University of Warsaw
Faculty of Economic Sciences

Working Papers
No. 20/2017 (249)

# HYPERBOLIC GRIDS AND DISCRETE RANDOM GRAPHS

Eryk Kopczyński
Dorota Celińska

Warsaw 2017

University of Warsaw
Faculty of Economic Sciences

# Hyperbolic grids and discrete random graphs

**Eryk Kopczyński**
Institute of Informatics
University of Warsaw
e-mail: erykk@mimuw.edu.pl

**Dorota Celińska**
Faculty of Economic Sciences
University of Warsaw
Narodowy Bank Polski
e-mail: dcelinska@wne.uw.edu.pl

**Abstract**

We present an efficient algorithm for computing distances in hyperbolic grids. We apply this algorithm to work efficiently with a discrete variant of the hyperbolic random graph model. This model is gaining popularity in the analysis of scale-free networks, which are ubiquitous in many fields, from social network analysis to biology. We present experimental results conducted on real world networks.

**Keywords:**

computational geometry, hyperbolic geometry, scale-free networks, hyperbolic random graphs

**JEL:**

C02, C55, C65

**Acknowledgements**

We are very grateful to the anonymous referees for their careful reading of an earlier version of this work. Many parts of the paper have been greatly improved as a result of their insightful and constructive comments. This work was supported by the National Science Centre, Poland, grant DEC-2016/21/N/HS4/02100.

**Note**

This is a work in progress. Some facts are not yet proven, although they intuitively should be true, or are true for the specific grids $G_7$ and $G_{67}$ we are working with. Furthermore, we believe that our algorithms (or at least our implementations) still can be improved.

**DOI:**

https://doi.org/10.26405/WP/WNE/2017/249/020

# 1 Introduction

Hyperbolic geometry has been discovered by 19th century mathematicians wondering about the nature of parallel lines. One of the properties of this geometry is that the amount of area in distance $d$ from a given point is exponential in $d$; intuitively, the metric structure of the hyperbolic plane is similar to that of an infinite binary tree, except that each vertex is additionally connected to two adjacent vertices on the same level. Anyone who has tried to draw a full binary tree of height 10 knows that there is not enough space in the Euclidean space; exponential expansion typical to hyperbolic geometry has found applications in visualization of hierarchical data [Lamping et al., 1995, Munzner, 1998].

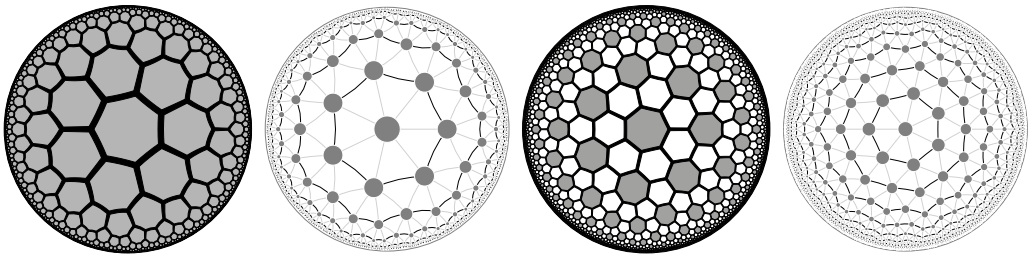

Figure 1: (a) order-3 heptagonal tiling, (b) the grid $G_7$, (c) truncated triangular tiling, (d) the grid $G_{67}$.

Figure 1 shows two tilings of the hyperbolic plane, the order-3 heptagonal tiling and the truncated triangular tiling, in the Poincaré disk model, together with their dual graphs, which we call $G_7$ and $G_{67}$. In the Poincaré model, the hyperbolic plane is represented as a disk. In the hyperbolic metric, all the triangles, heptagons and hexagons on each of these pictures are actually of the same size, and the points on the boundary of the disk are infinitely far from the center.

Recently, hyperbolic geometry has found application in the analysis of scale-free networks, which are ubiquitous in many fields, from network analysis to biology. [Papadopoulos et al., 2012]. In the hyperbolic random graph (HRG) model, each vertex is randomly assigned a point in the hyperbolic plane (according to some distribution), and then, we connect each pair of vertices with probability depending on the distance between the two points. HRGs turn out to have properties similar to that of scale-free networks. Efficient algorithms have been found for generating HRGs in time $O(n)$ [Bringmann et al., 2015] and for MLE embedding real world scale-free networks into the hyperbolic plane in time $\tilde{O}(n)$ [Bläsius et al., 2016], which was a major improvement over previous algorithms [Papadopoulos et al., 2015, von Looz et al., 2015].

In this paper, we present efficient algorithms for calculating distances in hyperbolic triangular grids, such as $G_{67}$ and $G_7$ from Figure 1. Our algorithm can be applied to work efficiently with discrete analogs of HRGs, which we call DHRGs, allowing us to compute the log-likelihood of an embedding in time and memory $O(nD^2)$, where $D$ is the radius of the grid and can be seen as logarithmic; this is a very important step for MLE embedders. Furthermore, it is possible to dynamically remap a vertex $v$ to another location and compute the log-likelihood of the new embeddding in time $O(D^2 + D \deg(v))$, which yields an efficient local search algorithm to improve the embedding – which might be useful on its own [Bringmann et al., 2016]. We can also generate DHRGs in time $O(nD^2 + mD)$. This matches the time and complexity of the best continuous algorithms listed in the previous section up to $D^{O(1)}$ factors.

We believe that such grid-based approach is interesting for the following reasons:

**Continuity is not essential in the HRG model.** For example, the Facebook graph from [Bläsius et al., 2016] has 4000 vertices, with typical distances between pairs of vertices connected by an edge are around 10 absolute units, which corresponds to roughly 18 steps in grid $G_{67}$. Thus, moving each point towards the nearest grid point does not significantly affect the distances between points.

**Approachability.** By avoiding continuous computations, our model is potentially more approachable to theoretical computer scientists who are more focused on discrete problems.

**Simplicity.** It is worth to note that the major breakthrough in [Bringmann et al., 2015] and [Bläsius et al., 2016] was achieved by using geometric structures based on partitioning hyperbolic disks into cells. This is in some sense similar to our grids. However, we believe that our approach is more elegant and simpler.

**Hyperbolic grids have other applications.** Regular grids in the hyperbolic plane have many applications, such as self-organizing maps [Ontrup and Ritter, 2001] or game design [Kopczyński et al., 2017]; in particular, we recommend playing our game HyperRogue as an intuitive introduction to hyperbolic grids and hyperbolic geometry in general. Hyperbolic grids arise naturally when working with bounded degree planar graphs; for example, many constructions in [Dawar and Kopczynski, 2016] are based on hyperbolic grids.

