Hyperbolic Embeddings for Near-Optimal Greedy Routing

THOMAS BLÄSIUS, TOBIAS FRIEDRICH, MAXIMILIAN KATZMANN, and ANTON KROHMER, Hasso Plattner Institute, University of Potsdam

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-world networks. For networks that are generally assumed to have a hidden underlying hyperbolic geometry, such as the Internet graph [3], 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.

In addition to measuring the stretch, we empirically evaluate our algorithms regarding the size of the coordinates of the resulting embeddings and observe how it impacts the success rate when coordinates are not represented with very high precision. Since numerical difficulties are a major issue when performing computations in the hyperbolic plane, we consider variations of our algorithm that improve the success rate when using coordinates with lower precision.

CCS Concepts: • **Theory of computation → Random projections and metric embeddings**; • **Mathematics of computing** → *Trees*; • **Networks** → *Transport protocols*; *Network experimentation*;

Additional Key Words and Phrases: Greedy routing, geographic routing, hyperbolic space, spanning trees, stretch, robustness

ACM Reference format:
Thomas Bläsius, Tobias Friedrich, Maximilian Katzmann, and Anton Krohmer. 2020. Hyperbolic Embeddings for Near-Optimal Greedy Routing. *J. Exp. Algorithmics* 25, 1, Article 1.3 (March 2020), 18 pages.
https://doi.org/10.1145/3381751

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

This research received funding from the German Research Foundation (DFG) under grant agreement no. FR 2988 (ADLON).
Authors' address: T. Bläsius, T. Friedrich, M. Katzmann, and A. Krohmer, Hasso Plattner Institute, University of Potsdam, Prof.-Dr.-Helmert-Str. 2-3, Potsdam, 14482, Germany; emails: {thomas.blasius, tobias.friedrich, maximilian.katzmann, anton.krohmer}@hpi.de.

1084-6654/2020/03-ART1.3 $15.00
https://doi.org/10.1145/3381751

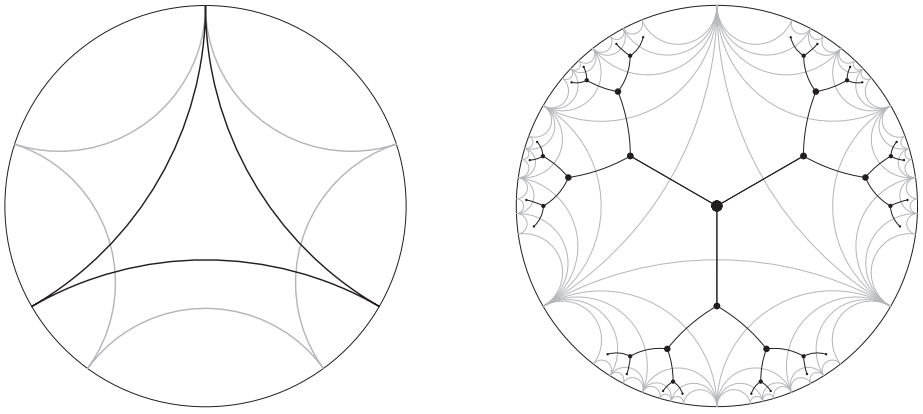
storing a part of the network topology in each node, such as in the form of forwarding tables. On its way to the destination, data is then passed from node to node using this information. Although in 1995 less than 1% of the world population used the Internet, today this number is close to 50%. As a result, the amount of information that needs to be stored in each node increases quickly. Eventually, it will become intractable to manage the vast amounts of data needed to successfully route messages between participants of the network.

In their seminal paper, Boguñá et al. [3] 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 node 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 Boguñá et al. [3] to create their hyperbolic embedding "require[d] substantial manual intervention and do[es] not scale to large networks" [10], there have been multiple attempts to recreate comparable results purely algorithmically [2, 11, 12]. These approaches are based on the assumption that the input graph has a hidden underlying hyperbolic geometry that has to be rediscovered. Although 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 Boguñá et al. [3]. In fact, recent experiments indicate that even a perfect recovery of the "lost" hyperbolic coordinates would only lead to success ratios of 80% [2].

Instead of trying to rediscover hidden information, we propose to use algorithms tailored towards greedy routing. As shown by Kleinberg [9], 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 three ways. Eppstein and Goodrich [6] address the issues of coordinates becoming too large. Cvetkovski and Crovella [5] show how to compute greedy embeddings that allow for changes in a dynamic network without having to recompute the whole embedding. Sarkar [15] considered embedding weighted trees into the hyperbolic plane such that the weights are realized as the hyperbolic distances between the corresponding nodes. Although Kleinberg [9], as well as Cvetkovski and Crovella [5], ran experiments on small test instances, the algorithms have not been evaluated with the focus on achieving a good stretch. Ban et al. [1] later took Kleinberg's approach and performed experiments on a small selection of larger real-world networks, observing that the choice of the spanning tree used for the embedding has an impact on the stretch.

In this article, we identify the main degrees of freedom in Kleinberg's embedding method (actually we use a slight variation, which is a specialized version version of the approach of Sarkar [15] and is more space efficient) and provide strategies of how to fill them to achieve a good stretch. We then evaluate the different methods regarding stretch, the size of the coordinates, success rate, and robustness to network failures. In our experiments, we consider generated graphs and real-world networks from different areas, such as biological networks, social networks, and infrastructural networks. For the Internet graph mentioned previously, our algorithm generates hyperbolic coordinates that enable greedy routing with 100% success ratio and 6% stretch. Furthermore, our experiments demonstrate that numerical difficulties are a major issue when performing computations in the hyperbolic plane. Our evaluation shows that coordinates may need to be represented with very high precision to reach perfect success rates. To counteract this issue, we consider how the representation of the embedding can be altered to obtain smaller coordinates.



