Algorithms for Computing Maximum Cliques in Hyperbolic Random Graphs

Eunjin Oh
Department of Computer Science, POSTECH, Pohang, South Korea

Seunghyeok Oh
Department of Computer Science, POSTECH, Pohang, South Korea

Abstract

In this paper, we study the maximum clique problem on hyperbolic random graphs. A hyperbolic random graph is a mathematical model for analyzing scale-free networks since it effectively explains the power-law degree distribution of scale-free networks. We propose a simple algorithm for finding a maximum clique in hyperbolic random graph. We first analyze the running time of our algorithm theoretically. We can compute a maximum clique on a hyperbolic random graph G in $O(m+n^{4.5(1-\alpha)})$ expected time if a geometric representation is given or in $O(m+n^{6(1-\alpha)})$ expected time if a geometric representation is not given, where n and m denote the numbers of vertices and edges of G, respectively, and α denotes a parameter controlling the power-law exponent of the degree distribution of G. Also, we implemented and evaluated our algorithm empirically. Our algorithm outperforms the previous algorithm [BFK18] practically and theoretically. Beyond the hyperbolic random graphs, we have experiment on real-world networks. For most of instances, we get large cliques close to the optimum solutions efficiently.

2012 ACM Subject Classification Theory of computation → Random network models

Keywords and phrases Maximum cliques, hyperbolic random graphs

Digital Object Identifier 10.4230/LIPIcs.ESA.2023.85

Related Version *Full Version*: `https://arxiv.org/abs/2306.16775`

Supplementary Material *Software (Source Code)*: `https://github.com/Menborong/HRG_maxClique` archived at `swh:1:dir:e721e2b75d1878c69a07cb1a383a6eff65dcfc5b`

Funding This work was supported by the National Research Foundation of Korea(NRF) grant funded by the Korea government(MSIT) (No.RS-2023-00209069).

1 Introduction

Designing algorithms for analyzing large real-world networks such as social networks, World Wide Web, or biological networks is a fundamental problem in computer science that has attracted considerable attention in the last decades. To deal with this problem from the theoretical point of view, it is required to define a mathematical model for real-world networks. For this purpose, several models have been proposed. Those models are required to replicate the salient features of real-world networks. One of the most salient features of real-world networks is *scale-free*. In general, a graph is considered as a scale-free network if its diameter is small, one connected component has large size, it has subgraphs with large edge density, and most importantly, *its degree distribution follows a power law*. Here, for an integer $k \leq n$, let $P(k)$ be the fraction of nodes having degree exactly k. If $P(k) \sim k^{-\beta}$, we say that the degree distribution of the graph follows a power law. In this case, β is called the *power-law exponent*.

One promising model for scale-free real-world networks is the hyperbolic random graph model. A hyperbolic random graph is constructed from two parameters. First, points in the hyperbolic plane are chosen from a certain distribution depending on the parameters. Then

we consider the points as the vertices of the constructed hyperbolic random graph. For two vertices whose distance is at most a certain threshold, we add an edge between them. For illustration, see Figure 1. It is known that the degree distribution of a hyperbolic random graph follows a power-law [15]. Moreover, its diameter is small with high probability [14, 16], and it has a giant connected component [10, 17]. Including these results, the structural properties of hyperbolic random graphs have been studied extensively [11]. However, only a few algorithmic results are known. In other words, the previous work focuses on *why* we can use hyperbolic random graphs as a promising model, but only a few work focuses on *how* to use this model for solving real-world problems. We focus on the latter type of problems.

In this paper, we focus on the maximum clique problem for hyperbolic random graphs from theoretical and practical point of view. The maximum clique problem asks for a maximum-cardinality set of pairwise adjacent vertices. For general graphs, this problem is NP-hard. Moreover, it is W[1]-hard when it is parameterized by the solution size, and it is APX-hard even for cubic graphs [2]. Therefore, the theoretical study on the clique problem focuses on special classes of graphs. In fact, this problem can be solved in polynomial time for special classes of graphs such as planar graphs, unit disk graphs and hyperbolic random graphs [7, 12, 13]. More specifically, the algorithm by Bläsius et al. [7] for computing a maximum clique of a hyperbolic random graph takes $O(mn^{2.5})$ *worst-case* time. The randomness in the choice of vertices is not considered in the analysis of the algorithm. One natural question here is to design an algorithm for this problem with improved *expected* time.

To analyze our algorithm, we use the *average-case analysis*. A traditional modeling choice in the analysis of algorithms is *worst-case analysis*, where the performance of an algorithm is measured by the worst performance over all possible inputs. Although it is a useful framework in the analysis of algorithms, it does not take into account the distribution of inputs that an algorithm is likely to encounter in practice. It is possible that an algorithm performs well on most inputs, but poorly on a small number of inputs that are rarely encountered in practice. In this case, the worst-case analysis can mislead the analysis of algorithms. The field of "beyond worst-case analysis" studies ways for overcoming these limitations [25]. One simple technique studied in this field is the average-case analysis. As the hyperbolic random graph model intrinsically defines an input distribution, the average-case analysis is a natural model for analyzing algorithms for hyperbolic random graphs.

Previous Work. While the structural properties of hyperbolic random graphs has been studied extensively, only a few algorithmic problems have been studied. The most extensively studied algorithmic problem on hyperbolic random graphs is the generation problem: Given parameters, the goal is to generate a hyperbolic random graph efficiently. The best-known algorithms run in expected linear time [5] and in worst-case subquadratic time [26]. Also, Bläsius et al. [8] studied the problem of embedding a scale-free network into the hyperbolic plane and presented heuristics for this problem.

Very recently, classical algorithmic problems such as shortest path problems, the maximum clique problem and the independent set problem have been studied. These problems can be solved significantly faster in hyperbolic random graphs. More specifically, given a hyperbolic random graph, the shortest path between any two vertices can be computed in sublinear expected time [4, 9]. A hyperbolic random graph admits a sublinear-sized balanced separator with high probability [6]. As applications, Bläsius et al. [6] showed that the independent set problem admits a PTAS for hyperbolic random graphs, and the maximum matching problem admits a subquadratic-time algorithm. Also, the clique problem can be solved in polynomial time for hyperbolic random graphs in the worst case [7].

The clique problem has been studied extensively because it has numerous applications in various field such as community search in social networks, team formation in expert networks, gene expression and motif discovery in bioinformatics and anomaly detection in complex networks [19]. From a theoretical perspective, the best-known exact algorithm runs in $2^{0.276n}$ time in [23]. However, it is not sufficiently fast for massive real-world networks, leading to the proposal of lots of exact algorithms and heuristics for this problem on real-world networks [1, 19, 21, 24]. While these algorithms and heuristics work efficiently in practice, there is no theoretical guarantee of their efficiency.

