Fellow Travelers Phenomenon Present in Real-World Networks

Abdulhakeem O. Mohammed[1(✉)], Feodor F. Dragan[2], and Heather M. Guarnera[3]

[1] Duhok Polytechnic University, Duhok, Kurdistan Region, Iraq
abdulhakeem.mohammed@dpu.edu.krd
[2] Computer Science Department, Kent State University, Kent, USA
dragan@cs.kent.edu
[3] Department of Mathematical and Computational Sciences, The College of Wooster, Wooster, USA
hguarnera@wooster.edu

Abstract. We investigate a metric parameter "Leanness" of graphs which is a formalization of a well-known Fellow Travelers Property present in some metric spaces. Given a graph $G = (V, E)$, the *leanness* of G is the smallest λ such that, for every pair of vertices $x, y \in V$, all shortest (x, y)-paths stay within distance λ from each other. We show that this parameter is bounded for many structured graph classes and is small for many real-world networks. We present efficient algorithms to compute or estimate this parameter and evaluate the performance of our algorithms on a number of real-world networks.

1 Introduction

Fellow Travelers Property in metric spaces states that two geodesics γ' and γ'' with the same ends x and y always stay close to each other, i.e., two travellers moving along γ' and γ'' from x to y at the same speed always stay at distance at most λ from each other [21]. In graphs, geodesics are shortest paths and this property is formulated as follows: two travelers moving along shortest paths $P'(x, y)$ and $P''(x, y)$ from vertex x to vertex y at the same speed always stay at distance at most λ from each other. Given a graph $G = (V, E)$, the *leanness* $\lambda(G)$ of G is the smallest λ such that, for every pair of vertices $x, y \in V$, all shortest (x, y)-paths stay within distance λ from each other (we give precise definition of this parameter in Definition 1). In this paper, we investigate this metric parameter on real-world networks and on structured families of graphs. We show that this parameter is bounded for many structured graph classes and is small for many real-world networks. This gives a structural indication that, in these networks with small leanness, shortest paths between the same two points tend to stay close together.

Related Work. Understanding the geometric properties of complex networks is a key issue in network analysis and geometric graph theory. In earlier empirical

R. M. Benito et al. (Eds.): COMPLEX NETWORKS 2021, SCI 1015, pp. 194–206, 2022.
https://doi.org/10.1007/978-3-030-93409-5_17

and theoretical studies researchers have mainly focused on features such as small world phenomenon, power law degree distribution, navigability, and high clustering coefficients. Those nice features were observed in many real-world complex networks and graphs arising in Internet applications, in the biological and social sciences, and in chemistry and physics. Although those features are interesting and important, as noted in [27], the impact of intrinsic geometric and topological features of large-scale data networks on performance, reliability and security is of much greater importance.

Recently, there has been a surge of empirical and theoretical work measuring and analyzing geometric characteristics of real-world networks [1,2,23,24,27,29]. One important such property is negative curvature [27], causing the traffic between the vertices to pass through a relatively small core of the network - as if the shortest paths between them were curved inwards. It has been empirically observed [23,27], then formally proved [10], that such a phenomenon is related to the value of the Gromov hyperbolicity (sometimes called also the global negative curvature) of the graph. A graph $G = (V, E)$ is said to be δ-*hyperbolic* [21] if for any four vertices u, v, x, y of V, the two larger of the three distance sums $d(u, v) + d(x, y)$, $d(u, x) + d(v, y)$, $d(u, y) + d(v, x)$ differ by at most 2δ. The smallest value δ for which G is δ-hyperbolic is called *the hyperbolicity* of G and denoted by $\delta(G)$. Hyperbolicity measures the local deviation of a metric from a tree metric. It has been shown that a number of data networks, including Internet application networks, web networks, collaboration networks, social networks, and others, have small hyperbolicity. Furthermore, graphs and networks with small hyperbolicities have many algorithmic advantages. They allow efficient approximate solutions for a number of optimization problems (see [8–11,18,25]).

For an n-vertex graph G, the definition of hyperbolicity implies a simple brute-force $O(n^4)$ algorithm for computing $\delta(G)$. This running time is too slow for computing the hyperbolicity of large graphs that occur in applications. Relying on matrix multiplication results, one can improve the upper bound on time-complexity to $O(n^{3.69})$ [19], but it still remains not very practical. The best known practical algorithm still has an $O(n^4)$-time worst-case bound but uses several clever tricks when compared to the brute-force algorithm [12] (see also [3]). There are also fast heuristics for estimating the hyperbolicity of a graph [12,24], and some efficient constant factor approximation algorithms [6,7,17,19] exist.