(a) (b)

Fig. 1. (a) An ideal triangle (black) and an ideal 5-gon (gray) centered at the origin of the Poincaré disk. All sides of an ideal polygon 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. The rest of the article is organized as follows. Section 2 provides an overview of the hyperbolic plane and the tools necessary to create greedy embeddings. Section 3 explains how these tools are applied to obtain greedy embeddings algorithmically, whereas Section 4 discusses how the degrees of freedom of this algorithmic approach can be filled 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. We conclude in Section 7.

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 almost all illustrations in this article 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 the *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 $\iota_0, \ldots, \iota_{k-1}$ 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 1(a) shows an ideal triangle. The ideal k-gons that 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 the 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 1(a) 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 1(b) 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.

Although 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 polar 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 node on the path is chosen to be the neighbor of the current node 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 node v has a neighbor that is closer to t than v itself (with respect to the metric). Thus, greedy routing *fails* at node 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—for instance, the embedding is also suitable for greedy routing in G (Observation II.1 in Kleinberg [9]). 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 (Section II-C in Kleinberg [9]) and extends the sufficient condition presented in Lemma 2 of Cvetkovski and Crovella [5] 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 a 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 node $t \in V$, at every node $s \in V$ there is a node $v \in V$ that is closer to t than s. Therefore, let $s \neq t$ be any two vertices in V. Furthermore, let $\pi_{st} = s, v, \dots, t$ 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,

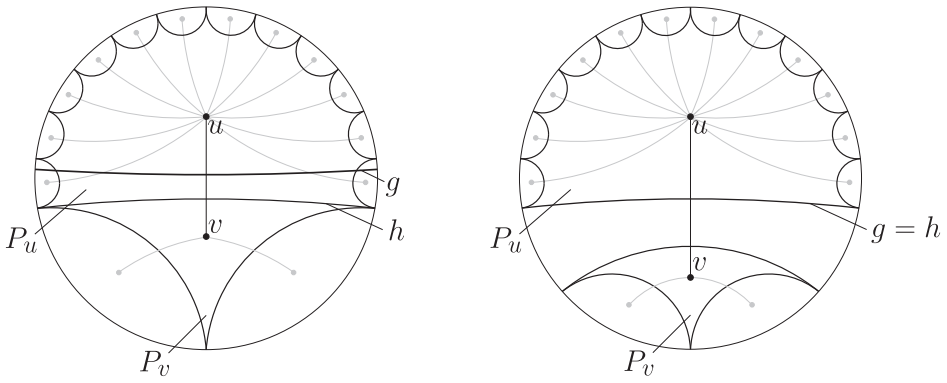


Fig. 2. (a) An ideal 13-gon P_u and an ideal triangle P_v are sharing the side h. Their perpendicular bisector g does not separate the tree into two connected components. (b) The distance between P_u and P_v is increased such that the perpendicular bisector g lies on the side h of the larger polygon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

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 an 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 to 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 therefore suited for greedy routing. This observation forms the basis of Kleinberg's embedding algorithm [9], 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 preceding 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 coordinates. Thus, setting k to the maximum degree of T seems to be a waste of space, particularly if most vertices of T have much smaller degree. 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 node degrees.

To explain the adaptive embedding, consider the tree $T = (V,E)$ with two adjacent vertices $u,v \in V$. The node u is placed at the center of an ideal $\deg(u)$-gon P_u, whereas 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 2(a), where h is the side shared by P_u and P_v. Observe that there is an important

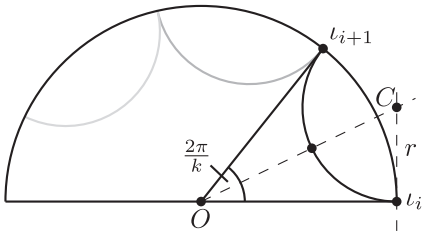


Fig. 3. Determining the distance between a side of an ideal k-gon and its center. Only half of the Poincaré disk is shown.

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 2(a) 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 2(a)). Thus, the embedding does not satisfy Theorem 3.1 and is not suitable for greedy routing. However, this issue can be 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 2(b), 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 3.2. *The distance $\ell(k)$ between the center and the sides of an ideal k-gon is given by*

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

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 afterward. Let $\iota_0, \dots, \iota_{k-1}$ 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). Without loss of generality, we can assume that ι_i is placed at $(1, 0)$ and that ι_{i+1} is its counter-clockwise successor (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. 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 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 preceding 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. Afterward, we proceed with each child independently. When processing a node v, we know the coordinates of its parent u and also know a side h of the $\deg(u)$-gon P_u with center u that should separate u from v (see Figure 2(b)). 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. Finally, we center an ideal $\deg(v)$-gon P_v at v and recursively proceed with the children of v.

Note that the choice of the distance $\ell_{u,v}$ ensures 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 3.3. *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. Additionally, we consider how the representation of the embedding can be altered to decrease issues with numerical difficulties.

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; second, the height of the spanning tree should be small to prevent numerical problems.

Breadth-first search trees. The first type of trees we consider are trees obtained from a breadth-first search (BFS) starting at a random node in the graph. The resulting BFS-trees have a small diameter (working toward our second goal). However, they are not optimized for achieving a good stretch. In the following, we thus propose a different choice for the tree focusing on the stretch.

Maximum betweenness centrality trees. 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* [4]. 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 child 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, decreasing the chances of a detour on the greedy path. Thus, we want adjacent vertices to be close and non-adjacent vertices farther apart.

Optimal cyclic arrangement order. The first strategy we propose to achieve this is to reorder the subtrees at each node in the spanning tree such that subtrees that are connected by more edges

are placed closer to each other. This problem can be modeled 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 $w : E \to \mathbb{N}$. 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 [8] can be easily reduced to it. In our experiments, we therefore use a heuristic that iteratively improves the order greedily. More precisely, for a given node v, the heuristic works as follows. We first choose a neighbor w of v uniformly at random. Afterward, among the $\deg(v) - 1$ positions in the cyclic order at which w could be placed, we determine the position that minimizes the edge costs, assuming that the order is fixed otherwise. In preliminary experiments, we determined that repeating this process $2 \deg(v)$ times for each node v yields a good balance between order quality and runtime.