Our results. We present algorithms for computing a maximum clique in a hyperbolic random graph and analyze their performances theoretically and empirically.

Given a hyperbolic random graph with parameters α and C together with its geometric representation, we can compute a maximum clique in $O(m+n^{4.5(1-\alpha)})$ expected time, where n and m denotes the numbers of vertices and edges of the given graph. Here, we have $1/2 < \alpha < 1$, and the O-notation hides a constant depending on α and C. With high probability, our algorithm outperforms the previously best-known algorithm by Bläsius et al. [7] running in $O(mn^{2.5})$ time. In the case that a geometric representation is not given, our algorithm runs in $O(m+n^{6(1-\alpha)})$ expected time. This is the first algorithms for the maximum clique problem on hyperbolic random graphs not using geometric representations.

Also, we implemented our algorithms and analyzed it empirically. We run our algorithms on both synthetic data (hyperbolic random graphs) and real-world data. For hyperbolic random graphs, since it is proved that our algorithm computes a maximum clique correctly, we focus on the efficiency of the algorithms. We observed that our algorithms perform efficiently; it takes about 100ms for $n = 10^6$. For real-world networks, our algorithm gives a lower bound on the optimal solution. We observed that our algorithm performs well especially for collaboration networks and web networks. These are typical scale-free real-world networks.

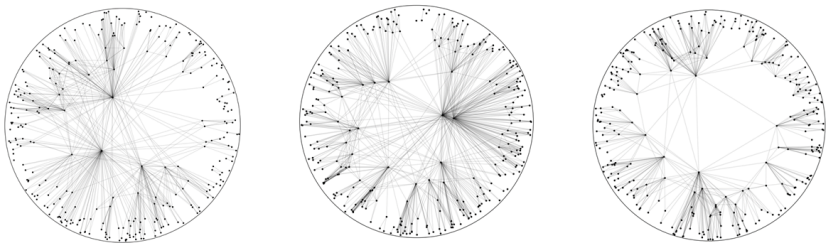
2 Preliminaries

Let $\mathbb{H}^2$ be the hyperbolic plane with curvature -1. We can handle hyperbolic planes with arbitrary (negative) curvatures by rescaling other model parameters which will be defined later. Thus it suffices to deal with the hyperbolic plane with curvature -1. Since the hyperbolic plane is isotropic, we choose an arbitrary point o and consider it as the origin of $\mathbb{H}^2$. Also, we fix a half-line ℓ_o from o going towards an arbitrary point, say w, as the *axis*. Then we can represent a point v of $\mathbb{H}^2$ as (r_v, ϕ_v) where r_v is the hyperbolic distance between v and o, and ϕ_v is the angle from ℓ_o to the half-line from o going towards v. We call r_v and ϕ_v the *radial* and *angular coordinates* of v.

For any two points x and y in $\mathbb{H}^2$, we use $d(x, y)$ to denote the distance in $\mathbb{H}^2$ between x and y. Then we have the following.

$$\begin{aligned} d(u,v) &= \cosh^{-1}(\cosh(r_u)\cosh(r_v) - \sinh(r_u)\sinh(r_v)\cos(\Delta\phi_{u,v}) \\ &\leq \cosh^{-1}(\cosh(r_u)\cosh(r_v) + \sinh(r_u)\sinh(r_v)), \end{aligned}$$

where $\Delta\phi_{u,v}$ denotes the small relative angle between v and u [3]. For a point $x \in \mathbb{H}^2$ and a value $r \geq 0$, we use $B_x(r)$ to denote the disk centered at x with radius r. That is, $B_x(r) = \{v \in \mathbb{H}^2 \mid d(v, x) \leq r\}$.



(a) $\alpha = 0.75$ and $C = 3$ (b) $\alpha = 0.75$ and $C = 1$ (c) $\alpha = 0.85$ and $C = 1$

Figure 1 HRGs with different parameters. As C gets larger, the average degree gets larger (See (a–b)). As α gets larger, the points gets closer to the boundary of D (See (b–c)).

2.1 Hyperbolic Random Graphs

A *hyperbolic unit disk graph (HRG)* is a graph whose vertices are placed on $\mathbb{H}^2$, and two vertices are connected by an edge if the distance between them on $\mathbb{H}^2$ is at most some threshold. This threshold is called the *radius threshold* of the graph. This is the same as the unit disk graph except that the hyperbolic unit disk graph is defined on $\mathbb{H}^2$ while the unit disk graph is defined on the Euclidean plane.

In this paper, we focus on the hyperbolic *random* graph model introduced by Papadopoulos et al. [20]. It is a family $\{\mathcal{G}_{n,\alpha,C}\}$ of distributions, indexed by the number n of vertices, a parameter C adjusting the average degree of a graph, and a parameter α determining the power-law exponent. A sample from $\{\mathcal{G}_{n,\alpha,C}\}$ is a hyperbolic unit disk graph on n points (vertices) chosen independently as follows. Let D be the disk centered at o with radius $R = 2\ln n + C$. To pick a point v in D, we first sample its radius r_v, and then sample its angular coordinate ϕ_v. The probability density $\rho(r)$ for the radial coordinate r_v is defined as

$$\rho(r) = \frac{\alpha \sinh(\alpha r)}{\cosh(\alpha R) - 1}. \tag{1}$$

Then the angular coordinate is sampled uniformly from $[0, 2\pi)$. In this way, we can sample one vertex with respect to parameters α and C, and by choosing n vertices independently and by computing the hyperbolic unit disk graph with radius threshold R on the n vertices, we can obtain a sample from the distribution $\{\mathcal{G}_{n,\alpha,C}\}$.

An intuition behind the definition of $\rho(r)$ is as follows. To choose a point v uniformly at random in D, we first choose its angular coordinate uniformly at random in $[0, 2\pi)$ as we did for $\mathcal{G}_{n,\alpha,C}$, and choose its radial coordinate according to the distribution with density function $\rho(r) = \frac{\sinh(r)}{\cosh(\alpha R)-1}$. But in this case, the power-law exponent of a graph is fixed. To add the flexibility to the model, the authors of [20] introduced a parameter α and defined the density function as in (1). Here, For $\alpha < 1$, this favors points closer to the center of D, while for $\alpha > 1$, this favors points closer to the boundary of D. For $\alpha = 1$, this corresponds to the uniform distribution [15]. For illustration, see Figure 1.

