On computing the Gromov hyperbolicity

Nathann Cohen, David Coudert, Aurélien Lancin

On computing the Gromov hyperbolicity[*]

Nathann Cohen[1], David Coudert[2,3], and Aurélien Lancin[2,3]

[1]LRI, Laboratoire de Recherche en Informatique, Université Paris-Sud 11, France
[2]Inria, France
[3]Univ. Nice Sophia Antipolis, CNRS, I3S, UMR 7271, 06900 Sophia Antipolis, France

Abstract

The Gromov hyperbolicity is an important parameter for analyzing complex networks which expresses how the metric structure of a network looks like a tree. It is for instance used to provide bounds on the expected stretch of greedy-routing algorithms in Internet-like graphs. However, the best known theoretical algorithm computing this parameter runs in $O(n^{3.69})$ time, which is prohibitive for large-scale graphs.

In this paper, we propose an algorithm for determining the hyperbolicity of graphs with tens of thousands of nodes. Its running time depends on the distribution of distances and on the actual value of the hyperbolicity. Although its worst case runtime is $O(n^4)$, it is in practice much faster than previous proposals as observed in our experimentations. Finally, we propose a heuristic algorithm that can be used on graphs with millions of nodes. Our algorithms are all evaluated on benchmark instances.

Keywords: Algorithms; Gromov Hyperbolicity; Networks.

1 Introduction

In the last years, extensive work has been carried out to better understand the structure of complex networks such as social networks, biological networks, citation networks, or the Internet. In [29] a geometric framework for studying the structure of complex networks has been proposed, which highlights the underlying hyperbolic geometry of complex networks and shows that topologies of hyperbolic geometry are robust and have an optimal structure for navigability. In particular, the topology of the Internet can be embedded into an hyperbolic space [4] in such a way that a simple greedy-forwarding algorithm offers good performances [37]. Separately, it has been shown that the efficiency of routing algorithms on given topologies depends on the hyperbolic nature of its metric space [10]. The notion of hyperbolicity has also been used for expressing the latency of the Internet as a tree metric [39]. In fact, the notion of hyperbolicity is widely used for different purposes. For instance, it is reported in [33] that the congestion induced by any shortest-path routing algorithm on the core of networks scales to $O(n^2)$ in Gromov-hyperbolic graphs. Furthermore, [26] use the notion of hyperbolicity for improving the reliability and security of networks. In a different context, [7, 18] measure the geodesic distance between phylogenetic trees prior to classification.

[*]This work has been partially supported by ANR project Stint under reference ANR-13-BS02-0007, ANR program "Investments for the Future" under reference ANR-11-LABX-0031-01, and by European project FP7 EULER (Grant No.258307).

The (Gromov) *hyperbolicity* of a graph, denoted δ, reflects how the metric space (distances) of a graph is close to the metric space of a tree. In this paper, we use the so-called *4-points condition* definition of the hyperbolicity as proposed by Gromov in [23] (see Section 2). With this definition, trees and cliques are both 0-hyperbolic graphs (which reflects the uniqueness of shortest paths), and as in general the hyperbolicity of a graph is the maximum hyperbolicity of its individual 2-connected components we know that *block* graphs (i.e. connected graphs whose 2-connected components are cliques) are also 0-hyperbolic. Cycles of order $n = 4p + \varepsilon$, with $p \geq 1$ and $\varepsilon \in \{0, 1, 2, 3\}$, are $(p - 1/2)$-hyperbolic when $\varepsilon = 1$, and p-hyperbolic otherwise [28]. $n \times m$ grids, with $2 \leq n \leq m$, are $(n - 1)$-hyperbolic. k-chordal graphs with $k \geq 4$ are $\lfloor k/4 \rfloor$-hyperbolic [47]. Finally, graphs with hyperbolicity at most one, which contain chordal graphs, have been fully characterized [5, 2, 28, 14].

From the 4-points condition [23], it is obvious that determining the hyperbolicity δ of a graph of order n can be done in time $O(n^4)$, by testing all the 4-tuples of vertices of the graph. This running time has recently been reduced to $O(n^{3.69})$ [21] using the fast (max,min)-matrix multiplication algorithm proposed in [20] (of which no implementation seems to be available). Furthermore, the computation time for large-scale graphs remains prohibitive. For instance, for a graph with 29 432 vertices (largest biconnected component of the November 2013 map of the Autonomous Systems (AS) of the Internet provided by CAIDA [46]), the algorithm has to perform $\approx 10^{16}$ operations which represents several weeks of computations on a standard computer. For this reason several heuristic algorithms have been proposed, which were used to determine the hyperbolicity of CAIDA AS maps and various forms of social networks [17, 27]. They were also applied to instances of Erdös-Rényi random graphs and colored random graphs, whose hyperbolicity is known to be unbounded [34, 40]. A 2-approximation algorithm running in cubic time is obtained by fixing one vertex and evaluating all possible 4-tuples containing that vertex [9]. Its running time has been reduced to $O(n^{2.69})$ [21]. Recently, a $2 + \varepsilon$-approximation algorithm running in $\tilde{O}(\varepsilon^{-1} n^{2.373})$ time has been proposed in [19]. To the best of our knowledge, none of the existing algorithms has been used to compute the exact value of the hyperbolicity of graphs with more than 10 000 nodes due to prohibitive running times or implementation work.

More precisely, the implementation of the basic $O(n^4)$ algorithm available in the distory package [6, 38] has only been used in [7] to evaluate and classify phylogenetic and hierarchical clustering trees with less than 500 nodes. Furthermore, the parallel implementation of the basic $O(n^4)$ algorithm used in [1] already took 13 295s ($\approx$ 3.5 hours) on a parallel computer equipped with 1 015 CPUs to compute the hyperbolicity of a graph with 8 104 nodes.

Our results In Section 3 we present a new algorithm for computing the hyperbolicity of a graph that is simple and efficient. Indeed, its running time on a single CPU outperforms by a factor 1 000 the running time of the parallel implementation used in [1]. Furthermore, we are able to compute the hyperbolicity of graphs with more than 30 000 nodes in a few hours on a standard computer while other methods would take days or even months. The running time of our algorithm depends on the shortest-path distances distribution and on the computed value of the hyperbolicity. Also, it runs in $O(n^4)$ time in the worst case, it is fast in practice since it uses bounds to drastically prune the search space. For instance, after a preliminary sorting of the pairs of vertices according to their distance, the hyperbolicity of the $(p \times q)$-grid is computed in time $O(1)$. We evaluate the performances of our algorithm in Section 4 on various large graphs. In particular, we show its efficiency with respect to other methods on CAIDA [46] and DIMES [41] AS maps, and various collaboration networks [30, 11]. We also evaluate the extra speedup offered by automatic parallelization tools.

The limitations of the proposed exact algorithm are on one hand the overall computation time, and on the other hand the memory requirements which are in $O(n^2)$ (though it is in this regard comparable to the $O(n^{3.69})$ algorithm from [21]). Consequently, our exact algorithm cannot be used on instances with hundreds of thousands of nodes. Therefore, we propose in Section 5 a heuristic algorithm for the hyperbolicity that is scalable for graphs with millions of nodes. This heuristic algorithm runs in $O(k^2(n+m))$ time and has a $O(n)$ memory requirement, where k is a parameter of the algorithm. The experimental performances of this heuristic with respect to our exact algorithm and others heuristics (e.g. [27]) are promising.

2 Definitions and known results

In this section, we fix some notations and recall some important definitions and results used in this paper.

Let $G = (V, E)$ be a connected graph with order $|V| = n$ and size $|E| = m$. Let $\mathtt{d}(a, b)$ denote the shortest path distance between vertices a and b in G. Let also $N(u)$ be the set of neighbors of vertex $u \in V$. The hyperbolicity of a graph has been defined by Gromov as follows.

Definition 1 (4-points condition [23]) *A graph G is δ-hyperbolic if for any vertices a, b, c, d of G, the two largest of the three sums $S_1 = \mathtt{d}(a, b) + \mathtt{d}(c, d)$, $S_2 = \mathtt{d}(a, c) + \mathtt{d}(b, d)$, and $S_3 = \mathtt{d}(a, d) + \mathtt{d}(b, c)$ differ by at most 2δ. The hyperbolicity $\delta(G)$ of a graph G is the smallest δ such that it is δ-hyperbolic.*

In addition, we will denote by $\delta_{\mathtt{diff}}(a, b, c, d)$ the difference between the two largest of the three sums S_1, S_2, and S_3, and so have $2\delta = \max_{a,b,c,d \in V} \delta_{\mathtt{diff}}(a, b, c, d)$. We will also use $\delta(a, b, c, d) = \delta_{\mathtt{diff}}(a, b, c, d)/2$.