The leanness $\lambda(G)$ and the hyperbolicity $\delta(G)$ of any graph G are related to each other. For any graph G (in fact, for any geodesic metric space), $\lambda(G) \leq 2\delta(G)$ holds [21]. Furthermore, in Helly graphs G (discrete analog of hyperconvex metric spaces), $\lambda(G)$ is almost $2\delta(G)$ $(2\delta(G) - 1 \leq \lambda(G) \leq 2\delta(G))$ [15]. For general graphs, however, $\lambda(G)$ and $2\delta(G)$ could be very far apart. For example, if G is an induced odd cycle of length $k = 4\ell+1$, then $\lambda(G) = 0$ and $2\delta(G) = 2\ell-1$. In the light of this, two natural questions arise: what other graph classes enjoy the property of the leanness being very close to twice the hyperbolicity; are parameters $\lambda(G)$ and $2\delta(G)$ close in real-world networks? If so, fast computations or estimations of $\lambda(G)$ could give very good estimations of $\delta(G)$.

Our Contribution. On theoretical side, we demonstrate that the leanness is small or bounded for many structured graph classes. Furthermore, we propose a dynamic programming algorithm that computes in $O(n^2m)$ time and $O(n^2)$ space the leanness $\lambda(G)$ of an arbitrary graph G with n vertices and m edges. Computing the exact leanness of a large graph with our theoretical $O(n^2m)$ time dynamic programming algorithm is prohibitively expensive. Therefore, on practical side, we provide an efficient practical algorithm to compute the exact leanness of large real-world networks. It has an $O(n^4)$-time worst-case bound but uses several tricks to speed-up the computation. It is very fast in practice as our experimental results show. We design also a very simple heuristic to estimate the leanness of a given graph. Lastly, we evaluate the performance of our algorithms on 26 real-world networks.

We also compute the exact hyperbolicity of those networks using the algorithm provided by Cohen et al. [12]. Notably, we found that $\lambda = 2\delta$ for all 26 networks investigated in this paper. In the light of this, we claim that our exact practical algorithm can be used to compute (or sharply estimate from below) the hyperbolicity of real-world networks. Furthermore, our heuristic can also be used to estimate the hyperbolicity. Our experimental results show that our exact practical algorithm is faster than the algorithm of Cohen et al. [12] in computing the exact hyperbolicity and that our heuristic used for estimation of the hyperbolicity outperforms two known heuristics from [12,24].

2 Basic Notions and Notations

All graphs appearing here are connected, finite, unweighted, undirected, and contain no self-loops nor multiple edges. The *distance* $d_G(u, v)$ between vertices u and v is the length of a shortest path connecting u and v in G. For any two vertices u, v of G, $I_G(u, v) = \{z \in V : d_G(u, v) = d_G(u, z) + d_G(z, v)\}$ is the (metric) *interval* between u and v, i.e., all vertices that lay on shortest paths between u and v. The set $S_k(x, y) = \{z \in I_G(x, y) : d_G(x, z) = k\}$ is called a slice of the interval $I_G(x, y)$ where $0 \leq k \leq d(x, y)$. The *diameter* $diam(G)$ of a graph G is the largest distance between a pair of vertices in G, i.e., $diam(G) = \max_{u,v \in V} d_G(u, v)$. The diameter $diam_G(S)$ of a set $S \subseteq V$ is defined as $\max_{u,v \in S} d_G(u, v)$. For a vertex v of G, $N_G(v) = \{u \in V : uv \in E\}$ is called the *open neighborhood* of v. We omit subscript G if no confusion arises.

Definition 1 (Interval Leanness $\lambda(G)$). *An interval $I_G(x, y)$ is said to be λ-lean*[1] *if $d_G(a, b) \leq \lambda$ for all $a, b \in I_G(x, y)$ with $d_G(x, a) = d_G(x, b)$. The smallest integer λ for which all intervals of G are λ-lean is called the* interval leanness (*or simply* leanness) *of G and denoted by $\lambda(G)$ (see Fig. 1).*

[1] This is known (see, e.g., [6]) also under the name λ-thin interval, but to differentiate graph thinness parameter based on the thinness of geodesic triangles from the graph thinness based on thinness of intervals, we use here the word lean.

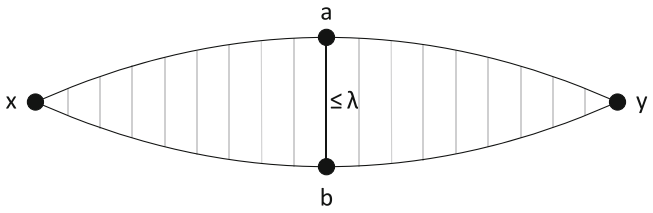

Fig. 1. A λ-lean interval.

