On Computing the Diameter of Real-World Undirected Graphs[☆]

P. Crescenzi[a,*], R. Grossi[b], M. Habib[c], L. Lanzi[a], A. Marino[a]

[a] *Dipartimento di Sistemi e Informatica, Università degli Studi di Firenze, Viale Morgagni 65, 50134 Firenze, Italy*
[b] *Dipartimento di Informatica, Università degli Studi di Pisa, Largo Bruno Pontecorvo 3, 56127 Pisa, Italy*
[c] *LIAFA, University Paris Diderot - Paris7, 175 rue du Chevaleret, 75013 Paris, France*

Abstract

In this paper we propose a new algorithm for computing the diameter of undirected unweighted graphs. Even though, in the worst case, this algorithm has complexity $O(nm)$, where n is the number of nodes and m is the number of edges of the graph, we experimentally show (on almost 200 real-world graphs) that in practice our method works in linear time. Moreover, we show how to extend our algorithm to the case of undirected weighted graphs and, even in this case, we present some preliminary very positive experimental results.

Keywords: breadth-first search, diameter, complex network

1. Introduction

This paper addresses the diameter computation problem in the case of undirected unweighted graphs, where the diameter D is defined as the maximum distance among all the pairs of nodes (the distance $d(u, v)$ between two nodes u and v is defined as the number of edges contained in the shortest path from u to v). In the context of real world networks, the textbook method, based on performing a breadth-first search (in short, BFS) from any node of the graph, requires a prohibitive cost of $O(nm)$ time, where n is the number of nodes and m is the number of edges of the graph: indeed, it is not rare that a real-world graph contains several millions of nodes and several millions of edges. Even more efficient methods, like the ones presented in [1, 2], turn out to be too much time consuming (for a comprehensive overview of results concerning distance and diameter computation, we refer to [3]).

Some simpler algorithms have also been recently proposed (see, for example, [4, 5]). These algorithms perform a sampling of the eccentricity of the nodes of the graph, by executing a fixed number of random BFSes (the eccentricty $ecc(u)$ of a node u is equal to $\max_v d(u, v)$): unfortunately, no useful bound on the performed error can be provided and even experimentally these algorithms turn out to be not always precise.

[☆]This research was supported in part by funds of the following projects or institutions: "DISCO" PRIN (National Research Project), "AlgoDEEP" PRIN (National Research Project), Florence section of INFN (National Institute of Nuclear Physics).

[*]Corresponding author
Email address: pierluigi.crescenzi@unifi.it (L. Lanzi)

The main contribution of this paper consists of showing that BFS *can indeed be an extremely powerful tool in order to compute the exact value of the diameter, whenever it is used in a more clever way.*

In particular, we present the *i*FUB (iterative fringe upper bound) algorithm to calculate the exact value of the diameter. This algorithm uses as a subroutine a lower bound computation method, called 4-SWEEP, which is a natural extension of the method described in [6, 7]. The *i*FUB algorithm, which is a generalization of the FUB method described in [8], uses the lower bound value computed by the 4-SWEEP algorithm in order to start an iterative procedure whose goal is both to refine the lower bound value itself and to properly tune an upper bound value, until the two values coincide (or are within a specified distance).

In the worst case, the time complexity of the *i*FUB algorithm is equal to the complexity of the textbook algorithm, that is, $O(nm)$. However, by performing BFSes in a specified "good" order, it turns out that we don't need to perform all the BFSes as in the case of the exhaustive method. This phenomenon is quite impressively evident in the case of real-world graphs: indeed, in these cases our method requires only few BFSes, so that its time complexity is, in practice, linear in the number of edges. In order to support this statement, we tested the algorithm on a dataset containing about 200 real-world graphs which cover several different application areas (including also the graphs used to validate several popular tools, such as the ones described in [9, 10, 11, 12]). Moreover, we also experimented the algorithm on synthetic graphs created by using some of the most well-known models for generating random complex networks. Finally, we adapted the *i*FUB algorithm in order to deal with undirected weighted graphs and we experimented this modification on a dozen of real-world weighted networks: even in this case, the performance of the *i*FUB approach is quite impressive. We implemented all our algorithms in the C language: the source code, the dataset, the logs of the experiments, and other tools for file format conversion, are available at our website `diameter.algoritmica.org`.

The paper is organized as follows. Section 2 describes the *i*FUB algorithm and analyses its complexity, while Section 3 shows some possible *ad hoc* bad cases. In Section 4 we describe the dataset, while in Section 5 we report the results of our experiments. In Section 6 we briefly present the modification of the *i*FUB algorithm in order to deal with undirected weighted graphs and we report some preliminary experimental results. We conclude in Section 7 by proposing some open questions.

2. The iterative fringe upper bound algorithm

Let $G = (V, E)$ be an undirected unweighted graph and let u be any node in V. We denote by T_u a breadth-first search (in short, BFS) tree rooted at node u, by $\mathrm{ecc}(u)$ the height of T_u (that is, the eccentricity of u), and by $F(u)$ the set of nodes whose distance from u is $\mathrm{ecc}(u)$. Moreover, let $F_i(u)$ be the *fringe* set of nodes whose distance from u is equal to i (note that $F(u) = F_{\mathrm{ecc}(u)}(u)$) and let $B_i(u) = \max_{z \in F_i(u)} \mathrm{ecc}(z)$. Observe that, for any x and y such that $x \in F_j(u)$ or $y \in F_j(u)$, we have that $d(x, y) \leq B_j(u)$: indeed,

ALGORITHM 1: *i*FUB

Input: A graph G, a node u, a lower bound l for the diameter, and an integer k
Output: A value M such that $D - M \leq k$

```
i ← ecc(u);
lb ← max{ecc(u), l};
ub ← 2ecc(u);
while ub − lb > k do
    if max{lb, B_i(u)} > 2(i − 1) then
        return max{lb, B_i(u)};
    else
        lb ← max{lb, B_i(u)};
        ub ← 2(i − 1);
    end
    i ← i − 1;
end
return lb;
```

$d(x, y) \leq \min\{\text{ecc}(x), \text{ecc}(y)\} \leq B_j(u)$. Observe also that, for any i and j with $1 \leq i, j \leq \text{ecc}(u)$ and for any x and y such that $x \in F_i(u)$ and $y \in F_j(u)$, $d(x, y) \leq i + j \leq 2\max\{i, j\}$.

