Theoretical Computer Science 514 (2013) 84–95

Contents lists available at ScienceDirect

Theoretical Computer Science

journal homepage: www.elsevier.com/locate/tcs

On computing the diameter of real-world undirected graphs[☆]

Pilu Crescenzi [a,*], Roberto Grossi [b], Michel Habib [c], Leonardo Lanzi [a], Andrea 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

ARTICLE INFO

Keywords:
Breadth-first search
Diameter
Complex network
Experimental analysis

ABSTRACT

We propose a new algorithm for the classical problem of computing the diameter of undirected unweighted graphs, namely, the maximum distance among all the pairs of nodes, where the distance of a pair of nodes is the number of edges contained in the shortest path connecting these two nodes. Although its worst-case complexity is $O(nm)$ time, where n is the number of nodes and m is the number of edges of the graph, we experimentally show that our algorithm works in $O(m)$ time in practice, requiring few breadth-first searches to complete its task on almost 200 real-world graphs.

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 and 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 (for short, BFS) from *every* 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 theoretical methods, like the ones presented in [1,2], turn out to be too much time consuming. We refer the reader to [3] for a comprehensive overview of results concerning distance and diameter computation.

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

The main contribution of this paper consists of showing that BFS can indeed be an extremely *powerful* tool to compute the *exact* value of the diameter, whenever it is used in a more clever way. In particular, we present the *iterative* Fringe Upper Bound (for short, *i*FUB) algorithm to calculate the exact value of the diameter, as a generalization of the FUB method described in [6]. This algorithm uses a subroutine to select a suitable node u and starts an iterative procedure from u: during this procedure, *i*FUB refines both a lower bound value lb and an upper bound value ub, such that $lb \leq D \leq ub$ at any time. It terminates when $lb = ub$ (or $ub - lb \leq k$ for a specified threshold k). The starting node u could be either a randomly chosen node, or a node with the highest degree, or a node returned by the 4-SWEEP algorithm, which is a natural extension of the method described in [7,8].

[☆] 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. Tel.: +39 055 4237452.
E-mail addresses: pcrescenzi@unifi.it, pierluigi.crescenzi@unifi.it (P. Crescenzi).

0304-3975/$ – see front matter © 2012 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2012.09.018

The worst-case 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 do *not* need to perform *all* the BFSes as in the case of the exhaustive textbook method. This phenomenon is quite evident in the case of real-world graphs: 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 [9–12], and synthetic graphs created by using some well-known models for generating random complex networks, meshes, and electronic circuits. We then compared the performance of *i*FUB with other approaches appeared in the literature, by using approximately 30 naturally undirected real-world networks taken from our dataset. We implemented our algorithm in C and in Java languages: both the source code and the dataset are available at our website amici.dsi.unifi.it/lasagne.

The paper is organized as follows. Section 2 describes the *i*FUB algorithm and analyzes 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 and in Section 6 we compare *i*FUB with other methods. We conclude in Section 7 by proposing some open questions.

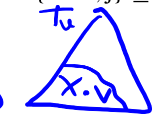
Largest distance from u to any $v \in G$
$\text{diam}(G) = \max_u \text{ecc}(u)$

2. The *i*FUB algorithm

Let $G = (V, E)$ be an undirected unweighted graph and let u be any node in V. We denote a BFS tree rooted at node u by T_u, the height of T_u by the eccentricity $\text{ecc}(u)$, and the set $\{v \mid d(u, v) = \text{ecc}(u)\}$ of nodes at maximum distance $\text{ecc}(u)$ from u by $F(u)$. In general, let $F_i(u)$ be the *fringe* set of nodes at distance i from u (note that $F(u) = F_{\text{ecc}(u)}(u)$) and let $B_i(u) = \max_{z \in F_i(u)} \text{ecc}(z)$ be the maximum eccentricity among these nodes. Observe that, for any x and y in V such that $x \in F_i(u)$ or $y \in F_i(u)$, we have that $d(x, y) \le B_i(u)$: indeed, $d(x, y) \le \min\{\text{ecc}(x), \text{ecc}(y)\} \le B_i(u)$. Observe also that, for any $1 \le i, j \le \text{ecc}(u)$ and for any $x \in F_i(u)$ and $y \in F_j(u)$, we have $d(x, y) \le i + j \le 2\max\{i, j\}$. We use this latter observation to give a termination condition for our *i*FUB algorithm as stated below.

Theorem 1. *For any* $1 \le i < \text{ecc}(u)$ *and* $1 \le k < i$, *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* $j \ge i$.