Properties of HRGs. Let $\mu(S)$ be the probability measure of a set $S \subseteq D$, that is,

$$\mu(S) = \int_{x \in S} \frac{\alpha \sinh(\alpha r_x)}{2\pi \cosh(\alpha R) - 1} dx.$$

For a vertex v of a hyperbolic random graph G with n vertices, the expected degree of v in G is $n \cdot \mu(B_v(R) \cap D)$ by construction. Moreover, notice that $\mu(B_v(R) \cap D) = \mu(B_{v'}(R) \cap D)$ for any two vertices v, v' with $r_v = r_{v'}$. Thus to make the description easier, we let $B_r(r')$ denote $B_{(r,0)}(r')$ if it is clear from the context. Note that $D = B_0(R)$.

Hyperbolic random graphs have all properties for being considered as scale-free networks mentioned above. In particular, hyperbolic random graphs with parameter α have power-law exponent β, where $\beta = 2\alpha + 1$ if $\alpha \geq 1/2$, and $\beta = 2$ otherwise. Most real-world networks have a power-law exponent larger than two. Thus we assume that $\alpha > 1/2$ in the paper.

2.2 Algorithms for the Maximum Clique Problem

In this section, we review the algorithm for this problem on hyperbolic random graphs described in [7], which is an extension of [12]. This algorithms requires geometric representations of hyperbolic random graphs. Let $G = (V, E)$ be a hyperbolic random graph.

For $\alpha \geq 1$, they showed that a hyperbolic random graph has $O(n)$ maximal cliques with high probability. Therefore, a maximum clique can be computed in linear time with high probability by just enumerating all the maximal cliques.

For $\frac{1}{2} < \alpha < 1$, they showed that the algorithm in [12] can be extended to hyperbolic random graphs. Assume first that, for a maximum clique K, we have two vertices u and v of K with maximum $r = d(u, v)$. Then all vertices in K are contained in the region $R_{uv} = B_u(r) \cap B_v(r)$. Then we can compute K by considering the vertices in R_{uv} as follows. We partition R_{uv} into R^1_{uv} and R^2_{uv} with respect to the line through u and v. They showed that the diameter of R^1_{uv} (and R^2_{uv}) is at most one, and thus $V \cap R^1_{uv}$ (and $V \cap R^2_{uv}$)) forms a clique. Therefore, the subgraph G_{uv} of G induced by $V \cap R_{uv}$ is the complement of a bipartite graph with bipartition $(V \cap R^1_{uv}, V \cap R^2_{uv})$. Moreover, K is an independent set of the complement of G_{uv}. Therefore, it suffices to compute an independent set of the complement of G_{uv}, and we can do this in in $O(n^{2.5})$ time using the Hopcroft-Karp algorithm. However, we are not given the edge uv in advance. Thus we apply this procedure for every edge uv of G, and then take the largest clique as a solution. This takes $\mathcal{O}(mn^{2.5})$ time in total.

Throughout this paper, we use $\mathbb{P}[A]$ to denote the probability that an event A occurs. For a random variable X, we use $\mathbb{E}[X]$ to denote the expected value of X. Due to lack of space, some proofs and details are omitted. Missing proofs and details can be found in the full version of this paper.

3 Efficient Algorithm for the Maximum Clique Problem

In this section, we present an algorithm for the maximum clique problem on a hyperbolic random graph drawn from $\mathcal{R}_{n,\alpha,C}$ running in $\mathcal{O}\left(m + n^{4.5(1-\alpha)}\right)$ expected time. This algorithm correctly works for any hyperbolic unit disk graph, but its time bound is guaranteed only for hyperbolic random graphs. As we only deal with the case that $1/2 < \alpha < 1$, this algorithm is significantly faster than the algorithm in [7].

A main observation is the following. A clique of size k consists of vertices of degree at least $k - 1$. That is, to find a clique of size at least k, removing vertices with degree less than $k - 1$ does not affect the solution. Thus once we have a lower bound, say k, on the size of a maximum clique, we can remove all vertices of degree less than k. Our strategy is to construct a sufficiently large clique (which is not necessarily maximum) as a preprocessing step so that we can remove a sufficiently large number of vertices of small degree. After applying a preprocessing step, we will see that the number of vertices we have decreases to $O(n^{1-\alpha})$ with high probability. Then we apply the algorithm in [7] to the resulting graph.

3.1 Computing a Sufficiently Large Clique Efficiently

In this section, we show how to compute a clique of size $\Omega(n^{1-\alpha})$ with probability $1-2^{-\Omega(n^{1-\alpha})}$. The algorithm is simple: Scan the vertices in the non-increasing order of their degrees, and maintain a clique Q, which is initially set as $\emptyset$. If the next vertex can be added to Q to form a larger clique, then add it, otherwise exclude it. We can sort the vertices with respect to their degrees in $O(n+m)$ time using counting sort. Also, we can construct the clique Q in $O(n+m)$ time. In the following, we call Q the *initial clique*.

Now, we show that the size of the initial clique is $\Omega(n^{1-\alpha})$ with probability $1-2^{-\Omega(n^{1-\alpha})}$. First, we show that a sufficient large clique can be found by collecting all vertices in $B_0(R/2)$ with high probability. in Lemma 1. For its proof, see the full version.

▶ **Lemma 1.** *For any constant $c'>0$, the vertices in $B_0(R/2-c')$ forms a clique of size $\Omega(n^{1-\alpha})$ with probability $1-2^{-\Omega(n^{1-\alpha})}$.*

Thus, we can get desired clique by choosing $\Omega(n^{1-\alpha})$ vertices in $B_0(R/2)$ with high probability. However, as we scan the vertices in the decreasing order of their degrees, this does not immediately imply that the size of the initial clique is $\Omega(n^{1-\alpha})$. In the following lemma, we show that the initial clique has the claimed size by showing that the $\Omega(n^{1-\alpha})$ vertices with highest degrees are contained in $B_0(R/2)$ with high probability.

