Fast generation of complex networks with underlying hyperbolic geometry

Moritz von Looz  Christian L. Staudt  Henning Meyerhenke  Roman Prutkin

Abstract
Complex networks have become increasingly popular for modeling various real-world phenomena. Realistic generative network models are important in this context as they avoid privacy concerns of real data and simplify complex network research regarding data sharing, reproducibility, and scalability studies. We investigate a model creating geometric graphs in hyperbolic space. Previous work provided empirical and theoretical evidence that this model creates networks with a hierarchical structure and other realistic features. However, the investigated networks in previous work were small, possibly due to a quadratic running time of a straightforward implementation.

In this paper we provide the first generator for a representative subset of these networks which has sub-quadratic running time. We prove a time complexity of $O\left(\frac{n^3}{2} + m \log n\right)$ with high probability for the generation process. This running time is confirmed by experimental data with our implementation. The acceleration stems primarily from the reduction of pairwise distance computations through a polar quadtree, which we adapt to hyperbolic space for this purpose. In practice we improve the running time of a previous implementation by at least two orders of magnitude this way. Networks with billions of edges can now be computed in less than half an hour. Finally, we evaluate the largest networks of this model published so far. Our empirical analysis shows that important features are retained over different graph sizes and densities.

Keywords: complex networks, hyperbolic geometry, generative graph model, polar quadtree, network analysis

1 Introduction
The algorithmic analysis of complex networks is a highly active research area since complex networks are increasingly used to represent phenomena as varied as the WWW, social relations, protein interactions, and brain topology [13]. Complex networks have several non-trivial topological features: They are usually scale-free, which refers to the presence of a few high-degree vertices (hubs) among many low-degree vertices. A degree distribution that occurs frequently in practice follows a power law [13, Chap. 8.4], i.e. the number of vertices with degree $k$ is proportional to $k^{-\gamma}$, for a fixed exponent $\gamma > 0$. Moreover, complex networks often have the small-world property, i.e. typical distance between two vertices is surprisingly small, regardless of network size and growth.

Generative network models play a central role in many complex network studies for several reasons: Real data often contains confidential information; it is then desirable to work on similar synthetic networks instead. Quick testing of algorithms requires small test cases, while benchmarks and scalability studies need bigger graphs. Graph generators can provide data at different user-defined scales for this purpose. Also, transmitting and storing a generative model and its parameters is much easier than doing the same with a gigabyte-sized network. A central goal for generative models is to produce networks with realistic features: Realism is understood as the ability to replicate relevant structural features of real-world networks such as degree distribution, spectral properties, community structure, and frequency of triangles [5]. Finally, generative models are an important theoretical part of network science, since realistic models can improve our understanding of the formation of complex networks.

Motivation. A model presented by Krioukov et al. [10] has provably high clustering coefficient [7], small diameter and a power-law degree distribution with adjustable exponent. It is based on hyperbolic geometry, which has negative curvature and is the basis for one of the three isotropic spaces. (The other two are Euclidean (flat) and spherical geometry (positive curvature).) Hyperbolic geometry has a theoretical connection to graphs with power-law degree distributions and has been...
studied with respect to routing applications [4]. In the generative model, vertices are distributed randomly on a hyperbolic disk and edges are inserted for every vertex pair with a probability depending on their hyperbolic distance. Calculating all pairwise distances has quadratic time complexity. This impedes the creation of massive networks and is likely the reason previously published networks based on hyperbolic geometry have been in the range of at most $10^4$ vertices. A faster generator is necessary to use this promising model for networks of interesting scales. Additionally, to judge the realism of these networks, more detailed parameter studies and comparisons from a network analysis point of view are necessary.

**Outline and Contribution.** We implement and study the model presented in [10] with a simplified edge probability (a binary step function) also used by Papadopoulos et al. [15]. We call the resulting model the hyperbolic unit-disk graph model and address gaps in terms of generation speed and network analysis. To lay the foundation, Section 2 introduces fundamentals of hyperbolic geometry. The main technical part starts with Section 3 in which we use the Poincaré disk model to relate hyperbolic to Euclidean geometry. This allows the use of a new spatial data structure, namely a polar quadtree adapted to hyperbolic space, to reduce both asymptotic complexity and running time of the generation. We analyze the time complexity of our generation process in Section 5 as well, resulting in a running time of $O((n^{3/2}+m)\log n)$ with high probability (w.h.p, i.e. $\geq 1-1/n$) for a graph with $n$ vertices and $m$ edges and sufficiently large $n$.

In Section 4 we add to previous studies a comprehensive network analytic evaluation of the hyperbolic unit-disk model. This evaluation shows many realistic features over a wide parameter range. The experimental results also confirm the theoretical expected running time. A graph with $10^7$ vertices and $10^9$ edges can be generated with sequential code in 22 minutes on our test machine. The generator will be made available in a future version of NetworKit [10], our open-source framework for large-scale network analysis. Material omitted due to space constraints can be found in the appendix.

2 Related Work and Preliminaries

Related generative graph models are discussed in Section 4.3, where we also compare them to the hyperbolic unit-disk graph model, in part by using empirical data.