**Precision errors.** Radial coordinates as used in [Papadopoulos et al., 2012, Gugelmann et al., 2012, Bläsius et al., 2016] are prone to precision errors. This is unavoidable in any typical coordinate system because of the exponential nature of hyperbolic geometry. Indeed, the circumference of a

hyperbolic circle of radius $r$ in $2\pi \sinh(r) = \Theta(e^r)$. Therefore, if we are using $b$ bits for the angular coordinate, two points on the circle of radius $b/\log(2)+\Theta(1)$ will be smashed into a single point, even if their exact distance is greater than 1. On the other hand, vertices in grids can be represented with paths from some specific "origin" vertex, thus avoiding these problems. Even if we are working with the continuous hyperbolic plane, a "hybrid" approach where each point is represented by a vertex of our grid together with the coordinates relative to that vertex. Such approach is used in HyperRogue [Rogue, 2012].

**Visualizations.** We can visualize a graph by mapping each of its vertices to a distinct vertex of our grid. Since the vertices in our grids are spaced regularly, this allows for aesthetically pleasant representations of graphs [Celińska and Kopczyński, 2017].

**Hyperbolic grids are interesting on their own.** It is well known that many algorithmic problems can be easily solved on trees; it is also well known that many graph problems admit very efficient algorithms on graphs that are similar to a tree, where similarity is most commonly measured using the notion of *tree width* [Robertson and Seymour, 1984]. For example, every fixed graph property definable in the monadic second order logic with quantification over sets of vertices and edges ($MSO_2$) can be checked in linear time on graphs of fixed tree width [Courcelle, 1990]. Hyperbolic metrics are very tree-like – however, tree width might be not the best measure of this, as the tree width of a neighborhood of radius $r$ in our grids is linear in $r$ rather than constant; while this is small (logarithmic in the size of the graph), it is not a constant. Other metrics, such as the Gromov hyperbolicity [Bermudo et al., 2013], which is constant, or the tree length [Chepoi et al., 2008], which is provably $O(\delta \log n)$ on graphs with $n$ vertices and Gromov hyperbolicity $\delta$ [Chepoi et al., 2008], might be more appropriate.

In the next section we present the general definition of the hyperbolic triangular grid we will be working with, and some basic facts about such grids. In section 3 we focus on grids whose structure is regular. In Section 4 we introduce our algorithms calculating distances in the grid. In Section 5 we study how the distances in our grids are related to distances in the underlying hyperbolic plane. In Section 6 we recall the definition of the HRG model, and define our own DHRG model, based on the intuitions from Section 5. We analyze the DHRG model in Section 7, then show how to apply our algorithms to work with DHRGs efficiently in Section 8. We have implemented some of the algorithms presented in Sections 4 and 6; the experimental results regarding the DHRG model are presented in Section 9. We discuss possible directions for further work in Section 10.

## 2 Hyperbolic triangular grids

In this paper by $\delta(v, w)$ we denote the distance between $v$ and $w$; in case if $v$ and $w$ are vertices of the graph it is the length of the shortest path between $v$ and $w$, and if they are points in the hyperbolic plane, it is their hyperbolic distance.

**Definition 2.1.** *A* **triangular grid** *is $G = (V, E, v_0)$ where:*

- *$(V, E)$ is an infinite planar graph such that every face is a triangle,*
- *for every $k > 0$, the $k$-th ring $R_k(G) = \{v \in V | \delta(v, v_0) = k\}$ is a cycle,*
- *$\lim_{k\to\infty} |R_k(G)| = \infty$.*
- *for every $v \in V$ except $v_0$, there exists at least one $w \in R_1(G)$ such that $\delta(v, w) > \delta(v, v_0)$.*

We assume that all the rings $R_k(G)$ are oriented clockwise around $v_0$. Thus, the $i$-th successor of $v$, denoted $v+i$, is the vertex obtained by starting from $v$ and going $i$ vertices on the cycle. The $i$-th predecessor of $v$, denoted $v - i$, is obtained by going $i$ vertices backwards on the cycle. A *segment* is the set $S = \{v, v + 1, \ldots, v + k\} \subsetneq R_k(G)$ for some $v \in V$ and $k \geq 0$; $v$ is called the leftmost element of $S$, and $v + k$ is called the rightmost element of $S$. By $[v, w]$ we denote the segment such that $v$ is its leftmost element, and $w$ is its rightmost element. For $v, w \in R_k(G)$, let $w - v$ be the smallest $i \geq 0$ such that $w = v + i$. We also denote $\delta_0(v) = \delta(v, v_0)$. By $B_k(G)$ we denote the $k$-th ball (neighborhood), i.e., $\bigcup_{i=0,\ldots,k} R_k(G) = \{v \in V | \delta(v, v_0) \leq k\}$.

We use tree-like terminology for connecting the rings. A vertex $w$ is a **parent** of $v$ if $\{v, w\} \in E$ and $\delta_0(v) = \delta_0(w) + 1$; in this case, $v$ is a **child** of $w$. Let $P(v)$ be the set of parents of $v \in R_k(G)$; it forms a segment of $R_{k-1}(G)$, and its leftmost and rightmost elements are called the **left parent** $p_L(v)$ and the **right parent** $p_R(v)$ respectively. The set of children $C(v)$, leftmost child $c_L(v)$ and rightmost child $c_R(v)$ are defined analogously.

Figure 2 presents the grid $G_{67}$ with named vertices. Both pictures use the Poincaré disk model, but the left picture is centered roughly at $v_0$ (labeled with $A$ in the picture), and the right picture is centered at a different location in the hyperbolic plane. Points drawn close to the boundary of the Poincaré disk are further away from each other than they appear – for example, vertices $T$ and $U$ appear very close in the left picture, yet in fact all the edges are roughly of the same length (in fact, there are two lengths – the distance between two vertices of degree 6 is slightly different than the distance between a vertex of degree 6 and a vertex of degree 7).

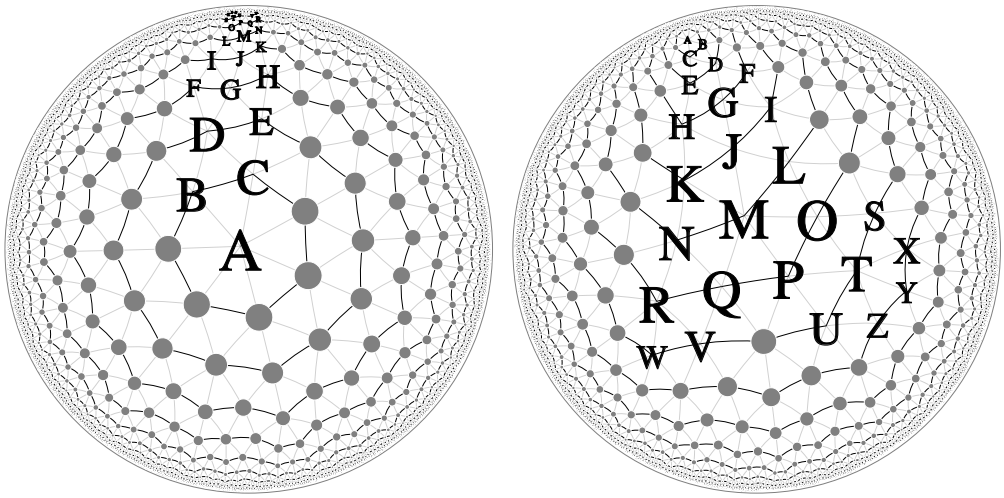


Figure 2: Grid $G_{67}$ with labeled vertices, in two perspectives.