A useful observation is that the hyperbolicity of a graph is the maximum of the hyperbolicity of its biconnected components. To see this, let x be a cut-vertex of G and let B_1 and B_2 be two components of G separated by x. Let now $a, b, c \in B_1$ and $d \in B_2$. We have $S_1 = \mathtt{d}(a, b) + \mathtt{d}(c, d) = \mathtt{d}(a, b) + \mathtt{d}(c, x) + \mathtt{d}(x, d)$, $S_2 = \mathtt{d}(a, c) + \mathtt{d}(b, d) = \mathtt{d}(a, c) + \mathtt{d}(b, x) + \mathtt{d}(x, d)$, and $S_3 = \mathtt{d}(a, d) + \mathtt{d}(b, c) = \mathtt{d}(a, x) + \mathtt{d}(x, d) + \mathtt{d}(b, c)$. The computed value for the 4-tuple a, b, c, d is the same as for the 4-tuple a, b, c, x (i.e., $\delta(a, b, c, d) = \delta(a, b, c, x)$). Moreover, when $a, b \in B_1$ and $c, d \in B_2$, the computed value is 0 (i.e., $\delta(a, b, c, d) = 0$). The biconnected components of a graph can be computed in linear time [44].

Other pre-processing methods for reducing the size of the input graph have been proposed. [42] proved that the hyperbolicity of G is equal to the maximum of the hyperbolicity of the graphs resulting from both a *modular* [22, 24] or a *split* [16, 15] decomposition of G. These decompositions can be computed in linear time [8]. Moreover, [12] show how to use the atoms of a decomposition of G by clique-minimal separators [45, 3]. The hyperbolicity of G is the maximum value of the hyperbolicity of modified versions of the atoms of the decomposition (see [12] for more details on this method). This decomposition can be obtained in time $O(nm)$.

3 Exact algorithm for computing the hyperbolicity

In this section, we formally describe a new exact algorithm for computing the hyperbolicity of a graph. We then give some hints of its time complexity and explain how to turn it into an approximation algorithm. Afterwards, we present in Section 3.2 a method for further reducing the time complexity of the algorithm.

3.1 Main algorithm

The core idea of our algorithm is to visit the most promising 4-tuples first, i.e. the most likely to yield a large hyperbolicity. Indeed, it has been proved in [42] that $\delta(a, b, c, d) \leq$

$\min_{x,y\in\{a,b,c,d\}} \mathtt{d}(x,y)$. In other words, $\delta(a,b,c,d)$ is small if two vertices among $\{a,b,c,d\}$ are close to each other. We provide a proof of this result for completion.

Lemma 1 ([42]) *Let $G=(V,E)$ be a connected graph and let $a,b,c,d\in V$. We have $\delta(a,b,c,d)\le \min_{x,y\in\{a,b,c,d\}} \mathtt{d}(x,y)$.*

Proof. Let $S_1=\mathtt{d}(a,b)+\mathtt{d}(c,d)$, $S_2=\mathtt{d}(a,c)+\mathtt{d}(b,d)$, and $S_3=\mathtt{d}(a,d)+\mathtt{d}(b,c)$, and assume w.l.o.g. that $S_1\ge S_2\ge S_3$. We have $\delta_{\mathtt{diff}}(a,b,c,d)=S_1-S_2=\mathtt{d}(a,b)+\mathtt{d}(c,d)-\mathtt{d}(a,c)-\mathtt{d}(b,d)$.

Using the triangular inequality, we deduce $\mathtt{d}(c,d)\le\mathtt{d}(a,c)+\mathtt{d}(a,b)+\mathtt{d}(b,d)$ and we obtain $\delta_{\mathtt{diff}}(a,b,c,d)\le 2\cdot\mathtt{d}(a,b)$. We obtain similarly that $\delta_{\mathtt{diff}}(a,b,c,d)\le 2\cdot\mathtt{d}(c,d)$.

Next, we use both $\mathtt{d}(a,b)\le\mathtt{d}(a,c)+\mathtt{d}(b,c)$ and $\mathtt{d}(c,d)\le\mathtt{d}(b,c)+\mathtt{d}(b,d)$ to obtain $\delta_{\mathtt{diff}}(a,b,c,d)\le 2\cdot\mathtt{d}(b,c)$. We obtain similarly that $\delta_{\mathtt{diff}}(a,b,c,d)\le 2\cdot\mathtt{d}(a,d)$.

Now, since $S_2\ge S_3$, we have $\delta_{\mathtt{diff}}(a,b,c,d)\le S_1-S_3\le\mathtt{d}(a,b)+\mathtt{d}(c,d)-\mathtt{d}(a,d)-\mathtt{d}(b,c)$. Then, we proceed as above to obtain $\delta_{\mathtt{diff}}(a,b,c,d)\le 2\cdot\mathtt{d}(a,c)$ and $\delta_{\mathtt{diff}}(a,b,c,d)\le 2\cdot\mathtt{d}(b,d)$. □

From Lemma 1 we know that promising 4-tuples are those with large distances between their vertices. Therefore, in our algorithm we visit the 4-tuple a,b,c,d before the 4-tuple a',b',c',d' if $\min\{\mathtt{d}(a,b),\ \mathtt{d}(c,d)\}>\min\{\mathtt{d}(a',b'),\mathtt{d}(c',d')\}$. This is based on Lemma 2, which also proves that $\delta(G)\le D/2$, where D is the diameter of the graph.

Lemma 2 *Let $G=(V,E)$ be a connected graph, let $a,b,c,d\in V$, let $S_1=\mathtt{d}(a,b)+\mathtt{d}(c,d)$, $S_2=\mathtt{d}(a,c)+\mathtt{d}(b,d)$, and $S_3=\mathtt{d}(a,d)+\mathtt{d}(b,c)$, and assume w.l.o.g. that $S_1\ge\max\{S_2,\ S_3\}$. We have $\delta_{\mathtt{diff}}(a,b,c,d)\le\min\{\mathtt{d}(a,b),\ \mathtt{d}(c,d)\}$.*

Proof. We have $S_2+S_3=\mathtt{d}(a,c)+\mathtt{d}(b,d)+\mathtt{d}(a,d)+\mathtt{d}(b,c)=(\mathtt{d}(a,c)+\mathtt{d}(b,c))+(\mathtt{d}(a,d)+\mathtt{d}(b,d))$. Using the triangular inequality, we deduce $S_2+S_3\ge 2\cdot\mathtt{d}(a,b)$. Since S_1 is the largest sum, we have $\delta_{\mathtt{diff}}(a,b,c,d)=S_1-\max\{S_2,\ S_3\}\le S_1-(S_2+S_3)/2\le S_1-\mathtt{d}(a,b)=\mathtt{d}(c,d)$. We obtain similarly that $\delta_{\mathtt{diff}}(a,b,c,d)\le\mathtt{d}(a,b)$. □

Assuming that `pairs` is the list of the $\binom{n}{2}$ pairs of vertices sorted by non increasing distances, this yields Algorithm 1.

Algorithm 1: Hyperbolicity

```
  Input: $G$ is a 2-connected graph
  Input: pairs is the list of the $\binom{n}{2}$ pairs of vertices sorted by decreasing distances.
  Result: $\delta$, the hyperbolicity of $G$ (observe that $2\delta = \mathrm{h}_{\mathtt{diff}}$).
1 Let $\mathrm{h}_{\mathtt{diff}} := 0$;
2 for $1 \le i < \binom{n}{2}$ do
3     $(a,b) :=$ pairs$[i]$;
4     for $0 \le j < i$ do
5         $(c,d) :=$ pairs$[j]$;
6         $\mathrm{h}_{\mathtt{diff}} := \max\{\mathrm{h}_{\mathtt{diff}},\ \delta_{\mathtt{diff}}(a,b,c,d)\}$ ;
7         if $\mathtt{d}(a,b) \le \mathrm{h}_{\mathtt{diff}}$ then
8             return $\mathrm{h}_{\mathtt{diff}}/2$
9 return $\mathrm{h}_{\mathtt{diff}}/2$
```

Thanks to Lemma 2 we know that at any step of the algorithm the value of $\mathtt{d}(a,b)$ is an upper bound on the value of $\delta_{\mathtt{diff}}(a,b,c,d)$. If the current lower bound is $\mathrm{h}_{\mathtt{diff}}$, none of the

4-tuples such that $\mathtt{d}(a,b) \leq \mathrm{h}_{\mathtt{diff}}$ can be used to improve the lower bound. We can thus stop exploration (line 8 of the algorithm) and return $\delta(G) = \mathrm{h}_{\mathtt{diff}}/2$.

Since we have $\binom{n}{2}$ pairs of vertices, and since Algorithm 1 considers pairs of pairs, the worst case time complexity of the algorithm is in $O(n^4)$ (and has a quadratic memory usage). However, we observe that the running time of the algorithm depends on the computed value of the hyperbolicity. More precisely, we have:

Proposition 1 *Given a δ-hyperbolic graph where $P_{\geq \ell}$ is the number of pairs of vertices at distance $\geq \ell$ from each other, the time complexity of Algorithm 1 is in $O\left(P^2_{\geq 2\delta}\right)$.*

Proof. Since the list `pairs` of pairs of vertices is sorted by decreasing distances, and since Algorithm 1 uses Lemma 2, it considers only pairs of pairs $((a,b),(c,d))$ such that $D \geq \mathtt{d}(c,d) \geq \mathtt{d}(a,b) \geq 2\delta$. $\square$

For instance, if the input graph is a $n \times n$ grid, with diameter $2n-2$ and hyperbolicity $\delta = n-1$ (so $\mathrm{h}_{\mathtt{diff}} = 2n-2$), the value of the hyperbolicity will be obtained with the first considered 4-tuple and the execution of the algorithm is immediately stopped. On the other hand, if the input graph is a $n \times 2$ grid, with diameter n and hyperbolicity $\delta = 1$ (so $\mathrm{h}_{\mathtt{diff}} = 2$), almost all 4-tuples will be considered. We will show in Section 3.2 how to reduce the worst case time complexity for grids to $O(1)$.