2.1 Graphs in Hyperbolic Geometry

Krioukov et al. [10] show how graphs generated with underlying hyperbolic geometry naturally develop a power-law degree distribution and other properties of complex networks. In their model, vertices are distributed randomly over a disk of radius $R$ in the hyperbolic plane, where vertex positions are generated as points in polar coordinates $(\phi, r)$. The angular coordinate $\phi$ is drawn from a uniform distribution over $[0, 2\pi]$, while the probability density for the radial coordinate $r$ is proportional to the circumference of a hyperbolic circle with radius $r$:

$$f(r) = \frac{2\pi \sinh(r)}{2\pi (\cosh(R) - 1)} = \frac{\sinh(r)}{\cosh(R) - 1} \propto \sinh(r)$$

The probability mass is thus equally spread over the hyperbolic space within the base disk. In the hyperbolic unit-disk graph model, an edge is inserted between two vertices $u$ and $v$ if their hyperbolic distance $\text{dist}_H(u, v)$ is below a threshold. (More general edge probabilities exist, so that we use the term unit-disk graph to make the distinction. The main difference in terms of the resulting graphs for the purpose of this paper is that, with a more general edge probability, the clustering coefficient can be adjusted.) The neighborhood of a point (= vertex) thus consists of the points lying in a hyperbolic circle around it. Gugelmann et al. [7] analyze this model theoretically and prove a low variation of the clustering coefficient for fixed parameters. Boguñá et al. [3] use vertex embedding in hyperbolic space to obtain virtual coordinates enabling greedy routing on internet topology networks, and Kleinberg [8] shows that every graph can be embedded in the hyperbolic space such that greedy routing always succeeds. Papadopoulos et al. [15] extend the generator of [10] with a dynamic growth model.

While [10] defines the disk radius $R$ as input parameter, we use the stretch parameter $s$ to determine it: $R = s \cdot \text{acosh}(n/(2\pi) + 1)$. The dispersion parameter $\alpha$ determines whether vertices tend to occur in the center or at the border of the hyperbolic disk. For this purpose, Eq. (2.1) is changed to $f(r) \propto \alpha \cdot \sinh(qr)$. We add a third parameter $t$ to the basic model, it determines the distance threshold for edge insertion. Two vertices are adjacent if their hyperbolic distance is below $tR$. An example graph with 500 vertices, $s = 1$, $\alpha = 0.8$ and $t = 0.2$ is shown in Figure 1a. The neighborhood of vertex $u$ (the bold blue vertex) consists of vertices $v$ where $\text{dist}_H(u, v) \leq 0.2 \cdot R$ (marked in blue). A previous generator code with a more general edge probability and quadratic time complexity is available [13]. We show in Section 4.4 that our implementation of the slightly more restricted model is at least two orders of magnitude faster in practice.

2.2 Poincaré Model

The Poincaré disk model is one of several representations of hyperbolic space within
Euclidean geometry and maps the hyperbolic plane onto the Euclidean unit disk \( D_1(0) \). The hyperbolic distance between two points \( p_E, q_E \in D_1(0) \) with radial coordinates \( r_{p_E} \) and \( r_{q_E} \), respectively, is then given by the Poincaré metric [3]:

\[
\text{dist}_H(p_E, q_E) = \text{acosh}\left(1 + 2 \frac{||p_E - q_E||^2}{(1 - r_{p_E}^2)(1 - r_{q_E}^2)}\right).
\]  

(2.2)

Figure 1b shows the same graph as in Figure 1a, but translated into the Poincaré model. This model is conformal, i.e. it preserves angles. More importantly for us, it maps hyperbolic circles onto Euclidean circles.

3 Fast Generation of Graphs in Hyperbolic Geometry

We proceed by showing how to relate hyperbolic to Euclidean geometry. Using this transformation, we are able to partition the Poincaré disk with a polar quadtree that supports efficient range queries. We adapt the network generation algorithm to use this quadtree and prove subsequently that it achieves subquadratic generation time.

3.1 Generation Algorithm

Transformation from hyperbolic geometry.

For a circular range query of radius \( \text{rad}_h \) around a query point \( u = (\phi_h, r_h) \), we are interested in the Euclidean circle \( E \) in the Poincaré disk which corresponds to the hyperbolic circle around \( u \). The center \( E_c \) and radius \( \text{rad}_E \) of \( E \) are almost always different from \( u \) and \( \text{rad}_h \), respectively. All points on the boundary of the Euclidean circle are also on the boundary of the hyperbolic circle and thus have hyperbolic distance \( tR \) from \( u \). Two of these points are straightforward to construct by keeping the angular coordinate fixed and choosing the radial coordinates to match the hyperbolic distance: \( (\phi_h, r_{e_1}) \) and \( (\phi_h, r_{e_2}) \), with \( r_{e_1}, r_{e_2} \neq r_h \) and \( \text{rad}_h = \text{acosh}(1 + 2(r_e - r_h)^2/(1 - r_h^2)(1 - r_e^2))) \). These points are directly below and above \( u \). It follows (for details see Appendix A):

**Proposition 3.1.** \( E_c \) is at \((\phi_h, \frac{2r_h}{ab + 2})\) and \( \text{rad}_E \) is \( \sqrt{(\frac{2r_h}{ab + 2})^2 - \frac{2r_h^2 - ab}{ab + 2}} \), with \( a = \cos(\text{rad}_h) - 1 \) and \( b = (1 - r_h^2) \).