Theorem 1. *For any i with $1 \leq i < \text{ecc}(u)$, for any k with $1 \leq k < i$, and for any $x \in F_{i-k}(u)$ such that $\text{ecc}(x) > 2(i-1)$, there exists $y_x \in F_j(u)$ such that $d(x, y_x) = \text{ecc}(x)$ with $j \geq i$.*

Proof. Since $\text{ecc}(x) > 2(i-1)$, then there exists y_x whose distance from x is greater than $2(i-1)$. If y_x was in $F_j(u)$ with $j < i$, then from the previous observation it would follow that $d(x, y_x) \leq 2\max\{i-k, j\} \leq 2\max\{i-k, i-1\} = 2(i-1)$, which is a contradiction. Hence, y_x must be in $F_j(u)$ with $j \geq i$. □

The previous result implies that if y is a node in $F_i(u) \cup F_{i+1}(u) \cup \ldots \cup F_{\text{ecc}(u)}(u)$ with maximum eccentricity and if $\text{ecc}(y) > 2(i-1)$, then the eccentricity of all nodes in $F_1(u) \cup F_2(u) \cup \ldots \cup F_{i-1}(u)$ is not greater than $\text{ecc}(y)$. This suggests to visit T_u in a bottom-up fashion, starting from the nodes in $F(u)$. At each level i, we can compute the eccentricities of all its nodes: if the maximum eccentricity e is greater than $2(i-1)$ then we can discard visiting the remaining levels, since the eccentricities of all their nodes cannot be greater than e. This idea suggests the following algorithm.

- Set $i = \text{ecc}(u)$ and $M = B_i(u)$.
- If $M > 2(i-1)$, then return M, else set $i = i - 1$ and $M = \max\{M, B_i(u)\}$, and repeat this step.

Note that at each iteration of the previous algorithm, we can also update a lower and an upper bound on the diameter. Indeed, the diameter is always greater than or equal to M (since M is always the maximum among a set of eccentricities) and it is always less than or equal to $2(i-1)$ (because of Theorem 1). This leads us to Algorithm 1 which receives as inputs the node u, an already known lower bound l, and a precision requirement k. The algorithm initializes the lower bound by setting it equal to the maximum between the already known one (that is, l) and the trivial bound obtained by performing the BFS starting from u (that is,

ALGORITHM 2: 4-SWEEP

Input: A graph G
Output: A lower bound for the diameter of G and a node with (hopefully) low eccentricity
$r_1 \leftarrow$ random node of G;
$a_1 \leftarrow argmax_{v \in V} d(r_1, v)$;
$b_1 \leftarrow argmax_{v \in V} d(a_1, v)$;
$r_2 \leftarrow$ the node in the middle of the path between a_1 and b_1;
$a_2 \leftarrow argmax_{v \in V} d(r_2, v)$;
$b_2 \leftarrow argmax_{v \in V} d(a_2, v)$;
$u \leftarrow$ the node in the middle of the path between a_2 and b_2;
$lowerb \leftarrow \max\{\text{ecc}(r_1), \text{ecc}(a_1), \text{ecc}(r_2), \text{ecc}(a_2)\}$;
return $lowerb$ and u;

ecc(u)). Moreover, it initializes the upper bound by setting it equal to the trivial one obtained by performing the BFS starting from u (that is, 2ecc(u)).

The running time of the algorithm is in the worst case $O(nm)$, as in the case of the exhaustive algorithm, since, in the worst case, we have to perform a BFS starting from "almost" every node of the graph (in Section 3 we will show some *ad hoc* cases in which this happens, while in Section 5 we will show that, in the case of real world graphs, by choosing the starting node u as described below this never happens). Indeed, observe that, at each iteration of the **while** loop, $ub - lb$ decreases at least by 2: this implies that the algorithm executes at most $\lceil \text{ecc}(u)/2 \rceil$ iterations (note that, since for any node u, ecc(u) $\leq D$, we have that the number of iterations is bounded by $D/2$).

This observation suggests that the starting node u affects the performance of *i*FUB. The 4-SWEEP method (see Algorithm 2) is a heuristic to find a good starting node u (that is, one with a low eccentricity), which requires always 4 BFSes. Let r_1 be a random node, let a_1 be a farthest node from r_1, and let b_1 be a farthest node from a_1; if r_2 is the node halfway between a_1 and b_1, then we define analogously a_2 and b_2. The starting node u is then defined as the middle node of the path between a_2 and b_2. Intuitively, u is a "center" of G, that is, a node whose eccentricity is close to the radius R of G, which is defined as the minimum eccentricity: as we will see in Section 5, in the case of real world graphs, $R \approx D/2$. Surprisingly, we will also see that this choice of u also provides, in the case of real world graphs, two other important features: a diametral node is always included in one of the first few fringe sets of u and these fringe sets are relatively small. Finally, let us note that, as shown in Algorithm 2, this selection of u also allows us to compute an initial lower bound l equal to $\max\{\text{ecc}(r_1), \text{ecc}(a_1), \text{ecc}(r_2), \text{ecc}(a_2)\}$.

3. Theoretical negative results

In this section we show that there exists an infinite family of graphs for which the 4-SWEEP algorithm does not compute the exact value of the diameter and that there exists an infinite family of graphs for which the *i*FUB algorithm requires $O(nm)$ time. To this aim, let us first note that, for any graph $G = (V, E)$ of diameter $D > 1$ and for any node $u \in V$, the graph $G' = (V \cup \{u'\}, E \cup E')$, where, for any $v \in V$, $(u', v) \in E'$

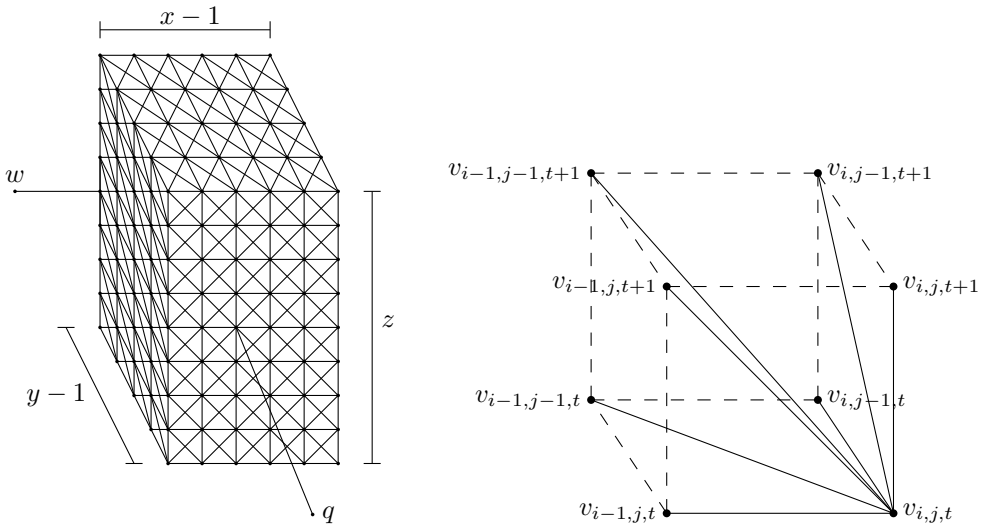


Figure 1: A bad network for the 4-Sweep algorithm (left) and a zoomed detail (right).

if and only if $(u, v) \in E$, has also diameter D: indeed, the only new distance which is introduced in G' is the one between u and u', which is equal to 2 and, hence, not greater than D. We will make use of this observation in the following bad graph constructions.