Maximum likelihood embedding order extraction. 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) [2, 11, 12]. As mentioned in Section 1, MLEs do typically not lead to successful greedy routing. However, such an embedding determines for each node a cyclic order of all of 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 MLE 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 [2].

4.3 Choice of the Root

Although the choice root of the tree is *not* a degree of freedom that changes the embedding itself, it does impact how the embedding is represented in the implementation. The root is placed in the origin of the hyperbolic plane. Therefore, considering different nodes as roots is equivalent to a translation of the embedding. Since a translation does not change the positions of the nodes relative to each other, in theory, changing the root does not impact the quality of the embedding. In practice, however, translating the coordinates can influence the precision that is required to avoid numerical difficulties. Since the area of a disk in the hyperbolic plane grows exponentially with its radius, the precision required to address points increases quickly with the distance to the origin.

Minimizing the maximum radius. We aim to choose a root such that the resulting embedding is enclosed in the smallest possible disk centered at the origin. The idea is to find a root that minimizes the depth of the tree (i.e., the node that halves the diameter). In the adaptive embedding, however, not all edges have the same length: their lengths are determined by the degrees of their endpoints. Consequently, we consider the lengths as edge weights. The root is then chosen as the node that halves the weighted diameter, which is determined using two executions of Dijkstra's algorithm.

5 EXPERIMENTAL EVALUATION

With our experimental evaluation, we want to answer the following main questions, with the third referring to the fact that our implementation[1] prevents numerical issues by using the multiple-precision library MPFR [7]:

[1] Our C++ implementation is available at https://hpi.de/friedrich/research/hyperbolicgreedyroutings.

- 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 node compared to multiple-precision coordinates)?
- How much more space efficient is the adaptive embedding compared to Kleinberg's approach?
- How do different spanning trees and child orders impact the robustness to network failures?

5.1 Experimental Setup

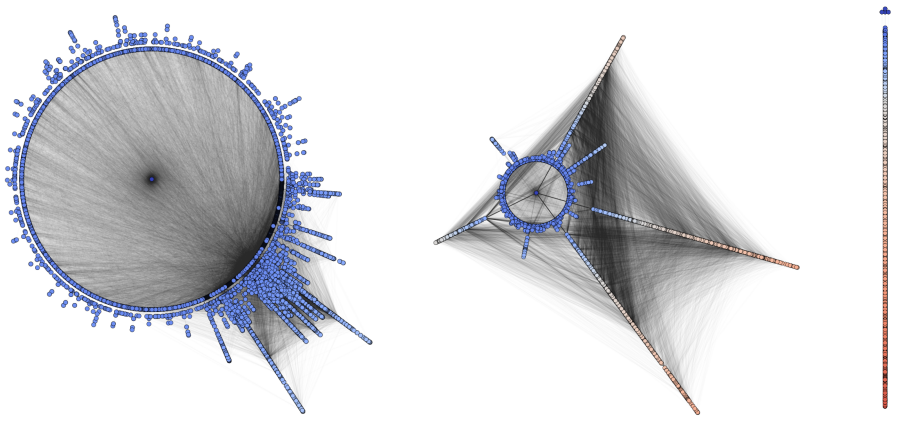
To evaluate the impact of the different degrees of freedom on the stretch, we implemented the adaptive embedding described in Section 3, as well as the different strategies of filling the degrees of freedom discussed in Section 4. Recall that these choices are as follows. For the spanning tree, we either choose a BFS-tree or a tree that maximizes the betweenness centrality, which we abbreviate as *BC-tree*. For the child order, we either heuristically solve an instance of OCA for each node of the tree to determine an order of its children, or we derive the orders from a single MLE. As the root of the spanning tree, we choose the node that minimizes the maximum radius of the coordinates (Min). To have a baseline to which we can compare these strategies, we also ran experiments with random spanning trees, random child orders, and random roots (Ran). Note that the strategies for choosing a spanning tree, child order, and root can be combined arbitrarily. Thus, with the three choices for the spanning tree (BFS, BC, Ran), the three choices for the node orders (OCA, MLE, Ran), and the two choices for the root (Min, Ran), we obtain 18 combinations in total. For combinations that do not involve randomness (e.g., [BC, OCA, Min]), we compute one embedding. Whenever one of the strategies does involve randomness (e.g., [Ran, Ran, Min]), we compute 20 embeddings instead and average the results. In total, we obtain 284 embeddings ($14 \cdot 20$ where at least one strategy involved randomness and 4 where none did) for each network.

The adaptive embedding method was implemented in C++ based on the NGraph data structure [13]. Previous experiments considered graphs with 50 nodes and an average degree of 3 that were obtained by randomly generating edges [5], or random unit disk graphs with 50 nodes [9]. Here we extend the work of Ban et al. [1] where a collection of five real-world networks was considered and examine a broader collection of larger real-world networks.

The instances we consider are a combination of real-world networks of different types and generated hyperbolic random graphs [10]. More precisely, we chose 21 real-world networks from the Network Repository [14] coming from different domains such as social networks, biological networks, or infrastructural networks. We also included networks with a naturally underlying Euclidean geometry (e.g., 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 [3]. 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 of these graphs, we embedded the largest connected component. Figure 4 shows three embeddings of the real-world network *soc-anybeat* obtained using different combinations of spanning trees, child orders, and roots.

After embedding these networks, we computed the stretch values by comparing the shortest path between every node pair to the corresponding route obtained by greedily navigating through the network. If the number of node pairs was larger than 10,000, we instead sampled 10,000 node pairs uniformly at random. We note that although each embedding is computed using only a spanning tree, the greedy routing is always performed using all edges in the graph.