**Algorithm.**

**Algorithm 3.1. (Graph generation)**

**Input:** \( n, t, \alpha, s \). **Output:** \( G = (V, E) \)
1. \( R = s \cdot \text{acosh}(n/(2\pi) + 1) \)
2. \( V = n \) vertices
3. \( T = \text{empty polar quadtree} \)
4. **For each** vertex \( v \in V \): 
5. \( \text{draw } \phi[v] \text{ from } U[0, 2\pi) \)
6. \( \text{draw } r_H[v] \text{ with density } f(r) \propto \alpha \sinh(\alpha r) \)
7. \( r_E[v] = \text{hyperbolicToEuclidean}(r_H[v]) \)
8. \( \text{insert } v \text{ into } T \text{ at } (\phi[v], r_E[v]) \)
9. **For each** vertex \( v \in V \): 
10. \( C_H = \text{circle around } (\phi[v], r_H[v]) \text{ with radius } tR \)
11. \( C_E = \text{transformCircleToEuclidean}(C_H) \)
12. **For each** vertex \( w \in T \).getVerticesInCircle(C_E) 
13. \( \text{add } (v, w) \) to \( E \)
14. **Return** \( G \)

The generation of \( G = (V, E) \) with \( n \) vertices and \( m \) edges in our model is shown in Algorithm 3.1. As in previous efforts, vertex positions are generated randomly (lines 5 and 6). Then, we map these positions into the Poincaré disk (line 7) and, as a new feature, store them in a polar quadtree (line 8). For each vertex \( v \) the hyperbolic circle defining the neighborhood is mapped into the Poincaré disk according to Proposition 3.1 (lines 10-11) – also see Figure 1b, where the neighborhood of \( v \) consists of exactly the vertices in the light blue Euclidean circle. Edges are then created by executing a Euclidean range query with the resulting circle in the polar quadtree (lines 12-13).

**Data Structure.** As mentioned above, our central data structure is a polar quadtree on the Poincaré disk. While Euclidean quadtrees are common [16], we are not aware of previous adaptations to hyperbolic space. A node in the quadtree is defined as a tuple \((\min_\phi, \max_\phi, \min_r, \max_r)\) with \( \min_\phi \leq \max_\phi \) and \( \min_r \leq \max_r \). It is responsible for a point \( p = (\phi_p, r_p) \in D_1(0) \) iff \( \min_\phi \leq \phi_p < \max_\phi \) and \( \min_r \leq r_p < \max_r \). Figure 2 shows a section of a polar quadtree where...
quadtree nodes are marked by dotted red lines. When a leaf cell is full, it is split into four children. Splitting in the angular direction is straightforward as the angle range is halved: \( \text{mid}_\phi = \frac{\max\phi + \min\phi}{2} \). For the radial direction, we choose the splitting radius to result in an equal division of space.

\[
(3.3) \quad \text{mid}_r = \text{acosh} \left( \frac{\cosh(\max_r) + \cosh(\min_r)}{2} \right)
\]

**Theorem 3.1.** Let \( R \) be the hyperbolic radius of the base disk. A node at depth \( i \) of the quadtree covers an area of \((2\pi(\cosh(R) - 1))/4^i\).

**Theorem 3.2.** Let \( T \) be a polar quadtree constructed as defined in Eq. (3.3) with \( n \) points distributed uniformly in hyperbolic space. Then, for \( n \) sufficiently large, \( \text{height}(T) = O(\log n) \) whp.

The above theorems hold for \( \alpha = 1 \), as the points are then distributed uniformly within the base disk. By setting the splitting radius in Eq. (3.3) to \( \text{acosh}(\alpha \max_r + \cosh(\alpha \min_r)/2) / \alpha \), we still have an equal division of probability mass even for \( \alpha \neq 1 \) and the balance argument holds. These results are useful for establishing the time complexity of the main quadtree operations.

### 3.2 Time Complexity

The time complexity of the generator, in turn, is determined by the operations of the polar quadtree. A complete quadtree with \( k \) levels has \( 4^k \) leaf cells, defined by \( 2^k \) angular and \( 2^k \) radial divisions. Let \( c \) be the number of points stored in a leaf, also called the leaf capacity.

**Quadtree Insertion.** The quadtree \( T \) is constructed one element at a time. Yet, we consider the tree after each point has been inserted. For each point \( p \), the time required to reach the final leaf is linear in the quadtree’s height, which is \( O(\log n) \) whp due to Theorem 3.2. This holds regardless of \( T \)'s height when inserting \( p \) since both traversing the tree to find the correct leaf and moving \( p \) in case of a leaf split have constant cost for \( p \). Hence the amortized time complexity is:

\[
(3.4) \quad T(\text{Insertion}) \in O(\log n) \text{ whp.}
\]