Now, since $\mathtt{d}(a,b)/2$ and $\mathrm{h}_{\mathtt{diff}}/2$ are respectively upper and lower bounds for the hyperbolicity, we can easily turn Algorithm 1 into an approximation algorithm. More precisely, we can insert one of the following statements after line 6.

- "If computation time is larger than allowed computation time, then stop computations and return $\mathrm{h}_{\mathtt{diff}}/2$ and $\mathtt{d}(a,b)/2$". We get $\mathrm{h}_{\mathtt{diff}} \leq 2\delta \leq \mathtt{d}(a,b)$;
- "If $\mathtt{d}(a,b) \leq apx \cdot \mathrm{h}_{\mathtt{diff}}$, then stop computations and return $\mathrm{h}_{\mathtt{diff}}/2$." This yields an approximation of the hyperbolicity with proven multiplicative factor *apx* (i.e., $\mathrm{h}_{\mathtt{diff}} \leq 2\delta \leq apx \cdot \mathrm{h}_{\mathtt{diff}}$);
- "If $\mathtt{d}(a,b) - \mathrm{h}_{\mathtt{diff}} \leq 2apx$, then stop computations and return $\mathrm{h}_{\mathtt{diff}}/2$." This yields an approximation of the value δ of the hyperbolicity with proven additive constant *apx* (i.e., $\frac{\mathrm{h}_{\mathtt{diff}}}{2} \leq \delta \leq \frac{\mathrm{h}_{\mathtt{diff}}}{2} + apx$).

In fact, the main part of the running time of the algorithm consists in closing the gap between lower bounds, that are generally found very quickly, and upper bounds (see Section 4).

3.2 Using far-apart pairs to reduce the number of visited 4-tuples

In this section, we present a method for reducing the number of pairs in the list `pairs`, thus reducing the running time of Algorithm 1. Using this method, we can formally establish the worst case time complexity of Algorithm 1 on particular graph classes.

The notion of *far-apart pairs* has been introduced in [42, 35] to reduce the number of 4-tuples to consider in the computation of the hyperbolicity. Roughly, we say that two vertices $u, v \in V$ are not *far apart* if there exists $w \in V$ such that either u lies on a shortest path from w to v, or v lies on a shortest path from u to w. More formally, we have:

Definition 2 *Given $G = (V,E)$, the pair u, v is far-apart if for every $w \in V \setminus \{u,v\}$, $\mathtt{d}(w,u) + \mathtt{d}(u,v) > \mathtt{d}(w,v)$ and $\mathtt{d}(w,v) + \mathtt{d}(u,v) > \mathtt{d}(w,u)$.*

The set of all far-apart pairs can be determined in time $O(nm)$ in unweighted graphs through a breadth-first search (BFS). Roughly, we initialize the set F of far-apart pairs with the set of all pairs. Then, pair (s, u) is removed from F during the execution of a BFS from s if some neighbor v of u is at distance $\mathtt{d}(s, u) + 1$ from s. At the end of the execution of all the BFS (one BFS per vertex $s \in V$), the set F contains only the far-apart pairs. The interest of far-apart pairs for our purposes is explained by the following lemma:

Lemma 3 *Let G be a connected graph. There exist two far-apart pairs (u, v) and (x, y) satisfying $\delta(u, v, x, y) = \delta(G)$.*

Proof. Given a 4-tuple u, v, x, y of G, write $S_1 = \mathtt{d}(u, v) + \mathtt{d}(x, y)$, $S_2 = \mathtt{d}(u, x) + \mathtt{d}(v, y)$ and $S_3 = \mathtt{d}(u, y) + \mathtt{d}(v, x)$. Assume furthermore that $S_1 \geq S_2 \geq S_3$. In particular, let u, v, x, y be such that $\delta(G) = \delta(u, v, x, y)$, maximizing S_1 w.r.t. this property.

We claim that u, v and x, y are far-apart pairs. By contradiction, assume that one of them is not and w.l.o.g., suppose that it is x, y. We can suppose w.l.o.g. that there exists $z \in N(x)$ satisfying $\mathtt{d}(z, y) = 1 + \mathtt{d}(x, y)$.

In such a case, write $S'_1 = \mathtt{d}(u, v) + \mathtt{d}(z, y)$, $S'_2 = \mathtt{d}(u, z) + \mathtt{d}(v, y)$ and $S'_3 = \mathtt{d}(u, y) + \mathtt{d}(v, z)$. Since $S_1 \geq S_2 \geq S_3$ by the hypothesis, it follows that $S'_1 = S_1 + 1 = \max\{S'_1, S'_2, S'_3\}$. In addition, we have $\max\{S'_2, S'_3\} \leq \max\{S_2, S_3\} + 1$. So, we have $\delta(u, v, x, z) \geq \delta(u, v, x, y)$ and $S'_1 > S_1$, contradicting the maximality of S_1. $\square$

Since the ordering in which Algorithm 1 visits the 4-tuples assumes that S_1 is the largest sum, Lemma 3 allows us to remove from `pairs` all non-far-apart pairs. This substantially reduces the number of visited 4-tuples as shown in the experiments reported in Section 4. Moreover, for particular graph classes it can be formally proved that the running time of Algorithm 1 is small (excluding the computation of the distance matrix and the set of far-apart pairs which are done in time $O(nm)$).

Proposition 2 *The computation time of Algorithm 1 using far-apart pairs is:*

1. *$O(1)$ when G is a $p \times q$-grid with $p, q \geq 2$.*
2. *$O(n^2)$ when G is a cycle of order $n \geq 4$.*

Proof.

1. Let $\{(i, j) \mid 1 \leq i \leq p,\ 1 \leq j \leq q\}$ be the set of vertices of a $p \times q$-grid. By symmetry, we have only to consider the following cases:

   - The pair $((i, j), (i', j'))$ with $i \leq i'$, $j \leq j'$, $(i, j) \neq (i', j')$, and $(i, j) \neq (1, 1)$ is not far-apart since we have $\mathtt{d}((i, j), (i', j')) + \mathtt{d}((1, 1), (i, j)) = (i' - i + j' - j) + (i + j - 2) = i' + j' - 2 = \mathtt{d}((1, 1), (i', j'))$.
   - The pair $((i, j), (i', j'))$ with $i \leq i'$, $j > j'$, $(i, j) \neq (i', j')$, and $(i, j) \neq (1, q)$ is not far-apart since we have $\mathtt{d}((i, j), (i', j')) + \mathtt{d}((1, q), (i, j)) = (i' - i + j - j') + (i - 1 + q - j) = i' - 1 + q - j' = \mathtt{d}((1, q), (i', j'))$.

   Finally, the grid has only two far-apart pairs, namely $((1, 1), (p, q))$ and $((p, 1), (1, q))$.

2. Let $\{i \mid 0 \leq i < n\}$ be the set of vertices of a cycle of order n, let $p = \left\lfloor \frac{n}{2} \right\rfloor$, and consider the pair $(0, i)$ with $1 \leq i < p$. This pair is not far-apart since i is on a shortest path from node 0 to node p. By symmetry, the same holds for all pairs (i, j) such that $\mathtt{d}(i, j) < p$.

Consequently, when $n = 2p$, the set of far-apart pairs is $L_{2p} = \{(i, i+p) \mid 0 \leq i < p\}$, and we have $|L_{2p}| = n/2$. When $n = 2p+1$ the set of far-apart pairs is $L_{2p+1} = \{(i, i+p), (i, i+p+1) \mid 0 \leq i < p\}$, and we have $|L_{2p}| = n$.

$\square$

4 Experimental performances

We now evaluate the performances of Algorithm 1. We start presenting our experimental settings in Section 4.1. Then, in Section 4.2 we compare the running time performances of our algorithm on a single CPU with respect to the running time of the massively parallel implementation proposed in [1]. We additionally evaluate the extra speedup resulting from the pre-processing method proposed in [12] and using a decomposition of the graph by clique-separators. We perform a deeper analysis of this pre-processing method in Section 4.4. Then, in Section 4.2 we analyze the performances of Algorithm 1 on a large set of empirical graphs, the largest of which has more than 30 000 nodes. We finally propose in Section 4.5 a parallel implementation of Algorithm 1 and compare the sequential running times presented in Section 4.3 with the parallel executions over 4, 8 and 16 threads.

4.1 Experimental settings

We have already included Algorithm 1 into the open-source mathematics software Sage [43] and the Java graph optimization library Grph [25]. A standalone C implementation of our algorithm is available from [13]. Therefore, anyone can now reproduce our experiments. We used the C implementation to conduct the experiments. All reported computations have been performed on computers equipped with 2 hexa-core 2.93GHz Intel Xeon X5670 processors and 96GB of RAM. Except for Section 4.5, all our computations have been done using a single core.