▶ **Lemma 2.** *The initial clique has size $\Omega(n^{1-\alpha})$ with probability $1-2^{-\Omega(n^{1-\alpha})}$.*

Proof. Here, we give a brief sketch of the proof. For details, see the full version. First, we show that no vertex lying outside of $B_0(R/2)$ has degree greater than $2ec_1\sqrt{n}$ with high probability for some constant c_1 depending only on C,α. Then we show that no vertex in $B_0(R/2-c_2)$ has degree smaller than $2ec_1\sqrt{n}$ with high probability for some constant c_2. To construct the initial solution, we scan the vertices in the decreasing order of their degrees. Therefore, with probability $1-2^{-\Omega(\sqrt{n})}$, we consider all vertices in $B_0(R/2-c_2)$ before considering any vertex lying outside of $B_0(R/2)$. Therefore, the initial clique contains all vertices in $B_0(R/2-c_2)$ with high probability. By Lemma 1, the initial clique has size at least $\Omega(n^{1-\alpha})$ with high probability. ◀

3.2 Removing All Vertices of Small Degree

In this section, we show how to remove a sufficiently large number of vertices, and show that the size of the remaining graph is $O(n^{1-\alpha})$ with high probability. This algorithm is also simple: given the initial clique of size k, we repeatedly delete all vertices of degree smaller than k. We call the resulting graph the *kernel*. Then no vertex in the kernel has degree smaller than k at the end of the process. This process can be implemented in linear time as follows: maintain the queue of vertices of degree smaller than k, and maintain the degree of each vertex. Then remove the vertices in the queue in order. Whenever a vertex v is removed, update the degree of each neighbor w of v and insert w to the queue if its degree gets smaller than k.

We show that the kernel has size $O(n^{1-\alpha})$. Notice that we do not specify the order of vertices we consider during the deletion process. Fortunately, the kernel size remains the same regardless of the choice of deletion ordering. A proof of Lemma 3 can be found in the full version of this paper.

▶ **Lemma 3.** *In any order of deleting vertices, we can get a unique kernel.*

Because of the uniqueness of the kernel, for analysis, we may fix a specific deletion ordering and slightly modify the deletion process as follows. Imagine that we scan the vertices in the decreasing order of their radial coordinates. If the degree of a current vertex (in the remaining graph) is at least the size of the initial clique size, then we terminate the deletion process. Otherwise, we delete the current vertex, and consider the next vertex. By Lemma 3, the number of remaining vertices is at least the size of the kernel. In the following lemma, we analyze the number of remaining vertices.

▶ **Lemma 4.** *Given an initial solution of size $\Omega(n^{1-\alpha})$, then the size of the kernel is $O(n^{1-\alpha})$ with probability $1-2^{-\Omega(n^{1-\alpha})}$.*

Proof. Here, we give a sketch of the proof only. For details, see the full version.

Let K be the initial solution, and let r' be a value such that $n\mu(B_0(r')\cap B_{r'}(R))$ is $\frac{|K|}{2e}$. Then we show that all the vertices with radial coordinates larger than r' are removed with probability $1-2^{-\Omega(n^{1-\alpha})}$. To do this, for a vertex v, we define the *inner degree* of v as the number of its neighboring vertices whose radial coordinates are smaller than the radial coordinate of v. The expected inner degree of a vertex with radial coordinate $r \geq r'$ is at most $n\mu(B_0(r')\cap B_{r'}(R)) = \frac{|K|}{2e}$.

Chernoff bound implies that for a vertex with radial coordinate larger than r', the probability that its inner degree is larger than the size of the initial clique is at most $2^{-|K|}$. By the union bound over at most n vertices with radial coordinates larger than r', the probability that no vertex with radial coordinate larger than r' has inner degree larger than than the size of the initial clique is at most $n2^{-|K|} = 2^{-\Omega(n^{1-\alpha})}$. In other words, with probability $1-n2^{-c_3 n^{1-\alpha}} = 1-2^{-\Omega(n^{1-\alpha})}$, all vertices with radial coordinates larger than r' have inner degree larger than the size of the initial clique.

If this event happens, we remove all vertices with coordinates larger than r'. We show that the number of vertices with coordinates at most r' is at most $O(n^{1-\alpha})$ with probability $1-2^{-\Omega(n^{1-\alpha})}$. Therefore, the probability that the number of remaining vertices after applying the deletion process is at most $O(n^{1-\alpha})$ is at least $1-2^{-\Omega(n^{1-\alpha})}$. ◀

Although the deletion process we use for analysis requires the geometric representation of G, the original deletion process does not require the geometric representation of G. By combining the argument in Section 3.1 and Section 3.2, we have the following theorem.

▶ **Theorem 5.** *Given a graph drawn from $\mathcal{G}_{n,\alpha,C}$ with $\frac{1}{2} < \alpha < 1$ and its geometric representation, we can compute its maximum clique in $\mathcal{O}\left(m + n^{4.5(1-\alpha)}\right)$ expected time.*

4 Efficient Robust Algorithm for the Maximum Clique Problem

In this section, we present the first algorithm for the maximum clique problem on hyperbolic random graphs which does not require geometric representations. In many cases, a geometric representation of a graph is not given. In particular, real-world networks such as social and collaboration networks are not defined based on geometry although they share properties with HRGs. As we want to use hyperbolic random graphs as a model for such real-world networks, it is necessary to design algorithms not requiring geometric representations.

Our main key tool in this section is the notion of *co-bipartite neighborhood edge elimination ordering* (CNEEO) introduced by Raghavan and Spinrad [22]. It can be considered as a variant of a perfect elimination ordering. Let G be an undirected graph. Let $L = (e_1, e_2, \dots e_m)$ be an edge ordering of all edges of G. Let $G_L[i]$ be the subgraph of G with the edge set $\{e_i, e_{i+1} \dots e_m\}$. For a vertex $v \in V$, let $N_G(v)$ denote the set of neighbors of v in G. Then

L is called a *co-bipartite neighborhood edge elimination ordering* (CNEEO) if for each edge $e_i = (u_i, v_i)$, the subgraph of G induced by $N_{G_L[i]}(u_i) \cap N_{G_L[i]}(v_i)$ is co-bipartite. Here, a *co-bipartite graph* is a graph whose complement is bipartite.

Raghavan and Spinrad [22] presented an algorithm for computing a CNEEO in polynomial time if it exists. It uses a simple greedy algorithm: compute the edges of a CNEEO one by one, and add an edge immediately if it does not violate the condition of a CNEEO. Moreover, they presented a polynomial-time algorithm for computing a maximum clique in polynomial time assuming a CNEEO is given. Since Raghavan and Spinrad [22] did not give an explicit time bound on their algorithm, we analyzed their algorithm and confirmed that their algorithm takes $O(m^3 + mn^{2.5})$ time in the full version. Note that this time bound holds for an arbitrary graph (not necessarily a hyperbolic unit disk graph) admitting a CNEEO.