**Quadtree Range Query.** The neighborhood of a vertex consists of the vertices within a hyperbolic circle of radius \( tR \) around it. For each neighbor found, at most \( c \) vertices have to be examined in the leaf cell. At most \( 4 \cdot \text{height}(T) \) inner nodes need to be visited, with \( \text{height}(T) \in O(\log n) \) whp. Some leaf cells are examined by the range query but yield no neighbors. These cells are cut by the boundary of the query circle, which allows an upper bound for their number. When following the circumference of a query circle, each newly cut leaf cell requires the crossing of an angular or radial division. Each division can be crossed at most twice, thus at most \( 4 \cdot 2^k = 2^{\log_4(n/c) + 2} = 4\sqrt{n/c} \) cells are cut. The amortized time complexity for a vertex \( v \) with degree \( \deg(v) \) is thus:

\[
(3.5) \quad T(\text{RQ}(v)) \in O \left( \left( \sqrt{n} + \deg(v) \right) \cdot \log n \right) \text{ whp.}
\]

**Graph Generation.** To generate a graph \( G \) with \( n \) vertices, the \( n \) points are distributed in a disc of radius \( s \cdot \text{acosh}(n/(2\pi) + 1) \) and inserted into the quadtree. The time complexity of this is \( n \cdot O(\log n) = O(n \log n) \) whp. In the next step, neighbors for all points are extracted. This has a complexity of

\[
T(\text{Edges}) = \sum_v O \left( \left( \sqrt{n} + \deg(v) \right) \cdot \log n \right)
= O \left( \left( n^{3/2} + m \right) \log n \right) \text{ whp.}
\]

This dominates the quadtree construction and thus total running time. We conclude:

**Theorem 3.3.** Generating networks with hyperbolic geometry can be done in \( O((n^{3/2} + n) \log n) \) time whp for sufficiently large \( n \), i.e., with probability \( \geq 1 - 1/n \).

### 4 Experimental Evaluation

We first discuss several structural properties of networks and use them to analyze the graphs generated by the model under different parameters. Comparisons to real-world networks and existing generators as well as an evaluation of the running time, also in comparison to a previous implementation, follow.

#### 4.1 Network Properties

We consider several important graph properties characteristic of complex networks. The **degree distribution** of many complex networks follows a **power law**. The **clustering coefficient** is the fraction of closed triangles to triads (paths of length 2) and measures how likely two vertices with a common neighbor are to be connected. **Degree assortativity** describes whether vertices have neighbors of similar degree. A value near 1 signifies subgraphs with equal degree, a value of -1 star-like structures. Many real networks have
multiple connected components, yet one large component is usually dominant. k-Cores are a generalization of components and result from iteratively peeling away vertices of degree k and assigning to each vertex the core number of the innermost core it is contained in. The diameter is the longest shortest path in the graph, which is often surprisingly small in complex networks. Complex networks also often exhibit a community structure, i.e., dense subgraphs with sparse connections between them. Modularity is a measure that quantifies how well a partition of the vertex set corresponds to the dense subgraphs.

The plots in Figure 3 illustrate the relationship between the model parameters α and s and a structural property of the resulting network. We focus on α and s instead of the threshold t to ease comparison with previous theoretical analysis about hyperbolic unit-disk graphs. We choose the range [0.5, 2] for the stretch s and [0.8, 3] for α to obtain graphs with not too unrealistic densities. As Figure 3a shows, the network becomes sparser with rising dispersion and stretch and the density ranges from 10^{-5} to 0.1. If α = 1, the number of edges depending on s is given by Table 2. The generated graphs are connected unless the average degree is below \( \approx 20 \). Except for very sparse graphs, the power-law exponent of the degree distribution is \( \gamma = 2\alpha + 1 \) (Figure 3b), as analyzed theoretically. The degree assortativity rises with α and s, above α = 1.2 it is positive for some, above α = 1.8 positive for all values of s (Fig. 3c). Diameter rises with thinner graphs, until the graph becomes disconnected and only the diameter of the largest component is measured (Figure 3d). The clustering coefficient (Fig. 5a) is above 0.75 for values of s below 1.9, since the hyperbolic unit-disk graph model inherently promotes the formation of triangles. The maximum core number corresponds closely to the density seen in Figure 3a, as a high density leads to a higher degree of connectedness and the emergence of a dense core (Fig. 3e). We analyze the community structure by applying a modularity-driven algorithm. Modularity is not independent of graph size but nonetheless indicates that the graphs have community structure. The size of communities decreases with the sparsity of the network (Figure 3f). Dense graphs with very few communities have a relatively low modularity. Finally, Figure 5c (appendix) shows how well a power law function fits the degree distribution. Except for very sparse graphs, a power-law fit is much more likely than an exponential fit.

4.2 Comparison with Real-world Networks We judge the realism of generated graphs by comparing them to a diverse set of real complex networks (Table 1). The Barabasi-Albert model can be found in Table 2. The Barabasi-Albert model implements a preferential attachment process to model
the growth of real complex networks. The probability
that a new vertex will be attached to an existing
vertex \(v\) is proportional to its degree, which results
in a power-law degree distribution. The produced
networks have a power-law degree distribution and a
fixed exponent of 3, which is roughly in the range of
real-world networks. However, the degree assortativity
is negative and the clustering coefficient low. The
running time is in \(\Theta(n^2)\), rendering the creation of
massive networks infeasible. The Dorogovtsev-Mendes
model is designed to model network growth with a fixed
average degree. It is very fast in theory (\(\Theta(n)\)) and
practice, but at the expense of flexibility. Clustering
coefficient, degree assortativity and power law exponent
of generated graphs are roughly similar to those of
real-world networks. The \textit{Recursive Matrix (R-MAT)}
model \[6\] was proposed to recreate properties of complex
networks including a power-law degree distribution,
the small-world property and self-similarity. The R-
MAT generator recursively subdivides the initially empty
adjacency matrix into quadrants and drops edges into it
according to given probabilities. It has a fast running
time in practice and \(\Theta(m \log n)\) asymptotic complexity.
At least the \textit{Graph500} benchmark parameters lead to
an insignificant community structure and clustering
coefficients, as no incentive to close triangles exists.