Vertices $X$, $Y$, and $Z$ are the children of $T$; its siblings are $S$ and $U$, and its parents are $O$ and $P$. The values of $P^k([Y])$ for consecutive values of $k$, i.e., the ancestor segments of $Y$, are: $[Y]$, $[T]$, $[O, P]$, $[L, M]$, $[I, K]$, $[F, H]$, $[D, E]$, $[B, C]$, $[A]$. Vertex $W$ has just a single ancestor on each level: $R$, $N$, $K$, $H$, $E$, $C$, $A$. Vertex $V$ has the following ancestor segments: $[Q, R]$, $[M, N]$, $[J, K]$, $[G, H]$, $[D, E]$, $[B, C]$, $[A]$. Note the tree-like nature of our grid: $[D, E]$ is the segment of ancestors for both $V$ and $Y$, and $[O, P]$ and $[Q, R]$ are already adjacent. This tree-like nature will be useful in the algorithms in the next section.

**Definition 2.2.** *A triangular grid $G$ is* **hyperbolic** *if:*

- *every vertex has at most two parents and at least two children,*
- *there exists a number $t(G)$ (denoted with $t$ for brevity) such that if $S \subseteq R_k(v)$ is a segment and $|S| \geq t$, $C(S) \geq t + 2$.*

Both grids $G_{67}$ and $G_7$ are hyperbolic. Probably the simplest, though geometrically less regular, example of a hyperbolic triangular grid is obtained by taking a full infinite binary tree, and additionally connecting each vertex to its cyclic left and right sibling, and additionally the right child of its left sibling.

**Proposition 2.3.** *Let $G$ be a hyperbolic triangular grid, and $v, w \in R_k(G)$. If $w - v \geq at - 1$, then $c_R(w) - c_L(v) \geq at + a$.*

*Proof.* Let $S_i = \{v + it, v + it + 1, \ldots, v + it + (t-1)\}$ for $i \in \{0, \ldots, a-1\}$. From the definition of the hyperbolic triangular grid, $|C(S_i)| \geq t + 2$. The segments $C(S_i)$ need not be disjoint, but the only case when they are not disjoint is that the rightmost element of $C(S_i)$ equals the leftmost element of $C(S_{i+1})$. Hence, $|\bigcup C(S_i)| \geq at + a + 1$. □

**Proposition 2.4.** *Let $G$ be a hyperbolic triangular grid, and $v, w \in R_k(G)$. There exists a number $t_k(G)$ such that if $w - v \geq t_k(G)$, then $p_R(w) - p_L(v) \leq w - v - k$.*

*Proof.* Should be similar. Not written yet. □

**Proposition 2.5.** *For a hyperbolic triangular grid $G$, $|R_k(G)|$ is at least exponential in $k$.*

*Proof.* From the definition ($\lim_k |R_k(G)| = \infty$) and Proposition 2.3, we can easily show that $|R_k(G) = \Omega((1 + \frac{1}{t})^k)$. □

# 3 Uniform hyperbolic grids

Triangular grids as defined in the previous section did not necessarily have any regular structure. In this section we define *uniform* triangular grids; their structure allows us both to efficiently handle them algorithmically, and to obtain combinatorial results.

By $T^*$ we denote the set of all words over an alphabet $T$. A function $c : T \to T^*$ can be naturally extended to a homomorphism, which we also denote with $c : T^* \to T^*$. Let $\#(w) : T \to \mathbb{N}$ be the vector counting the number of occurences of each letter in $w \in T^*$. It is easy to see that $\#(cw) = C\#(w)$ for some matrix $C$. Since we already

**Definition 3.1.** *We say that a triangular grid $G$ is uniform iff there exists a finite set of types $T$, a function $c : T \to T^*$, and an assignment $t : V(G) \to T$ of types to vertices, such that for each $v \in V(G)$, the sequence of types of all children of $v$ from left to right except the rightmost child is given by $c(t(v))$. Furthermore, there is a density function $g : T \to \mathbb{R}$ such that, for each $t, u \in T$, $\lim_{n\to\infty} \#_u c^n(t) = g(u)$.*

For example, in the grid $G_7$, we take $T = \{0, 1, 2\}$, where $t(v)$ is the number of parents of $v$. The root has no parents, and has seven children of type 1. A vertex of type 1 has the leftmost child of type 2, followed by two children of type 1. A vertex of type 2 has the leftmost child of type 2, followed by a child of type 1.

The situation is more complicated in the grid $G_{67}$. We have 7 types, with $t(v)$ depending on: the degree of $v$, the number of parents of $v$, and the degree of the left sibling of $v$. Given that each vertex of degree 7 has 7 neighbors of degree 6, and the neighbors of vertices of degree 6 alternate between degree 6 and 7, we can compute the sequence of types of children for each of the 7 types.

**Proposition 3.2.** *In a uniform hyperbolic triangular grid $G$, for each $v \in G$, $|C_k(v)| = \Theta(\gamma^k)$ for some $\gamma$, where $\gamma$ depends only on $G$.*

*Proof.* It is easy to see that $|C_k(G)| = |c^k(t(v))|+1$, and $\#(cw) = C\#(w)$ for some matrix $C$. Since we already have an exponential lower bound from 2.5, our result now can be obtained easily using well known properties of matrix exponentiation. The existence of the density function $g$ ensures that $\gamma$ does not depend on $t(v)$. □

We have $\gamma \approx 2.6180339$ for $G_7$ and $\gamma \approx 1.72208$ for $G_{67}$ [Kopczyński et al., 2017].

It is not feasible to represent all vertices in, say, $B_{100}(G_{67})$ in computer memory – there are more than $10^{23}$ of them! However, we can generate the vertices in our uniform triangular grid lazily. that is, represent our vertices with pointers, start from the root, and generate other vertices when asked for them. In particular, each vertex $v$ is represented with a pointer to a structure which contains $\delta_0(v)$, the type of $v$, the pointers to $p_L(v)$, $p_R(v)$, $v-1$, $v+1$, $c_L(v)$, and the index of $v$ among the children of $p_R(v)$; the last three pointers are NULL if the given neighbor has not yet been computed. Such a structure allows us to compute all the neighbors of the given vertex in amortized time $O(1)$ for a fixed grid. This is similar to the approach used in [Margenstern, 2013] and [Rogue, 2012] for $G_7$ (though in [Rogue, 2012] a different algorithm is used for $G_{67}$, based on subdividing $G_7$ rather than working with the rings in $G_{67}$ directly).

# 4 Computing distances in hyperbolic grids

In this section we show how to compute distances in the hyperbolic grid, based on the data structure outlined in the previous section.