We show that a hyperbolic unit disk graph admits a CNEEO. This immediately leads to an $O(m^3 + mn^{2.5})$-time algorithm for the maximum clique problem. Its proof can be found in the full version. In the case of hyperbolic *random* graphs, we can solve the problem even faster. As we did in Section 3, we compute an initial clique, remove vertices of small degrees, and then obtain a kernel of small size. Recall that these procedures do not require geometric representations. Note that the kernel is also a hyperbolic unit disk graph because we remove vertices only. The number of vertices of the kernel is $O(n^{1-\alpha})$ and the edges is $O(n^{2-2\alpha})$ in probability $1 - 2^{-\Omega(n^{1-\alpha})}$. Thus, we have the following theorem.

▶ **Corollary 6.** *Given a graph drawn from $\mathcal{G}_{n,\alpha,C}$ with $\frac{1}{2} < \alpha < 1$, we can compute a maximum clique in $\mathcal{O}\left(m + n^{6(1-\alpha)}\right)$ expected time without its geometric representation.*

Heuristics for real-world networks. A main motivation of the study of hyperbolic random graphs is to obtain heuristics for analyzing real-world networks. Many real-world networks share salient features with hyperbolic random graphs, but this does not mean that many real-world networks *are* hyperbolic random graphs. Because the algorithm in Corollary 6 is aborted for a graph not admitting a CNEEO, one cannot expect that this algorithm works correctly for many real-world networks. In fact, only a few real-world networks admit CNEEO as we will see in Section 6. That is, for most of real-world networks, the algorithm in Corollary 6 is aborted.

However, in this case, we can obtain a lower bound on the optimal clique sizes, and moreover, we can reduce the size of the graph. Although we do not have any theoretical bound here, our experiments showed that the size of the clique we can obtain is close to the optimal value for many instances. For details, see the full version.

5 Improving Performance through Additional Optimizations

For implementation, we introduce the following two minor techniques for improving the performance of the algorithms. Although these techniques do not improve the performance theoretically, they improve the performance empirically. Recall that our algorithms consist of two phases: Computing a kernel of size $O(n^{1-\alpha})$, and then computing a maximum clique of the kernel. As the first phase can be implemented efficiently, we focus on the second phase here. Again, the second phase has two steps. With geometric representations, we first compute a CNEEO, and then compute a maximum clique using the CNEEO. Without geometric representations, we consider every edge, and then compute a maximum clique in the subgraph induced by the common neighbors of the endpoint of the edge. The first technique applies to both of the two steps, and the second technique applies to the first step.

5.1 Skipping Vertices with Low Degree

The main observation of our kernelization algorithm is that, for any lower bound k on the size of a maximum clique, a vertex of degree less than k does not participate in a maximum clique. The first technique we use in the implementation is to make use of this observation also for computing a CNEEO, and for computing a maximum clique using the CNEEO.

While computing a CNEEO, the lower bound k we have does not change; it is the size of the initial clique. Whenever we access a vertex, we check if its degree is less than k. If so, we remove this vertex from the kernel, and do not consider it any more. One can consider this as "lazy deletion." Then once we have a CNEEO, we scan the edges in the CNEEO, and for each edge, we compute a maximum clique of the subgraph defined by the edge. If it is larger than the lower bound k we have, we update k accordingly. In this process, whenever we find a vertex of degree less than k, we remove it immediately. Moreover, if the subgraph defined by each edge of CNEEO has vertices less than k, we skip this subgraph as it does not contain a clique of size larger than k.

5.2 Introducing the Priority of Edges

Recall that the second phases consists of two steps: computing a CNEEO, and computing a maximum clique using the CNEEO. In this section, we focus on the first step.

With Geometric Representations. In this case, we use the $O(m'n'^{2.5})$-time algorithm by [7] for computing a maximum clique of the kernel with n' vertices and m' edges. Although it is the theoretically best-known algorithm, we observed that computing a maximum clique using a CNEEO is more efficient practically. By the proof in the full version, the list of the edges sorted in the non-decreasing order of their lengths is a CNEEO. Without using a CNEEO, for each edge uv, we have to compute the subgraph of G induced by the common neighbors of u and v in G. On the other hand, once we have a CNEEO, it suffices to consider the subgraph of G induced by the common neighbors of u and v in G_L, where G_L is the subgraph of G with the edges coming after uv in the CNEEO. If uv lies close to the last edge in the CNEEO, the number of common neighbors of u and v in G_L can be significantly smaller than the number of their common neighbors in G. This can lead to the performance improvement.

Without Geometric Representations. In this case, we compute a CNEEO in a greedy fashion. Starting from the empty sequence, we add the edges one by one in order. For each edge e not added to the current ordering, we check if the common neighbors of the endpoints of e in the kernel is co-bipartite. If an edge passes this test, we add it to the ordering. It is time-consuming especially when only a few edges can pass the test. To avoid considering the same edge repeatedly, we use the following observation. Once an edge e fails this test, it cannot pass the test unless one of its incident edges are added to the ordering. Using this observation, we classify the edges into two sets: active edges and inactive edges. In each iteration, we consider the active edges only. Once an edge fails the test, then it becomes inactive. Once an edge passes the test, we make all its incident edges active. In this way, we can significantly improve the running time especially for graphs that do not accept CNEEO.

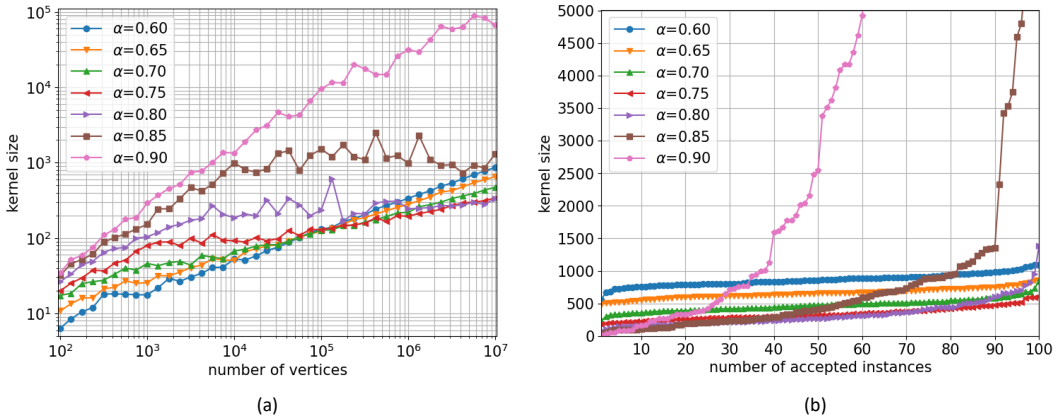


Figure 2 (a) Comparison of the kernel sizes with varying α. Here, $\delta = 10$. (b) Cactus plot of the kernel size versus the number of accepted instances for each value of α with fixed $n = 10^7$.