Given a degree sequence, the \textit{Chung-Lu (CL) model} \[11\]
creates edges \((u,v)\) with a probability of \(p(u,v) = \frac{\deg(u)\deg(v)}{\sum_{k} \deg(k)}\), which recreates the degree sequence in
expectation. The model can be conceived as a weighted
version of the well-known Erdős-Rényi (ER) model, and
has been shown to have similar capabilities as the R-
MAT model \[18\]. Implementations exist with \(\Theta(n + m)\)
time complexity \[11\]. It succeeds in matching the degree
distributions of the first four graphs in Table\[1\] but in all
results both clustering coefficient and degree assortativity
are near zero and the diameter too small.

\textit{BTER} \[9\] is a two-stage structure-driven model. It
uses the standard ER model to form relatively dense sub-
graphs, thus forming distinct communities. Afterwards,
the CL model is used to add edges, matching the
desired degree distribution in expectation \[17\]. The BTER
generator achieves an asymptotic complexity of \(\Theta(n + m \log d_{\max})\), where \(d_{\max}\) is the maximum vertex degree.
We test it with the \textit{PGPgiantcompo}, \textit{caidaRouterLevel},
\textit{citationCiteseer} and \textit{coPapersDBLP} networks. The
degree distributions and clustering coefficients are
matched with a deviation of \(\approx 5\%\). Generated com-
munities have a size of 5-45 on average, which is smaller.
than typical real communities and those of the hyperbolic unit disk model.

As indicated by Table 2, the hyperbolic unit disk generator can match a degree distribution exponent, has stronger clustering than the Chung-Lu and R-MAT generator and is more scalable and flexible than the Barabasi-Albert generator. Diameter (without additional random edges) and number of connected components are less realistic than those produced by BTER, but community structure is closer to typical real communities.

4.4 Performance Measurements Figure 4 shows the sequential running times for networks with $10^4$ to $10^7$ vertices and up to $6 \cdot 10^8$ edges. Measurements were made on one core of a server with 256 GB RAM and 2x8 Intel Xeon E5-2680 cores at 2.7 GHz. We achieve a throughput of up to 20 million edges/s. Even at only $10^4$ vertices, our implementation is two orders of magnitude faster than the implementation at [14]. (Note that their implementation supports a more general edge magnitude faster than the implementation at [14]. (Note that their implementation supports a more general edge magnitude faster than the implementation at [14]. (Note that their implementation supports a more general edge magnitude faster than the implementation at [14]. (Note that their implementation supports a more general edge magnitude faster than the implementation at [14]. (Note that their implementation supports a more general edge magnitude faster than the implementation at [14]. (Note that their implementation supports a more general edge magnitude faster than the implementation at [14]. (Note that their implementation supports a more general edge magnitude faster than the implementation at [14]. (Note that their implementation supports a more general edge magnitude faster than the implementation at [14]. (Note that their implementation supports a more general edge magnitude faster than the implementation at [14]. (Note that their implementation supports a more general edge magnitude faster than the implementation at [14]. (Note that their implementation supports a more general edge magnitude faster than the implementation at [14]. (Note that their implementation supports a more general edge magnitude faster than the implementation at [14]. (Note that their implementation supports a more general edge magnitude faster than the implementation at [14]. (Note that their implementation supports a more general edge magnitude faster than the implementation at [14]. (Note that their implementation supports a more general edge magnitude faster than the implementation at [14]. (Note that their implementation supports a more general edge magnitude faster than the implementation at [14]. (Note that their implementation supports a more general edge magnitude faster than the implementation at [14]. (Note that their implementation supports a more general edge magnitude faster than the implementation at [14]. (Note that their implementation supports a more general edge magnitude faster than the implementation at [14]. (Note that their implementation supports a more general edge magnitude faster than the implementation at [14]. (Note that their implementation supports a more general edge magnitude faster than the implementation at [14]. (Note that their implementation supports a more general edge magnitude faster than the implementation at [14]. (Note that their implementation supports a more general edge magnitude faster than the implementation at [14]. (Note that their implementation supports a more general edge magnitude faster than the implementation at [14]. (Note that their implementation supports a more general edge magnitude faster than the implementation at [14]. (Note that their implementation supports a more general edge magnitude faster than the implementation at [14]. (Note that their implementation supports a more general edge magnitude faster than the implementation at [14]. (Note that their implementation supports a more general edge magnitude faster than the implementation at [14]. (Note that their implementation supports a more general edge magnitude faster than the implementation at [14]. (Note that their implementation supports a more general edge magnitude faster than the implementation at [14]. (Note that their implementation supports a more general edge magnitude faster than the implementation at [14]. (Note that their implementation supports a more general edge magnitude faster than the implementation at [14]. (Note that their implementation supports a more general edge magnitude faster than the implementation at [14]. (Note that their implementation supports a more general edge magnitude faster than the implementation at [14]. (Note that their implementation supports a more general edge magnitude faster than the implementation at [14]. (Note that their implementation supports a more general edge magnitude faster than the implementation at [14]. (Note that their implementation supports a more general edge magnitude faster than the implementation at [14]. (Note that their implementation supports a more general edge magnitude faster than the implementation at [14]. (Note that their implementation supports a more general edge magnitude faster than the implementation at [14]. (Note that their implementation supports a more general edge magn...
$n = 10^4$, our impl.
$n = 10^5$, our impl.
$n = 10^6$, our impl.
$n = 10^7$, our impl.
$n = 10^8$, impl. at [14]
$n = 10^9$, impl. at [14]
$n = 10^4$, theoretical fit
$n = 10^5$, theoretical fit
$n = 10^6$, theoretical fit