Definition 2 (Hyperbolicity $\delta(G)$ (Gromov 4-point condition)). *Let $\delta(u,v,x,y)$ denote the hyperbolicity of a quadruple $u,v,x,y \in V$ defined as half the difference between the biggest two of the three distance sums $S_1 = d_G(x,y) + d_G(u,v)$, $S_2 = d_G(x,v) + d_G(u,y)$, and $S_3 = d_G(x,u) + d_G(v,y)$. The hyperbolicity of a graph G is equal to $\delta(G) = \max_{u,v,x,y \in V} \delta(u,v,x,y)$.*

3 Theoretical Results

Interval Leanness of Structured Classes of Graphs. As for every graph $\lambda(G) \leq 2\delta(G)$ holds, based on bounds on hyperbolicity known in the literature (see [1,4,6,8,16,20,21,32] and papers cited therein), we may also derive several bounds on leanness $\lambda(G)$. Clearly, $\lambda(G) = 0$ if and only if G is a graph where shortest paths are unique. In particular, $\lambda(G) = 0$ when G is an odd induced cycle or when G is a block graph (i.e., every biconnected component is a clique). The leanness is bounded by a small constant on chordal graphs (≤ 1), on AT-free graphs, dually chordal graphs, HHD-free graphs and distance-hereditary graphs (≤ 2 in all), and on many other structured classes of graphs. The leanness of any graph is upper bounded also by twice the slimness and by the thinness, i.e., $\lambda(G) \leq \min\{2\delta(G), \tau(G), 2\varsigma(G)\}$, where $\tau(G)$ and $\varsigma(G)$ are the thinness and the slimness of a graph G, respectively (two other negative curvature parameters of graphs that are within constant factors from the hyperbolicity; see, e.g., [6] for exact definitions). Furthermore, the leanness of a graph G satisfies the inequality $\lambda(G) \leq \min\{\lfloor k(G)/2 \rfloor, 2tl(G), 2\Delta_s(G)\}$, where $k(G)$, $tl(G)$ and $\Delta_s(G)$ are, respectively, the chordality, the tree-length and the cluster diameter of the layering partition of G with respect to a vertex s (other metric tree-likeness parameters of graphs; see, e.g., [1] for exact definitions).

Surprisingly, we have found that equality $\lambda(G) = 2\delta(G)$ holds for all real-world networks that are considered in this paper. We will discuss this in more details in Sect. 4.

On Exact Computation of Interval Leanness. It is obvious from the definition that the leanness $\lambda(G)$ can be computed in $O(n^4)$ time and $O(n^2)$ space for a graph G with n vertices. Here, we propose a dynamic programming algorithm that computes in $O(n^2m)$ time and $O(n^2)$ space the leanness $\lambda(G)$ of a given graph G with n vertices and m edges. We have

Theorem 1. *For every graph G with n vertices and m edges, the interval leanness $\lambda(G)$ of G can be computed in $O(n^2m)$ time and with $O(n^2)$ space.*

We highlight the main ideas of our algorithm next. Let $\lambda_x(G) = \max\{d(w,w') : w,w' \in I(x,y), d(x,w) = d(x,w'), y \in V\}$ be the leanness of G at a fixed vertex x (here, the maximum is taken over all $y \in V$ and $w,w' \in I(x,y)$ with $d(x,w) = d(x,w')$). Notice that $\lambda(G) = \max_{x \in V} \lambda_x(G)$. Additionally, for every ordered pair x,y and every vertex $w \in I(x,y)$, let $M_w(x,y) = \max\{d(w,w') : w' \in I(x,y) \text{ and } d(x,w) = d(x,w')\}$. Since $\lambda(G) = \max_{x \in V} \lambda_x(G)$, given an algorithm for computing $\lambda_x(G)$ in $O(T(n,m))$ time, we can compute $\lambda(G)$ in $O(nT(n,m))$ time, by calling n times this algorithm. Two simple lemmas are crucial to our algorithm.

Lemma 1. *For any $x \in V$, $\lambda_x(G) = \max\{M_w(x,y) : y \in V, w \in I(x,y)\}$.*

The algorithm for computing $\lambda_x(G)$ works as follows. First, we compute the distance matrix of G in $O(nm)$ time. Next, we compute $M_w(x,y)$ for all y and all $w \in I(x,y)$ in time $O(nm)$. Finally, we compute $\max\{M_w(x,y) : y \in V, w \in I(x,y)\}$ in $O(n^2)$ time. By Lemma 1, the obtained value is exactly $\lambda_x(G)$. Hence, we are just left with proving that we can compute $M_w(x,y)$ for all y and all $w \in I(x,y)$ in time $O(nm)$.

To do that, we extend the definition of $M_w(x,y)$ to all vertices $w \in V$. Set $M_w(x,y) = \max\{d(w,w') : w' \in I(x,y) \text{ and } d(x,w) = d(x,w')\}$ if $d(x,w) \leq d(x,y)$ and set $M_w(x,y) = 0$ if $d(x,w) > d(x,y)$. Now, to compute $M_w(x,y)$ for any fixed $x,w \in V$, we can use the following recursive formula [6]:

$$M_w(x,y) = \begin{cases} 0, & \text{if } d(x,y) < d(x,w) \\ d(w,y), & \text{if } d(x,y) = d(x,w) \\ \max\{M_w(x,y') : y' \in N(y) \cap I(x,y)\}, & \text{otherwise.} \end{cases}$$

Since the distance matrix of G is available, using a standard dynamic programming approach, the values $M_w(x,y)$ for all $y \in V$ can be computed in $O(\sum_y \deg(y)) = O(m)$ time. Hence,