To determine how using multiple precision (we used 300 decimal digits per coordinate) impacted the quality of the obtained embeddings, we also conducted the same evaluation experiments after



(a) (b) (c)

Fig. 4. Three exemplary adaptive embeddings of the *soc-anybeat* network with embedding strategy combinations of [BC, OCA, Min] (a), [Ran, Ran, Min] (b), and [Ran, Ran, Ran] (c). All embeddings are visualized using the native representation of the hyperbolic plane. Node colors represent the radii of the nodes with the same colors as in Figure 6.

rounding all coordinates to Double values. The compiler that was used in our experiments complies to the IEEE-754 standard, meaning Double values are represented with 15 decimal digits. We note that computations still used the multiple-precision library. Nevertheless, the success rate may not reach 100% due to rounding the coordinates afterward.

To determine how much more space efficient our adaptive embedding technique is compared to Kleinberg's approach, we computed a Kleinberg counterpart embedding for each adaptive one. This counterpart uses the same inputs (tree, child order, and root) as the original embedding, but instead of adapting the size of the ideal polygons to the degrees of the nodes, it always uses the maximum degree of the tree. Afterward, we compare the maximum radii of the resulting embeddings.

Finally, to measure how robust our embeddings are against node or edge failures, we simulated such outages by removing random parts of different sizes from the network. Recall that we initially computed 284 embeddings for each network. When simulating network failures of different extends and for each perform multiple iterations of evaluations in order to obtain an average, the number of evaluations becomes intractable. Therefore, we reduced the number of embeddings by choosing a representative embedding for each combination of spanning trees and child orders that involved randomness. (Since these evaluations were only performed using high-precision coordinates, there was no need for considering different choices for the root.) In particular, for each strategy combination where we initially computed multiple embeddings, we now chose the embedding with the median stretch as representative. Consequently, we obtained nine embeddings for each network. For each embedding, we then performed 10 evaluations of deleting 5%, 10%, and 15% of randomly chosen nodes in the network. Since the number of evaluations was still quite large, we additionally reduced the path samples in each evaluation to 5,000. The same process was used when simulating edge failures. Since the removal of nodes and edges may disconnect the graph into multiple components, we resampled node pairs that were not in the same connected component.

The experiments were run on a GNU/Linux Server with 48 Intel Xeon CPUs with 2.60 GHz and 256 GB of RAM. To accommodate for the large number of experiments, we used GNU Parallel [16] to schedule multiple embedding and evaluation processes simultaneously.

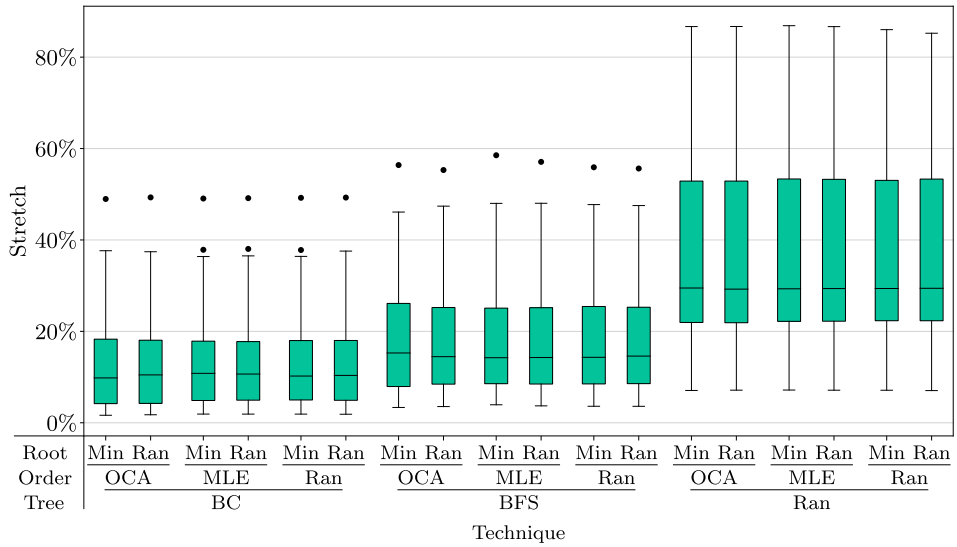


Fig. 5. Using different kinds of spanning trees as the basis of an embedding has an impact on the stretch. Varying the node order influences the stretch only marginally.

5.2 Evaluation

Impact of degrees of freedom on stretch. Concerning the first question, Figure 5 shows the resulting stretch values for the different combinations (using high-precision coordinates). For each network and each strategy combination, we obtained a single data point by measuring the stretch (for strategies involving randomness, the results were averaged). Figure 5 contains the box plots summarizing the values for all networks. The x-axis shows the different strategies, and the y-axis denotes the stretch. Boxes show the median value at the center and extend to the 25th and 75th percentiles, whereas the whiskers denote the 10th and 90th percentiles.

One can clearly see that the choice of the spanning tree has a significant impact on the stretch while the child order appears to be irrelevant. On average, using random spanning trees results in a stretch of about 37.5%. Using a BFS-tree improves this value to about 18.2%. This can be reduced even further by using spanning trees that maximize the betweenness centrality, resulting in an average stretch of 14.0%. As expected, the choice of the root does not affect the stretch. Slight deviations that are visible in the figure can be explained by differences in the random sampling of node pairs during the evaluation.

