Dominer pour calculer l’hyperbolicité des graphes
David Coudert, André Nusser, Laurent Viennot

To cite this version:
David Coudert, André Nusser, Laurent Viennot. Dominer pour calculer l’hyperbolicité des graphes. AlgoTel 2022 - 24èmes Rencontres Francophones sur les Aspects Algorithmiques des Télécommunications, May 2022, Saint-Rémy-Lès-Chevreuse, France. hal-03648264v2

HAL Id: hal-03648264
https://hal.archives-ouvertes.fr/hal-03648264v2
Submitted on 21 Apr 2022

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
Hyperbolicity is a graph parameter introduced by Gromov [Gro87] to measure how different the distances in a graph are to distances in a tree. It is used to classify complex networks and to design efficient routing schemes as it provides information on the dispersion of the shortest paths (distance between the vertices of two shortest paths from s to t). Hyperbolicity is usually defined through a 4-points condition that associates to each quadruple a δ-value defined through distances between the four nodes (see Section 2). The hyperbolicity of a graph is the maximum of the δ-values over all quadruples in the graph. This definition trivially results in a Θ(n^4) algorithm for computing hyperbolicity, and the best known theoretical complexity is O(n^3.69), relying on an optimized (max,min)-matrix product [FIV15]. However, the algorithms exhibiting the best performances in practice have time complexity in O(n^4) [BCCM15, CCL15, CNV21a].

The quest for computing the hyperbolicity of large graphs has led to several breakthroughs in the last years. Indeed, the first attempts are based on brute force implementations of the trivial Θ(n^4) algorithm enabling to compute the hyperbolicity of graphs with few hundreds of nodes on a single core [CH12] and up to 8 000 nodes using massive parallelism (up to 1 000 cores) [ASHM13]. The first noticeable progress is due to [CCL15] which introduces pruning techniques to drastically reduce the number of quadruples to consider. With the addition of refined pruning techniques [BCCM15], it enables to compute the hyperbolicity of graphs with up to 50 000 nodes. Furthermore, this algorithm is orders of magnitude faster using a single core than the running times reported in [ASHM13]. However, this algorithm reaches a memory bottleneck as it has space complexity in O(n^2). To go beyond this bottleneck, [CNV21a] engineered an algorithm, along with suitable data structures, that consumes significantly less memory while offering good performances in practice. It enables to compute the hyperbolicity of graphs with more than 100 000 nodes. Nonetheless, the memory usage of this algorithm is still high, which limits its scalability. In this paper (extended abstract of [CNV22]), we propose a new approach that uses a hierarchy of distance-k dominating sets to both reduce memory usage and further prune the search space, resulting in an algorithm enabling to compute for the first time the hyperbolicity of graphs with up to a million nodes.
2 Definitions and notations

We consider only finite, connected, unweighted and simple undirected graphs. However, the results presented in this paper extend easily to weighted graphs. The graph \( G = (V, E) \) has \( n = |V| \) vertices and \( m = |E| \) edges. Given two vertices \( u \) and \( v \), a \( uv \)-path of length \( \ell \geq 0 \) is a sequence of vertices \( (u = v_0, v_1, \ldots, v_{\ell} = v) \) such that \( \{v_i, v_{i+1}\} \) is an edge for every \( i \). In particular, a graph \( G \) is connected if there exists a \( uv \)-path for all pairs \( u, v \in V \), and in such a case the distance \( d(u, v) \) is defined as the minimum length of a \( uv \)-path in \( G \).