Figure 4: Comparison of running times to generate networks with 10^4-10^6 vertices. Circles represent running times of our implementation, diamonds the running times of the implementation at [14]. The running times of our implementation are fitted with the equation $T(n) = (1.09 \cdot 10^{-6}n + 2.29 \cdot 10^{-8}n^{3/2} + 1.51 \cdot 10^{-8}m) \log n$ seconds.

[3] James W Anderson. Hyperbolic geometry; 2nd ed. Springer undergraduate mathematics series. Springer, Berlin, 2005.

[4] Marián Boguñá, Fragkiskos Papadopoulos, and Dmitri Krioukov. Sustaining the internet with hyperbolic mapping. Nature Communications, (62), September 2010.

[5] Deepayan Chakrabarti and Christos Faloutsos. Graph mining: Laws, generators, and algorithms. ACM Computing Surveys (CSUR), 38(1):2, 2006.

[6] Deepayan Chakrabarti, Yiping Zhan, and Christos Faloutsos. R-mat: A recursive model for graph mining. In SDM, volume 4, pages 442–446. SIAM, 2004.

[7] Luca Gugelmann, Konstantinos Panagiotou, and Ueli Peter. Random hyperbolic graphs: Degree sequence and clustering - (extended abstract). In Automata, Languages, and Programming - 39th International Colloquium, ICALP 2012, Proceedings, Part II, 2012.

[8] Robert Kleinberg. Geographic routing using hyperbolic space. In INFOCOM, 2007.

[9] Tamara G. Kolda, Ali Pinar, Todd Plantenga, and C. Seshadhri. A scalable generative graph model with community structure. SIAM J. Scientific Computing, 36(5):C424–C452, Sep 2014.

[10] Dmitri Krioukov, Fragkiskos Papadopoulos, Maksim Kitsak, Amin Vahdat, and Marián Boguñá. Hyperbolic geometry of complex networks. Physical Review E, 82, Sep 2010.

[11] Joel C Miller and Aric Hagberg. Efficient generation of networks with given expected degrees. In Algorithms and Models for the Web Graph, pages 115–126. Springer, 2011.

[12] Michael Mitzenmacher and Eli Upfal. Probability and computing: Randomized algorithms and probabilistic analysis. Cambridge University Press, 2005.

[13] Mark Newman. Networks: an introduction. Oxford University Press, 2010.

[14] Chiara Orsini and Rodrigo Aldecoa. Hyperbolic generator. https://github.com/named-data/Hyperbolic-Graph-Generator/releases/tag/v.1.0.1

[15] Fragkiskos Papadopoulos, Dmitri Krioukov, Marián Boguñá, and Amin Vahdat. Greedy forwarding in dynamic scale-free networks embedded in hyperbolic metric spaces. In INFOCOM, 2010 Proceedings IEEE, pages 1–9. IEEE, 2010.

[16] Hanan Samet. Foundations of Multidimensional and Metric Data Structures. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2005.

[17] C Seshadhri, Tamara G Kolda, and Ali Pinar. Community structure and scale-free collections of erdős-renyi graphs. Physical Review E, 85(5):056109, 2012.

[18] C. Seshadhri, Ali Pinar, and Tamara G. Kolda. The similarity between stochastic kronecker and chung-lu graph models. In Proc. 2012 SIAM Intl. Conf. on Data Mining (SDM), pages 1071–1082, 2012.

[19] C. Staudt and H. Meyerhenke. Engineering parallel algorithms for community detection in massive networks. Parallel and Distributed Systems, IEEE Transactions on, 2015. doi: 10.1109/TPDS.2015.2390633.
A Derivation of Proposition 3.1

The radial coordinates $r_{c_1,2}$ can be derived with several transformations from the definition of the hyperbolic distance:

\begin{equation}
\cosh(r_{h}) - 1 = \frac{2(r_e - r_h)^2}{(1 - r_h^2)(1 - r_e^2)}
\end{equation}

\begin{equation}
(\cosh(r_{h}) - 1)(1 - r_h^2) = \frac{2(r_e^2 - 2r_hr_e + r_h^2)}{1 - r_h^2}
\end{equation}