Bad graphs for 4-Sweep. In Section 5, we will show how we have experimentally found that the lower bound returned by one execution of the 4-Sweep algorithm coincides with the diameter in a large majority of the real-world graphs: however, there are very few cases in which this value does not coincides with the diameter, even if the algorithm is executed ten times (see Table 1). Indeed the idea of the counterexample presented in [8] can be generalized in order to obtain an example with an arbitrary diameter value in which the 4-Sweep algorithm computes a wrong estimate with high probability.

To this aim we define a graph $G = (V, E)$ (see Figure 1) in which $V = \{v_{i,j,t} : 1 \leq i \leq x-1,\ 1 \leq j \leq y-1,\ 1 \leq t \leq z\} \cup \{w, q\}$ and E is defined as follows. For any two nodes $v_{i,j,t}$ and $v_{i',j',t'}$, $(v_{i,j,t}, v_{i',j',t'}) \in E$ if and only if $\max\{|i-i'|, |j-j'|, |t-t'|\} = 1$ (see the right part of Figure 1). Moreover the node w is linked to $v_{1,1,(z+1)/2}$ and the node q is linked to $v_{(x+1)/2,y-1,(z+1)/2}$.

Observe that for any node $v_{i,j,t}$, with $1 \leq i \leq x-1$, $1 \leq j \leq y-1$, and $1 \leq t \leq z$, we have that, for any other node $v_{i',j',t'}$, with $1 \leq i' \leq x-1$, $1 \leq j' \leq y-1$, and $1 \leq t' \leq z$, $d(v_{i,j,t},\ v_{i',j',t'}) = \max\{|i-i'|, |j-j'|, |t-t'|\} \leq \max\{x-2, y-2, z-1\}$, and that $d(v_{i,j,t}, w) = d(v_{i,j,t}, v_{1,1,(z+1)/2})+1 \leq \max\{x-2, y-2, (z-1)/2\}+1$, while $d(v_{i,j,t}, q) = d(v_{i,j,t}, v_{(x+1)/2,y-1,(z+1)/2})+1 \leq \max\{(x-1)/2, y-2, (z-1)/2\}+1$. Moreover $d(w, q) = d(v_{1,1,(z+1)/2}, v_{(x+1)/2,y-1,(z+1)/2}) + 2 = \max\{(x-1)/2, y-2\} + 2$. Thus the diameter of G is $\max\{x-1, y, z-1\}$.

If x, y, and z are such that $z > x > y+1 > 3$, the diameter is equal to $z-1$. Moreover if x, y, and z

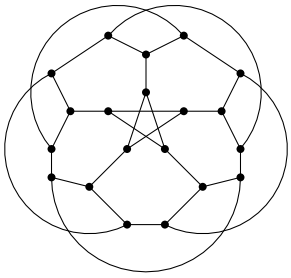


Figure 2: A bad network for the *i*FUB algorithm.

are odd and such that $y-1 > (z+1)/2$, $y-1 > (x+1)/2$, and $x-1 > (z+1)/2$, then the lower bound computed by the 4-SWEEP algorithm can be $x-1$ instead of $z-1$. Indeed, referring to Algorithm 2, the following computation might be performed.

- $r_1 = v_{x-1,1,(z+1)/2}$.
- $a_1 = w$ since $x-1 > (z+1)/2-1$ and $x-1 > y-1$.
- $b_1 = v_{x-1,1,(z+1)/2}$ since $x-1 > (z+1)/2$ and $x-1 > y$.
- $r_2 = v_{(x+1)/2,1,(z+1)/2}$, that is the node opposite to q with respect to the x axis.
- $a_2 = q$ since $y-1 > (z+1)/2-1$ and $y-1 > (x+1)/2-1$.
- $b_2 = w$, since $y > (z+1)/2$ and $y > (x+1)/2$.

Thus $\mathrm{ecc}(r_1) = \mathrm{ecc}(a_1) = x-1$, $\mathrm{ecc}(r_2) = y-1$, and $\mathrm{ecc}(a_2) = y$. Hence the lower bound computed by the 4-SWEEP algorithm is equal to $\max\{x-1, y-1, y\} = x-1$. Observe that the approximation ratio of the value returned by the algorithm is asymptotically close to 2, that is the same ratio obtained by performing a BFS and returning the diameter of the corresponding tree.

The example can be modified in order to make the starting bad choice more probable, by creating several copies of the node $v_{x-1,1,(z+1)/2}$. Moreover, it can be easily generalized to higher dimensions, in order to become a *bad graph* for natural extensions of the 4-SWEEP algorithm, such as the $2k$-SWEEP algorithm, for $k > 2$.

*Bad graphs for i*FUB. In [8], it was observed that there exist graphs (see graphs available at [13]), such as the one shown in Figure 2, such that, for any node u, the size of the fringe $F(u)$ is linear in n. This immediately implies that, in the case of these graphs, the complexity of *i*FUB is $O(nm)$. In the previous section we have already observed that, when we apply *i*FUB, the number of iterations depends on the choice of the starting node u. In particular, let R be the radius of the graph, and let C be the center of the graph (that is, the set of nodes v for which $\mathrm{ecc}(v)$ is equal to R). Then, the minimum number of iterations performed by *i*FUB

is obtained whenever $u \in C$: in this case, indeed, the upper bound on the iterations is minimum and equal to $R - D/2 + 1$. On the other hand, more iterations are needed whenever the eccentricity of u is close to the diameter. Unfortunately, graphs like the one shown in Figure 2 have the property that all nodes have eccentricity equal to the diameter, which implies that $D/2 + 1$ iterations will always be executed: that is, in these cases the *i*FUB algorithm executes the maximum number of iterations. In real-world networks, however, the difference between R and D seems to be very high (actually close to the maximum, that is, $D/2$). Moreover, in these cases 4-SWEEP seems to be able to choose a starting node u with low eccentricity (that is, close to R), and with very small fringe sizes.

4. Dataset

We collected about 200 real-world graphs, which have been chosen in order to cover the largest possible set of network typologies: as far as we know, this is the largest examined dataset of real-world graphs. Note that, in the case of several of these graphs, the exact value of the diameter was still unknown or only approximated (as in the case of the values given in [5]). We also considered some synthetic graphs obtained from well-known generative models. In particular, we classify our networks according to the following categories.