In Table 1, we list for each graph the technique that led to the best stretch values together with these stretch values. There, each entry corresponds to a single embedding. In other words, for strategies that involve randomness, the table does not show the average but the embedding that yielded the best results. One can see that for 83.9% of the graphs, using the BC-tree leads to the best stretch. We note that for 69.2% of these graphs, the OCA node order yielded the best results, indicating that (possibly) the child order does have an impact. However, for all but one of the remaining graphs where BC-trees performed best, the best result was obtained using a random child order. Additionally, the difference between the stretch values for BC-trees with the OCA child order and BC-trees with random child orders is only about 0.00167 on average. Both support the fact that the child order does not have an impact on the stretch. A possible explanation for the irrelevance of the child order is a property of the hyperbolic plane: from the point of view of a single node, most other nodes lie in a small cone, as can be seen in Figure 4(c). Therefore,

Table 1. Stretch Values and Success Rates for All Graphs

Network Name	Nodes	Avg. Deg.	Stretch (MPFR)	Technique (MPFR)	Success (Double)	Stretch (Double)	Technique (Double)
bio-celegans	453	8.9	4.1%	BC OCA	100%	4.1%	BC OCA
bn-fly-drosophila-medulla-1	1,770	10.1	1.6%	BC OCA	100%	1.6%	BC OCA
bn-mouse-retina-1	1,076	168.8	12.5%	BFS OCA	100%	12.5%	BFS OCA
chem-ENZYMES-g118	95	2.5	17.9%	BFS Ran	100%	17.9%	BFS Ran
chem-ENZYMES-g123	90	2.8	35.6%	Ran OCA	100%	35.6%	Ran OCA
ca-CSphd	1,025	2.0	4.4%	BC MLE	100%	7.6%	BC Ran
ca-Erdos992	4,991	3.0	24.9%	BC Ran	100%	29.5%	BFS OCA
ca-GrQc	4,158	6.5	29.1%	BFS Ran	100%	29.5%	BFS Ran
ia-email-EU	32,430	3.4	13.3%	BC OCA	100%	14.6%	BFS Ran
ia-email-univ	1,133	9.6	36.1%	BC OCA	100%	37.0%	BC Ran
ia-reality	6,809	2.3	7.7%	BC OCA	100%	8.2%	BC OCA
inf-euroroad	1,039	2.5	19.6%	BC Ran	100%	20.1%	BC Ran
inf-openflights	2,905	10.8	11.5%	BC OCA	100%	11.7%	BFS Ran
inf-power	4,941	2.7	13.2%	BC Ran	99.81%	14.5%	BC OCA
lp-afiro	51	4.0	21.4%	Ran MLE	100%	21.4%	Ran MLE
Internet	13,204	6.8	6.3%	BC OCA	100%	6.3%	BC OCA
as-caida	26,475	4.0	7.4%	BC OCA	100%	9.8%	BFS Ran
soc-advogato	5,054	15.6	15.2%	BC OCA	100%	15.3%	BC Ran
soc-anybeat	12,645	7.8	3.7%	BC OCA	100%	3.9%	BC OCA
soc-gplus	23,613	3.3	7.9%	BC Ran	99.99%	13.6%	BFS OCA
soc-hamsterster	2,000	16.1	21.9%	BC OCA	100%	22.2%	BC OCA
soc-wiki-Vote	889	6.6	14.5%	BC Ran	100%	14.5%	BC OCA
HRG(0.1, 2.1)	4,932	4.5	1.5%	BC OCA	100%	1.5%	BC OCA
HRG(0.1, 2.5)	9,527	8.0	2.0%	BC OCA	100%	2.0%	BC OCA
HRG(0.1, 2.9)	9,747	8.5	2.6%	BC OCA	100%	2.7%	BC OCA
HRG(0.5, 2.1)	4,567	3.9	2.3%	BC OCA	100%	2.3%	BC OCA
HRG(0.5, 2.5)	9,712	7.8	3.5%	BC OCA	100%	3.5%	BC OCA
HRG(0.5, 2.9)	9,863	6.7	9.1%	BC Ran	100%	9.8%	BC OCA
HRG(0.9, 2.1)	3,592	3.0	2.9%	BC OCA	100%	2.9%	BC OCA
HRG(0.9, 2.5)	8,254	3.7	9.6%	BC OCA	100%	9.8%	BC OCA
HRG(0.9, 2.9)	8,456	3.4	16.4%	BC Ran	100%	25.4%	BFS Ran

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. (For random spanning trees and random node orders, these values do *not* represent the average but the best value obtained.) All values refer to the largest connected component in the graph.

reordering the nodes within such a cone only results in very small angular deviations that do not make a difference during the routing.

Performance on real-world networks. In addition to showing which embedding strategies performed best, Table 1 also shows for which networks greedy routing is well suited. Besides the Internet graph with a stretch of 6.3%, it is worth noting that there are other networks with very low stretch values. With 1.5%, the smallest value is obtained for a hyperbolic random graph. Additionally, the best stretch obtained on a real-world network was 1.6% for the biological network

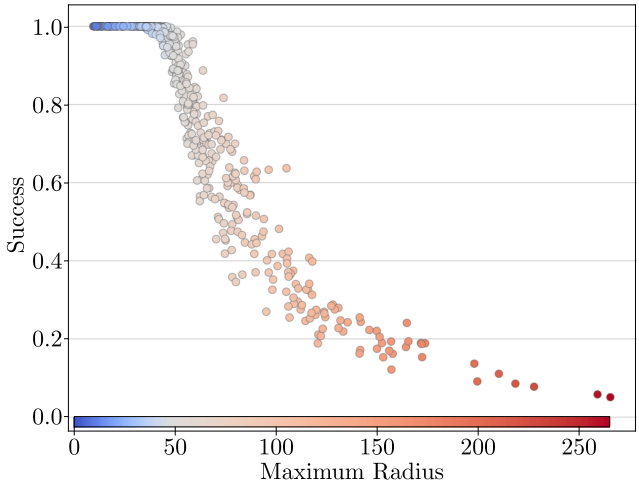


Fig. 6. Numerical issues can occur when coordinates are rounded to Double values. With increasing maximum radius of the embedding, the success rate decreases.

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. Yet there are graphs with small average degree that lead to a high stretch. For example, *inf-euroroad* has an average degree of 2.5 and a stretch value of 19.6%. 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.