Proof. Since $\text{ecc}(x) > 2(i - 1)$, then there exists y_x whose distance from x is equal to $\text{ecc}(x)$ and, hence, 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 $d(x, y_x) \le 2\max\{i - k, j\} \le 2\max\{i - k, i - 1\} = 2(i - 1)$, which is a contradiction. Hence, y_x must be in $F_j(u)$ with $j \ge i$. □



2.1. *Exploring the* BFS *from node u*

identify v s.t. $\text{ecc}(v) < \text{ecc}(u)$ don't need BFS(v)

As previously said, Theorem 1 gives a termination condition for our *i*FUB algorithm: indeed, it implies that if y is a node in $F_i(u) \cup F_{i+1}(u) \cup \cdots \cup F_{\text{ecc}(u)}(u)$ with maximum eccentricity $\text{ecc}(y) > 2(i - 1)$, then the eccentricity of all nodes in $F_1(u) \cup F_2(u) \cup \cdots \cup F_{i-1}(u)$ is not greater than $\text{ecc}(y)$. This suggests to traverse the BFS tree 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 traversing the remaining levels, since the eccentricities of all their nodes cannot be greater than e. This idea suggests the following approach for a node u.

- 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 above approach, 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 $2(i - 1)$ when $M \le 2(i - 1)$. This leads us to Algorithm 1 which combines all of these observations. The algorithm receives as inputs the node u, an already known lower bound l, and a precision threshold k. It 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, $\text{ecc}(u)$). Moreover, it initializes the upper bound by setting it equal to the trivial one obtained by performing the BFS starting from u (that is, $2\text{ecc}(u)$). Then, it performs a **while** loop by refining the current lower and upper bounds as discussed so far. It terminates when $ub - lb \le k$ and $M = lb$ is returned (note that, by setting $k = 0$, we obtain the actual diameter D).

The running time of the algorithm is $O(nm)$ in the worst case, as in the case of the textbook algorithm: indeed, in the worst case, we have to perform a BFS starting from "almost" every node of the graph. Specifically, at each iteration of the **while** loop, $ub - lb$ decreases by at least 2, where initially $ub - lb \le \text{ecc}(u) \le D$: this implies that the algorithm executes at most $\text{ecc}(u)/2 \le D/2$ iterations. Thus, letting $N_{\ge h}(u)$ be the number of nodes v such that $d(u, v) \ge h$, we have that *i*FUB performs at most $N_{\ge D/2}(u)$ BFSes. Since $N_{\ge D/2}(u) \le n$, we have that $N_{\ge D/2}(u)$ is a better theoretical upper bound on the number of BFSes performed by *i*FUB. In Section 3 we will show some *ad hoc* cases in which $N_{\ge D/2}(u)$ is close to n, while in Section 5 we will show that the quantity $N_{\ge D/2}(u)$ is very low in the case of real-world graphs, by choosing the starting node u as described below.

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 $D - M \leq k$
$i \leftarrow \mathrm{ecc}(u)$;
$lb \leftarrow \max\{\mathrm{ecc}(u), l\}$;
$ub \leftarrow 2\mathrm{ecc}(u)$;
while $ub - lb > k$ do
    if $\max\{lb, B_i(u)\} > 2(i-1)$ then
        return $\max\{lb, B_i(u)\}$;
    else
        $lb \leftarrow \max\{lb, B_i(u)\}$;
        $ub \leftarrow 2(i-1)$;
    end
    $i \leftarrow i - 1$;
end
return $lb$;
```

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 $G$ or node with the highest degree;
$a_1 \leftarrow argmax_{v \in V} d(r_1, v)$;
$b_1 \leftarrow argmax_{v \in V} d(a_1, v)$;
$r_2 \leftarrow$ 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 $b_2$;
$lowerb \leftarrow \max\{\mathrm{ecc}(a_1), \mathrm{ecc}(a_2)\}$;
return $lowerb$ and $u$;
```

2.2. Selecting the starting node *u*

Since the starting node u affects the performance of *i*FUB, we describe here some ways of choosing u while postponing their experimental evaluation to Section 5.

Random selection. A first very simple method of choosing the starting node u is by picking it uniformly at random.

Degree selection. A second simple way of selecting u is choosing a node with the highest degree. The intuition behind this greedy choice is that a node u with high degree is more likely to yield a BFS tree whose upper levels are dense. Hopefully, $N_{\geq D/2}(u)$ is a small fraction of the number of nodes in this case.