1. *Biological networks.* These graphs refer to databases of physical, genetic and biological interactions [14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24].
2. *Citations networks.* Nodes represent papers and edges represent citations [5, 25, 26].
3. *Collaboration networks.* Nodes represent people and edges represent collaborations (co-authors, actors, software developers teams , and so on) [5, 14, 27].
4. *Communication networks.* Nodes represent people and edges represent communication among them [5, 28, 29].
5. *Product co-purchasing networks.* Nodes represent products and edges link commonly co-purchased products [5].
6. *Autonomous systems graphs.* These are the graphs of the Internet, typically referring to connections among Internet Service Providers [5, 14].
7. *Internet peer-to-peer networks.* Nodes represent computers and edges represent communication among them [5, 30].
8. *Web graphs.* Nodes represent web pages and edges are hyperlinks [5, 30, 31, 32].
9. *Social networks.* Nodes represent people and edges represent interactions between them [5, 27, 28, 29, 33].
10. *Road networks.* Nodes represent intersections and endpoints and edges represent roads connecting these intersections or road endpoints [5].

Networks	Total	Number of graphs in which x over 10 experiments 4-Sweep has returned a tight lower bound			
		$x = 10$	$10 > x \geq 5$	$5 > x > 0$	$x = 0$
Biological networks	46	34	6	4	2
Citations networks	5	4	1	0	0
Collaboration networks	13	11	0	2	0
Communication networks	38	29	7	2	0
Product co-purchasing networks	4	4	0	0	0
Autonomous systems graphs	2	1	1	0	0
Internet peer-to-peer networks	2	2	0	0	0
Web graphs	11	10	1	0	0
Social networks	11	10	1	0	0
Road networks	3	2	1	0	0
Words adjacency networks	4	4	0	0	0
Meshes and electronic circuits	34	26	5	2	1
Synthetic graphs	18	15	2	1	0
Others	8	6	2	0	0
Total	199	158	27	11	3

Table 1: A summary of the results obtained by ten executions of the 4-Sweep algorithm.

11. *Words adjacency networks.* Nodes represent words and edges represent their adjacency in the text [15, 34].
12. *Meshes and electronic circuits.* These graphs correspond to adjacency matrices derived from finite element meshes or to stiffness matrices, or they are calculated during simulations for path optimization in digital electronic circuit projects [15, 35].
13. *Synthetic graphs.* These graphs are generated according to the following evolution models: Erdős-Rényi [36], random geometric [37], forest-fire [11] and Kronecker [10].

An important feature is that almost all graphs in our dataset are sparse (that is, $m = O(n)$).

5. Experiments

Our computing platform is a machine with a Pentium Dual-Core CPU (Intel(tm) E5200 @ 2.50GHz), with a 8GB shared memory. The operating system is a Debian GNU/Linux 6.0, with a Linux kernel version 2.6.32 and gcc version 4.4.5.

Obtaining a lower bound via random sampling.. When the graphs are so huge that no exhaustive computation can be done, a random set of nodes can be randomly sampled and the maximum eccentricity among these nodes can be returned as a lower bound for the diameter. We have performed 1000 random BFSes for each of the 199 graphs (this is the same number of BFSes used to compute the values reported in [5]). For 101 graphs the returned lower bound coincides with the diameter of the graph: however, only 34 of these graphs have more than 10000 nodes. On the other hand, in the remaining 98 graphs for which the lower bound is not tight, 91 graphs have more than 10000 nodes. It means that, as expected, the size of the random sample, that is the number of BFSes that have to be performed, has to depend on the size of the graph.

v	Number of graphs in which iFUB has performed v BFSes on the average					
		Number n of vertices				
	Total	$n \leq 10^3$	$10^3 < n \leq 10^4$	$10^4 < n \leq 10^5$	$10^5 < n \leq 10^6$	$10^6 < n$
$v = 5$	29	2	8	9	10	0
$5 < v \leq 100$	123	17	44	43	11	8
$100 < v \leq 1000$	21	1	3	10	4	3
$1000 < v \leq 10^4$	18	0	4	12	1	1
$10^4 < v \leq 10^5$	8	0	0	3	3	2

Table 2: A summary of the results obtained by ten executions of iFUB.

Obtaining tight lower bound via 4-Sweep. For each of the 199 graphs, we executed the 4-Sweep algorithm ten times (see Table 1).[1]

- For 158 graphs, in all the ten experiments the lower bound computed by 4-Sweep is equal to the diameter.
- For 185 graphs, in more than four among the ten experiments, the lower bound computed by 4-Sweep is equal to the diameter.
- For 196 graphs, in at least one among the ten experiments, the lower bound computed by 4-Sweep is equal to the diameter.
- In 3 networks, no experiment has returned a lower bound equal to the diameter.

Computing the diameter via iFUB. For each of the 199 graph, we executed the iFUB algorithm ten times. In Table 2 we show some of the results obtained by our experiments: in particular, we consider the average number of visits, among the ten experiments, employed by iFUB to calculate the exact value of diameter (that is, with $k = 0$). For each graph and for each number v, we report the number of graphs in which iFUB has performed v BFSes on the average. Observe that given a graph with n nodes, the number of BFSes ranges between 5 (that is, the case in which Algorithm 1 returns the exact value of diameter without entering the while loop), and $O(n)$ (that is, the case in which iFUB degenerates to the textbook algorithm).

In the figures shown in the appendix, we visually show the performance of iFUB by grouping the results with respect to the category of the graphs. On the x-axis is reported the number of nodes, while on the y-axis is reported the average ratio (among the ten experiments) between the number of BFSes performed by the algorithm and the number of nodes: the inverse of this ratio corresponds to the *gain* of the algorithm with respect to the textbook method. The ratio can range between $5/n$ (represented by the continuous line) and 1 (represented by the upper border of the plot).

[1] No significant variance was observed also on more experiments because central nodes are easily detected by the 4-Sweep algorithm: they usually are the same in all the experiments, even if we perform random choices.

- 22 graphs are such that *i*FUB is forced to perform more than 10% of the BFSes of the exhaustive method: these networks are some meshes and electronic circuit simulation networks, some random graphs, one peer-to-peer network, and one communication network;
- 37 graphs are such that *i*FUB performs between 1% and 10% of the BFSes of the exhaustive method: these networks are some biological networks, the road networks, some synthetic networks, and some meshes and electronic circuit simulation networks ;
- 59 graphs are such that *i*FUB performs between 0.1% and 1% of the BFSes of the exhaustive method: these networks are mainly biological networks, collaboration networks, and some synthetic networks;
- 49 graphs are such that *i*FUB performs between 0.01% and 0.1% of the BFSes of the exhaustive method: these are communication networks;
- 32 graphs are such that *i*FUB performs less than 0.01% of the BFSes of the exhaustive method.