**Proposition 4.1.** *Let $v, w \in V(G)$ for a hyperbolic triangular grid $G$, and $\delta(v, w) = d$. Then at least one of the following is true:*

- $v \in P^d(w)$,

- $w \in P^d(v)$,
- $p_R^a(v) + b = p_L^c(w)$, *where* $a + b + c = d$,
- $p_R^a(w) + b = p_L^c(v)$, *where* $a + b + c = d$.

*Furthermore,* $b \leq t_3$, *where* $t_3$ *is from Proposition 2.4.*

In other words, the shortest path between any pair of two vertices $(v, w)$ can always be obtained by going some number of steps toward $v_0$, moving along the ring, and going back away from $v_0$. The cases where one of the vertices is an ancestor of the other one had to be listed separatedly because it is possible that $|P^a(v)| > 2$ for $a > 1$, thus $w$ might be neither the leftmost not the rightmost ancestor.

Such a situation happens in the grid $G_{67}$ for the pair of vertices labeled $(J, O)$ in Figure 2, even though $|P^a(v)| \leq 3$ always holds. Let's analyze the grid $G_{67}$ in more detail. Take vertices $v$ and $v + k$, where $k \geq 0$ and $\delta_0(v) = d$. The distance $\delta(v, v + k)$ could be $k$, or it could be achieved by an inner path, i.e., one that goes through $B_{d-1}$; denote the length of the shortest inner path by $\delta_I(v, v + k)$. For $k < 2$ we have $\delta(v, v + k) = k < \delta_I(v, v + k)$, for $k = 2$ we have $\delta(v, v + k) = k \leq \delta_I(v, v + k)$, for $k \geq 4$ we have $k \geq \delta_I(v, v + k) = \delta(v, v + k)$. For $k = 3$ both $k$ and $\delta_I(v, v + k)$ can be smaller – it is possible that $p_R(v_d) = p_L(v_{d+2})$ (e.g. $V$ and $V + 3$ are both children of $R$), but it is also possible that $p_R(v_d) - p_L(v_{d+2}) = 2$ (as happens for $U$ and $W$ in Figure 2). The situation is similar, though a bit simpler, in the grid $G_7$. These observations allow us to create an efficient algorithm computing distances between vertices of our grid, which can be generalized to any uniform hyperbolic triangular grid.

*Proof of Proposition 4.1.* Not written yet. □

**Proposition 4.2.** *Fix a hyperbolic triangular grid* $G$. *Then* $\delta(v, w)$ *can be computed for* $v, w \in G$ *in time* $O(\delta(v, w))$.

*Proof.* Our algorithm uses five integer variables $a, c, d_v, d_w, d$ and four vertex variables $v_L, v_R, w_L, w_R$, satisfying the following invariant:

- $\delta_0(v_L) = \delta_0(v_R) = d_v, \delta_0(w_L) = \delta_0(w_R) = d_w$
- $v_L = p_L^a(v), v_R = p_R^a(v), w_L = p_L^c(w), w_R = p_R^c(w)$
- $d$ is the upper bound on $\delta(v, w)$, more precisely, the length of the shortest path between $v$ and $w$ among those which are of the form from Proposition 4.1, and such that $a$ and $c$ from $\delta(v_L, v_0) = \delta(v_R, v_0) = d_v$ are smaller than our $a$ and $c$.

Start by setting $d_v := \delta_0(v)$, $d_w := \delta_0(w)$, $a := 0$, $c := 0$, $v_L := v$, $v_R := v$, $w_L = w$ and $w_R = w$, $d := \infty$. The invariant is satisfied.

While $d_v > d_w$, we push $v$ down, i.e., set $a := a + 1$, $v_L := p_L(v_L)$ and $v_R := p_R(v_R)$. While $d_w > d_v$, we push $w$ down, i.e., set $c := c + 1$, $w_L := p_L(w_L)$ and $w_R := p_R(w_R)$. The invariant is still satisfied.

Now, $d_v = d_w$, hence $v_L$, $v_R$, $w_L$ and $w_R$ are all in $d_v$-th ring. If $a+c > d$, we are sure that $d$ is the correct value of $\delta(v, w)$, and thus we terminate the algorithm. Otherwise, we can check in $O(1)$ whether $w_L - v_R \leq f(t)$, $v_L - w_R \leq f(t)$, $w$ is between $v_L$ and $v_R$, or $v$ is between $w_L$ and $w_R$ – if yes, this gives us a new upper bound on $\delta(v, w)$. However, it is still possible that a better path is to be found on one of the following levels, thus we push down both $v$ and $w$. The invariant is still satisfied, and we repeat.

The algorithm runs in time $O(\delta(v, w))$, because the values of both $a$ and $c$ in the optimal path are smaller than $\delta(v, w)$, and we do at most $t_3$ extra iterations after finding the optimal path. □

Remark: with some preprocessing, we can optimize to $O(\log \delta(v, w))$ per query – precompute $p_L^a(v)$ for each $v \in V$ and $a \in \mathbb{N}$, or for each $a$ that is a power of two.

A **distance tally counter** for a graph $G = (V, E)$ represents a function $f : V \to \mathbb{R}$ with the following operations:

- Initialize: $f$ is initialized with the constant 0 function
- Add($v$, $k$): add $k$ to $f(v)$
- Tally($v$): return an array $A$ such that $A[d] = \sum_{w \in V : \delta(v,w) = d} f(w)$

**Theorem 4.3.** *If $G$ is a hyperbolic triangular grid, a distance tally counter can be implemented working in memory $O(\sum_{w \in W : f(w) \neq 0} \delta_0(w)^2)$, initialization in time $O(1)$, and Add($v$) and Tally($v$) in time $O(\delta_0(v)^2)$.*

It is also possible to optimize either memory or Add time by a factor of $\delta_0(v)$. In our applications $\delta_0(v)$ is small, and thus could be considered to be logarithmic in the size of the graph.

*Proof.* A segment is **good** if it is of form $P^d([v, v])$ for some $v \in V$ and $d \in \mathbb{N}$. Note that the algorithm from the proof of Proposition 4.2 can be seen as follows: we start with two segments $[v, v]$ and $[w, w]$, and then apply the operation $P$ to each of them until we obtain good vertex pairs which are close. Our algorithm will optimize this by representing all the good vertex pairs coming from vertices $v$ added to our structure.

A vertex or good segment is called *active* if it is represented in the memory of our algorithm. For each active vertex $v \in V$ we keep two lists of active segments $S$ such that $v$ is respectively the leftmost and rightmost element of $S$. Each active segment $S$ also has a pointer to $P(S)$, which is also active (and thus, all the ancestors of $S$ are active too), and a dynamic array of integers $a(S)$. Initially, there are no active vertices or good vertex pairs; when we activate a segment $S$, its $a(S)$ is initially filled with zeros.

The operation Add($v$, $k$) activates $v$, and $S = [v, v]$ together with all its ancestors. Then, for each $c = 0, \ldots, \delta_0(v)$, it adds $k$ to $a(p^d(S))[c]$.