Most of our comparisons are in terms of number of *visited 4-tuples*, that is the number of 4-tuples for which the algorithm has effectively computed $\delta_{\texttt{diff}}(a, b, c, d)$. This number is implementation and computer independent and so can be reused by others to perform fair comparisons. Reported running times in seconds can certainly be reduced using improved implementations and faster computers. Nonetheless, we use running time to give indications on the orders of magnitude we gain using our algorithm compared to other proposals and implementations. In particular, since we are not aware of any available implementation of the algorithm proposed in [21] respecting the $O(n^{3.69})$ time complexity, we have estimated its running time as $n^{3.69}/10^{10}$ seconds. We believe this is particularly fair since with our implementation of Algorithm 1 we evaluate no more than 300 million 4-tuples per second.

As explained in Section 2, the hyperbolicity of a graph is the maximum value over all its biconnected components. We thus proceed as follows: 1) We decompose the graph G into biconnected components and sort them by decreasing numbers of vertices; 2) We run Algorithm 1 sequentially on each of these biconnected components, unless the diameter D_H of a component H is less or equal to the maximum value $\mathrm{h}_{\texttt{diff}}$ we have computed on previous components (recall that $\delta(H) \leq D_H/2$). In all our experiments, we have observed that the value $\delta(G)$ is found within the Largest Biconnected Component (LBC) of the graph.

We also use as pre-processing a decomposition of G by clique-minimal separators [45, 3], since the hyperbolicity of G is the maximum value of the hyperbolicity of modified versions of the atoms of the decomposition (see [12] for more details on this method). This pre-processing method has time complexity in $O(nm)$, that is the same time complexity than the computation of the distances between all pairs of nodes. Note that this decomposition by clique-minimal separators can only be used for graphs such that $\delta(G) \geq 1$, which is the case in our experiments. As for the biconnected components, $\delta(G)$ is always found with the first considered modified atom (which is the largest). We denote LMA this largest modified atom.

To get a better understanding of the behavior of our algorithm, we have selected a set of graphs from different sources with different sizes and structural properties. We computed the hyperbolicity of :

- Maps of the relationships between the Autonomous Systems (AS) of the Internet. These graphs have been collected by the Cooperative Association for the Internet Data Analysis (CAIDA) from January 2000 to November 2013 [46] and by the Distributed Internet Measurements & Simulations (DIMES) project from 2010 to 2012 [41];
- 5 collaboration networks of different scientific research communities, namely `ca-AstroPh` for the astrophysics community, `ca-CondMat` for the condensed matter physics community, `ca-GrQc` for the general relativity and quantum cosmology community, `ca-HepPh` for the high energy physics-phenomenology, and `ca-HepTh` for the high energy physics-theory community (see [30]).
- The `Gnutella09` snapshot of the peer-to-peer network of the Gnutella file sharing network of August 9, 2002 (see [30]).
- The `loc-Brightkite` friendship network of the Brightkite location-based social networking service provider. This graph is composed of 58 228 nodes, the largest graph for which we computed the exact hyperbolicity (see [30]).

Since our first motivation is to better understand the behavior of our algorithm, sections 4.2 and 4.3 focus on the performances of Algorithm 1. Thus, the computation times given in Tables 1 and 2 do not include the computation time required by the pre-processing steps (i.e., the computation times of the distance matrix, the far-apart pairs, the LBC or the LMA). We provide in Section 4.4 an analysis of the whole processus.

4.2 Performance comparison on benchmark instances

Massive parallelism has been used in [1] to compute the hyperbolicity of some graphs. They have used OpenMP [36] for parallelizing the loops of the basic $O(n^4)$ algorithm and executed the code on a machine that scales up to 1 015 threads. Note that the primary goal of this study was to compare the performances of worksharing and tasking models with OpenMP, using the computation of the hyperbolicity only as a case study. However, it is interesting to compare the running times reported in [1] with our running times.

In Table 1, we have reported the running time obtained in [1, Table 2] for the tasking model only, since it is always faster than the worksharing model, and for respectively 512 and 1015 CPUs. We have also reported the running time using our implementation of Algorithm 1 for the same graphs with different settings. Clearly, the combination of decomposition methods to reduce the size of the input graph with the pruning of 4-tuples operated by Algorithm 1 offers drastic reduction of the computation time. Our algorithm is between 3 and 6 orders of magnitude faster on these graphs, and it uses a single CPU.

To get a better insight on the performances of Algorithm 1 on these graphs, we have reported in Table 1 for the LBC of each graph the number of nodes and the number $P_{=\ell}$ of pairs at distance ℓ. By Proposition 1, we know that the worst case time complexity of the algorithm is in $O(P^2_{\geq 2\delta})$. However, we have observed for many graphs that a 4-tuple (a, b, c, d) such that

	ca-GrQc				as20000102				Gnutella09	
n	4 158				6 474				8 104	
δ	3.5				2.5				3	
[1]										
Time in sec.										
512 CPUs	1 916				9 039				26 888	
1015 CPUs	3 691				11 419				13 295	
	LBC		LMA		LBC		LMA		LBC = LMA	
n	2 651		2 107		4 009		2 991		5 606	
Total 4-tuples	$2.0\,10^{12}$		$8.2\,10^{10}$		$1.1\,10^{13}$		$3.3\,10^{12}$		$4.1\,10^{13}$	
ℓ	$P_{=\ell}$	$F_{=\ell}$	$P_{=\ell}$	$F_{=\ell}$	$P_{=\ell}$	$F_{=\ell}$	$P_{=\ell}$	$F_{=\ell}$	$P_{=\ell}$	$F_{=\ell}$
1	10 480	1 129	7 135	182	10 101	–	7 993	–	23 510	–
2	50 508	5 538	38 484	2 293	1 231 628	519 224	412 563	64 871	292 458	3
3	209 027	24 879	160 255	12 028	3 721 215	2 235 874	1 873 697	846 725	2 101 638	1 999
4	625 349	109 394	463 779	59 800	2 569 590	1 943 404	1 765 412	1 248 401	7 222 588	451 789
5	1 111 919	354 964	752 072	204 664	475 366	413 824	388 989	334 668	5 420 722	3 376 888
6	983 190	512 472	567 187	274 411	25 648	24 017	22 453	21 001	645 387	629 722
7	418 341	287 452	193 841	127 957	485	472	435	422	4 490	4 404
8	91 502	73 153	32 858	25 825	3	3	3	3	22	22
9	11 332	9 800	2 890	2 441	–	–	–	–	–	–
10	886	811	165	150	–	–	–	–	–	–
11	41	41	5	5	–	–	–	–	–	–
$P_{\geq 2\delta}$	522 102	371 257	229 759	156 378	501 502	438 316	411 880	356 094	649 899	634 148
$P_{\geq 2\delta+1}$	103 761	83 805	35 918	28 421	26 136	24 492	22 891	21 426	4 512	4 426
Visited 4-tuples	$5.4\,10^{9}$	$3.5\,10^{9}$	$6.4\,10^{8}$	$4.0\,10^{8}$	$3.4\,10^{8}$	$3.0\,10^{8}$	$2.6\,10^{8}$	$2.3\,10^{8}$	$1.0\,10^{7}$	$9.8\,10^{6}$
Time (in sec.)	13	9	1.58	1	0.85	0.74	0.65	0.57	0.029	0.028

Table 1: Comparison of running time (in sec.) of Algorithm 1 with [1].

$\delta(a,b,c,d) = \delta(G)$ is found very early and so that the algorithm ends as soon as $\mathtt{d}(a,b) = 2\delta(G)$ (line 8). Consequently, in these experiments, the number of visited 4-tuples is $\binom{P_{\geq 2\delta+1}}{2}$. Given the respective repartition of distances and values of hyperbolicity of these graphs, this explains why, although the `Gnutella09` graph is twice as large as the `ca-GrQc` graph, the running time of the algorithm is 400 times smaller for `Gnutella09`.

We have also reported in Table 1 the number $F_{=\ell}$ of far-apart pairs at distance ℓ in the LBC of each graph. As expected from the definition of far-apart pairs, the number of pairs at small distances is drastically reduced, thus pruning 4-tuples with small values. Also, we observe a substantial reduction on the number of visited 4-tuples, and hence of the computation time.

Moreover, we have reported in Table 1 the number of nodes and both the number $P_{=\ell}$ of pairs and the number $F_{=\ell}$ of far-apart pairs at distance ℓ in the largest modified atom (LMA) of each of these graphs. This pre-processing method helps further reducing the computation time, except for `Gnutella09` since the LMA is exactly the LBC and so the decomposition by clique-separators is not helpful in this case. For `ca-GrQc`, the number of visited 4-tuples is 13 times less when combining clique-decomposition and far-apart pairs than for the LBC without far-apart pairs. For `as20000102`, we reduce by a factor 1.5 the number of visited 4-tuples.