Lemma 2. *For any fixed $x,w \in V$, one can compute the values of $M_w(x,y)$ for all $y \in V$ in $O(m)$ time.*

4 Interval Leanness in Real-World Networks

In this section, we investigate the leanness in real-world networks. We first provide two algorithms: an efficient practical algorithm to compute the exact leanness of a network and a heuristic to estimate its leanness. We show the performance of our algorithms on a collection of real-world networks and find that the interval leanness tends to be small (≤ 8 in all 26 networks considered). We observe with our experimental results that our heuristic can be used to estimate the hyperbolicity of a real-world network (recall that $\delta(G) \geq \frac{\lambda(G)}{2}$). We found that our heuristic consistently runs faster than two well-known heuristics presented in [12,24] for estimating the hyperbolicity of a graph.

Practical Exact Algorithm. Computing the exact leanness of a large graph with our theoretical $O(n^2m)$ time dynamic programming algorithm is prohibitively expensive. Therefore, we provide next an efficient practical algorithm to compute the exact leanness of a real-world network. Since it is more likely to find the leanness of a graph G in the interval between pairs of vertices with large distance, we first sort the $\binom{n}{2}$ vertex pairs by distances and process them in non-increasing order. The search is complete once a pair x, y is found such that the current computed value of leanness λ is greater than or equal to $d(x, y)$, and the current computed value is reported as the exact leanness of a given graph. A formal description is given in Algorithm 1. Notice that the leanness of a graph G is bounded by the diameter of G, i.e., $\lambda(G) \leq diam(G)$. The following simple lemmas are also used to reduce the running time of our algorithm.

Lemma 3. *Let x, y be a pair of vertices of a graph G and λ be an integer. If $d(x, y) \leq \lambda$, then the leanness of interval $I(x, y)$ is at most λ.*

Lemma 4. *Let x, y be a pair of vertices of a graph G. Then, for each $u, v \in V$ with $d(x, y) = d(x, u) + d(u, v) + d(v, y)$, the leanness of interval $I(u, v)$ is upper bounded by the leanness of interval $I(x, y)$.*

Lemma 5. *Let λ be the current computed value of leanness and let p, q be the next vertex pair to be checked by Algorithm 1 (see line 2). Let $t_1 = \lfloor \lambda/2 \rfloor$, $t_2 = d(p, q) - \lfloor \lambda/2 \rfloor$ and $S[i] := S_i(p, q)$, $i := 0, 1, 2, ..., d(p, q)$, be the slices of the interval $I(p, q)$. Then, we can start the search with slice $S[t_1]$ and stop when we reach slice $S[t_2]$ for a pair p, q.*

Algorithm 1 runs in $O(n^4)$ time in the worst case. For a fixed vertex pair x, y, the interval $I(x, y)$ can be computed in linear time and line 11 runs in $O(n^2)$ time, since for any pair x, y, $\sum_{i=0}^{d(x,y)} |S[i]| \leq n$. Although there is no improvement to the worst case time complexity of this algorithm compared to our first ($O(n^2m)$ time) algorithm, it is very fast in practice as our experimental results show.

Heuristic. Computing the exact leanness is computationally and memory intensive. The distance matrix alone requires $O(n^2)$ space, which is prohibitive for large graphs. Therefore, we design a very simple heuristic to estimate the leanness of a given graph. Indeed, we will show later that our heuristic can also be used to estimate the hyperbolicity of a real-world network.

We have observed that it is more likely to find the leanness of a graph on the interval of vertex pairs with large distance between them. Hence, we use 2-sweep breadth-first search (BFS) [13] to find a pair of vertices x, y that are far from each other in linear time. Next, we estimate (by sampling) the leanness of interval $I(x, y)$ and report that as an estimation of the leanness of graph G.

One iteration of the heuristic is as follows. Choose an arbitrary vertex z. Run BFS at z and let x be a vertex maximizing $d(z, x)$, then run BFS at x and let y be a vertex maximizing $d(x, y)$. Finally, run BFS at y; now all distances

Algorithm 1: Computes the exact leanness of a graph.

Input : A connected graph $G = (V, E)$ with precomputed distance matrix. List Q of all vertex pairs $\{x, y\}$ of G sorted in non-increasing order with respect to $d(x, y)$.
Output: λ, the leanness of G.

```
λ := 0
foreach pair {x, y} ∈ Q in sorted order do
    if d(x, y) ≤ λ then
        return λ
    Create empty sets S[i], i := 0, 1, 2, ..., d(x, y)
    foreach w in V do
        if d(x, y) := d(x, w) + d(y, w) then
            insert w into S[d(x, w)], remove pairs {x, w}, {w, y} from Q
    end
    Start := ⌊λ/2⌋, End := d(x, y) − ⌊λ/2⌋
    for i:=Start to End do
        foreach u, v in S[i] do
            if λ < d(u, v) then
                λ := d(u, v)
        end
    end
end
return λ.
```