Performance with small coordinates. 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, yielding 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 a 100% success rate (for all others, the success rates are still very high). Moreover, the stretch values do not change much.

Although we did find an embedding with a perfect (or almost perfect) success rate for each network, even when rounding the coordinates to Double values, Figure 6 shows that for a large portion of embeddings, the success rate is rather low. It shows the maximum radius among the nodes in the considered embeddings on the x-axis and the success rate of the greedy routing with Double coordinates on the y-axis. One can see that the success rate is always 100% for embeddings where the maximum radius is small. Starting with maximum radii close to 50, the success rate drops quickly and decreases with increasing radius. Note that the small success rates basically render the resulting embeddings unusable for greedy routing, even for relatively small radii ($\sim$100) that Double coordinates can represent just fine. This is because the problem is only indirectly described by the radii: since the area of a disk in the hyperbolic plane increases exponentially with its radius, the distance between two points with fixed radii increases quickly with the angular distance between them (the larger the radii, the stronger the effect). However, in the adaptive embedding, the distances between adjacent nodes are usually small, meaning nodes need to be placed at small angular distances. With large enough radii, the required angular distances become so small that the precision of Double values is not sufficient to distinguish between the corresponding angular

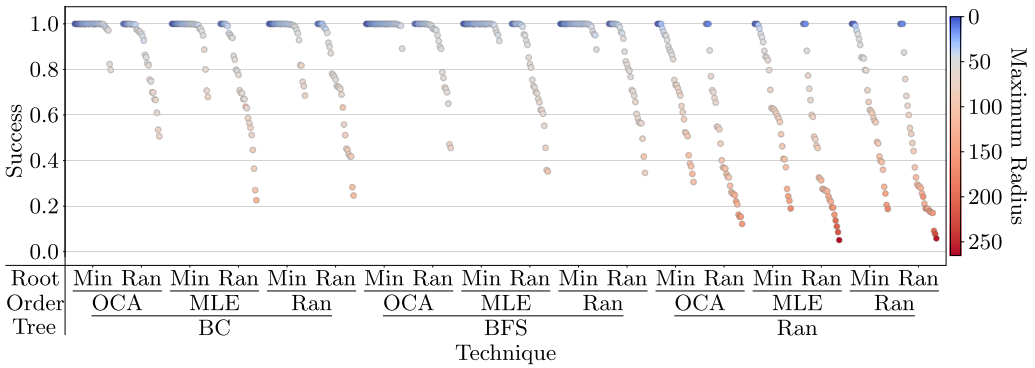


Fig. 7. The combination of strategies used for the embeddings impacts the size of the coordinates and with it the success rate, if coordinates are rounded to Double values.

coordinates. This situation can be observed in Figure 4, which shows three different embeddings of the same network *soc-anybeat*. The color of each node corresponds to its radius, using the same colors as in Figure 6. Recall that in the adaptive embedding, the neighbors of a node are placed at equidistant angles around it. In each embedding in Figure 4, this can be observed for the root (dark blue) that is placed in the origin of the hyperbolic plane. With increasing radii, the angles between the nodes then decrease quickly and most nodes seem to be lying on lines instead of circles. For the [BC, OCA, Min] embedding (Figure 4(a)), all nodes are in the blue range, meaning their radii are below approximately 50, and although many nodes seem to have the same angular coordinate, the precision of Double coordinates is sufficient to distinguish their angles. For this embedding, we obtained a success rate of 100% with rounded coordinates. For the [Ran, Ran, Min] embedding (Figure 4(b)), fewer nodes are colored blue and there are several subgraphs extending into light red lines. For this embedding, the success rate dropped to 55%. For the [Ran, Ran, Ran] embedding (Figure 4(c)), it is even worse: from the point of view of the root at the top, almost all other nodes basically lie on a single line extending into darker red. The required resolution to distinguish the angular coordinates of the nodes is far from what Double values can represent, yielding a success rate of 14%.

Since different embeddings of the same network lead to different success rates, we wanted to see whether some of our embedding strategies are more susceptible to numerical difficulties than others and plotted the success rates for the embeddings with Double coordinates depending on the used strategy combination in Figure 7. There, each point corresponds to one network and the colors indicate the maximum radii of the embeddings. One can see that random spanning trees appear to be less robust to numerical difficulties than the BC- and BFS-trees while the child order still does not have a significant impact. For BC- and BFS-trees, the OCA child order seems to perform only slightly better than MLE or Ran. The choice of the root, however, is more important. On BC- and BFS-trees, our strategy described in Section 4.3 (i.e, choosing a root that minimizes the maximum radius) yielded the desired results: smaller coordinates and higher success rates. Most notably, the strategy combination [BFS, OCA, Min] delivered the best results where almost all success rates are very close to 100%.

It is obvious that the precision of Double coordinates (with 15 decimal digits) is not sufficient to guarantee a success rate of 100%. However, it is unclear how precise the coordinates need to be. In fact, among the 8,804 embeddings computed with high-precision coordinates (300 decimal digits), one embedding did not yield a success rate of 100%. For the largest considered network *ia-email-EU* (with 32,430 nodes), the strategy combination [Ran, Ran, Ran] yielded an embedding