The operation Tally($w$) activates $w$ and $S = [w, w]$ together with all its ancestors. Then, we look at $p^a(S)$ for $a = 0, \ldots, \delta_0(v)$, and for each $p^d(S)$, we look at close good vertex pairs $q'$ on the same level. The intuition here is as follows: Algorithm 4.2, on reaching $(v_L, v_R) = S$ and $(w_L, w_R) = S'$, would find out that these two pairs are close enough and return $a + b + c$; in our case, for each $c$ such that $a(S') \neq 0$, we will instead add $a(S')[c]$ to $A[a+b+c]$. Since Algorithm 4.2 stops in that situation and our $a$ goes all the way down to $\delta_0(v)$, we also have to make sure that we do not count vertices which have been already counted. □

**Remark.** It easily follows from Proposition 4.1 that the uniform hyperbolic triangular grids are hyperbolic graphs in the sense of Gromov [Bermudo et al., 2013]. (Should follow – not formally checked yet.) Our algorithms could be seen as an algorithmic application of Gromov hyperbolicity.

# 5 Grid distances versus hyperbolic distances

We assume that our grid is mapped to the hyperbolic plane $\mathbb{H}^2$ via a function $j : V(G) \to \mathbb{H}^2$; in case of $G_{67}$ and $G_7$ this mapping corresponds to the regular tesselation that these grids are based on; furthermore, $j(v)$ is obtained by applying $d = \delta_0(v)$ isometries to $j(v_0)$, with $i$-th isometry depending only on the type of $p_R^{d-i+1}(v)$ and the index of $p_R^{d-i}(v)$ among its children.

**Intuition 5.1.** *For $v, w \in V(G)$, let $d = \delta(v, w)$, and $r = \delta(j(v), j(w))$. Then $d$ and $r$ are approximately proportional.*

Stating and proving this intuition formally appears to be challenging, as we have to deal both with the discrete structure of the triangular grid, and the continuous hyperbolic geometry. From the regularity of our tesselation we get that $d = \Theta(r)$; we cannot give a better estimate (e.g., $d = \alpha r + \Theta(1)$) because the density of rings depends on the direction. However, we can guess that, on average, $r \approx d \log \gamma$. This is because, in the hyperbolic plane, the

area and circumference f a circle of radius $r$ given in absolute units is given by $\cosh(r) - 1$ and $\sinh(r)$ respectively, which are $\Theta(e^r)$; from Proposition 3.2 we know that this corresponds to $\Theta(\gamma^d)$ elements of the grid, yielding $r \approx d \log \gamma$ after taking the logarithm of both sides.

We can also expect this approximation to be better than the corresponding Euclidean one. Consider the regular triangular grid $G_6$ on the Euclidean plane, in the standard embedding wher every edge has length 1. Let $v = v_0$ and $W$ be a random vertex in $R_d(G_6)$. From basic geometry we obtain that $r \in [\frac{\sqrt{3}+1}{2}d, d]$. The standard deviation of $r$ will be linear in $d$, because the ratio $r/d$ depends on the angle between the line $(v_0, W)$ and the grid lines. However, because of the exponential expansion in the hyperbolic plane, this angle constantly changes as the line $(v_0, W)$ traverses the grid, leading to the following conjecture:

**Conjecture 5.2.** *Let $G = G_{67}$ or $G_7$, and $W \in R_d(G)$ be randomly chosen. Then $\delta(j(v_0), j(W)) = c_1 d + c_0 + X$, where $EX = o(1)$, $Var\ X = \Theta(d)$, and the limit distribution of $X/\sqrt{d}$ is normal.*

The results of experimental verification agree with the conjecture both for $G_{67}$ and $G_7$, although it appears that $c_1$ is slightly larger than $\log \gamma$ in both cases.

# 6 Hyperbolic random graphs

We start by presenting the hyperbolic random graph (HRG) model [Krioukov et al., 2010, Gugelmann et al., 2012, Bläsius et al., 2016]. Then, we will use our intuitions from the previous section to define the discrete hyperbolic random graph model (DHRG).

Fix a radial coordinate system in the hyperbolic plane $\mathbb{H}^2$, where every point is represented by two coordinates $(r, \phi)$, where $r$ is the distance from some fixed center, and $\phi$ is the angle. The hyperbolic random graph model has four parameters: $n$, $R$, $T$, and $\alpha$. Each vertex $v \in V(H) = \{1, \dots, n\}$ is independently randomly assigned a point $\mu(v) = (r_v, \phi_v)$, where the distribution of $\phi_v$ is uniform in $[0, 2\pi]$, and the density of the distribution of $r_v \in [0, R]$ is given by $f(r) = \frac{\alpha \sinh(\alpha r)}{\cosh(\alpha R)-1}$. Then, for each pair of vertices $v_1, v_2 \in V(H)$, they are connected with probability $p(\delta(\mu(v_1), \mu(v_2)))$, where $p(d) = \frac{1}{1+e^{(d-R)/2T}}$.

It is known that, for correctly chosen values of $n$, $R$, $T$ and $\alpha$, the parameters of hyperbolic random graph, such as its degree distribution and clustering coefficient, are similar to those of real world scale-free networks [Gugelmann

et al., 2012]. We can also embed real world scale-free networks into the hyperbolic plane using the maximum likelihood (MLE) method, that is, for a graph $H$ representing a real world scale-free network we can find a mapping $\mu : V(H) \to \mathbb{H}^2$ which predicts the edges as accurately as possible.

**DHRG model.** In our model, we map vertices $v \in V(H)$ not to points in the continuous hyperbolic plane, but to the vertices of our uniform hyperbolic triangular grid $G$, i.e., $\mu : V(H) \to V(G)$. The density function $f(r)$ from the HRG model cannot be reproduced exactly, but we can use $f(r) = \alpha e^{\alpha r}/(e^{\alpha r} - 1)$, which is a very good approximation (it only slightly changes the low probability of placing a vertex very close to the center). To approximate this in our grid, we choose $m(v)$ such that $\delta_0(m(v)) = d$ with probability proportional to $e^{d\alpha}$. The exact vertex $\mu(v)$ is then chosen uniformly from $R_d(G)$. The formula for edge probability remains the same, except that we are using the grid distance now.

DHRG mappings can be converted to HRG by composing $\mu$ with $j$, and the other conversion can be done by finding the nearest grid point to $\mu(v)$ for each $v \in V(G)$. From Conjecture 5.2 we expect tha DHRG parameters $\alpha$, $R$, $T$, and $n$ to be related to the HRG parameters by the factor of $\log \gamma$.

# 7 Properties of DHRGs

This section is not yet completed.

Conjecture 5.2 suggests that, if we take matching values of parameters, the DHRG and HRG should have similar properties; in particular, for DHRGs, the power law degree distribution, average degree, and clustering coefficient should behave similarly as for HRGs [Gugelmann et al., 2012]. In this section we are going to prove this formally. We work with a DHRG with fixed parameters, over a fixed uniform hyperbolic triangular grid $G$.