from x and y to every vertex in G are known. Consider the interval $I(x, y)$. We compute the slices $S_i(x, y)$, $i = 0, 1, ..., d(x, y)$, in total linear time (for each $w \in V$, if $d(x, y) = d(x, w) + d(y, w)$ and $d(x, w) = i$, then $w \in S_i(x, y)$). For each slice $S_i(x, y)$, pick k vertices as a sampling and then for each vertex v in the sample, run BFS and compute the distance from v to every vertex in the same slice. Let $\check{\lambda}$ be the largest distance between a sample vertex and a vertex of the same slice. Starting with a new arbitrary vertex z, we repeat this process for t iterations, and return the largest computed value $\check{\lambda}$ as an estimation for leanness of G.

Let d be the maximum distance over all t of the x, y pairs that are considered by our heuristic. Notice that the maximum number of slices to be checked by our heuristic for any fixed interval is at most d. Thus, the running time of our heuristic is $O(tdk(n + m))$. We can significantly speed up the heuristic in practice by using Lemma 5 to skip some slices. Algorithm 2 describes our heuristic formally.

Experimental Results. We implemented our practical algorithm to compute exact leanness and heuristic to estimate leanness. We evaluate the performance of our algorithms on 26 real-world networks. We also computed the exact hyperbolicity of those networks using the algorithm provided by Cohen et al. [12]; we refer to this algorithm as $CCLE$. Notably, we found that $\lambda = 2\delta$ for all 26 networks investigated in this paper. In the light of this, our exact algorithm can

Algorithm 2: Heuristic to estimate the leanness of a graph.

Input : A connected graph $G = (V, E)$. Two integer variables t and k
Output: $\check{\lambda}$, a lower bound on the leanness of G.

```
1  λ̌ := 0
2  for i:=1 to t do
3  |  Choose an arbitrary vertex z. Let x be the last vertex visited by a BFS
   |   starting at z. Let y be the last vertex visited by a BFS starting at x.
4  |  Run BFS at y to get the distances from y to every other vertex in G.
5  |  Create empty sets S[i], i := 0, 1, 2, ..., d(x, y)
6  |  foreach w in V do
7  |  |  if d(x, y) := d(x, w) + d(y, w) then
8  |  |  |  insert w into S[d(x, w)]
9  |  end
10 |  Start := ⌊λ̌/2⌋, End := d(x, y) − ⌊λ̌/2⌋
11 |  for i:=Start to End do
12 |  |  P := pick from S[i] k random vertices
13 |  |  foreach u in P do
14 |  |  |  BFS(u)
15 |  |  |  foreach v in S[i] do
16 |  |  |  |  if λ̌ < d(u, v) then
17 |  |  |  |  |  λ̌ := d(u, v)
18 |  |  |  end
19 |  |  end
20 |  end
21 end
22 return λ̌.
```

also be used to compute the hyperbolicity of real-world networks. Additionally, we compare our heuristic to estimate the hyperbolicity of real-world networks to two other heuristics that estimate hyperbolicity, denoted by author initials as $CCLH$ [12] and KSN [24]. The run time of CCLH is $O(kt(n+m))$, parameterized by integer inputs k and t; it is described briefly as follows. Given a graph $G = (V, E)$, find a pair of vertices x and y with the largest distance between them using 2-sweep BFS. Next, let K be a set of k random vertices from the set $S_x \cap S_y$, where S_x (S_y) is the set of vertices at distance $\lceil d(x, y)/2 \rceil$ from x (y, respectively). Then, for each vertex $u \in K$, run a BFS from u and for each vertex v visited during that BFS, compute $\delta(x, y, u, v)$. Finally, repeat all previous steps t times and return the largest computed value as an estimation of the hyperbolicity of G. On the other hand, the KSN heuristic picks k total vertex quadruples, computes the hyperbolicity of each, then reports the largest computed value as an estimation of the hyperbolicity of graph G.

Datasets. We consider 26 real-world networks of a variety of sizes, structural properties, and domains, including seven collaboration networks, seven autonomous systems (AS) networks, four peer-to-peer networks, two biological networks, two web networks, and four social networks. See Table 1. Autonomous

systems networks studied here are collected by the Cooperative Association for the Internet Data Analysis (CAIDA) [5] and Distributed Internet Measurements and Simulations (DIMES) project [28]. Web networks are obtained from [22]. Douban and Reactome networks are collected from the KONECT library [26] and Erdos, Geom, and Yeast are form the Pajek dataset [31]. All other networks are available as part of the Stanford Large Network Dataset Collection [30]. Each graph represents the largest connected component of the original dataset, as some datasets consist of one large connected component and many very small ones. Furthermore, one can decompose the graph into biconnected components and find the leanness of those components since it is clear that the interval leanness is realized on its biconnected component. Since most of those graphs have one large biconnected component and many small ones, we run our algorithms on the largest biconnected component of each of those graphs.