4.3 Large-scale instances and decompositions

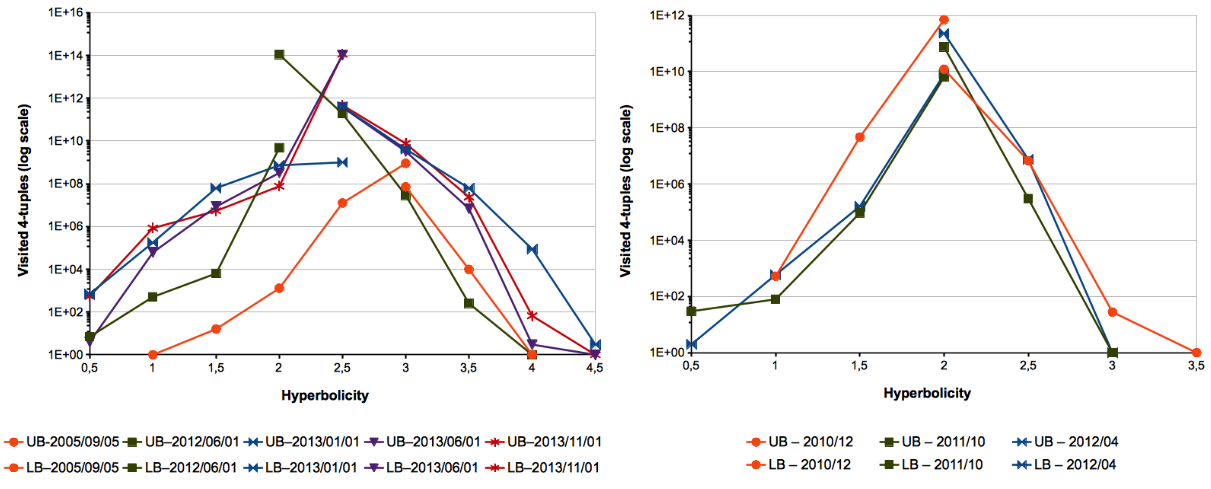
We now report on experiments performed on large scale instances, namely CAIDA [46] and DIMES [41] AS maps, collaboration networks of scientific communities (`ca-*`) [30], and a location-based online social networks (`loc-brightkite`) [11]. For each considered graph, we have reported in Table 2 its number n of nodes, the value δ of its hyperbolicity, the number n_B of nodes of its LBC, the diameter D_B of its LBC, and the number n_A of nodes of its LMA. We have also reported for the LMA the total number of 4-tuples in this component (Tot.), the number of visited 4-tuples using Algorithm 1 (Vis.), and the running time in seconds (T_{seq}). Columns T_4, T_8, and T_{16} relate to a parallel implementation as will be discussed in Section 4.5.

				Number of nodes			4-tuples		Computation time (in sec.)			
	Instance name	δ	D_B	n	n_B	n_A	Tot.	Vis.	T_{seq}	T_4	T_8	T_{16}
CAIDA	`2004/01/05`	2.5	8	16 301	10 424	7 963	$1.7\,10^{14}$	$2.2\,10^{10}$	57	25	13	7
	`2004/06/07`	2	8	17 306	11 100	8 568	$2.2\,10^{14}$	$4.5\,10^{12}$	14 442	5 606	3 726	2 242
	`2005/09/05`	3	8	20 344	12 957	9 751	$3.8\,10^{14}$	$9.0\,10^{8}$	2	1	0.5	0.5
	`2010/01/20`	2	8	33 508	20 940	17 057	$3.5\,10^{15}$	$5.6\,10^{13}$	201 795	166 101	48 852	38 571
	`2011/01/16`	2	8	36 878	23 214	18 934	$5.4\,10^{15}$	$4.7\,10^{13}$	185 191	67 148	41 836	38 543
	`2012/01/01`	2	8	40 109	25 614	20 768	$7.7\,10^{15}$	$1.0\,10^{14}$	334 078	156 973	88 799	84 902
	`2012/06/01`	2	8	41 203	25 815	21 196	$8.4\,10^{15}$	$1.1\,10^{14}$	352 480	152 552	95 575	89 513
	`2013/01/01`	2.5	10	43 274	27 454	22 347	$1.0\,10^{16}$	$4.0\,10^{11}$	1 563	449	235	135
	`2013/06/01`	2.5	9	44 611	28 654	23 331	$1.2\,10^{16}$	$1.0\,10^{14}$	386 574	164 369	114 896	86 990
	`2013/11/01`	2.5	9	45 427	29 432	23 978	$1.4\,10^{16}$	$1.1\,10^{14}$	363 770	151 634	98 518	90 419
DIMES	`2010/12`	2	7	26 235	18 764	14 930	$2.0\,10^{15}$	$7.0\,10^{11}$	2 171	825	436	246
	`2011/10`	2	7	25 683	17 137	13 366	$1.3\,10^{15}$	$7.5\,10^{10}$	214	89	45	23
	`2012/04`	2	7	25 367	16 907	13 217	$1.3\,10^{15}$	$2.3\,10^{11}$	717	299	137	75
	`loc-brightkite`	3	11	58 228	33 187	32 040	$4.4\,10^{16}$	$1.9\,10^{12}$	7 710	3 274	1 424	858
	`ca-AstroPh`	3	10	18 772	15 929	13 407	$1.3\,10^{15}$	$7.8\,10^{9}$	22	9	5	3
	`ca-CondMat`	3.5	12	23 133	17 234	13 643	$1.4\,10^{15}$	$3.6\,10^{10}$	101	42	24	12
	`ca-GrQc`	3.5	11	5 242	2 651	2 107	$8.2\,10^{12}$	$4.0\,10^{8}$	1	0.4	0.2	0.1
	`ca-HepPh`	3	11	12 008	9 025	7 170	$1.1\,10^{14}$	$1.9\,10^{10}$	50	31	11	6
	`ca-HepTh`	4	11	9 877	5 898	5 236	$3.1\,10^{13}$	$1.2\,10^{8}$	0.3	0.1	0.07	0.04

Table 2: Performances of proposed algorithms on some large instances.

We observe as in Section 4.2 that the number of visited 4-tuples decreases drastically when δ increases. For instance, the algorithm visits less 4-tuples for the `loc-brightkite` graph than for the CAIDA AS map of 2004/06/07 while the latter has 4 times less vertices. However, we also observe important variations in the number of visited 4-tuples between graphs with same numbers of nodes and hyperbolicity. For instance, we visit 3 times more 4-tuples for the DIMES map 2012/04 than for the DIMES map 2011/10. This can be explained by the differences in the distribution of the distances of the far-apart pairs. The map 2012/04 has more pairs at distance $\geq 2\delta$, and more precisely $\geq 2\delta + 1$, than the map 2011/10.

Moreover, except for the CAIDA maps 2005/09/05, 2013/06/01, 2013/11/01 and the DIMES map 2010/12, we have observed that the number of visited 4-tuples when using Algorithm 1 on the LMA is equal to $\binom{F_{\geq 2\delta+1}}{2}$ where $F_{\geq \ell}$ is the number of far-apart pairs at distance at least ℓ from each other. It indicates that a 4-tuple such that $\delta(a, b, c, d) = \delta$ is found before the algorithm starts considering a far-apart pair (a', b') such that $\mathtt{d}(a', b') = 2\delta$. Indeed, recall



(a) CAIDA AS maps since 2004.

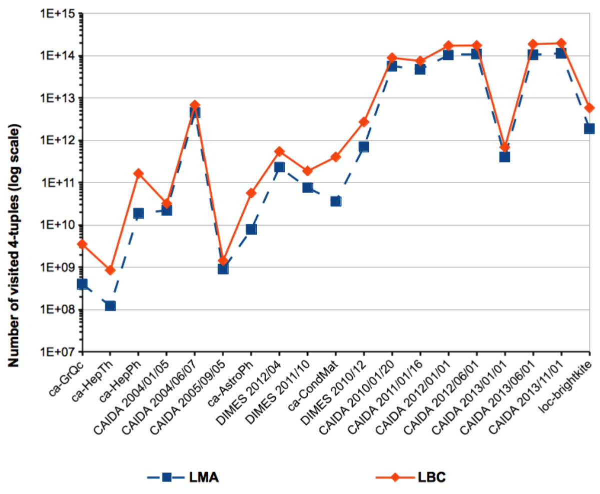
(b) DIMES AS maps since 2010.

Figure 1: Number of visited 4-tuples to reach lower and upper bounds. Number of 4-tuples for lower bounds are plotted from left to right, and from right to left for upper bounds.

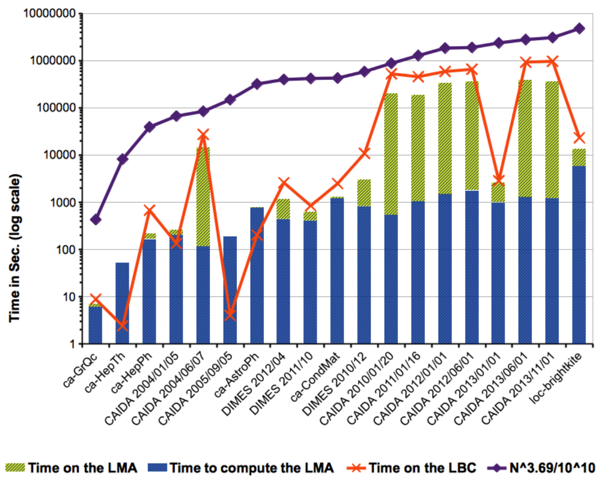
that in Algorithm 1 $\mathtt{d}(a,b)$ is the upper bound on δ, and $\mathrm{h}_{\mathtt{diff}}/2$ is the current lower bound. Also, as soon as such a pair (a',b') is considered, the optimality of the solution is proved and the algorithm ends and returns the result. It directly implies that only 4-tuples with $\mathtt{d}(a,b) \geq 2\delta+1$ are visited.