(Should be relatively easy for regular trees. Our grids are more difficult for two reasons: first, vertices of different types have different number of children, and second, we can take shortcuts as stated in 4.1. However, these complications should not affect the results qualitatively. Experimental results on $G_{67}$ and $G_7$ should be helpful too.)

# 8 Algorithms for DHRG

We show how the algorithms from Section 4 allows us to deal with the DHRG model efficiently.

**Computing the likelihood.** Let $H = (V(H), E(H))$ be a network. The quality of an embedding $m : V(H) \to V(G)$ is measured with its *likelihood*, which describes how good the given embedding is at predicting whether two vertices are connected by an edge. The likelihood $L(m)$ is the probability that, if we connect vertices with edges according to the model outlined above, we exactly obtain $E(H)$; since this number is usually very small, we usually compute its logarithm, or *log-likelihood*. The log-likelihood can be computed with $\log L(m) = \sum_{v,w \in V(H)} \log p_{\{v,w\} \in E}(\delta(\mu(v), \mu(w)))$, where $p_\phi(d) = p(d)$ if $\phi$ is true and $1 - p(d)$ if $\phi$ is false.

Computing the log-likelihood in the continuous model is difficult, because we need to compute the sum over $O(n^2)$ pairs; a better algorithm was crucial for efficient embedding of large real world scale-free networks [Bläsius et al., 2016]. The algorithms from the previous section allows us to compute it quite easily in the DHRG model. To compute the log-likelihood of our embedding of a network $H$ with $n$ vertices and $m$ edges, such that $\delta_0(v) \leq D$ for each $v \in V(H)$, we:

- for each $d$, compute $\textsc{Tally}[d]$, which is the number of pairs $(v, w)$ such that $\delta(v, w) = d$ – the distance tally counter allows doing this in a straightforward way (simply by doing $\text{Add}(\mu(v), 1)$ for each $v \in V(H)$), in time $O(nD^2)$.

- for each $d$, compute $\textsc{EdgeTally}[d]$, which is the number of pairs $(v, w)$ connected by an edge such that $\delta(v, w) = d$ – this can be done in time $O(mD)$ simply using the distance algorithm for each edge.

After computing these two values for each $d$, computing the log-likelihood is straightforward. One of the advantages over [Bläsius et al., 2016] is that we can then easily compute the log-likelihood obtained from other values of $R$ and $T$, or from a function $p(d)$ which is not necessarily logistic.

**Improving the embedding.** A continuous embedding can be improved by a *spring embedder* [Kobourov and Wampler, 2004]. Imagine that there are attractive forces between connected pairs of vertices, and repulsive forces between unconnected pairs. The embedding $m$ will change in time as the forces push the vertices towards locations in such a way that the quality of the embedding, measured by log-likelihood, is improved. However, computationally, spring embedders are very expensive – there are $O(n^2)$ forces, and potentially, many steps of our simulation could be necessary.

A similar effect can be obtained quite easily in our grid-based model, and our algorithms allow us to do it efficiently. We use a local search algorithm. Suppose we have computed the log-likelihood, and on the way we

have computed the vectors TALLY and EDGETALLY, as well as the distance tally counter where every $\mu(v)$ has been added. Now, let $v' \in V(H)$ be a vertex of our embedding, and $w \in V(G)$. Let $\mu'$ be the new embedding given by $\mu'(v') = w$ and $\mu'(v) = \mu(v)$ for $v \neq v'$. Our auxiliary data allows us then to compute the log-likelihood of $\mu'$ in time $O(D^2 + D \deg(w))$.

This allows us to try to improve the embedding in the following way: in each step, for each $v \in V(G)$, consider all neighbors of $\mu(v)$, compute the log-likelihood for all of them, and if for some $\mu'$ we have $\log L(\mu') > \log L(\mu)$, replace $\mu$ with $\mu'$. Assuming the bounded degree of $G$, this can be done in time $O(D^2 n + Dm)$.

Our experiments have shown that the locally optimal mapping is achieved after a small number of steps.

**Generating a random graph.**

Generating large HRGs is not trivial – a naive algorithm works in $\Theta(n^2)$; algorithms working in $O((n^{3/2} + m) \log n)$ and $O(n)$ [von Looz et al., 2015, Bringmann et al., 2015] are known. Our algorithms allow to generate DHRGs quite easily in $O(nD^2 + mD)$.

The first step is to generate the vertices. For each vertex $v$ of the generated graph $H$, we choose $d = \delta_0(v)$ (according to the given distribution), and then we have to randomly choose $v$ from the $|R_d(v)|$ possibilities. This can be done iteratively: we create a sequence of vertices $v_0, \ldots, v_d$, where $v_0$ is the root, and $v_{k+1}$ is a non-rightmost child of $v_k$. The probability of choosing the particular $v$ as $v_k$ should be proportional to $a_{d-k}(v)$, where $a_i(v) = |C^{d-k}(v)| - 1 = |c^{d-k}(t(v))|$ can be obtained by matrix multiplication ($O(D)$ preprocessing).

The second step is to generate the edges. This can be done by modifying the algorithm computing the vector TALLY$[d]$ – when we add $k$ to TALLY$[d]$, we now also add each of the edges with the probability $p(d)$. Thus, we need to choose a subset of $S = \{1, \ldots, k\}$ where each element is independently chosen with probability $p$. $\min S$ has a geometric distribution Geo$(p)$, except in the cases where $S =$ are represented by Geo$(p)$¿$k$; assuming that Geo$(p)$ can be sampled in $O(1)$, this allows us to generate $\min S$ in time $O(1)$, and the rest of $S$ can then be generated in the same way. Then, trace the elements of $S$ back to their original vertices, which can be done in $O(D)$ per edge by following the tree of active segments back. The whole algorithm works in time $O(nD^2 + mD)$, where $n$ is the number of vertices and $m$ is the number of generated edges.

# 9 Experimental results

We have implemented the log-likelihood and local search algorithms outlined in the previous section, and conducted experiments on real world network data. The implementation is based on the routines implemented for HyperRogue [Rogue, 2012]. The source code, data, and experimental results are available at `http://www.mimuw.edu.pl/~erykk/dhrg-v2.zip`;here, we present a short summary of the results.

**Facebook social circle network.** First, we test our model on a relatively small network. We have chosen the Facebook social circle network, coming from the SNAP database [Leskovec and Krevl, 2014] and included with the hyperbolic embedder implementing the algorithm by Bläsius et al [Bläsius et al., 2016], which we will refer to as BFKL. This network has $N = 4039$ nodes and $M = 88234$ edges. BFKL has mapped this graph to the hyperbolic plane, using parameters $R = 12.576$, $\alpha = 0.755$, $T = 0.1$. We have computed the log-likelihood as $L_1 = -516534$. This looks extremely bad at first, as it is worse than the log-likelihood of the trivial model where each edge exists with probability $M/\binom{N}{2}$, which is $L_0 = -487133$; however, this is because the influence of the parameter $T$ on the quality of the embedding is small [Papadopoulos et al., 2015], and thus BFKL uses a small value of $T = 0.1$, which does not necessarily correspond to the network. The best log-likelihood of $L_2 = -176132$ is obtained for $R_2 = 11.09358$ and $T_2 = 0.54336$.