Evaluation and Results. Our experimental results are summarized in Table 2. Let us first look at value of hyperbolicity and leanness of networks. Recall that shortest paths in negatively curved graphs stay close to each other, that is, $\lambda \leq 2\delta$. Surprisingly, we found that leanness is exactly two times the hyperbolicity for all 26 real-world networks investigated in this paper, that is, $\lambda = 2\delta$ (see columns 2 and 4 in Table 2). Our experiments show that the hyperbolicity in real-world networks is governed by the interval leanness.

We next evaluate the performance of our heuristic to estimate leanness $\check{\lambda}$. We observed that our estimate is very close to exact leanness λ and has a very small computation time. Of the 26 networks, our estimate matches the exact leanness for 9 networks ($\check{\lambda} = \lambda$), it is one less than exact leanness for 15 networks ($\check{\lambda} = \lambda - 1$), and in only two networks, Reactome and Github, our estimate is two less than exact leanness ($\check{\lambda} = \lambda - 2$). Notice that the running time $T_{\check{\lambda}}$ on some graphs might be larger than for other graphs with close numbers of nodes and edges since this depends on the distance between a pair x, y returned by our heuristic and the number of vertices in the slices of $I(x, y)$.

We now compare the performance of our heuristic to the CCLH and KSN heuristics. We have tested both our heuristic and CCLH heuristic on our dataset of real-world networks. An implementation of CCLH is available from [14]. All reported computations have been performed on a computer equipped with Intel Xeon Gold 6240R CPU 2.40 GHz and 50 GB of RAM. For the KSN heuristic, we report the values available for the 9 overlapping networks from [24]. Before analysing the performance of our heuristic and CCLH, we have to set a number of parameters for both heuristics to get the best bounds and also to make a fair comparison. As previously stated, the performance of both heuristics mainly depends on a number of BFS runs. For CCLH, we set $k = 50$ and $t = 25$. In other words, we choose 50 random vertices from the set $S_x \cap S_y$ for each pair x, y, and repeat that process for 25 iterations. So, the total number of BFS runs would be 1325 $(= 25(3+50))$. For our heuristic, we set $\mathrm{k} = 15$ and $\mathrm{t} = 15$. That is to say we choose 15 random vertices for each slice of $I(x, y)$ for each pair x, y, and repeat that process for 15 iterations. So, the total number of BFS runs for our heuristic is $225\ell + 45$ $(= 15(3+15\ell))$, where ℓ is the number of slices of each pair

Table 1. Graph datasets and their parameters: number of vertices n; number of edges m, diameter D. n_{lbc}, m_{lbc}, and D_{lbc} indicate number of vertices, edges, and diameter of the largest biconnected component.

Network name	Type	n	m	D	n_{lbc}	m_{lbc}	D_{lbc}
CA-AstroPh	Collaboration	17903	196972	14	15929	193922	10
CA-CondMat		21363	91286	15	17234	84595	12
CA-GrQc		4158	13422	17	2651	10480	11
CA-HepPh		11204	117619	13	9025	114046	11
CA-HepTh		8638	24806	18	5898	20983	11
Erdos		6927	11850	4	2145	7067	4
GEOM		3621	9461	14	1901	6816	10
DIMES-AUG052009	AS	26655	84900	6	18344	76557	6
DIMES-AUG052010		26496	93175	8	18840	85488	6
DIMES-AUG052011		26163	97614	9	18439	89859	7
DIMES-AUG052012		25357	74999	10	16907	66489	7
CAIDA-20130101		43274	140532	11	27454	124672	10
CAIDA-20170201		56667	239339	11	36296	218906	10
CAIDA-20170301		57015	245287	12	36649	224860	9
Gnutella04	P2P	10876	39994	10	8379	37497	7
Gnutella09		8104	26008	10	5606	23510	8
Gnutella24		26498	65359	11	15519	54380	9
Gnutella31		62561	147878	11	33812	119127	9
Yeast	PPI	2224	6609	11	1465	5839	8
Reactome		5973	145778	24	5306	144537	13
EPA	Web	4253	8897	10	2163	6776	8
California		5925	15770	13	3694	13445	10
Brightkite	Social	56739	212945	18	33187	188577	11
Facebook		22470	170823	15	19153	167296	12
GitHub		37700	289003	11	32416	283696	7
Douban		154908	327162	9	51634	223878	8

x, y. For instance, for the graph CA-Astroph, our heuristic returned a pair x, y with $d(x, y) = 9$. So, the number of BFS runs for CA-Astroph would be around 2070. Fortunately, using Lemma 5, we can skip some slices for next pairs. Our heuristic to estimate leanness also provides an estimate for hyperbolicity given that $\lambda \geq \check{\lambda}$ and $\delta \geq \frac{\check{\lambda}}{2}$ holds for any graph.