6 Experimental Evaluation

In this section, we evaluate the performance of our algorithm mainly on hyperbolic random graphs and real-world networks.

Environment and data. We implemented our algorithm using C++17. The code were compiled with GNU GCC version 11.3.0 with optimization flag "-O2". All tests were run on a desktop with Rygen 7 3800X CPU, 32GB memory, and Ubuntu 22.04LTS.

We evaluate the performance of our algorithm on hyperbolic random graphs and real-world networks. For hyperbolic random graphs, we generate graphs using the open source library GIRGs [5] by setting parameters differently. Recall that we have three parameters n, C and α. Here, instead of C, we use the average degree, denoted by δ, as a parameter because δ can be represented as a function of C and α. As we consider the average performance of our algorithm, we sampled 100 random graphs for fixed parameters n, δ and α, and then calculate the average results (the size of kernels or the running times).

For real-world networks, we use the SNAP dataset [18]. It contains directed graphs and non-simple graphs as well. In this case, we simply ignore the directions of the edges and interpret all directed graphs as undirected graphs. Also, we collapse all multiple edges into a single edge and remove all loops.

6.1 Experiment on Hyperbolic Random Graphs: Kernel Size

We showed that the size of the kernel of the hyperbolic random graph is $O(n^{1-\alpha})$ with probability $1 - 2^{\Omega(n^{1-\alpha})}$. In this section, we evaluated the tendency on the size of the kernel experimentally as n, α and δ change. Here, α controls the power-law exponent, and δ is the average degree of the graph. For experiments for δ, see the full version.

Figure 2 shows the tendency on the size of the kernel as α changes. Here, we fix $\delta = 10$. Figure 2(a) shows a plot of the kernel size versus the number of vertices of a graph on a log–log scale for each value of α. We generate 100 instances randomly and take the average of their results for each point in the plot. Figure 2(b) shows a cactus plot of the kernel size versus the number of accepted instances for each value of α. Here, for a fixed kernel size k, an instance is said to *accepted* if our algorithm returns a kernel of size at most k for this instance. Here, we fix $n = 10^7$ and $\delta = 10$.

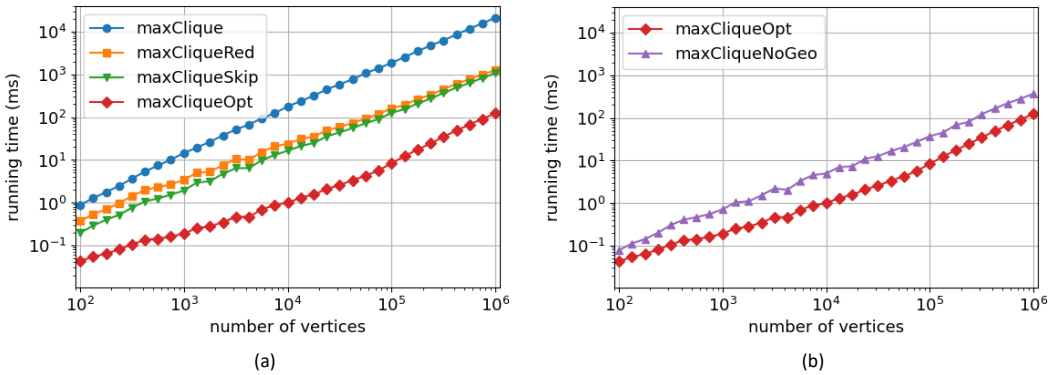


Figure 3 (a) Comparison of running times of different versions of our algorithm. (b) Comparison of running times with and without geometric representations. Here, $\alpha = 0.75$ and $\delta = 10$.

Table 1 Running time for each operation. The unit of time is a millisecond.

	INIT	KERNEL	CNEEO	CONST	INDEP	OTHER	TOTAL
`MaxClique`	-	-	-	21 516.34	4 470.38	8.18	25 994.90
`MaxCliqueRed`	15.67	89.27	-	1 126.95	37.92	2.88	1 272.70
`MaxCliqueSkip`	15.59	88.62	-	925.33	30.77	2.88	1 063.19
`MaxCliqueOpt`	15.64	88.61	1.07	12.55	1.19	2.88	121.94
`MaxCliqueNoGeo`	15.32	89.25	258.45	5.82	0.93	2.96	372.74

For $\alpha \leq 0.8$, the size of the average kernel decreases for sufficiently large n, say $n = 10^7$, as α increases in Figure 2(a). Also, the kernel sizes for all instances are concentrated on the average kernel size for each $\alpha \leq 0.8$ in Figure 2(b). This is consistent with Lemma 4 stating that the kernel size is $O(n^{1-\alpha})$ with high probability. However, this fact does not hold for $\alpha > 0.8$ in Figure 2(a), and the reason for this can be seen in Figure 2(b). At $\alpha = 0.85$, approximately 10% of instances did not have kernels of size at most 1500, and at $\alpha = 0.9$, over 60% of instances did not have such kernels. Notice that the plot sharply increases when the kernel size exceeds 1500. The success probability stated in Lemma 4 is $1 - 2^{\Omega(n^{1-\alpha})}$, which decreases as α increases. In other words, if n is not sufficiently large, it is possible that the success probability $1 - 2^{\Omega(n^{1-\alpha})}$ is not sufficiently large for $\alpha > 0.8$. That is, if we increase the number of vertices on our experiments, we would get the desired tendency on the kernel size for all values α. Nevertheless, at $n = 10^7$, our algorithm removes a significant number of vertices, leaving only 0.01% of vertices at $\alpha = 0.85$ and only 1% at $\alpha = 0.9$.

6.2 Experiment on Hyperbolic Random Graphs: Running Time

In this section, we conducted experiments for evaluating the running times of different versions of our algorithms. Figure 3 shows a plot of the number of vertices versus the running time of each version of our algorithm. Here, we fix $\alpha = 0.75$ and $\delta = 10$. Each point of the plot is averaged for 100 instances. Figure 3(a) shows a plot for the algorithm, denoted by `MaxClique`, by Bläsius et al. [7] and three different versions of the algorithm using geometric representations: `MaxCliqueRed`, `MaxCliqueSkip`, and `MaxCliqueOpt`. More specifically, `MaxCliqueRed` denotes the algorithm described in Section 4. `MaxCliqueSkip` denotes the algorithm described in Section 5.1 that skips low-degree vertices. `MaxCliqueOpt` denotes the algorithm described in Section 5.2 that introduces priorities of edges. As expected, `MaxCliqueOpt` outperforms all other versions of the algorithms in this experiment.

Table 2 The performance of our algorithm on the real-world data.