Now, we convert this embedding into the DHRG model, by finding the nearest vertex of $G_{67}$ for each $v \in V(H)$. The best log-likelihood $L_3 = 179125$ is obtained for $R_3 = 20.393945$ and $T_3 = 1.012944$; As predicted in the previous section, $T_2/T_3 \approx R_2/R_3 \approx \log \gamma$. Our log-likelihood $L_3$ is slightly worse than $L_2$, but this is not surprising – first, our edge predictor has lost some precision in the input because of the discrete nature of our grid, and second, the original prediction was based on $r$ while our prediction is based on $d$, and the ratio of $r$ and $d$ depends on the direction. We also compute the log-likelihood obtained by a model where the edge probability is $p(d) = \textsc{Edgetally}[d]/\textsc{Tally}[d]$, which corresponds to using the best possible function $p(d)$ (not necessarily logistic); we obtain $L_4 = -177033$, which is only slightly better than $L_3$. This shows that the logistic function is close to the optimum.

Now, we try our local search algorithm. The points stopped moving in the $k$-th iteration, for $k = 24$. This allows us to improve the log-likelihood of $L_5 = -167808$, again for the best values of $R_5 = 20.837295$ and $T_5 = 0.966040$, and the optimal log-likelihood to $L_6 = -165154$.

Now, we convert our mapping back to the HRG model, obtaining the

log-likelihood of $L_7 = -168298$ for the optimal values of $R_7 = 11.25917$ and $T_7 = 0.52659$. Note that $L_7$ is significantly better than $L_2$; hence, despite converting from HRG to DHRG and back, our method was successful at finding a better continuous embedding.

The running time of parts of our algorithm were: $t_1$=0.24 s (converting), $t_2$=0.3 s (computing EDGETALLY), $t_3$=0.03 ms (computing TALLY), $t_4$=35 s (local search). The BFKL embedder computes the log-likelihood in 0.3 seconds, which is comparable. However, their spring embedder working in quadratic time is much slower than our local search.[1]

We have also computed the log-likelihood of our vertex placement, assuming that a vertex is an element of $R_d(G)$ with probability $q(d) = |r_d(G) \cap m(V(H))|/N$, and that each element of $R_d(G)$ is equally probable. We obtain $L_8 = -63787$ for the placement after the local search. (Since our local search did not place an upper limit on $\delta_0$, we have decided to not compare against the probability function $q(d)$ used by the DHRG model.) This is interesting from the information theoretic perspective: using arithmetic encoding, an event with probability $p$ could be encoded with $-\log_2 p$ bits, and thus $(L_6 + L_8)/L_0 = 0.475$ could be interpreted as the compression ratio achieved by representing the graph in our model; thus, our algorithm could be applied as a method of compression of scale-free networks. This has no direct analog in the continuous HRG model, since we have to choose the precision reported for the coordinates. Note that our implementation optimizes $L_5$ rather than $L_6 + L_8$, so a better compression should be achievable.

The respective values obtained by the grid $G_7$ were: $t_1 = 0.23s$, $t_2 = 0.2s$, $t_3 = 0.02s$, $k = 18$, $t_4 = 15s$, $L_3 = -181919$, $L_4 = -180201$, $L_5 = -169205$, $L_6 = -167048$, $L_7 = -170473$, $L_8 = -59147$. $G_7$ is coarser than $G_{67}$, hence it is not surprising that its results are slightly worse; also the smaller size and greater simplicity of the grid improves the running time. Yet, the general qualitative effects are similar. The compression ratio is slightly better at 0.469, suggesting that the coarser grid $G_7$ is slightly more efficient at compression.

**GitHub following graph.** To benchmark our algorithm on a large network, we study the embedding of a social network observed in GitHub

[1]For $T = 0.54336$ and seed 123456789 the BFKL spring embedder reported the log-likelihood of -131634, which is better than ours; however, our implementation reports $L_1 = -211454$ and $L_2 = -174465$, which our local search still manages to improve to $L_7 = -157026$. This appears to to be a problem in their approximation (which also affects the fast embedder, and smaller values of $T$). Indeed, replacing their optimized log-likelihood function with a $\Theta(n^2)$ one from `hyperbolic.cpp` reports log-likelihood equal to ours. [Actually, it reports double our result, but this seems to be caused by counting each pair of vertices twice, which is easy to fix and irrelevant for the optimized embedder.]

repository hosting service. In GitHub convention *following* means that a registered user agreed to be sent notifications about other user's activity within the service. This relationship can be represented by the means of the graph of following $\mathcal{G}_f$. There is an edge in $\mathcal{G}_f$ between A and B if and only if A follows B. Decision about following a particular user can be simultaneously driven by their popularity within the network and the similarity to the interested user, which suggests hyperbolic geometry can be intrinsic in development of $\mathcal{G}_f$. $\mathcal{G}_f$ was also proved to show power-law-like scale behavior [Celińska, 2016], that is why we believe it is a sound benchmark for our analysis. Since the complete download of GitHub data is impossible, our dataset is combined from two sources: GHTorrent project [Gousios, 2013] and GitHubArchive project [Grigorik, 2012]. The analyzed network contains information about the following relationships that occurred in the service from 2008 to 2009.

The graph has $n$=74946 vertices and $m$=537952 edges (since we are working with an undirected graph, an edge appears between A and B if either A follows B or B follows A). The BFKL embedder has chosen parameters $R = 20.9037$ and $\alpha = 0.855$, and computes the log-likelihood in 5 seconds. The results for $G_7$ grid are as follows: $t_1 = 6$ s, $t_2 = 12$ s, $t_3 = 2s$, $L_0 = -4364526$, $L_3 = -3972006$, $T_3 = 1.390037$, $R_3 = 9.180628$, $L_4 = -3855721$. After 6 iterations of local search (27 minutes) the results have been improved to $L_5 = -3542063$, $L_6 = -3520703$, $L_8 = -1997284$. The time $t_2+t_3$ is still comparable to BFKL.[2] The results are generally worse than in the previous graph, which is probably caused by less order in data (not all accounts correspond to real people, avoiding direction in a directed graph, non-random missing data patterns).

Unfortunately, parts of the current implementation of our algorithm were not designed for representing that large graphs, and thus are not memory efficient for our use – this prevents us from running more iterations of local search, or analyzing later years in GitHub. Some of the structures essential for the game are irrelevant for us. Possible further improvements include: freeing memory which is no longer used, or avoid allocating memory just for temporary use; representing (some) dynamic arrays $a$ as lists since most of them are sparse; path compression to avoid using extra memory and time on long paths which lead to a single vertex. Additionally, since the hyperbolic distances between vertices are large, a grid coarser than $G_7$ should be more efficient. This is going to be improved in future work.

[2] As with the smaller graph, we suspect that our value is more accurate than BFKL.