For a selection of 4 graphs, we have reported in Fig. 1 the number of already visited 4-tuples when new lower and upper bounds are obtained. Recall that in Algorithm 1 $\mathtt{d}(a,b)$ is the current upper bound on δ, and $\mathrm{h}_{\mathtt{diff}}/2$ is the current lower bound. When considering the plots in Fig. 1b for DIMES maps 2011/10 and 2012/04, we observe that the lower bound is increased to $\mathrm{h}_{\mathtt{diff}}/2 = 2$ before the upper bound can be reduced to 2. Hence, the number of visited 4-tuples is $\binom{F_{\geq 2\delta+1}}{2}$ for these maps. However, for the DIMES map 2010/12, the upper bound is reduced to 2 before a 4-tuple such that $\delta(a,b,c,d) = 2$ is found. Hence, the number of visited 4-tuples is much larger in this case. We observe the same behavior in Fig. 1a for the CAIDA maps of 2005/09/05, 2013/06/01 and 2013/11/01. This allows us to explain for instance the huge difference in numbers of visited 4-tuples, and so running times, between the CAIDA maps 2013/01/01 and 2013/06/01. Indeed, the number of nodes of the LMA of these maps differ by less than 5% (see Table 2), they both have hyperbolicity 2.5 and approximately $1.6\,10^7$ far-apart pairs at distance $\geq 2\delta = 5$ and $8.9\,10^5$ far-apart pairs at distance $\geq 2\delta+1 = 6$. So these maps have very similar structure. The main difference is that the algorithm is able to find a 4-tuple such that $\delta(a,b,c,d) = 2.5$ using a far-apart pair (a,b) with $\mathtt{d}(a,b) \geq 6$ for the map 2013/01/01 while for the map 2013/06/01 it has to visit far-apart pairs at distance $\mathtt{d}(a,b) \geq 5$. So the algorithm has to visit 10^{13} times more 4-tuples for the map 2013/06/01.

Finally, we can observe from the plots of Fig. 1 that the optimality gap (distance between lower and upper bounds) is quickly reduced to a small interval. This suggests that using the algorithm as a heuristic by bounding the allowed computation time or the number of visited 4-tuples, as explained in Section 3.1, would yield very good solutions. For the graphs of Fig. 1, one hour of computation is sufficient to either prove optimality or to return a solution with optimality gap below 1/2.



(a) Visited 4-tuples on the LBC and the LMA



(b) Computation times

Figure 2: Comparisons of the computation time on the LMA (including the time to compute the LMA) with the computation time on the LBC and the estimated computation time of [21]

4.4 The cost of pre-processing

In this section, we show that the cost of pre-processing, and more precisely the computation time of the LMA from the LBC, is greatly balanced by the reduction of the computation time of the hyperbolicity on the LMA compared to the computation time on the LBC for most graphs.

Fig. 2a shows (in logarithmic scale) the number of visited 4-tuples when computing the hyperbolicity on the LMA and in the LBC. In this figure, the graphs are ordered by increasing number of nodes in the LBC (see Table 2). For all graphs, we observe that the algorithm visits up to 10 times less 4-tuples on the LMA than on the LBC, even in the case of the CAIDA map 2005/09/05 and the DIMES map 2010/12. This was expected since the LMA has significantly less nodes than the LBC, as can be seen from Table 2.

Fig. 2b shows the running times for computing the LMA and its hyperbolicity for the instances in our test set. The plot also shows a comparison of the total running times of our approach with the running times for computing the hyperbolicity directly on the LBC. We first observe that the computation of the LMA requires a non-negligeable amount of time which is sometimes larger than the computation time of the hyperbolicity on the LMA. This is for instance the case for `ca-HepTh`, `ca-AstroPh`, the CAIDA maps of 2004/01/05 and 2005/09/05. Moreover, for these four graphs the computation time of the hyperbolicity on the LBC is smaller than the computation time of the LMA. In fact, the only cases where computing the hyperbolicity directly on the LBC rather than first computing the LMA and then its hyperbolicity are the smallest graphs for which the computation of the hyperbolicity is quick. Nonetheless, for most of the tested instances the sum of the computation times of the LMA and its hyperbolicity is smaller than the computation time of the hyperbolicity on the LBC. For instance, it takes 1h 38min to compute the LMA of `loc-brightkite` and 2h 8min to compute its hyperbolicity for a total computation time of 3h 46min, while 6h 28min of computation are needed to compute the hyperbolicity directly on the LBC. So overall, using the LMA results in a net saving of 3 hours. For the CAIDA map 2013/06/01, it takes 21min to compute the LMA and 5 days to compute its hyperbolicity, while it takes 11 days to compute the hyperbolicity on the LBC. The overall computation time has been divided by 2 for a net saving of 6 days.

Fig. 2b also shows the estimated computation time of the algorithm proposed in [21] with time complexity in $O(n^{3.69})$. Recall that we estimate the running time of this algorithm as $n^{3.69}/10^{10}$ seconds, while Algorithm 1 visits no more than $3.0\,10^8$ 4-tuples per second. With this setting, our algorithm is significantly faster than [21].

4.5 Parallelism

We have seen in previous sections that Algorithm 1 is orders of magnitude faster than both the expected running time of the algorithm proposed in [21] with time complexity in $O(n^{3.69})$, and the massively parallel implementation used in [1]. However, for some instances such as CAIDA AS maps, the computation time remains large. Also, we show in this section that a significant speedup can be obtained using parallelism.

Indeed, we observe that all the operations performed in Algorithm 1 are independent from each other, except for the update of the lower bound $h_{\texttt{diff}}$ which is a rare event. Therefore, we can rely as in [1] on automatic loop parallelization tools such as OpenMP [36] to distribute the computations among multiple threads. We have reported in Algorithm 2 the instructions given to OpenMP to parallelize the first loop (line 2), to declare the update of variable $h_{\texttt{diff}}$ as critical (line 9), and to propagate the new bound among all the threads (line 10).

Algorithm 2: Parallel

```
   Input: $G$ is a 2-connected graph
   Input: pairs is the list of the $\binom{n}{2}$ pairs of vertices sorted by decreasing distances.
   Result: $\delta$, the hyperbolicity of $G$ (observe that $2\delta = h_{\texttt{diff}}$).
1  Let $h_{\texttt{diff}} := 0$;
2  #pragma omp parallel shared($h_{\texttt{diff}}$, pairs) private($h$, $j$)
3      for $1 \leq i < \binom{n}{2}$ do
4          $(a, b) :=$ pairs$[i]$;
5          for $0 \leq j < i$ do
6              $(c, d) :=$ pairs$[j]$;
7              Let $h := \delta_{\texttt{diff}}(a, b, c, d)$;
8              if $h < h_{\texttt{diff}}$ then
9                  #pragma omp critical;
10                 #pragma omp flush($h$ as $h_{\texttt{diff}}$);
11             if $\texttt{d}(a, b) \leq h_{\texttt{diff}}$ then
12                 GOTO line 14
13 #pragma omp barrier;
14 return $h_{\texttt{diff}}/2$
```

We have reported in Table 2, for the LMA of each graph, the running time of Algorithm 2 when using respectively 4, 8, and 16 threads (columns T_4, T_8, and T_{16}). Compared to the sequential version, the parallel implementation offers interesting speedups[1]. More precisely, the average speedups are respectively 2.13 with 4 threads, 3.7 with 8 threads, and 4.3 with 16 threads. This is less than what we could expect and further analysis and optimization of the code could certainly result in better performances.

[1]Recall that the speedup of a parallel algorithm is defined as $S_p = \frac{T_1}{T_p}$, and its efficiency as $E_p = \frac{T_1}{pT_p}$, where T_i is the execution time of the algorithm over p processing units.

5 Heuristic for the hyperbolicity

In Section 3 we have presented an exact algorithm for computing the hyperbolicity of graphs. One of the main limitations of this algorithm, apart from the computation time, is the overall memory requirement. Indeed, it requires the distance matrix of the graph and so has space complexity in $O(n^2)$, assuming that the largest distance can be stored on a constant number of bits. The same limitation holds for the algorithm proposed in [21]. Therefore, in this section, we design a heuristic for the hyperbolicity with time complexity in $O(k^2(n+m))$, where k is a parameter of the algorithm, and space complexity in $O(n)$. This algorithm can thus be used on very large graphs (millions of vertices).

