy routing. To that end, we proposed the adaptive embedding as an extension of Kleinberg's algorithm [8] that is more space efficient. Moreover, we proposed several methods to fill its degrees of freedom and evaluated their impact on the stretch. It turned out that a careful choice for the spanning tree greatly improved the quality of the resulting embeddings.

Although, our algorithms generated embeddings with 100% success rate for most graphs, even when rounding the coordinates to double values, numerical difficulties are a major issue. As can be seen in Figure 4b, the child orders had a big impact on these difficulties. It would thus be interesting to explore which types of orderings prevent numerical problems. From a theoretical point of view, a proof that small coordinates (e.g., with $\log n$ bits) are sufficient would be very interesting. As noted in Section 6, we believe that this is an open problem.

References

[1] Thomas Bläsius, Tobias Friedrich, Anton Krohmer, and Sören Laue. Efficient embedding of scale-free graphs in the hyperbolic plane. In *24th ESA*, volume 57, pages 16:1–16:18. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2016. doi:10.4230/LIPIcs.ESA.2016.16.

[2] Marián Boguná, Fragkiskos Papadopoulos, and Dmitri Krioukov. Sustaining the internet with hyperbolic mapping. *Nature Communications*, 1:62, 2010. doi:10.1038/ncomms1063.

[3] Michele Borassi and Emanuele Natale. KADABRA is an ADaptive Algorithm for Betweenness via Random Approximation. In *24th ESA*, pages 20:1–20:18, 2016. doi:10.4230/LIPIcs.ESA.2016.20.

[4] Andrej Cvetkovski and Mark Crovella. Hyperbolic embedding and routing for dynamic graphs. In *28th*

INFOCOM, pages 1647–1655, 2009. doi:10.1109/INFCOM.2009.5062083.

[5] David Eppstein and Michael T. Goodrich. Succinct greedy geometric routing using hyperbolic geometry. *IEEE Trans. Comput.*, 60(11):1571–1580, 2011. doi:10.1109/TC.2010.257.

[6] Laurent Fousse, Guillaume Hanrot, Vincent Lefèvre, Patrick Pélissier, and Paul Zimmermann. MPFR: A Multiple-precision Binary Floating-point Library with Correct Rounding. *ACM Trans. Math. Softw.*, 33(2), 2007. doi:10.1145/1236463.1236468.

[7] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1979.

[8] Robert Kleinberg. Geographic routing using hyperbolic space. In *26th INFOCOM*, pages 1902–1909. IEEE, 2007. doi:10.1109/INFCOM.2007.221.

[9] 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.

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

[11] Fragkiskos Papadopoulos, Constantinos Psomas, and Dmitri Krioukov. Network mapping by replaying hyperbolic growth. *IEEE/ACM Trans. Netw.*, (99):198–211, 2014. doi:10.1109/TNET.2013.2294052.

[12] Ryan A. Rossi and Nesreen K. Ahmed. The Network Data Repository with Interactive Graph Analytics and Visualization. In *29th AAAI*, 2015. URL: http://networkrepository.com.