	$\|V\|$	$\|E\|$	$\|V_{\text{kernel}}\|$	$\|V_{\text{left}}\|$	$\|E_{\text{left}}\|$	runtime	ω_{kernel}	ω_{eval}	ω
as-skitter	1 696,415	11 095,298	28 787	17 033	693 272	342.63	37	≥ 63	67
ca-AstroPh	18 771	198 050	3 679	0	0	1.18	23	57	57
ca-CondMat	23 133	93 439	13 464	0	0	0.07	4	26	26
ca-HepPh	11 204	118 489	0	0	0	0.00	239	239	239
com-amazon	334 863	925 872	255 473	0	0	0.55	3	7	7
com-dblp	317 080	1 049 866	1 716	0	0	0.21	26	114	114
com-lj	3 997 962	34 681 189	1 713 237	126 388	4 587 418	2 316.20	7	≥ 289	327
com-youtube	1 134 890	2 990 443	36 716	7 919	211 051	53.48	13	≥ 14	17
Gnutella31	62 586	147 892	33 816	0	0	0.05	2	4	4
Slashdot0811	77 360	469 180	14 315	1 503	40 418	7.94	10	≥ 17	26
Slashdot0902	82 168	504 229	13 964	1 543	42 215	9.24	11	≥ 17	27
soc-Epinions1	75 879	405 740	9 337	3 717	148 354	26.09	10	≥ 22	23
soc-pokec	1 632 803	22 301 964	1 252 317	54 101	924 531	288.37	4	≥ 29	29
web-BerkStan	685 230	6 649 470	27 058	25 593	589 913	1 224.22	18	≥ 201	201
web-Google	875 713	4 322 051	193 406	2 068	20 426	8.60	10	≥ 44	44
web-NotreDame	325 729	1 090 108	51 227	760	8 181	4.86	6	≥ 155	155
web-Stanford	281 903	1 992 636	32 123	10 721	249 272	177.79	18	≥ 61	61
WikiTalk	2,394 385	4 659 563	70 130	10 421	520 338	447.69	7	≥ 16	26
Wiki-Vote	7 115	100 762	2 913	1 802	62 893	6.34	9	≥ 13	17

For a precise analysis, we evaluated the running time of each task for our algorithms and reported them in in Table 1. More specifically, the algorithms conduct six tasks: INIT denotes the task of finding an initial solution. KERNEL denotes the task of finding the kernel. CNEEO denotes the task of computing a CNEEO. CONST denotes the task of constructing a co-bipartite graph by considering the common neighbors of the endpoints of each edge. INDEP denotes the task of computing a maximum independent set of the complement of a co-bipartite graph. OTHER denotes all the other tasks such as the initialization for variables and caches. TOTAL denotes the entire tasks of our algorithm.

As expected, `MaxCliqueRed` outperforms `MaxClique` significantly. However, CONST is still a time-consuming task for `MaxCliqueRed`. Thus we focus on optimization techniques for CONST and provides `MaxCliqueSkip` and `MaxCliqueOpt` in Section 5. Although `MaxCliqueSkip` gives a performance improvement, it still takes a significant amount of time in CONST and INDEP. `MaxCliqueOpt` computes a CNEEO by sorting the edges with respect to their lengths. This allows us to manage degree efficiently and apply low-degree skip technique to a larger number of vertices. This significantly improves the running time of `MaxCliqueSkip` for CONST and INDEP. This algorithm runs in about 100ms even at $n = 10^6$, exhibiting a performance improvement over 200 times compared to `MaxClique`.

Next, we compared the running times of two algorithms with and without geometrical representations. In the case that a geometric representation is given, we use `MaxCliqueOpt`. If a geometric representation is not given, we use the algorithm in Section 4 and denote it by `MaxCliqueNoGeo`. In `MaxCliqueOpt`, we can quickly compute a CNEEO by sorting the edges in non-decreasing order of length. However, `MaxCliqueNoGeo` computes a CNEEO in a greedy approach which incurs significant overhead. Despite this, the performance of `MaxCliqueNoGeo` in Figure 3(b) does not show a significant difference compared to `MaxCliqueOpt`, and it even outperforms `MaxCliqueSkip`.

6.3 Experiment on Real-World Dataset

Our algorithm can heuristically find large cliques for real-world data. We conducted experiments on several real-world datasets and recorded these results in Table 2. The unit of the running time is a second. $|V|$ and $|E|$ denote the numbers of vertices and edges of the input graph, respectively. $|V_{\text{kernel}}|$ denotes the number of vertices of the kernel. $|V_{\text{left}}|$ and $|E_{\text{left}}|$ denote the numbers of vertices and edges of the remaining graph. Also, ω_{kernel} denotes the size of the initial clique, ω_{eval} denotes the size of the clique computed from our algorithm, and ω denotes the size of the maximum clique of the graph. Here, ω is the correct answer given by the dataset. If a given graph accepts a CNEEO, it is theoretically guaranteed that ω_{eval} is the exact solution and $|V_{\text{left}}| = |E_{\text{left}}| = 0$. Otherwise, ω_{eval} is a lower bound on the exact solution, and $G_{\text{left}} = (V_{\text{left}}, E_{\text{left}})$ has a maximum clique if ω_{eval} is strictly smaller than the exact solution.

The collaboration networks such as ca-AstroPh, CondMat, HepPh, and com-dblp are one of the well-known scale-free networks. These networks accept a CNEEO, allowing us to find the exact maximum clique. Moreover, we were able to find a CNEEO considerably faster for these networks than for other graphs in our experiments. Web graphs such as web-BerkStan, web-Google, web-Notre Dame, and web-Stanford are also one of the well-known scale-free networks. Although these graphs do not accept a CNEEO, we were able to reduce the number of vertices and edges significantly, and we obtained maximum cliques. For the other graphs we tested, we were able to obtain lower bounds that were close to the maximum clique size in most cases, and we were able to significantly reduce the size of the graphs.

7 Conclusion

We presented improved algorithms for the maximum clique problem on hyperbolic random graphs. Our algorithms find a sufficiently large initial solution and find a sufficiently small kernel in linear time, which greatly improves the average time complexity and practical running time. Also we gave the first algorithm for the maximum clique problem on hyperbolic random graphs without geometrical representations. Beyond the hyperbolic random graph, we applied these algorithms to real-world dataset and obtained lower bounds close to the optimum solutions for most of instances.