# 10 Conclusion

We have shown efficient algorithms for computing the distances between points in hyperbolic triangular grids, and distances between a given point and a set of points. We have shown how to apply these algorithms to work with the DHRG model efficiently, and how our DHRG model can be used to improve the results of the BFKL embedder.

The results in Sections 2 and 4 have been phrased for arbitrary hyperbolic triangular grids, even though in our applications and implementation we currently only use grids $G_{67}$ and $G_7$. It is interesting to what extend these algorithms can be generalized to wider classes of hyperbolic graphs, such as graphs with Gromov hyperbolic constant of at most $\delta$ [Bermudo et al., 2013].

We believe that an efficient embedder which embeds real world scale-free networks in the DHRG model could be created; however, such embedder would be probably based on ideas similar to those that BFKL is based on, and for this reason we have decided to simply convert the results of BFKL in this paper. Creating an efficient direct DHRG embedder is an area of further research.

Finally, a combined approach could be used for HRGs, where each vertex is mapped both to a point in $\mathbb{H}^2$ and the nearest vertex of our grid. Our algorithms for computing log-likelihood and generating random graphs produce all pairs of vertices which are close enough that their contribution to log-likelihood is significant, or they have a good chance of being connected. This potentially allows to use the grid to efficiently compute log-likelihood and generate HRGs in the continuous case.

# References


[Bermudo et al., 2013] Bermudo, S., Rodríguez, J. M., Sigarreta, J. M., and Vilaire, J.-M. (2013). Gromov hyperbolic graphs. *Discrete Mathematics*, 313(15):1575 – 1585.

[Bläsius et al., 2016] Bläsius, T., Friedrich, T., Krohmer, A., and Laue, S. (2016). Efficient embedding of scale-free graphs in the hyperbolic plane. In *European Symposium on Algorithms (ESA)*, pages 16:1–16:18.

[Bringmann et al., 2015] Bringmann, K., Keusch, R., and Lengler, J. (2015). Geometric inhomogeneous random graphs. *CoRR*, abs/1511.00576.

[Bringmann et al., 2016] Bringmann, K., Keusch, R., Lengler, J., Maus, Y., and Molla, A. R. (2016). Greedy routing and the algorithmic small-world phenomenom. *CoRR*, abs/1612.05539.

[Celińska, 2016] Celińska, D. (2016). Information and influence in social network of Open Source community. In *9th Annual Conference of the EuroMed Academy of Business*.

[Celińska and Kopczyński, 2017] Celińska, D. and Kopczyński, E. (2017). Programming languages in GitHub: a visualization in hyperbolic plane. In *Proceedings of ICWSM 2017, Montreal, Canada, May 16-18, 2017.*, pages 727–728. `http://coin.wne.uw.edu.pl/dcelinska/en/pages/rogueviz-langs.html`.

[Chepoi et al., 2008] Chepoi, V., Dragan, F., Estellon, B., Habib, M., and Vaxès, Y. (2008). Diameters, centers, and approximating trees of delta-hyperbolicgeodesic spaces and graphs. In *Proceedings of the Twenty-fourth Annual Symposium on Computational Geometry*, SCG '08, pages 59–68, New York, NY, USA. ACM.

[Courcelle, 1990] Courcelle, B. (1990). The monadic second-order logic of graphs. i. recognizable sets of finite graphs. *Inf. Comput.*, 85(1):12–75.

[Dawar and Kopczynski, 2016] Dawar, A. and Kopczynski, E. (2016). Bounded degree and planar spectra. *CoRR*, abs/1609.01789.

[Gousios, 2013] Gousios, G. (2013). The ghtorrent dataset and tool suite. In *Proceedings of the 10th Working Conference on Mining Software Repositories*, MSR '13, pages 233–236, Piscataway, NJ, USA. IEEE Press.

[Grigorik, 2012] Grigorik, I. (2012). Github Archive. `https://www.githubarchive.org/`.

[Gugelmann et al., 2012] Gugelmann, L., Panagiotou, K., and Peter, U. (2012). Random hyperbolic graphs: Degree sequence and clustering. In *Proceedings of the 39th International Colloquium Conference on Automata, Languages, and Programming - Volume Part II*, ICALP'12, pages 573–585, Berlin, Heidelberg. Springer-Verlag.

[Kobourov and Wampler, 2004] Kobourov, S. and Wampler, K. (2004). *Non-euclidean spring embedders*, pages 207–214.

[Kopczyński et al., 2017] Kopczyński, E., Celińska, D., and Čtrnáct, M. (2017). HyperRogue: playing with hyperbolic geometry. Accepted to Bridges 2017: Mathematics, Music, Art, Architecture, Education, Culture. `http://www.roguetemple.com/z/hyper/papers/hyperrogue.pdf`.

[Krioukov et al., 2010] Krioukov, D., Papadopoulos, F., Kitsak, M., Vahdat, A., and Boguñá, M. (2010). Hyperbolic geometry of complex networks. *Phys. Rev. E*, 82:036106.

[Lamping et al., 1995] Lamping, J., Rao, R., and Pirolli, P. (1995). A focus+context technique based on hyperbolic geometry for visualizing large hierarchies. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '95, pages 401–408, New York, NY, USA. ACM Press/Addison-Wesley Publishing Co.

[Leskovec and Krevl, 2014] Leskovec, J. and Krevl, A. (2014). SNAP Datasets: Stanford large network dataset collection. `http://snap.stanford.edu/data`.

[Margenstern, 2013] Margenstern, M. (2013). *Small universal cellular automata in hyperbolic spaces: A collection of jewels*, volume 4. Springer Science & Business Media.

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

[Ontrup and Ritter, 2001] Ontrup, J. and Ritter, H. (2001). Hyperbolic self-organizing maps for semantic navigation. In *Proceedings of the 14th International Conference on Neural Information Processing Systems: Natural and Synthetic*, NIPS'01, pages 1417–1424, Cambridge, MA, USA. MIT Press.

[Papadopoulos et al., 2015] Papadopoulos, F., Aldecoa, R., and Krioukov, D. (2015). Network geometry inference using common neighbors. *Phys. Rev. E*, 92:022807.

[Papadopoulos et al., 2012] Papadopoulos, F., Kitsak, M., Serrano, M. A., Boguñá, M., and Krioukov, D. (2012). Popularity versus Similarity in Growing Networks. *Nature*, 489:537–540.

[Robertson and Seymour, 1984] Robertson, N. and Seymour, P. (1984). Graph minors. iii. planar tree-width. *Journal of Combinatorial Theory, Series B*, 36(1):49 – 64.

[Rogue, 2012] Rogue, Z. (2012). HyperRogue: programming. `http://www.roguetemple.com/z/hyper/dev.php` (as of Jan 27, 2017).

[von Looz et al., 2015] von Looz, M., Meyerhenke, H., and Prutkin, R. (2015). *Generating Random Hyperbolic Graphs in Subquadratic Time*, pages 467–478. Springer Berlin Heidelberg, Berlin, Heidelberg.

Faculty of Economic Sciences
University of Warsaw
44/50 Długa St.
00-241 Warsaw
www.wne.uw.edu.pl