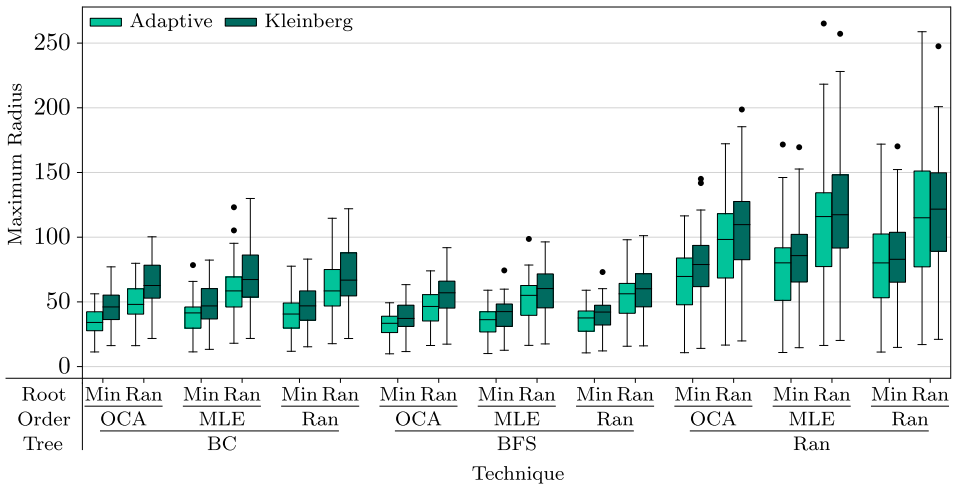


Fig. 8. The combination of strategies used for the embeddings impacts the size of the coordinates. Most of the time, the maximum radius is smaller in adaptive embeddings than when using Kleinberg's approach.

with a maximum radius of 362, where even the high-precision coordinates resulted in a success rate of "only" 99.58%.

Size of the embeddings. Recall that the main difference between Kleinberg's approach and the adaptive embedding is that in the latter, the size of the ideal polygon that each node v is placed in is adapted to the degree of v instead of always using the maximum degree of the tree. Since the distance between the center and the sides of an ideal k-gon increases with k (see Lemma 3.2), our adaptive method should yield embeddings with smaller coordinates. To confirm this, we computed a Kleinberg counterpart embedding for each adaptive embedding (using the same tree, child order, and root) and compared the maximum radii. As before, we obtained one data point for each network and embedding technique (averaging results whenever randomness is involved). The resulting values are summarized as box plots in Figure 8. Clearly, the adaptive method yielded embeddings with smaller radii most of the time. The best results were obtained using the strategy combination [BFS, OCA, Min], which also led to the best success rates (see Figure 7).

We note that the adaptive embedding did not always produce smaller coordinates than Kleinberg's approach. By not adapting the size of the ideal polygons to the degrees of the nodes, the angular distances between the neighbors of a node are smaller. These neighbors are placed closer to the node (and therefore closer to the origin) than when using the adaptive embedding. Most of the times, however, the space saved due to the smaller edge lengths outweighs the space wasted by using larger angular distances.

Robustness to network failures. Since real-world networks like the Internet are prone to node or edge failures, our last question aims to find out how different spanning trees and child orders influence the robustness of the embeddings to such outages. Therefore, we observed how the quality of the greedy routing is affected when portions of the network are removed. Recall that (for this experiment), we reduced the number of embeddings per graph by choosing a representative embedding for each combination of spanning tree and child order that involves randomness. Furthermore, we only considered embeddings with high-precision coordinates, where the choice of the root is irrelevant. For the resulting nine embeddings, we performed 10 evaluations of deleting random parts of the network and afterward computed the average of the resulting success rates. Consequently, we obtained nine data points for every network. The box plots in Figure 9

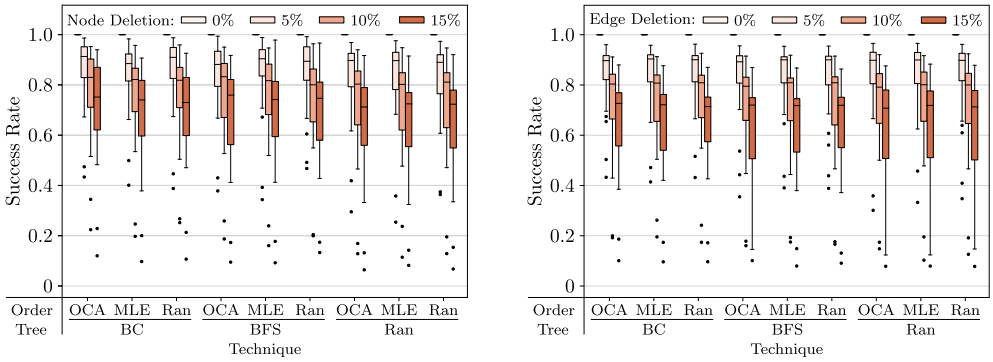


Fig. 9. For each technique, there are four box plots that denote (from left to right) the success rate after deleting 0%, 5%, 10%, and 15% of nodes (a) and edges (b) in the networks. These values were obtained using high-precision coordinates.

summarize the resulting values for all networks when deleting nodes and edges, respectively. Non-surprisingly, the success rate decreases as the size of the failed portion of the network increases. Additionally, neither the type of spanning tree nor the chosen child order seem to affect the robustness of the embedding.

Although the adaptive embedding technique produced embeddings of the Internet graph with perfect success rates and very good stretch, they were not as robust as the existing embedding by Boguñá et al. [3] when it comes to network failures. On our BC OCA embedding that is also shown in Table 1, greedy routing succeeds with a success rate of 91.44% when deleting 5% of the nodes in the network. The removal of 10% and 15% reduces the success rates to 86.99% and 84.09%, respectively. The success rates for the embedding by Boguñá et al. [3] are about 95% when deleting up to 10% of the nodes in the network and above 85% when deleting up to 20%. In contrast to our adaptive embedding, they place hub nodes (nodes that have a large degree) very close to each other in the so-called core of the embedding. Greedy routing between two vertices usually leads to this densely connected core before moving toward the target node. As a consequence, the success rate decreases to about 80% even when removing only 10 nodes from the core [3]. However, since the core only consists of a small fraction of the nodes in the network, it is unlikely that removing nodes uniformly at random destroys the core. In that case, the success rate is not influenced much. Our adaptive embedding behaves differently in this scenario. Removing *any* node *always* destroys the underlying spanning tree, and deleting only 5% of the graph (which constitutes about 660 nodes for the considered Internet graph) already does a lot of damage to the tree and therefore impacts the success rate.