**Domination.** Given an integer \( k > 0 \), we say that a node \( u \) \( k \)-dominates a node \( v \) if \( d(u, v) \leq k \). We define a \( k \)-dominating set \( D \) as a set \( D \subseteq V \) of nodes such that any node \( v \in V \) is \( k \)-dominated by some node \( u \in D \). Given a \( k \)-dominating set \( D \), we associate to any node \( v \in V \) such a \( k \)-dominating node \( D(v) \) in \( D \), and \( D(v) \) is called the associated dominator of \( v \). We denote \( D^{-1}(u) \) as the set of vertices that are \( k \)-dominated by \( u \in D \), i.e., \( D^{-1}(u) = \{ v \in V : D(v) = u \} \). For each node \( u \in D \), we define its domination radius as \( k_u = \max_{v \in D^{-1}(u)} d(u, v) \). When a node \( v \) is \( k \)-dominated by several nodes of \( D \), the choice of its associated dominator \( D(v) \) can be arbitrary. However, we will typically choose \( D(v) \) as the closest node to \( v \) in \( D \) as a heuristic to obtain smaller values of \( k_u \) for \( u \in D \).

**Hyperbolicity.** This notion has been introduced to measure how the shortest-path metric space \((V, d)\) of a connected graph \( G = (V, E) \) deviates from a tree metric when its vertices are mapped to the vertices of an edge-weighted tree. This additive stretch of the distances, denoted \( \delta \), is called the hyperbolicity of the graph and is said to be \( \delta \)-hyperbolic if it satisfies the 4-point condition below.

**Definition 1 (4-points Condition, [Gro87])** Let \( G \) be a connected graph. For every quadruple \( u, v, x, y \) of vertices of \( G \), we define \( \delta(u, v, x, y) \) as half of the difference between the two largest sums among \( S_1 = d(u, v) + d(x, y), S_2 = d(u, x) + d(v, y), \) and \( S_3 = d(u, x) + d(v, x) \).

The hyperbolicity of \( G \), denoted by \( \delta(G) \), is equal to \( \max_{u, v, x, y \in V(G)} \delta(u, v, x, y) \). Moreover, given a value \( \delta \), we say that \( G \) is \( \delta \)-hyperbolic whenever \( \delta(G) \leq \delta \).

Note that if \( G \) is a tree or a clique, we have \( \delta(G) = 0 \). If \( G \) is a cycle of order \( n = 4p + \varepsilon \), with \( p \geq 1 \) and \( 0 \leq \varepsilon < 4 \), then \( \delta(G) = p - 1/2 \) if \( \varepsilon = 1 \), and \( \delta(G) = p \) otherwise. If \( G \) is an \( n \times m \) grid, with \( 2 \leq n \leq m \), then we have \( \delta(G) = n - 1 \). Other definitions of hyperbolicity have been proposed [BRSV13, dLHG90, Gro87] and differ only by a small constant factor.

3 Approach

We first show in Lemma 1 how to obtain an additive \( 4k \) approximation of hyperbolicity from a \( k \)-dominating set, and then show how to use it to prune the search space and design an exact algorithm.

**Lemma 1 ([CNV22])** Given a \( k \)-dominating set \( D \) of \( G \) and a quadruple \( u', v', x', y' \in V \) with respective associated dominators \( u, v, x, y \in D \), we have \( \delta(u, v, x, y) - K_4 \leq \delta(u', v', x', y') \leq \delta(u, v, x, y) + K_4 \) where \( K_4 = k_u + k_v + k_x + k_y \leq 4k \).

**Algorithm.** We can now present an exact algorithm that exploits the notion of \( k \)-domination to prune the search space and significantly reduce the memory usage compared to the algorithms proposed in [BCCM15, CCL15, CNV21a]. It takes as input a connected graph \( G \) and a sequence \( k_t, k_{t-1}, \ldots, k_0 \) of domination distances, with \( i \geq 0 \) and \( k_i = k > k_{i-1} > \cdots > k_0 = 0 \). The main idea is to use nested dominating sets \( D_t, \ldots, D_0 = V \) with respective domination distances \( k_t, \ldots, k_0 \) for exploring in a recursive manner the quadruples of the graph while maintaining a lower bound \( \delta_L \) on \( \delta(G) \) based on the quadruples scanned so far. This lower bound is used in accordance with Lemma 1 to prune the exploration, that is, skip quadruples for which we can infer that their \( \delta(L) \) value is \( \delta_L \) at most.