There are two possible directions for further improvement on our algorithms. First, we compute a maximum clique of hyperbolic random graphs using the framework for computing a maximum clique of unit disk graphs in [12]. Recently, Espenant et al. [13] improved the algorithm [12] and presented an $O(n^{2.5} \log n)$-time algorithm for the maximum clique problem on unit disk graphs. It would be interesting if this technique can be applied to hyperbolic geometry. Second, the bottleneck of our algorithm lies in constructing a CNEEO. Especially, for most of real-world dataset, most of the running time is devoted to constructing a CNEEO. Thus to speed up the overall performance, this step must be improved.

References

1 James Abello, Mauricio GC Resende, and Sandra Sudarsky. Massive quasi-clique detection. In *LATIN 2002: Theoretical Informatics: 5th Latin American Symposium Cancun, Mexico, April 3–6, 2002 Proceedings 5*, pages 598–612. Springer, 2002.

2 Paola Alimonti and Viggo Kann. Some APX-completeness results for cubic graphs. *Theoretical Computer Science*, 237(1-2):123–134, 2000.

3 James W Anderson. *Hyperbolic geometry*. Springer Science & Business Media, 2006.

4 Thomas Bläsius, Cedric Freiberger, Tobias Friedrich, Maximilian Katzmann, Felix Montenegro-Retana, and Marianne Thieffry. Efficient shortest paths in scale-free networks with underlying hyperbolic geometry. *ACM Transactions on Algorithms (TALG)*, 18(2):1–32, 2022.

5 Thomas Bläsius, Tobias Friedrich, Maximilian Katzmann, Ulrich Meyer, Manuel Penschuck, and Christopher Weyand. Efficiently generating geometric inhomogeneous and hyperbolic random graphs. *Network Science*, 10(4):361–380, 2022.

6 Thomas Bläsius, Tobias Friedrich, and Anton Krohmer. Hyperbolic random graphs: Separators and treewidth. In *24th Annual European Symposium on Algorithms (ESA 2016)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2016.

7 Thomas Bläsius, Tobias Friedrich, and Anton Krohmer. Cliques in hyperbolic random graphs. *Algorithmica*, 80(8):2324–2344, 2018.

8 Thomas Bläsius, Tobias Friedrich, Anton Krohmer, and Sören Laue. Efficient embedding of scale-free graphs in the hyperbolic plane. *IEEE/ACM transactions on Networking*, 26(2):920–933, 2018.

9 Thomas Bläsius, Tobias Friedrich, and Christopher Weyand. Efficiently computing maximum flows in scale-free networks. In *29th Annual European Symposium on Algorithms (ESA 2021)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2021.

10 Michel Bode, Nikolaos Fountoulakis, and Tobias Müller. On the largest component of a hyperbolic model of complex networks. *The Electronic Journal of Combinatorics*, pages P3–24, 2015.

11 Elisabetta Candellero and Nikolaos Fountoulakis. Clustering and the hyperbolic geometry of complex networks. In *Algorithms and Models for the Web Graph: 11th International Workshop, WAW 2014, Beijing, China, December 17-18, 2014, Proceedings 11*, pages 1–12. Springer, 2014.

12 Brent N. Clark, Charles J. Colbourn, and David S. Johnson. Unit disk graphs. *Discrete Mathematics*, 86(1):165–177, 1990. doi:10.1016/0012-365X(90)90358-O.

13 Jared Espenant, J. Mark Keil, and Debajyoti Mondal. Finding a maximum clique in a disk graph. In Erin W. Chambers and Joachim Gudmundsson, editors, *39th International Symposium on Computational Geometry, SoCG 2023, June 12-15, 2023, Dallas, Texas, USA*, volume 258 of *LIPIcs*, pages 30:1–30:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPIcs.SoCG.2023.30.

14 Tobias Friedrich and Anton Krohmer. On the diameter of hyperbolic random graphs. *SIAM Journal on Discrete Mathematics*, 32(2):1314–1334, 2018.

15 Luca Gugelmann, Konstantinos Panagiotou, and Ueli Peter. Random hyperbolic graphs: degree sequence and clustering. In *Automata, Languages, and Programming: 39th International Colloquium, ICALP 2012, Warwick, UK, July 9-13, 2012, Proceedings, Part II 39*, pages 573–585. Springer, 2012.

16 Marcos Kiwi and Dieter Mitsche. A bound for the diameter of random hyperbolic graphs. In *2015 Proceedings of the Twelfth Workshop on Analytic Algorithmics and Combinatorics (ANALCO)*, pages 26–39. SIAM, 2014.

17 Marcos Kiwi and Dieter Mitsche. On the second largest component of random hyperbolic graphs. *SIAM Journal on Discrete Mathematics*, 33(4):2200–2217, 2019. doi:10.1137/18M121201X.

18 Jure Leskovec and Andrej Krevl. SNAP Datasets: Stanford large network dataset collection. http://snap.stanford.edu/data, June 2014.

19 Can Lu, Jeffrey Xu Yu, Hao Wei, and Yikai Zhang. Finding the maximum clique in massive graphs. *Proceedings of the VLDB Endowment*, 10(11):1538–1549, 2017.

20 Fragkiskos Papadopoulos, Dmitri Krioukov, Marián Boguná, and Amin Vahdat. Greedy forwarding in dynamic scale-free networks embedded in hyperbolic metric spaces. In *2010 Proceedings IEEE Infocom*, pages 1–9. IEEE, 2010.

21 Bharath Pattabiraman, Md Mostofa Ali Patwary, Assefaw H Gebremedhin, Wei-keng Liao, and Alok Choudhary. Fast algorithms for the maximum clique problem on massive sparse graphs. In *Algorithms and Models for the Web Graph: 10th International Workshop, WAW 2013, Cambridge, MA, USA, December 14-15, 2013, Proceedings 10*, pages 156–169. Springer, 2013.

22 Vijay Raghavan and Jeremy Spinrad. Robust algorithms for restricted domains. *Journal of algorithms*, 48(1):160–172, 2003.

23 John Michael Robson. Algorithms for maximum independent sets. *Journal of Algorithms*, 7(3):425–440, 1986.

24 Ryan A Rossi, David F Gleich, Assefaw H Gebremedhin, and Md Mostofa Ali Patwary. Fast maximum clique algorithms for large graphs. In *Proceedings of the 23rd International Conference on World Wide Web*, pages 365–366, 2014.

25 Tim Roughgarden. *Beyond the worst-case analysis of algorithms*. Cambridge University Press, 2021.

26 Moritz von Looz, Henning Meyerhenke, and Roman Prutkin. Generating random hyperbolic graphs in subquadratic time. In *Algorithms and Computation: 26th International Symposium, ISAAC 2015, Nagoya, Japan, December 9-11, 2015, Proceedings*, pages 467–478. Springer, 2015.