When deleting 5%, 10%, and 15% of edges, our adaptive embedding of the Internet graph yielded success rates of 91.52%, 83.93%, and 75.85%, respectively (compared to 91.44%, 86.99%, and 84.09% when deleting nodes). However, in the embedding of Boguñá et al. [3], the success rate decreased to only slightly less than 85% when removing up to 30% of the nodes while decreasing below 75% when deleting up to 30% of the edges. Thus, removing edges affects the success rate stronger than when removing nodes for both types of embeddings. However, the adaptive embedding is not as robust to such network failures.

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 node 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 node 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 7), 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 [6] gave an algorithm that generates succinct greedy embeddings in which node positions are represented with only $O(\log n)$ bits per node. Unfortunately, our implementation of their algorithm failed to produce embeddings with a 100% success rate, and we in fact believe that the proof is flawed, as briefly argued in the following.

The embedding algorithm [6] first embeds a spanning tree T into a so-called dyadic tree metric that 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, the tie-breakers decide in which direction to move on π: if $t < s$, the routing walks toward u, and if $s < t$, the routing walks toward 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 upward on π. However, when routing from s to u', the tie-breaker lets us walk toward u, and when routing from s to v', the tie-breaker lets us walk toward v, which cannot both be upward on π. Thus, at least one of these two routing queries fails. Although 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 toward greedy routing. To that end, we proposed the adaptive embedding, which is a specialized version of Sarkar's algorithm [15] and an extension of Kleinberg's algorithm [9] 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 an issue. As can be seen in Figure 6, the success rate correlates with the maximum radius in the embedding. It would therefore be interesting to further investigate the precision required to guarantee a perfect success rate for an embedding of a given size. 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.

Furthermore, since the adaptive embeddings of the Internet graph were not as robust as the one by Boguñá et al. [3], it would be interesting whether one can combine the advantages of the two

types of embeddings. On the one hand, having a small, densely connected core seems to make the embedding more robust to network failures. On the other hand, one can guarantee a perfect success rate by using a tiling of the hyperbolic plane as the basis for the embedding and reduce the stretch by carefully choosing how the graph is embedded in this tiling.

REFERENCES

[1] Xiaomeng Ban, Jie Gao, and Arnout van de Rijt. 2010. Navigation in real-world complex networks through embedding in latent spaces. In *Proceedings of the Meeting on Algorithm Engineering and Expermiments (ALENEX'10)*. 138–148. DOI: https://doi.org/10.1137/1.9781611972900.13

[2] Thomas Bläsius, Tobias Friedrich, Anton Krohmer, and Sören Laue. 2016. Efficient embedding of scale-free graphs in the hyperbolic plane. In *24th Annual European Symposium on Algorithms (ESA 2016). Leibniz International Proceedings in Informatics (LIPIcs)*, P. Sankowski and C. Zaroliagis (Eds.), Vol. 57. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, Article 16, 18 pages. DOI: https://doi.org/10.4230/LIPIcs.ESA.2016.16

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

[4] Michele Borassi and Emanuele Natale. 2016. KADABRA is an ADaptive algorithm for betweenness via random approximation. In *24th Annual European Symposium on Algorithms (ESA 2016). Leibniz International Proceedings in Informatics (LIPIcs)*, P. Sankowski and C. Zaroliagis (Eds.), Vol. 57. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, Article 20, 18 pages. DOI: https://doi.org/10.4230/LIPIcs.ESA.2016.20

[5] A. Cvetkovski and M. Crovella. 2009. Hyperbolic embedding and routing for dynamic graphs. In *Proceedings of the 28th IEEE International Conference on Computer Communications (IEEE INFOCOM'09)*. 1647–1655. DOI: https://doi.org/10.1109/INFCOM.2009.5062083

[6] David Eppstein and Michael T. Goodrich. 2011. Succinct greedy geometric routing using hyperbolic geometry. *IEEE Transactions on Computers* 60, 11 (2011), 1571–1580. DOI: https://doi.org/10.1109/TC.2010.257

[7] Laurent Fousse, Guillaume Hanrot, Vincent Lefèvre, Patrick Pélissier, and Paul Zimmermann. 2007. MPFR: A multiple-precision binary floating-point library with correct rounding. *ACM Transactions on Mathematical Software* 33, 2 (June 2007), Article 13, 14 pages. DOI: https://doi.org/10.1145/1236463.1236468

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

[9] R. Kleinberg. 2007. Geographic routing using hyperbolic space. In *Proceedings of the 26th IEEE International Conference on Computer Communications (IEEE INFOCOM'07)*. 1902–1909. DOI: https://doi.org/10.1109/INFCOM.2007.221

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

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

[12] F. Papadopoulos, C. Psomas, and D. Krioukov. 2015. Network mapping by replaying hyperbolic growth. *IEEE/ACM Transactions on Networking* 23, 1 (Feb. 2015), 198–211. DOI: https://doi.org/10.1109/TNET.2013.2294052

[13] Roldan Pozo. (n.d.). NGraph: A Simple (Network) Graph Library in C++. Retrieved February 21, 2020 from http://math.nist.gov/~RPozo/ngraph/ngraph_index.html.

[14] Ryan A. Rossi and Nesreen K. Ahmed. 2015. The network data repository with interactive graph analytics and visualization. In *Proceedings of the 29th AAAI Conference on Artificial Intelligence (AAAI'15)*. 4292–4293. http://networkrepository.com.

[15] Rik Sarkar. 2012. Low distortion Delaunay embedding of trees in hyperbolic plane. In *Graph Drawing*. Berlin, Germany, 355–366.

[16] O. Tange. 2011. GNU parallel—The command-line power tool. *;login: The USENIX Magazine* 36, 1 (Feb. 2011), 42–47. DOI: https://doi.org/10.5281/zenodo.16303

Received August 2018; revised November 2019; accepted January 2020