To keep the notation short, define $a = \cosh(r_{h}) - 1$ and $b = (1 - r_h^2)$. It follows:

\begin{equation}
a - r_h^2(a) = \frac{2(r_e^2 - 2r_hr_e + r_h^2)}{b}
\end{equation}

\begin{equation}
a = r_e^2(a) + \frac{2(r_e^2 - 2r_hr_e + r_h^2)}{b}
\end{equation}

\begin{equation}
a = \frac{r_e^2(a) + 2r_h}{b} + \frac{2r_h}{b}
\end{equation}

\begin{equation}
0 = r_e^2(a) + \frac{2r_h}{b} - a
\end{equation}

\begin{equation}
0 = r_e^2(a) + \frac{2r_h}{b(a + \frac{2}{a})} - a
\end{equation}

Solving this quadratic equation, we obtain:

\begin{equation}
r_{c_1,2} = \frac{2r_h}{ab + 2} \pm \sqrt{\left(\frac{2r_h}{ab + 2}\right)^2 - \frac{4r_h^2 - ab}{ab + 2}}
\end{equation}

Since $(\phi_h, r_{c_1})$ and $(\phi_h, r_{c_2})$ are different points on the border of $E$, the center $E_c$ needs to be on the perpendicular bisector. Its radial coordinate $r_{E_c}$ is thus $(r_{c_1} + r_{c_2})/2 = \frac{2r_h}{ab + 2}$. To determine the angular coordinate, we need the following lemma:

**Lemma A.1.** Let $H$ be a hyperbolic circle centered at $(\phi_h, r_h)$ and radius $r_{h}$. The center $E_c$ of the corresponding Euclidean circle $E$ is on the ray from $(\phi_h, r_h)$ to the origin.

**Proof.** Let $p$ be a point in $H$, meaning $\text{dist}_H(p, (\phi_h, r_h)) \leq r_{h}$. Let $p'$ be the mirror image of $p$ under reflection on the ray going through $(\phi_h, r_h)$ and $p$. $(\phi_h, r_h)$ is on the ray and unchanged under reflection: $(\phi_h, r_h) = (\phi_h, r_h)'$. Since reflection on the equator is an isometry in the Poincaré disk model and preserves distance, we have $\text{dist}_H(p', (\phi_h, r_h)) = \text{dist}_H(p, (\phi_h, r_h)) \leq r_{h}$. If $p' \in H$. The Euclidean circle $E$ is then symmetric with respect to the ray and $E_c$ must lie on it.

The radius of the circle is then derived from the distance of the center to $(\phi_h, r_{c_1})$ and $(\phi_h, r_{c_2})$, which is $\sqrt{\left(\frac{2r_h}{ab + 2}\right)^2 - \frac{4r_h^2 - ab}{ab + 2}}$. With both radial and angular coordinates of $E_c$ fixed, Proposition 3.1 follows.

B Proof of Theorem 3.1

To prove Theorem 3.1, we use the area for circles in hyperbolic space and an auxiliary lemma. The area of a circle is proportional to the cosine hyperbolic of its radius $R$:

\begin{equation}
\text{area}(R) = (2\pi\cosh(R) - 1)
\end{equation}

**Lemma B.1.** The area of a quadtree cell delimited by $\min_r, \max_r, \min_{\phi}, \max_{\phi}$ is

\begin{equation}
\text{area}(\min_r, \max_r, \min_{\phi}, \max_{\phi}) = (\max_{\phi} - \min_{\phi}) \cdot \left(2\pi\cosh(\max_r) - \cosh(\min_r)\right)
\end{equation}

**Proof.** In the case of $\min_{\phi} = 0$ and $\max_{\phi} = 2\pi$, the resulting cell is a ring, delimited by $\min_r$ and $\max_r$. The area is then the area of the outer circle excluding the area of the inner circle:

\begin{equation}
\text{area}(\min_r, \max_r, 0, 2\pi) = (2\pi(\cosh(\max_r) - 1)) - (2\pi(\cosh(\min_r) - 1))
\end{equation}

If the angular range of a quadtree cell is smaller than $2\pi$, the resulting area is a fraction of the ring area:

\begin{equation}
\text{area}(\min_r, \max_r, 0, 2\pi) = \frac{\max_{\phi} - \min_{\phi}}{2\pi} \cdot \left(2\pi\cosh(\max_r) - \cosh(\min_r)\right)
\end{equation}

Note that Lemma B.1 yields Eq. (2.14) if the quadtree cell in question covers the whole disk:

\begin{equation}
\text{area}(0, R, 0, 2\pi) = (2\pi - 0) \cdot (\cosh(R) - \cosh(0)) = 2\pi(\cosh(R) - 1).
\end{equation}

We proceed to prove Theorem 3.1 by induction.

**Proof.** Start of induction ($i = 0$): At level 0, only the root cell exists and covers the whole disk. The requirement is fulfilled with Eq. (2.17).