4-Sweep selection. A third and more complex way to select u is by using the 4-SWEEP method shown in Algorithm 2, whose aim is finding a "good" (i.e. with low eccentricity) starting node u, by using always four BFSes. Let r_1 be a node in V, let a_1 be one of the farthest nodes from r_1, and let b_1 be one of the farthest nodes 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. Our node u is then defined as the middle node of the path between a_2 and b_2. In this paper, we will consider two different ways of selecting node r_1: one method chooses r_1 uniformly at random, while the other method chooses r_1 as a node with the highest degree. Intuitively, the node u resulting from the 4-SWEEP algorithm 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, $R \approx D/2$ for real-world graphs). 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, note that Algorithm 2 computes also an initial lower bound $l = \max\{\mathrm{ecc}(a_1), \mathrm{ecc}(a_2)\}$.

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 always compute the exact value of the diameter and that there exists an infinite family of graphs for which the *i*FUB algorithm requires $\Theta(nm)$ time. We will make use of the following observation to build our examples. For any graph $G = (V, E)$ of diameter $D > 1$ and for any node $u \in V$, let us consider the graph $G' = (V \cup \{u'\}, E \cup E')$ where u' is a new node and $E' = \{(u', v) \mid (u, v) \in E\}$. Clearly, we have that G' 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.

Bad graphs for 4-SWEEP. The idea of the counterexample presented in [6] can be generalized in order to obtain an example with an arbitrary diameter value in which the 4-SWEEP algorithm can compute a lower bound which is not tight. For any

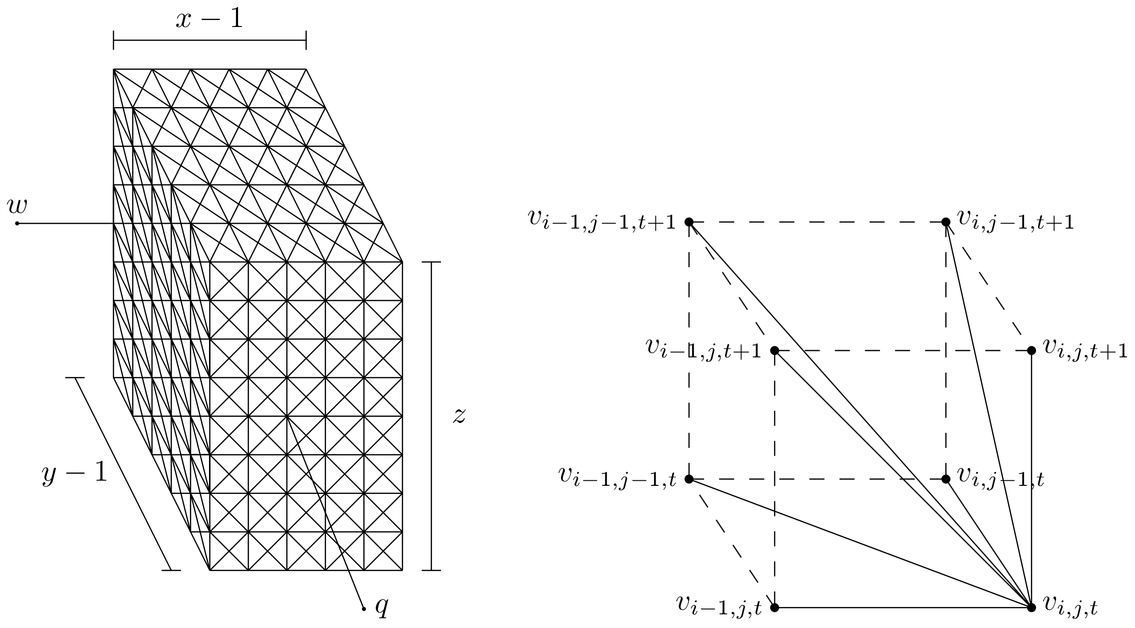


Fig. 1. A bad network for the 4-SWEEP algorithm (left) and a zoomed detail (right).

three odd numbers x, y and z greater than 2, we define a graph $G = (V, E)$ in which $V = \{v_{i,j,t} : 1 \le i \le x-1,\ 1 \le j \le y-1,\ 1 \le t \le z\} \cup \{w, q\}$ and E is defined as follows (see the left part of Fig. 1, where $x = 7$, $y = 7$, and $z = 9$). 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 Fig. 1). Moreover node w is connected to $v_{1,1,(z+1)/2}$ and node q is connected to $v_{(x-1)/2,y-1,(z+1)/2}$.