It is worth observing that, even if sometime the plotted values are scattered over order of magnitudes, the relative cost of the algorithm seems to decrease with respect to n for almost all the networks except very few categories.

6. Extension to weighted graphs

Theorem 1 can be easily extended to the case of undirected weighted graphs. Indeed, let T_u^{w} denote a shortest path (in short, SP) tree rooted at node u (computed, for instance, by means of the Dijkstra algorithm [38]), $\mathrm{ecc}^{\mathrm{w}}(u)$ denote the weight of the longest path from u to one of the leaves of T_u^{w} (that is, the weighted eccentricity of u), and $F_d^{\mathrm{w}}(u)$ denote the set of nodes whose weighted distance from u is equal to d (hence, $F_d^{\mathrm{w}}(u) \neq \emptyset$ if and only if there exists at least one node x in T_u^{w} such that the weight of the path from u to x is equal to d). Moreover, let $d_1, d_2, \ldots, d_h$ be the sequence of distinct values d such that $F_d^{\mathrm{w}}(u) \neq \emptyset$ ordered in increasing order, that is, $d_1 < d_2 < \cdots < d_h$: note that $d_h = \mathrm{ecc}^{\mathrm{w}}(u)$. We then have the following result.

Theorem 2. *For any i with $1 < i < h$, for any k with $1 \leq k < i$, and for any $x \in F_{d_{i-k}}^{\mathrm{w}}(u)$ such that $\mathrm{ecc}^{\mathrm{w}}(x) > 2d_{i-1}$, there exists $y_x \in F_{d_j}^{\mathrm{w}}(u)$ such that $d(x, y_x) = \mathrm{ecc}^{\mathrm{w}}(x)$ with $j \geq i$.*

Proof. The proof is very similar to the one of Theorem 1. Since $\mathrm{ecc}^{\mathrm{w}}(x) > 2d_{i-1}$, then there exists y_x whose distance from x is greater than $2d_{i-1}$. If y_x was in $F_{d_j}^{\mathrm{w}}(u)$ with $j < i$, then it would follow that $d(x, y_x) \leq d_{i-k} + d_j < 2d_i$ (since both $i - k$ and j are smaller than i), which is a contradiction. Hence, y_x must be in $F_{d_j}^{\mathrm{w}}(u)$ with $j \geq i$. □

ALGORITHM 3: *i*FUB

Input: A weighted graph G, a node u, a lower bound l for the diameter, and an integer k
Output: A value M such that $D - M \leq k$
Let $d_1 < d_2 < \ldots < d_h$ be the sequence of distinct values d such that $F^{\mathrm{w}}_d(u) \neq \emptyset$
$i \leftarrow h$;
$lb \leftarrow \max\{\mathrm{ecc}^{\mathrm{w}}(u), l\}$;
$ub \leftarrow 2\mathrm{ecc}^{\mathrm{w}}(u)$;
while $ub - lb > k$ **do**
  if $\max\{lb, B^{\mathrm{w}}_{d_i}(u)\} > 2d_{i-1}$ **then**
    return $\max\{lb, B^{\mathrm{w}}_{d_i}(u)\}$;
  else
    $lb \leftarrow \max\{lb, B^{\mathrm{w}}_{d_i}(u)\}$;
    $ub \leftarrow 2d_{i-1}$;
  end
  $i \leftarrow i - 1$;
end
return lb;

We can then appropriately modify the *i*FUB algorithm in order to deal with undirected weighted graphs. Indeed, by defining $B^{\mathrm{w}}_{d_i}(u) = \max_{z \in F^{\mathrm{w}}_{d_i}(u)} \mathrm{ecc}^{\mathrm{w}}(z)$, we can adapt Algorithm 1 as shown in Algorithm 3. In order to start the execution of the algorithm, we can also modify the 4-SWEEP algorithm by using four single source shortest path algorithm executions instead of four BFSes. We have experimented this modification of the two algorithms on 12 undirected weighted real-world graphs: these graphs have been downloaded either from the weighted network dataset available at [39] or from the web site of the 9th DIMACS Implementation Challenge on shortest paths [40]. Most of these networks were originally directed and we made them undirected by setting the weight of each new (undirected) edge equal to the average of the weights of the two original (directed) edges. Our experimental results are summarized in Table 3: as it is shown in the table, these results turn out to be very promising, since the diameter is most of the times found in less than 10 executions of the shortest path algorithm, and the number of these executions is always much less than 10% of the number of all nodes in the largest strongly connected component. It is also worth observing that in all networks the modified 4-SWEEP algorithm computes the exact value of the diameter: as already observed in [8], the lesson learned in our experiments is thus that a good starting point is to run the 4-SWEEP method to have an educated guess of what is the diameter of a large graph, and that the *i*FUB algorithm can be used in order to validate this finding in many cases.

v	Number of graphs in which the weighted *i*FUB has performed v shortest path computations on the average					
	Total	Number n of vertices				
		$10^2 \leq n \leq 10^3$	$10^3 < n \leq 10^4$	$10^4 < n \leq 10^5$	$10^5 < n \leq 10^6$	$10^6 < n$
$5 < v \leq 10$	8	2	2	1	1	2
$10 < v \leq 100$	3	0	1	2	0	0
$10^3 < v \leq 10^4$	1	0	0	0	1	0

Table 3: A summary of the results obtained by ten executions of *i*FUB on weighted graphs.

7. Conclusions and future work

The main contribution of this paper is the definition of a new algorithm, called *i*FUB, for the computation of the diameter of undirected unweighted graphs. Even if, in the worst case, the time complexity of *i*FUB is $O(nm)$, this algorithm in practice executes in $O(m)$ time when applied to real-world networks (and, thus, in $O(n)$ time because of the sparsity of these networks). These impressive performances are also, but not only, due to the performance of the 4-Sweep method, that by means of only 4 BFSes is able to compute a lower bound for the diameter, which is in practice almost always tight: intuitively, the good performance of 4-Sweep lowers the variability of the performances of *i*FUB. As a result, the combination of these two algorithms allowed us to compute the diameter of huge real-world networks whose diameter was still unknown. We have also shown that the *i*FUB algorithm can be easily modified in order to deal with undirected unweighted graphs and we presented some preliminary very promising experimental results.

The *i*FUB algorithm has been recently integrated into the WebGraph Java library [9], and it has been used in order to compute the exact diameter of several quite huge subgraphs of the Facebook graph: the good news is that a highly parallel version of our method was able to compute the diameter of the largest subgraph (approximately 149.1M of nodes and 15.9G of edges) in twenty minutes [41].