Inductive step ($i \to i + 1$): Let $d$ be a node at level $i$, $d$ is delimited by the radial boundaries $\min_r$ and $\max_r$,
as well as the angular boundaries \( \min_\phi \) and \( \max_\phi \). It has four children at level \( i + 1 \), separated by \( \text{mid}_r \) and \( \text{mid}_\phi \). Let \( d' \) be without loss of generality the south west child of \( d \). With Lemma 3.1 the area of \( d' \) is:

\[
(2.18) \quad \text{area}(d') = (\text{mid}_\phi - \min_\phi) \cdot (\cosh(\text{mid}_r) - \cosh(\min_r))
\]

The angular range is halved and \( \text{mid}_r \) is selected according to Eq. (3.3). This results in an area of \( \frac{1}{2}(\max_\phi - \min_\phi) \)

\[
\cdot (\cosh(\acosh(\frac{\cosh(\max_r) + \cosh(\min_r)}{2}) - \cosh(\min_r))
\]

\[
= \frac{1}{2}(\max_\phi - \min_\phi)
\]

\[
\cdot (\cosh(\max_r) + \cosh(\min_r))/2 - \cosh(\min_r))
\]

\[
= \frac{1}{2}(\max_\phi - \min_\phi) \cdot \frac{1}{2}(\cosh(\max_r) - \cosh(\min_r))
\]

\[
= \frac{1}{4} \text{area}(d)
\]

As per the induction hypothesis, \( \text{area}(d) \) is \( 2\pi(\cosh(R) - 1)/4^i \) and \( \text{area}(d') \) thus \( \frac{1}{4} \cdot 2\pi(\cosh(R) - 1)/4^i = 2\pi(\cosh(R) - 1)/4^{i+1} \).

\[
\square
\]

C Proof of Theorem 3.2

We say “with high probability” when referring to a probability of at least \( 1 - 1/n \) (for sufficiently large \( n \)). While previous results exist for the height and cost of two-dimensional quadtrees [16], these quadtrees differ from our polar hyperbolic approach in important properties and the results are not easily transferable. For example, the side length of a cell is not proportional to its area. We thus make use of a lemma from the theory of balls into bins instead:

\textbf{Lemma C.1.} (12) \textit{When \( n \) balls are thrown independently and uniformly at random into \( n \) bins, the probability that the maximum load is more than \( 3 \ln n / \ln \ln n \) is at most \( 1/n \) for \( n \) sufficiently large.}

\textit{Proof.} [of Theorem 3.2] In a complete quadtree, \( 4^k \) cells exist at height \( k \). For analysis purposes only, we construct such a complete but initially empty quadtree of height \( k = \lfloor \log_4(n) \rfloor \), which has at least \( n \) leaf cells. As seen in Theorem 3.1 each tree cell at a given height covers an equally-sized area of hyperbolic space. Since the points (= vertices) are distributed uniformly in hyperbolic space, a given point has thus an equal chance to land in each leaf cell. Hence, we can apply Lemma C.1 with each leaf cell being a bin and a point being a ball. (The fact that we can have more than \( n \) leaf cells only helps in reducing the average load.) From this we can conclude that, for \( n \) sufficiently large, no leaf cell of the current tree contains more than \( 3 \ln n / \ln \ln n \) points with high probability (whp). Even if we had to construct a subtree below a current leaf \( l \) to store points whose number exceeds the capacity of \( l \), the height of this subtree cannot exceed the number of points in the corresponding area, which is at most \( 3 \ln n / \ln \ln n \) whp. Consequently, the total quadtree height does not exceed \( O(\log n) \) whp.

Let \( T' \) be the quadtree as constructed in the previous paragraph and let \( T \) be the quadtree created in our algorithm. The construction of \( T \) creates only quadtree nodes that are necessary due to point insertions. In \( T' \) we have all necessary nodes as well, but potentially more. Thus, \( T \) does not have nodes that do not exist in \( T' \). Consequently, its height is bounded by \( O(\log n) \) whp as well.

\[
\square
\]
D Further Graph Property Parameter Studies

Figure 5: Further parameter studies, omitted from Figure 3 due to space constraints.

E Effect of Additional Random Edges

Figure 6: Effect of random long-range edges on diameter ranges. Baseline graphs are generated with an average degree of 10, values are averaged over 5 runs. Black circles correspond to PGPgiantcompo, caidaRouterLevel, citationCiteseer and as-Skitter from Table I which have a comparable density.

F Parameter Studies with Varying Threshold Factor $t$

Figure 8 shows the properties of generated graphs depending on parameters $\alpha$ and $t$. It is similar to Figure 3 but varies $t$ instead of $s$. The effect is almost exactly inverted, similar graphs can be generated by choosing a low $t$ or by choosing a high $s$. We focus on $s$ in the rest of the paper, since its effect is better analyzed theoretically. The vertex dispersion $\alpha$ starts from 0.8 (lower values tend to result in a complete graph) and goes up to 3. The threshold factor $t$ is in the range $(0.1, 1)$. Choosing values over 1 results in very dense graphs quickly, a complete graph is generated for $t = 2^3$.

---

Each vertex has a hyperbolic distance to the origin of at most $R$. Due to the triangle inequality, the maximal distance between two vertices is at most $2R$. 

11
Figure 7: Side effects of adding random edges to generated graphs. Baseline graphs are generated with an average degree of 10, values are averaged over 5 runs. The clustering coefficient changes by less than 0.03, the two lines of the likelihood of a power-law degree distribution are nearly identical.

Figure 8: Properties of graphs generated with the hyperbolic generator for 100,000 vertices and different values for the parameters $\alpha$ and $t$. Higher values of $\alpha$ cause fewer edges to be generated, as do lower values of $t$. With too few edges, the graphs become disconnected and noise of other property measurements increases.