We observed that for seven graphs, our heuristic indeed returned a better estimation on the hyperbolicity compared to the value computed by CCLH. CCLH computes a better estimation for only two networks (CAIDA-20170301 and Reactome). The values coincide for the remaining 17 networks. Our heuristic

returned a better estimation on the hyperbolicity compared to the value computed by the KSN heuristic for eight of the nine networks that were evaluated with KSN, and the values coincide for the remaining network. See columns 7, 9, and 11 for the time performance of all heuristicss. Above all, our heuristic is faster than CCLH and KSN.

Table 2. CCLE computes exact hyperbolicity δ_{CCLE} in $T_{\delta_{CCLE}}$ seconds. Our Algorithm 1 computes exact leanness λ in T_λ seconds. KSN estimates hyperbolicity $\check{\delta}_{KSN}$ in $T_{\check{\delta}_{KSN}}$ seconds. CCLH estimates hyperbolicity $\check{\delta}_{CCLH}$ in $T_{\check{\delta}_{CCLH}}$ seconds. Our Algorithm 2 estimates leanness $\check{\lambda}$ (and consequently estimates hyperbolicity) in $T_{\check{\lambda}}$ seconds.

Network name	Exact				Estimate					
	δ_{CCLE}	$T_{\delta_{CCLE}}$	λ	T_λ	$\check{\delta}_{KSN}$ [24]	$T_{\check{\delta}_{KSN}}$ [24]	$\check{\delta}_{CCLH}$	$T_{\check{\delta}_{CCLH}}$	$\check{\lambda}$	$T_{\check{\lambda}}$
CA-AstroPh	3.0	80.91	6	17.07	2.0	15960	2.5	3.14	5	1.46
CA-CondMat	3.5	589.70	7	59.90	2.5	9480	3.0	1.97	6	0.83
CA-GrQc	3.5	4.81	7	0.85	3.0	8580	3.0	0.21	6	0.06
CA-HepPh	3.0	229.03	6	15.69	2.0	46980	3.0	1.43	6	0.55
CA-HepTh	4.0	1.76	8	0.74	3.0	14760	3.5	0.52	7	0.18
Erdos	2.0	0.04	4	0.01			1.5	0.14	4	0.02
GEOM	3.0	0.49	6	0.13			2.5	0.15	5	0.05
DIMES-AUG052009	2.0	19.63	4	8.02			1.5	1.75	3	0.49
DIMES-AUG052010	2.0	70.87	4	16.39			1.5	1.95	3	0.57
DIMES-AUG052011	2.0	1358.29	4	9.11			1.0	1.87	3	0.53
DIMES-AUG042012	2.0	1529.12	4	44.39			1.5	1.51	3	0.47
CAIDA-20130101	2.5	1157.94	5	84.15			2.5	1.85	5	0.51
CAIDA-20170201	2.5	3567.73	5	175.13			1.5	2.16	5	0.82
CAIDA-20170301	2.5	4926.91	5	152.21			2.5	4.02	4	1.09
Gnutella04	3.0	0.02	6	0.18	2.5	10680	2.5	0.88	6	0.11
Gnutella09	3.0	0.02	6	0.09	2.5	8760	2.5	0.54	6	0.15
Gnutella24	3.0	334.37	6	31.47	2.5	5520	2.5	1.68	6	0.21
Gnutella31	3.5	11.23	7	15.99	2.5	12600	3.0	3.82	6	0.89
Yeast	2.5	0.22	5	0.12			2.5	0.11	5	0.03
Reactome	4.0	0.28	8	0.32			3.5	0.71	6	0.37
EPA	3.0	0.09	6	0.02			2.5	0.16	5	0.06
California	3.0	5.67	6	1.11			2.5	0.27	6	0.08
Brightkite	3.0	13618.60	6	214.03			2.5	4.38	5	1.87
Facebook	4.0	25.93	8	13.65			3.0	1.58	6	0.82
Github	2.5	5197.01	5	13.31			2.0	3.05	4	1.48
Douban	3.0	4.51	6	15.21			2.5	6.69	5	2.45

References

1. Abu-Ata, M., Dragan, F.F.: Metric tree-like structures in real-world networks: an empirical study. Networks **67**(1), 49–68 (2016)
2. Adcock, A.B., Sullivan, B.D., Mahoney, M.W.: Tree-like structure in large social and information networks. In: 13th ICDM 2013, pp. 1–10. IEEE (2013)