The impressive performances of our algorithm seems to be related to the structure of the real-world networks, in which we have observed a very particular phenomenon: the radius of these networks is usually close to the minimum possible value with respect to their diameter, and, moreover, the set of the farthest nodes from nodes with low eccentricity is usually small. Understanding the reasons of such structural properties in real-world networks is an interesting open problem to be addressed in future.

The reason of the accuracy of the 4-Sweep algorithm while computing lower bounds for the diameter and nodes with low eccentricity is also an interesting open problem. Indeed, a way to characterize graphs for which 4-Sweep returns the exact value of the diameter is still unknown even if some steps has been performed by [42]. If this will be better understood, then it could be interesting to understand also the reasons for which the real-world networks exhibit this characterizing behavior.

References

[1] Z. Galil, O. Margalit, All pairs shortest distances for graphs with small integer length edges, Information and Computation 134 (2) (1997) 103 – 139.

[2] R. Seidel, On the all-pairs-shortest-path problem in unweighted undirected graphs, J. Comput. Syst. Sci. 51 (1995) 400–403.

[3] U. Zwick, Exact and Approximate Distances in Graphs - a survey, in: F. M. a. d. Heide (Ed.), ESA '01: Proceedings of the 9th Annual European Symposium on Algorithms, Springer-Verlag, London, UK, 2001, pp. 33–48.

[4] A. Mislove, M. Marcon, K. Gummadi, P. Druschel, B. Bhattacharjee, Measurement and Analysis of Online Social Networks, in: Proceedings of the 5th ACM/USENIX Internet Measurement Conference (IMC'07), ACM, New York, NY, USA, 2007.

[5] SNAP, Stanford Network Analysis Package (SNAP) Website, `http://snap.stanford.edu` (2009).

[6] D. Corneil, F. Dragan, M. Habib, C. Paul, Diameter determination on restricted graph families, Discrete Applied Mathematics 113 (2-3) (2001) 143–166.

[7] C. Magnien, M. Latapy, M. Habib, Fast computation of empirically tight bounds for the diameter of massive graphs, Journal of Experimental Algorithmics 13.

[8] P. Crescenzi, R. Grossi, C. Imbrenda, L. Lanzi, A. Marino, Finding the Diameter in Real-World Graphs: Experimentally Turning a Lower Bound into an Upper Bound, in: Proc. ESA, Vol. 6346 of LNCS, 2010, pp. 302–313.

[9] P. Boldi, S. Vigna, The WebGraph Framework I: Compression Techniques, in: Proceedings of the 13th International World Wide Web Conference, ACM Press, Manhattan, USA, 2003, pp. 595–601.

[10] J. Leskovec, D. Chakrabarti, J. Kleinberg, C. Faloutsos, Z. Ghahramani, Kronecker Graphs: An Approach to Modeling Networks, Journal of Machine Learning Research 11 (2010) 985–1042.

[11] J. Leskovec, J. Kleinberg, C. Faloutsos, Graph Evolution: Densification and Shrinking Diameters, ACM Trans. Knowl. Discov. Data 1 (1).

[12] C. R. Palmer, P. B. Gibbons, C. Faloutsos, ANF: a Fast and Scalable Tool for Data Mining in Massive Graphs, in: Proceedings of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2002, pp. 81–90.

[13] R. G. on Graph Theory, D. o. t. U. P. d. C. U. Combinatorics, The degree diameter problem for general graphs, `http://www-mat.upc.es/grup_de_grafs/` (2010).

[14] C. Sommer, Christian Sommer's homepage, `http://www.sommer.jp/graphs/` (2009).

[15] U. Aron, Uri Alon Lab, `http://www.weizmann.ac.il/mcb/UriAlon/` (2002).

[16] MetExplore, MetExplore: a web server to link metabolomic experiments and genome-scale metabolic networks, `http://metexplore.toulouse.inra.fr/metexplore/` (2010).

[17] J. Wu, T. Vallenius, K. Ovaska, J. Westermarck, T. Makela, S. Hautaniemi, Integrated network analysis platform for protein-protein interactions, Nature methods 6 (2009) 75–77.

[18] J. Duch, A. Arenas, Community identification using Extremal Optimization, Physical Review E 72 (2005) 027104.

[19] HPRD, Human Protein Reference Database, http://www.hprd.org/ (2003).

[20] The Jena Protein-Protein Interaction Website, http://ppi.fli-leibniz.de/ (2009).

[21] iPfam, iPfam: the Protein Domain Interactions Database, http://ipfam.sanger.ac.uk/ (2009).

[22] Integrated protein-protein interaction database of Synechocystis sp. PC 6803, http://bioportal.kobic.re.kr/SynechoNET/ (2007).

[23] STRING, Functional protein association networks, http://string-db.org/ (2000).

[24] A PPI graph from Intact DB, http://prabi2.inrialpes.fr/bamboo/ (2010).

[25] CiteSeer, CiteSeer Website, http://citeseer.ist.psu.edu/citeseer.html (1997).

[26] Cora, The Cora dataset, http://www.cs.umd.edu/∼sen/lbc-proj/data/ (2007).

[27] TrustLet, TrustLet Website, ://www.trustlet.org (2007).

[28] B. Y. Zhao, CURRENT LAB: social networking project, Data available, as explained at http://current.cs.ucsb.edu/facebook/ (2010).

[29] Sandbox, Webscope from Yahoo! Labs, Data available, as explained at http://sandbox.yahoo.com/ (2010).

[30] M. Latapy, FTP public directory, ftp://www-rp.lip6.fr/pub/latapy/Measurement/ (2007).

[31] P. Boldi, B. Codenotti, M. Santini, S. Vigna, UbiCrawler: A Scalable Fully Distributed Web Crawler, Software: Practice & Experience 34 (8) (2004) 711–726.

[32] Clusters and Communities, overlapping dense groups in networks, http://hal.elte.hu/cfinder/ (2005).

[33] A. Stoica, Alina Stoica homepage, http://www.liafa.jussieu.fr/~stoica/.

[34] Pajek dataset, http://vlado.fmf.uni-lj.si/pub/networks/data/default.htm (2006).

[35] C. Walshaw, The University of Greenwich Graph Partitioning Archive, http://staffweb.cms.gre.ac.uk/~c.walshaw/partition/ (2000).

[36] P. Erdős, A. Rényi, On random graphs, I, Publ. Math. Debrecen 6 (1959) 290–297.

[37] M. Penrose, Random Geometric Graphs, Oxford Studies in Probability, 2003.

[38] E. Dijkstra, A note on two problems in connexion with graphs, Numer. Math. 1 (1959) 269–271.

[39] Network datasets, `http://toreopsahl.com/datasets/` (2009).