Observe that for any two distinct nodes $v_{i,j,t}$ and $v_{i',j',t'}$, we have $d(v_{i,j,t}, v_{i',j',t'}) = \max\{|i-i'|, |j-j'|, |t-t'|\} \le \max\{x-2, y-2, z-1\}$. Also, $d(v_{i,j,t}, w) = d(v_{i,j,t}, v_{1,1,(z+1)/2}) + 1 \le \max\{x-2, y-2, (z-1)/2\} + 1$, while $d(v_{i,j,t}, q) = d(v_{i,j,t}, v_{(x-1)/2,y-1,(z+1)/2}) + 1 \le \max\{(x-1)/2, y-2, (z-1)/2\} + 1$. Moreover $d(w, q) = d(v_{1,1,(z+1)/2}, v_{(x-1)/2,y-1,(z+1)/2}) + 2 = \max\{(x-3)/2, y-2\} + 2$.

Thus the diameter of G is $\max\{x-1, y, z-1\}$. If x, y, and z are such that $z > x > y+1 > 3$, then the diameter is equal to $z-1$. Moreover, if $y-1 > (z+1)/2$, $y-1 > (x+1)/2$, 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, according 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 $y-1 > (x-1)/2$.
- $b_2 = w$, 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, which is the same ratio obtained by simply performing one BFS and returning the diameter of the corresponding tree.

Observe that the above example can be modified (without changing the diameter) so that the starting bad choice is more probable (by creating several copies of the node $v_{x-1,1,(z+1)/2}$), or so that the unique node of highest degree is a bad choice (by creating several dummy neighbors of $v_{x-1,1,(z+1)/2}$ and by properly adapting the position of q). Moreover, the example can be easily generalized to higher dimensions and become a *bad graph* for natural extensions of the 4-SWEEP algorithm, such as the 2k-SWEEP algorithm for $k > 2$.

Bad graphs for iFUB. There exist graphs, such as the cycle shown in the left part of Fig. 2 and other graphs available at [13], where *i*FUB employs $\Theta(n)$ BFSes. These graphs are characterized by an extreme regularity: the BFS trees at their nodes are very similar, with the radius and the diameter being the same (namely, $R = D$), and all nodes having eccentricity equal to the diameter. This implies that $D/2 + 1$ iterations will always be executed in the *i*FUB algorithm, and that $N_{\ge D/2}(u)$ is also close to n. Thus in the case of these graphs, the complexity of *i*FUB is $\Theta(nm)$. For instance, a cycle with n nodes (where n is odd) has diameter $\frac{n-1}{2}$, and any of its nodes has the same BFS tree, whose height is $\frac{n-1}{2}$ (see the right part of Fig. 2). Thus, referring to Algorithm 1, *i*FUB repeats its loop until $2(i-1) \ge \frac{n-1}{2}$, that is $i \ge \frac{n+3}{4}$, and stops the first time that $2(i-1) < \frac{n-1}{2}$: the

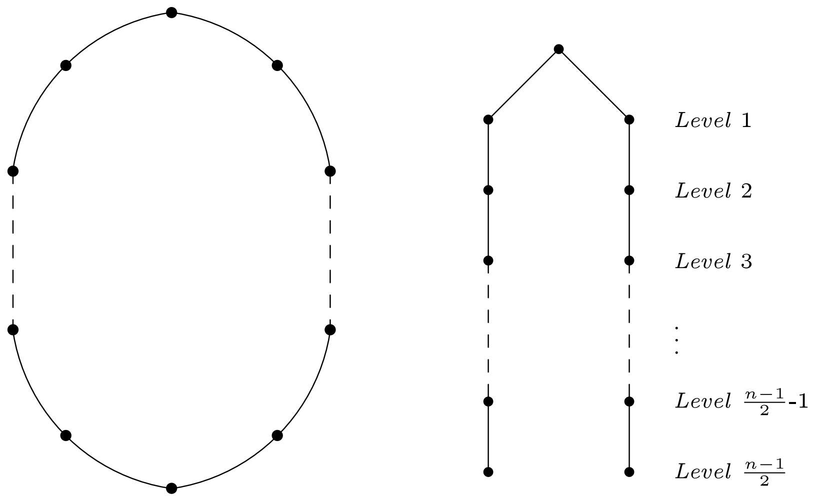


Fig. 2. A cycle, which is a bad graph for the *i*FUB algorithm, (left) and a BFS tree from any of its n nodes, with n odd (right).

total number of iterations is equal to $\frac{n-1}{2} - \frac{n+3}{4} + 2 = \frac{n+3}{4}$. Since each level has two nodes, the number of BFSes performed by *i*FUB is equal to $\frac{n+3}{2}$ and, thus, linear in the number of nodes.