We have seen with Lemma 1 [42] that $\delta(a,b,c,d) \leq \min_{x,y\in\{a,b,c,d\}} \mathtt{d}(x,y)$. In other words, $\delta(a,b,c,d)$ is small if two vertices among $\{a,b,c,d\}$ are close to each other. Furthermore, we have seen with Lemma 2 that a 4-tuple with largest hyperbolicity has two pairs (a,b) and (c,d) such that $2\delta \leq \mathtt{d}(a,b) \leq \mathtt{d}(c,d)$. Consequently, we are interested in 4-tuples composed of vertices which are far from each other. To find such 4-tuples in a graph $G=(V,E)$ without computing the distance matrix, we propose the following approach which has the flavour of the 2-sweep heuristic proposed in [32] to quickly find a pair of vertices that are as far from each other as possible. We proceed as follows:

1. Choose at random a vertex $x \in V$, compute the distances from x using a BFS and let a be a vertex at largest distance from x (i.e., the last visited vertex of the BFS). Then compute distances from a using BFS and let b be a vertex at largest distance from a. Vertices a and b are thus selected using the 2-sweep heuristic [32]. Let also S_a be the set of vertices at distance $\mathtt{d}(a,b)/2$ from a;

2. Compute the distances from b using a second BFS and let S_b be the set of vertices at distance $\mathtt{d}(a,b)/2$ from b;

3. For at most k (randomly selected) vertices $c \in S_a \cap S_b$, compute distances from c using a third BFS, and for each vertex d visited during that BFS compute $\delta(a,b,c,d)$.

4. Repeat previous steps k times and return the largest computed value δ_h.

The running time of this heuristic is thus $O(k^2(n+m))$. Note that the exploration starting from a randomly chosen vertex x is stopped if $\min\{\mathtt{d}(a,b), \mathtt{d}(a,c), \mathtt{d}(b,c)\} \leq \delta_h$, where δ_h is the best solution found so far. It is also stopped if $\max\{\mathtt{d}(a,b), \mathtt{d}(a,c), \mathtt{d}(b,c)\} \leq 2\delta_h$ by Lemma 1. Furthermore, to ensure that $S_a \cap S_b$ is sufficiently large, we use a parameter ε allowing sets S_a and S_b to contain vertices at distance ℓ such that $\mathtt{d}(a,b)/2-\varepsilon \leq \ell \leq \mathtt{d}(a,b)/2+\varepsilon$ from a and b. This heuristic uses two integer arrays of size n to store the distances from a and b, and a third array of size n to store $S_a \cap S_b$. Distances from c are not stored. Overall, the space complexity of the heuristic is $O(n)$.

We have reported in Table 3 the results obtained with this heuristic on the largest biconnected component of the graphs of Section 4.3, and on the largest biconnected components of three road networks (`roadNet-*`) from [31]. For each graph, we have set the parameter k to $k=50$, and so we have repeated the process from 50 randomly chosen vertices and only considered 50 vertices among $S_a \cap S_b$. We have also set $\varepsilon = 1$. Column $\langle\delta_h\rangle$ indicates the average value over 100 executions of heuristic with this parameters, and column $\hat{\delta}_h$ the best value returned over all executions. Column T_h indicates the average running times.

To confirm the pertinence of choosing first a pair of vertices which are far from each other, along with a set of vertices at mid-distance, we compare our heuristic to a random heuristic. This heuristic selects uniformly at random three vertices a, b, and c, and then computes the

Instance name		n_B	D_B	δ	δ_s	$\hat{\delta}_h$	$\langle\delta_h\rangle$	$\hat{\delta}_r$	$\langle\delta_r\rangle$	T_h (in sec.)
CAIDA	`2004/01/05`	10 424	8	2.5	–	2	2	2	1.5	3.0
	`2004/06/07`	11 100	8	2	–	2	1.6	2	1.5	3.2
	`2005/09/05`	12 957	8	3	–	2.5	2.2	2	1.5	4.0
	`2010/01/20`	20 940	8	2	–	2	1.7	2	1.5	10.7
	`2011/01/16`	23 214	8	2	–	2	2	1.5	1.5	14.5
	`2012/01/01`	25 614	8	2	–	2	1.5	1.5	1.5	16.6
	`2012/06/01`	25 815	8	2	–	2	1.5	2	1.5	19.6
	`2013/01/01`	27 454	8	2.5	–	2.5	2.5	2	1.5	11.5
	`2013/06/01`	28 654	10	2.5	–	1.5	1.5	1.5	1.5	20.1
	`2013/11/01`	29 432	9	2.5	–	2	1.5	1.5	1.5	19.9
DIMES	`12/2010`	18 764	7	2	–	1.5	1.5	1.5	1.1	14.7
	`10/2011`	17 137	7	2	–	1.5	1.5	1.5	1.1	11.3
	`04/2012`	16 907	7	2	–	1.5	1.5	1.5	1.2	12.4
`loc-brightkite`		33 187	11	3	–	2.5	2.4	2	1.8	29.9
`ca-AstroPh`		15 929	10	3	2	2.5	2.5	2.5	2	39.8
`ca-CondMat`		17 234	12	3.5	2.5	3	3	2	2.3	14.9
`ca-GrQc`		2 651	11	3.5	3	3.5	3	3	2.6	1.1
`ca-HepPh`		9 025	11	3	2	3	3	2.5	2.1	12.7
`ca-HepTh`		5 898	11	4	3	3.5	3.3	3	2.6	2.5
`roadNet-PA`		863 105	793	–	195.5	166.5	158.5	158	143.8	329.2
`roadNet-TX`		1 050 434	1 063	–	222	225	203.3	221.5	205.4	371.9
`roadNet-CA`		1 563 362	862	–	208.5	220	218.7	223	202.9	582.4

Table 3: Comparison of our heuristic (column δ_h) with the sampling process of [27] (column δ_s) and a random heuristic (column δ_r)

value $\delta(a, b, c, d)$ for each vertex $d \in V \setminus \{a, b, c\}$. This process is repeated until the overall computation time equals the computation time of the first heuristic, and the largest computed value, δ_r, is returned. The resulting values are reported in Columns $\hat{\delta}_r$ and $\langle\delta_r\rangle$ of Table 3 (best value over 100 executions of the heuristic, and average value).

We compare our solutions to the values obtained in [27] with another sampling process. Roughly, the proposed heuristic samples 4-tuples until a certain level of confidence in the mean value is reached. Then, the largest computed value δ_s is returned. We have reported in Column δ_s of Table 3 the values available from [27]. However, there is no indication in [27] on the sample sizes or running times.

We have also reported in Table 3 for each graph the order n_B of its largest biconnected component, its diameter D_B, and the exact value δ of its hyperbolicity, when this value is known (i.e., we where able to compute it using Algorithm 1).

Let us first consider in Table 3 the graphs for which the exact value of δ is known. We observe that our heuristic was able to find the exact value of the hyperbolicity for 8 of them, and furthermore that for 3 of them the average value of the heuristic is also the exact one. The positive impact of the initial selection process of distant pairs in our heuristic can be seen when comparing the average values $\langle\delta_h\rangle$ with $\langle\delta_r\rangle$. Indeed, $\langle\delta_h\rangle$ is generally larger than $\langle\delta_r\rangle$. Our heuristic also finds on average larger values for the `ca-*` graphs compared to δ_s. Furthermore, these results have been obtained with small computation time compared to the results reported in Table 2 (recall that computations in Table 2 are performed on the LMA of the graphs which is smaller than the LBC). The running time on the `ca-AstroPh` graph is larger than for other graphs with similar numbers of nodes since this graph has average degree 24, which increases the number of visited edges during the BFS compared to the DIMES map `4/2012` which has average degree 8. For the CAIDA map `2013/01/01`, the running time is smaller than for similar maps due to a better usage of the pruning rules.

We now consider the road networks (`roadNet-*`) for which the exact value of the hyperbolicity is unknown. One very good result of our heuristic is that the average value $\langle\delta_h\rangle$ computed on `roadNet-CA` is larger than δ_s and $\langle\delta_r\rangle$. However 223, which is the best value, has been obtained using the random heuristic. This indicates that for this particular graph, the 4-tuples maximizing $\delta(a, b, c, d)$ might not involve pairs at largest possible distances, and so more diversity as offered by the random heuristic is needed. This result is somewhat mitigated when looking at `roadNet-PA` where $\hat{\delta}_h$ is significantly smaller than δ_s, and we are unable to explain this behavior. Observe however that our heuristic is better than the random one in this case, and furthermore that $\langle\delta_h\rangle > \hat{\delta}_r$ which indicates that the most promising 4-tuples are those involving pairs at large distances in this case. For `roadNet-TX`, we observe that the random heuristic provides on average slightly better results than our heuristic.

Overall, the results obtained with our heuristic on `RoadNet-*` graphs are good for a heuristic designed to target specific 4-tuples, based on observations done on CAIDA maps. These graphs have different structural properties like power-law degree distribution and small diameters compared to the `RoadNet-*` graphs which are nearly planar with a poisson degree distribution and very large diameters (see [27]). Moreover, we were able to find new bounds on the hyperbolicity of these graphs, improving [27]. Thus, this heuristic is a simple way to obtain good lower bounds on the hyperbolicity of very large graphs, since it has linear time and space complexity.

6 Conclusion

We have proposed a simple algorithm to compute the hyperbolicity of graphs that offers good computational performances, and we have analyzed the impact of the hyperbolicity of a graph and its distance distribution on the computation time of the algorithm. We have also shown how a decomposition by clique-separators can help reducing the overall computation time. Combining these methods with automatic parallelization tools, we are now able to compute efficiently the exact hyperbolicity of graphs with tens of thousands of nodes. We have also proposed a fast heuristic providing good lower bounds on the hyperbolicity of graphs with millions of nodes.