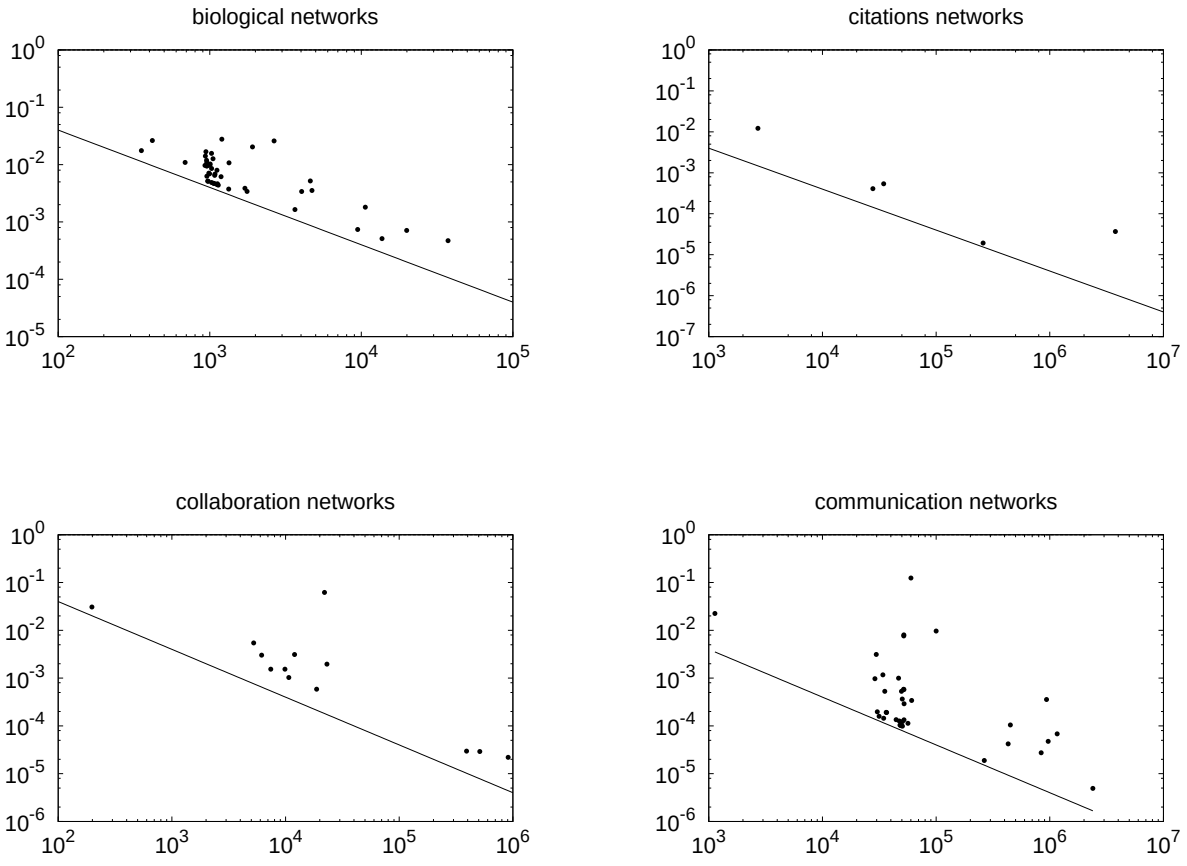
[40] 9th DIMACS Implementation Challenge - Shortest Paths, `http://www.dis.uniroma1.it/~challenge9/` (2006).

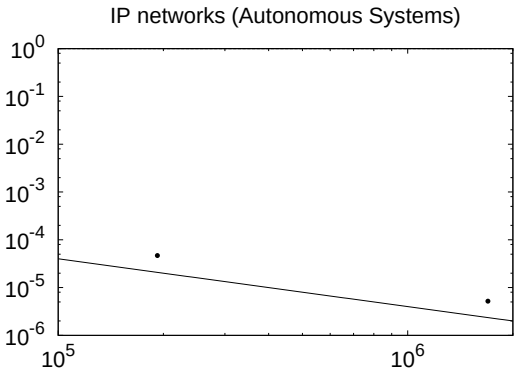
[41] L. Backstrom, P. Boldi, M. Rosa, J. Ugander, S. Vigna, Four Degrees of Separation, arXiv:1111.4570v1.

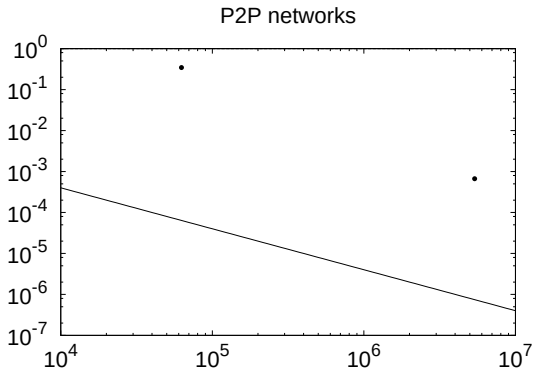
[42] V. Chepoi, F. Dragan, B. Estellon, M. Habib, Y. Vaxès, Diameters, centers, and approximating trees of delta-hyperbolic geodesic spaces and graphs, in: Proc. SCG, 2008, pp. 59–68.

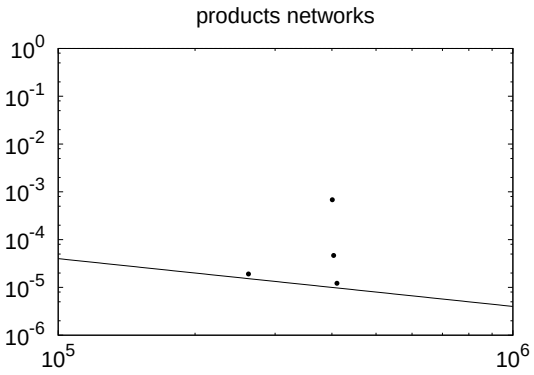
Appendix A. Figures summarizing our experimental results

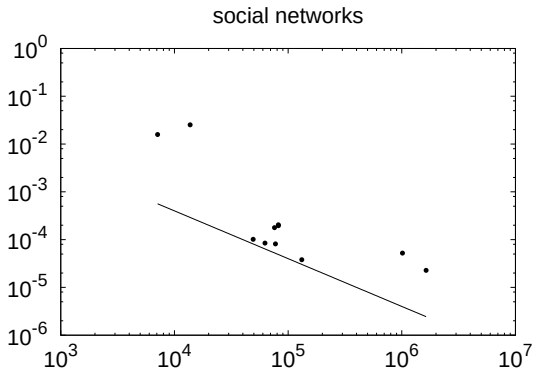
The following plots summarize our experimental results for the different network classes: in the plots, the relative cost is expressed by the ratio between the average number of BFSes executed by *i*FUB and the number of nodes.