More precisely, the algorithm first starts by scanning the quadruples in the coarsest dominating set \( D_t \) (with largest domination distance \( k_t \)). Then, thanks to Lemma 1, we know that if a quadruple \( u, v, x, y \in D_t \) is such that \( \delta(u, v, x, y) + 4k_t \leq \delta_L \), then all quadruples \( u', v', x', y' \) it dominates, that is such that \( D(u') = u \), \( D(v') = v \), \( D(x') = x \) and \( D(y') = y \), must satisfy \( \delta(u', v', x', y') \leq \delta_L \) and can be skipped. Otherwise, if \( \delta(u, v, x, y) + 4k_t > \delta_L \), then \( u, v, x, y \) may dominate a quadruple \( u', v', x', y' \) such that \( \delta(u', v', x', y') > \delta_L \). We thus start a recursive exploration of the quadruples dominated by \( u, v, x, y \) as follows. We scan those
that are in the next level dominating set \(D_{i-1}\). For that purpose, we use for each node \(w \in D_i\) the pre-computed list \(D_{i-1,w}\) of nodes it dominates in \(D_{i-1}\). We then similarly skip the quadruples \(u', v', x', y'\) for which \(\delta(u', v', x', y') + 4k_{i-1} \leq \delta_{i-1}\). Otherwise, we proceed recursively for smaller and smaller domination radii \(k_j\) with \(0 < j < i\) as long as the condition of Lemma 1 requiring further exploration is satisfied for \(k_j\). For that purpose, similar pre-computed lists for lower levels are stored. We also use other lemmas, generalizing to some extent the approach of [BCCM15], and \(K_4\) instead of \(4k\) as in Lemma 1, to prune even more quadruples. These lemmas are omitted for the sake of brevity and can be found in [CNV22].

4 Experimental evaluation

We consider web graphs (NotreDame, web-BerkStan, web-Stanford), road networks (t.CAL, t.FLA, roadNet-PA), a 3D triangular mesh (buddha), a graph from a computer game (froz), and a grid-like graph from VLSI (z-value7065). We also use synthetic graphs (grid300-10, grid500-10) which are square grids with respective sides 301 and 501 where 10% of the edges have been randomly deleted. Each graph is taken as an undirected unweighted graph and we consider only its largest biconnected component. The data is available from [CNV21b]. The chosen graphs have a large number of nodes compared to the graph sizes that were feasible for previous algorithms. Furthermore, our approach relies on the fact that pruning of quadruples is actually possible. Thus, the graphs that we consider do not exhibit a very low hyperbolicity — the lowest is 8 (NotreDame).

To evaluate the improvement of our algorithm over previous work, we compare to [CNV21a], which was shown to outperform the algorithm of [BCCM15]. See [CNV21a, Figures 1 and 2] for a comparison of the running time and memory consumption of [BCCM15] and [CNV21a]. In particular, the memory consumption of [BCCM15] is prohibitive for all the graph sizes considered in this work except for the graph z-value7065, which we mainly use to conduct experiments with different parameter choices.

The code of the algorithms (in C++) is available online [CNV21b]. Important implementation details are given in [CNV22]. For instance, we use a hub labeling of the graph to answer distance queries [AIY13] and we cache some distances in small matrices to reduce the number of queries to the hub labeling. We define the sequence of domination distances of the hierarchy of dominating sets using two parameters: the largest consumption of [BCCM15] is prohibitive for all the graph sizes considered in this work except for the graph z-value7065, which we mainly use to conduct experiments with different parameter choices.

We use a computer equipped with Intel Xeon Gold 6240 CPUs operating at 2.6 GHz and 192 GB RAM. All computations were conducted using a single thread. Running times reported in Table 1 include all steps of the program, from reading the data to returning the result. A time limit of 60 hours and a memory limit of 192 GB was set for the algorithm of [CNV21a] as it would not terminate anyway for large instances.