3. Borassi, M., Coudert, D., Crescenzi, P., Marino, A.: On computing the hyperbolicity of real-world graphs. In: Bansal, N., Finocchi, I. (eds.) ESA 2015. LNCS, vol. 9294, pp. 215–226. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-48350-3_19
4. Bridson, M., Haefliger, A.: Metric Spaces of Non-Positive Curvature. Grundlehren der mathematischen Wissenschaften, vol. 319. Springer, Heidelberg (1999). https://doi.org/10.1007/978-3-662-12494-9
5. Center for Applied Internet Data Analysis (CAIDA). CAIDA AS Relationships Dataset (2017). http://www.caida.org/data/active/as-relationships/
6. Chalopin, J., Chepoi, V., Dragan, F.F., Ducoffe, G., Mohammed, A., Vaxès, Y.: Fast approximation and exact computation of negative curvature parameters of graphs. Discret. Comput. Geom. **65**(3), 856–892 (2021)
7. Chalopin, J., Chepoi, V., Papasoglu, P., Pecatte, T.: Cop and robber game and hyperbolicity. SIAM J. Discret. Math. **28**(4), 1987–2007 (2014)
8. Chepoi, V., Dragan, F., Estellon, B., Habib, M., Vaxès, Y.: Diameters, centers, and approximating trees of δ-hyperbolic geodesic spaces and graphs. In: Proceedings of the 24th Annual Symposium on Computational Geometry, pp. 59–68. ACM (2008)
9. Chepoi, V., Dragan, F.F., Estellon, B., Habib, M., Vaxès, Y., Xiang, Y.: Additive spanners and distance and routing labeling schemes for hyperbolic graphs. Algorithmica **62**(3–4), 713–732 (2012)
10. Chepoi, V., Dragan, F.F., Vaxes, Y.: Core congestion is inherent in hyperbolic networks. In: Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 2264–2279. SIAM (2017)
11. Chepoi, V., Estellon, B.: Packing and covering δ-hyperbolic spaces by balls. In: Charikar, M., Jansen, K., Reingold, O., Rolim, J.D.P. (eds.) APPROX/RANDOM -2007. LNCS, vol. 4627, pp. 59–73. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-74208-1_5
12. Cohen, N., Coudert, D., Lancin, A.: On computing the Gromov hyperbolicity. J. Exp. Algorithmics (JEA) **20**, 1–6 (2015)
13. Corneil, D.G., Dragan, F.F., Köhler, E.: On the power of BFS to determine a graph's diameter. Networks **42**(4), 209–222 (2003)
14. Coudert, D.: Gromov hyperbolicity of graphs: C source code (2014). http://www-sop.inria.fr/members/David.Coudert/code/hyperbolicity.shtml
15. Dragan, F.F., Guarnera, H.M.: Obstructions to a small hyperbolicity in Helly graphs. Discret. Math. **342**(2), 326–338 (2019)
16. Dragan, F.F., Mohammed, A.: Slimness of graphs. Discret. Math. Theor. Comput. Sci. **21**(3) (2019)
17. Duan, R.: Approximation algorithms for the Gromov hyperbolicity of discrete metric spaces. In: LATIN, pp. 285–293 (2014)
18. Edwards, K., Kennedy, S., Saniee, I.: Fast approximation algorithms for p-centers in large δ-hyperbolic graphs. In: Bonato, A., Graham, F.C., Prałat, P. (eds.) WAW 2016. LNCS, vol. 10088, pp. 60–73. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-49787-7_6
19. Fournier, H., Ismail, A., Vigneron, A.: Computing the Gromov hyperbolicity of a discrete metric space. eprint arXiv:1210.3323 (2012)
20. Ghys, E., de la Harpe, P. (eds.): Sur les groupes hyperboliques d'après M. Gromov. Progress in Mathematics, vol. 83 (1990)
21. Gromov, M.: Hyperbolic groups: essays in group theory. MSRI **8**, 75–263 (1987)
22. Jon Kleinberg. Jon Kleinberg's web page. http://www.cs.cornell.edu/courses/cs685/2002fa/

23. Jonckheere, E.A., Lou, M., Bonahon, F., Baryshnikov, Y.: Euclidean versus hyperbolic congestion in idealized versus experimental networks. Internet Math. **7**(1), 1–27 (2011)
24. Kennedy, W.S., Saniee, I., Narayan, O.: On the hyperbolicity of large-scale networks and its estimation. In: 2016 IEEE International Conference on Big Data (Big Data), pp. 3344–3351. IEEE (2016)
25. Krauthgamer, R., Lee, J.R.: Algorithms on negatively curved spaces. In: 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2006), pp. 119–132. IEEE (2006)
26. Kunegis, J.: Konect: the Koblenz network collection. In: WWW 2013 Companion, pp. 1343–1350. Association for Computing Machinery, New York (2013)
27. Narayan, O., Saniee, I.: Large-scale curvature of networks. Phys. Rev. E **84**(6), 066108 (2011)
28. Shavitt, Y., Shir, E.: DIMES: let the internet measure itself. ACM SIGCOMM Comput. Commun. Rev. **35**(5), 71–74 (2005)
29. Shavitt, Y., Tankel, T.: Hyperbolic embedding of internet graph for distance estimation and overlay construction. IEEE/ACM Trans. Netw. **16**(1), 25–36 (2008)
30. Stanford Large Network Dataset Collection (SNAP). Stanford large network dataset. http://snap.stanford.edu/data/index.html
31. Vladimir Batagelj. Pajek datasets. http://vlado.fmf.uni-lj.si/pub/networks/data/
32. Wu, Y., Zhang, C.: Hyperbolicity and chordality of a graph. Electr. J. Comb. **18**(1), Paper #P43 (2011)