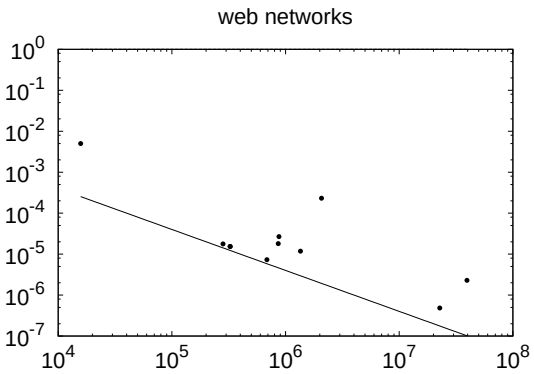


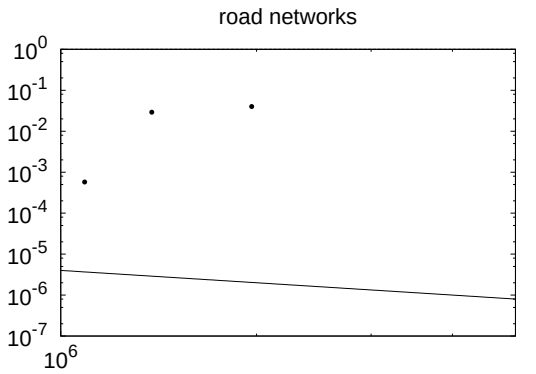

IP networks (Autonomous Systems)
10^0
10^{-1}
10^{-2}
10^{-3}
10^{-4}
10^{-5}
10^{-6}
10^5
10^6

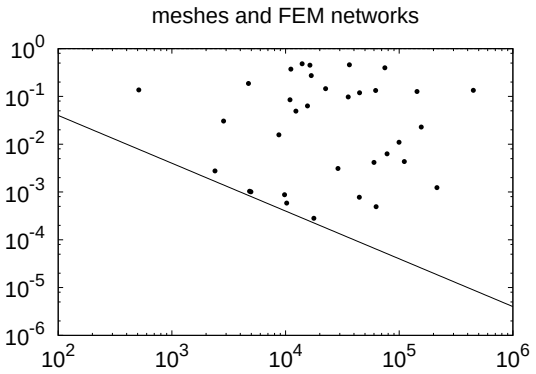

P2P networks
10^0
10^{-1}
10^{-2}
10^{-3}
10^{-4}
10^{-5}
10^{-6}
10^{-7}
10^4
10^5
10^6
10^7

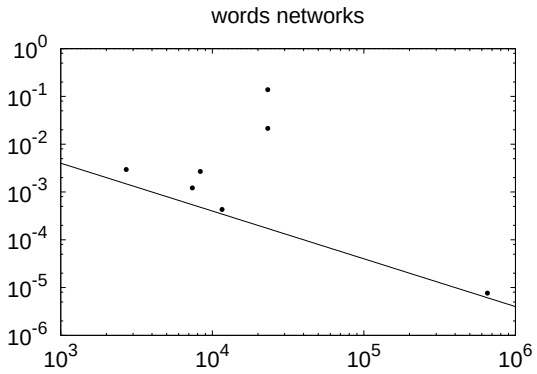

products networks
10^0
10^{-1}
10^{-2}
10^{-3}
10^{-4}
10^{-5}
10^{-6}
10^5
10^6

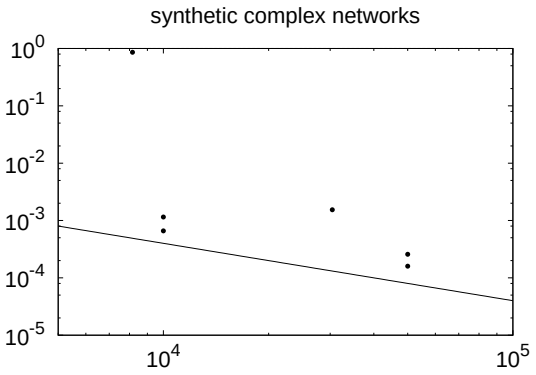

social networks
10^0
10^{-1}
10^{-2}
10^{-3}
10^{-4}
10^{-5}
10^{-6}
10^3
10^4
10^5
10^6
10^7

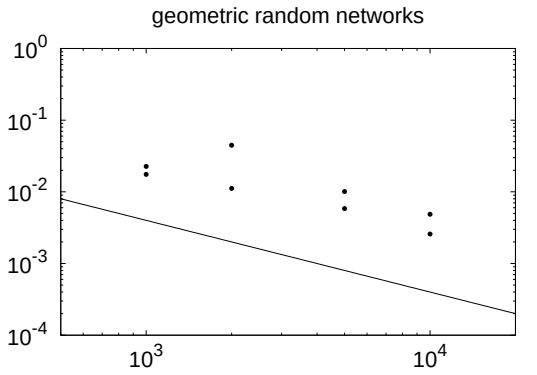

web networks
10^0
10^{-1}
10^{-2}
10^{-3}
10^{-4}
10^{-5}
10^{-6}
10^{-7}
10^4
10^5
10^6
10^7
10^8

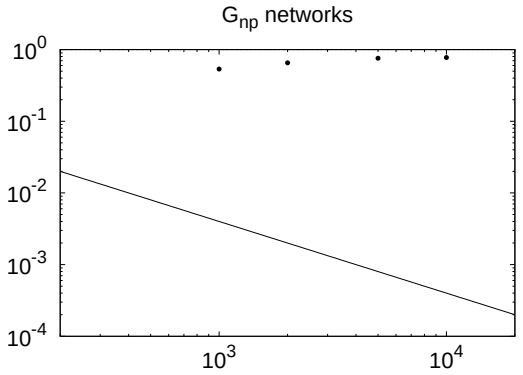

road networks
10^0
10^{-1}
10^{-2}
10^{-3}
10^{-4}
10^{-5}
10^{-6}
10^{-7}
10^6


meshes and FEM networks
10^0
10^{-1}
10^{-2}
10^{-3}
10^{-4}
10^{-5}
10^{-6}
10^2
10^3
10^4
10^5
10^6


words networks
10^0
10^{-1}
10^{-2}
10^{-3}
10^{-4}
10^{-5}
10^{-6}
10^3
10^4
10^5
10^6


synthetic complex networks
10^0
10^{-1}
10^{-2}
10^{-3}
10^{-4}
10^{-5}
10^4
10^5


geometric random networks
10^0
10^{-1}
10^{-2}
10^{-3}
10^{-4}
10^3
10^4


G_{np} networks
10^0
10^{-1}
10^{-2}
10^{-3}
10^{-4}
10^3
10^4