Our study opens the gates for further studies. Firstly, the speedups we have observed when executing our parallel version of Algorithm 1 on a computer equipped with 16 cores can certainly be improved. Indeed, a careful placement of data in memory would be helpful. Furthermore, the order in which instructions are executed, and so the order of the elements in the list `pairs`, could be refined. Secondly, the performances of the heuristic we have proposed can certainly be improved. In particular, a fast method cutting the search space when selecting the fourth node (node d) would speedup the whole process. Moreover, tools for evaluating the optimality gap (apart from the distance with the upper bound, that is $D_G/2$) would help deciding when to stop computations.

References

[1] A. B. Adcock, B. D. Sullivan, O. R. Hernandez, and M. W. Mahoney. Evaluating openmp tasking at scale for the computation of graph hyperbolicity. In *9th International Workshop on OpenMP (IWOMP)*, volume 8122 of *Lecture Notes in Computer Science*, pages 71–83, Canberra, Australia, 2013. Springer.

[2] H.-J. Bandelt and V. Chepoi. 1-hyperbolic graphs. *SIAM Journal on Discrete Mathematics*, 16(2):323–334, 2003.

[3] A. Berry, R. Pogorelcnik, and G. Simonet. An introduction to clique minimal separator decomposition. *Algorithms*, 3(2):197–215, 2010.

[4] M. Boguñá, F. Papadopoulos, and D. V. Krioukov. Sustaining the Internet with hyperbolic mapping. *Nature Communications*, 1(62):1–18, Oct. 2010.

[5] G. Brinkmann, J. H. Koolen, and V. Moulton. On the hyperbolicity of chordal graphs. *Annals of Combinatorics*, 5(1):61–69, 2001.

[6] J. Chakerian and S. Holmes. distory: Distance between phylogenetic histories, 2010.

[7] J. Chakerian and S. Holmes. Computational tools for evaluating phylogenetic and hierarchical clustering trees. *Journal of Computational and Graphical Statistics*, 21(3):581–599, 2012.

[8] P. Charbit, F. de Montgolfier, and M. Raffinot. Linear time split decomposition revisited. *SIAM Journal on Discrete Mathematics*, 26(2):499–514, 2012.

[9] V. Chepoi, F. F. Dragan, B. Estellon, M. Habib, and Y. Vaxès. Notes on diameters, centers, and approximating trees of δ-hyperbolic geodesic spaces and graphs. *Electronic Notes in Discrete Mathematics*, 31:231–234, 2008.

[10] V. Chepoi, F. F. Dragan, B. Estellon, M. Habib, Y. Vaxès, and Y. Xiang. Additive spanners and distance and routing labeling schemes for hyperbolic graphs. *Algorithmica*, 62(3-4):713–732, 2012.

[11] E. Cho, S. A. Myers, and J. Leskovec. Friendship and mobility: user movement in location-based social networks. In *ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1082–1090, San Diego, CA, 2011. ACM.

[12] N. Cohen, D. Coudert, G. Ducoffe, and A. Lancin. Applying clique-decomposition for computing Gromov hyperbolicity. Research Report RR-8535, Inria, May 2014.

[13] D. Coudert. Gromov hyperbolicity of graphs: C source code. http://www-sop.inria.fr/members/David.Coudert/code/hyperbolicity.shtml, May 2014.

[14] D. Coudert and G. Ducoffe. Recognition of C_4-free and 1/2-hyperbolic graphs. *SIAM Journal on Discrete Mathematics*, 28(3):1601–1617, Sept. 2014.

[15] W. H. Cunningham. Decomposition of directed graphs. *SIAM Journal on Algebraic Discrete Methods*, 3(2):214–228, 1982.

[16] W. H. Cunningham and J. Edmonds. A combinatorial decomposition theory. *Canadian Journal of Mathematics*, 32(3):734–765, 1980.

[17] F. de Montgolfier, M. Soto, and L. Viennot. Treewidth and hyperbolicity of the internet. In *10th IEEE International Symposium on Network Computing and Applications (NCA)*, pages 25–32, Boston, 2011. IEEE.

[18] A. Dress, K. Huber, J. Koolen, V. Moulton, and A. Spillner. *Basic Phylogenetic Combinatorics.* Cambridge University Press, Cambridge, UK, Dec. 2011.

[19] R. Duan. Approximation algorithms for the gromov hyperbolicity of discrete metric spaces. In *11th Latin American Theoretical Informatics Symposium (LATIN)*, volume 8392 of *Lecture Notes in Computer Science*, pages 285–293, Montevideo, Uruguay, 2014. Springer.

[20] R. Duan and S. Pettie. Fast algorithms for (max, min)-matrix multiplication and bottleneck shortest paths. In *20th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 384–391, New York, NY, USA, 2009. SIAM.

[21] H. Fournier, A. Ismail, and A. Vigneron. Computing the Gromov hyperbolicity of a discrete metric space. *Information Processing Letters*, 115(6-8):576–579, 2015.

[22] T. Gallai. Transitiv orientierbare graphen. *Acta Mathematica Hungarica*, 18(1):25–66, 1967.

[23] M. Gromov. Hyperbolic groups. In S. Gersten, editor, *Essays in Group Theory*, volume 8 of *Mathematical Sciences Research Institute Publications*, pages 75–263. Springer, New York, 1987.

[24] M. Habib and C. Paul. A survey of the algorithmic aspects of modular decomposition. *Computer Science Review*, 4(1):41–59, 2010.

[25] L. Hogie et al. *Grph, the high performance graph library for Java (Version 1.4.17).* The Grph Development Team, 2014.

[26] E. A. Jonckheere and P. Lohsoonthorn. Geometry of network security. In *American Control Conference*, volume 2, pages 976–981, Boston, MA, USA, 2004. IEEE.

[27] W. S. Kennedy, O. Narayan, and I. Saniee. On the hyperbolicity of large-scale networks. Technical Report arXiv:1307.0031, ArXiv, 2013.

[28] J. H. Koolen and V. Moulton. Hyperbolic bridged graphs. *European Journal of Combinatorics*, 23(6):683–699, 2002.

[29] D. V. Krioukov, F. Papadopoulos, M. Kitsak, A. Vahdat, and M. Boguñá. Hyperbolic geometry of complex networks. *Physical Review E*, 82(3):036106, Sept. 2010.

[30] J. Leskovec, J. Kleinberg, and C. Faloutsos. Graph evolution: Densification and shrinking diameters. *ACM Transactions on Knowledge Discovery from Data*, 1(1):1–41, Mar. 2007.

[31] D. Lokshtanov. Finding the longest isometric cycle in a graph. *Discrete Applied Mathematics*, 157(12):2670–2674, 2009.

[32] C. Magnien, M. Latapy, and M. Habib. Fast computation of empirically tight bounds for the diameter of massive graphs. *ACM Journal of Experimental Algorithmics*, 13:10:1.10–10:1.9, Feb. 2009.

[33] O. Narayan and I. Saniee. The large scale curvature of networks. *Physical Review E*, 84:066108, Dec. 2011.

[34] O. Narayan, I. Saniee, and G. H. Tucci. Lack of hyperbolicity in asymptotic Erdös-Renyi sparse random graphs. *Internet Mathematics*, 11(3):277–288, 2015.

[35] D. Noguès. δ-hyperbolicité et graphes. Master's thesis, MPRI, Université Paris 7, 2009.

[36] OpenMP Architecture Review Board. OpenMP API version 3.0, May 2008.

[37] F. Papadopoulos, D. V. Krioukov, M. Boguñá, and A. Vahdat. Greedy forwarding in scale-free networks embedded in hyperbolic metric spaces. *ACM SIGMETRICS Performance Evaluation Review*, 37(2):15–17, Sept. 2009.

[38] R Core Team. *R: A Language and Environment for Statistical Computing.* R Foundation for Statistical Computing, Vienna, Austria, 2014.

[39] V. Ramasubramanian, D. Malkhi, F. Kuhn, M. Balakrishnan, A. Gupta, and A. Akella. On the treeness of Internet latency and bandwidth. *ACM SIGMETRICS Performance Evaluation Review*, 37(1):61–72, June 2009.

[40] Y. Shang. Lack of Gromov-hyperbolicity in colored random networks. *PanAmerican Mathematical Journal*, 21(1):27–36, 2011.

[41] Y. Shavitt and E. Shir. Dimes: Let the internet measure itself. *ACM SIGCOMM Computer Communication Review*, 35(5):71–74, Oct. 2005.

[42] M. A. Soto Gómez. *Quelques propriétés topologiques des graphes et applications à internet et aux réseaux.* PhD thesis, Univ. Paris Diderot (Paris 7), 2011.

[43] W. A. Stein et al. *Sage Mathematics Software (Version 6.1).* The Sage Development Team, 2014.

[44] R. E. Tarjan. Depth-first search and linear graph algorithms. *SIAM Journal on Computing*, 1(2):146–160, 1972.

[45] R. E. Tarjan. Decomposition by clique separators. *Discrete Mathematics*, 55(2):221–232, 1985.

[46] The Cooperative Association for Internet Data Analysis (CAIDA). The CAIDA AS relationships dataset. `http://www.caida.org/data/active/as-relationships/`, 2013.

[47] Y. Wu and C. Zhang. Hyperbolicity and chordality of a graph. *The Electronic Journal of Combinatorics*, 18(1):P43, 2011.