The most noticeable points on the experiments reported in Table 1 are:

- For NotreDame, web-Stanford and web-BerkStan, we reduce the memory consumption compared to [CNV21a] by factors of 16.2, 7.2, and 23.1. The running time is reduced by factors of 18.1, 549.9, and 1104, i.e., up to 3 orders of magnitude.
- Computing the hyperbolicity of t.CAL, t.FLA-w, roadNet-PA, buddha and froz was not feasible using previous algorithms but it can be computed using our approach. Especially, we are able to compute the hyperbolicity of a real-world graph with more than a million nodes for the first time. We want to highlight that the memory consumption of our algorithm is below 25 GB for all graphs except buddha. Even in the cases where the algorithm of [CNV21a] hits the time limit of 60 hours and not the memory limit, increasing the time limit most probably does not make these graphs attainable, as the lower and upper bounds are still very far from matching.
- Our algorithm fails to outperform previous work in grid-like graphs (grid300-10, grid500-10, z-value7065). This is probably due to the notion of far-apart pairs used in [BCCM15, CNV21a], which we do not use in our approach. Computing the hyperbolicity iterating over far-apart pairs is very fast on grid-like graphs as they contain only very few far-apart pairs — a perfect grid actually just contains two far-apart pairs, which also form the quadruple whose \(\delta\)-value yields its hyperbolicity.
Table 1: Experiments on various graphs using algorithm [CNV21a] with time limit of 60h (which is 216,000 seconds) and memory limit of 192GB and the algorithm proposed in this paper. Skulls (-country-) indicate that the time or memory limit was reached before terminating, in which case we report the best found lower and upper bounds on hyperbolicity.

<table>
<thead>
<tr>
<th>Graph</th>
<th># nodes</th>
<th># edges</th>
<th>Algorithm [CNV21a]</th>
<th>This paper</th>
</tr>
</thead>
<tbody>
<tr>
<td>NotreDame</td>
<td>134958</td>
<td>833732</td>
<td>4514  53.02 GB 8.0</td>
<td>249 3.37 GB 8.0 2 2</td>
</tr>
<tr>
<td>web-Stanford</td>
<td>181906</td>
<td>1676077</td>
<td>8249  23.28 GB 23.0</td>
<td>15 3.22 GB 23.0 8 2</td>
</tr>
<tr>
<td>web-BerkStan</td>
<td>489296</td>
<td>5939242</td>
<td>65134 76.93 GB 23.0</td>
<td>59 3.33 GB 23.0 4 4</td>
</tr>
<tr>
<td>t.CAL</td>
<td>1267004</td>
<td>1671989</td>
<td>29358</td>
<td>[379.0, 1025.5]</td>
</tr>
<tr>
<td>t.FLA</td>
<td>691175</td>
<td>941893</td>
<td>143.3 GB [81.0, 818.0]</td>
<td>1199907 18.04 GB 229.5 25 1.5</td>
</tr>
<tr>
<td>roadNet-PA</td>
<td>863105</td>
<td>1313732</td>
<td>148.2 GB [109.0, 370.5]</td>
<td>1357512 23.32 GB 170.5 20 1.5</td>
</tr>
<tr>
<td>buddha</td>
<td>543652</td>
<td>1631574</td>
<td>88.35 GB [93.0, 211.5]</td>
<td>134421 52.84 GB 112.0 8 1.5</td>
</tr>
<tr>
<td>froz</td>
<td>749520</td>
<td>2895228</td>
<td>106.4 GB [387.5, 599.0]</td>
<td>160111 11.74 GB 401.5 27 1.5</td>
</tr>
<tr>
<td>grid300-10</td>
<td>90211</td>
<td>162152</td>
<td>10 1.08 GB 280.0</td>
<td>23 5.28 GB 280.0 10 1.5</td>
</tr>
<tr>
<td>grid500-10</td>
<td>250041</td>
<td>449831</td>
<td>95 2.99 GB 463.0</td>
<td>98 6.14 GB 463.0 10 2</td>
</tr>
<tr>
<td>z-alue7065</td>
<td>34040</td>
<td>54835</td>
<td>33 431.18 MB 138.0</td>
<td>1927 3.48 GB 138.0 2 2</td>
</tr>
</tbody>
</table>

References