*Real-world networks: potentially good for i*FUB. In real-world networks, the difference between D and R seems to be very high (actually close to the maximum, that is, $D/2$). Hence the number of iterations depends on the choice of the starting node u. In particular, let $C = \{v \in V \mid \mathrm{ecc}(v) = R\}$ be the set of centers of the graph. Then, the minimum number of iterations performed by *i*FUB is obtained whenever $u \in C$: the number of iterations is minimum and equals $R - D/2 + 1$, and so the upper bound $N_{\geq D/2}(u)$ is also minimum.

4. Dataset

We collected 196 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. For 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. We classify our graphs according to the following categories.

1. *Biological networks*. These graphs refer to databases of physical, genetic and biological interactions [14–24].
2. *Citations networks*. Nodes represent papers and edges represent citations [5,25,26].
3. *Collaboration networks*. Nodes represent people and edges represent collaborations among them [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 represent 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 [5,30–32].
9. *Social networks*. Nodes represent people and edges represent interactions between them [5,27–29,33].
10. *Road networks*. Nodes represent intersections and endpoints and edges represent roads connecting these intersections and endpoints [5].
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 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 of our dataset is that almost all graphs in it are sparse, that is, $m = O(n)$.

5. Experiments

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 196 graphs (this is the same number of BFSes used to compute the values

Table 1
A summary of the results obtained by ten executions of the 4-Sweep*r* algorithm, i.e. the 4-Sweep where r_1 is randomly chosen, and by the execution of the 4-Sweep*hd* algorithm, i.e. the 4-Sweep where r_1 is a node with the highest degree.

Networks	Total	Number of graphs in which x over 10 experiments 4-Sweep*r* has returned a tight lower bound				Number of graphs in which 4-Sweep*hd* has returned a tight lower bound
		$x = 10$	$10 > x \geq 5$	$5 > x > 0$	$x = 0$	
Autonomous systems graphs	2	1	1	0	0	2
Biological networks	48	35	7	4	2	47
Citations networks	5	4	1	0	0	5
Collaboration networks	13	11	0	2	0	13
Communication networks	38	29	7	2	0	37
Internet peer-to-peer networks	1	1	0	0	0	1
Meshes and electronic circuits	34	26	5	2	1	22
Product co-purchasing networks	4	4	0	0	0	4
Road networks	3	2	1	0	0	2
Social networks	11	10	1	0	0	11
Synthetic graphs	18	15	2	1	0	15
Web graphs	9	8	1	0	0	9
Words adjacency networks	4	4	0	0	0	4
Others	6	5	1	0	0	3
Total	196	155	27	11	3	175

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 10 000 nodes. On the other hand, in the remaining 95 graphs for which the lower bound is not tight, 88 graphs have more than 10 000 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.

5.1. *Obtaining tight lower bound via* 4-Sweep

The 4-Sweep algorithm computes a lower bound on the diameter by using always 4 BFSes, independently of the size of the graph. Nevertheless, its performance is much better than the random sampling method. In particular, we have verified this statement on all the 196 graphs by considering the two different strategies for selecting the starting node r_1, described at the end of Section 2.2.

4-Sweep with random starting node (for short 4-Sweepr). For each of the 196 graphs, we executed the 4-Sweep algorithm ten times (see the central columns of Table 1) by choosing r_1 uniformly at random.[1] For 155 graphs, in all the ten experiments the lower bound computed by 4-Sweep is equal to the diameter and for 193 graphs in at least one among the ten experiments the lower bound is tight. Hence, in the case of 3 networks (i.e. a mesh and two biological networks), no experiment has returned a lower bound equal to the diameter.

4-Sweep with highest degree starting node (for short 4-Sweephd). For each of the 196 graphs, we executed the 4-Sweep algorithm by choosing r_1 as a node with the highest degree. In almost all graphs in the dataset there is one node of maximum degree: the exceptions are 19 meshes (where a linear number of nodes have the same degree), 2 road networks (with respectively 4 and 5 nodes with maximum degree), 7 synthetic networks (with at most 3 nodes with maximum degree), 2 biological networks (with respectively 5 and 8 nodes with maximum degree). In these particular cases ties are broken by considering the node with the smallest identifier: however similar results can be observed by considering other choices. For 175 graphs the computed lower bound is tight (see the rightmost column of Table 1). In the case of 16 graphs the error is at most 2, while in the case of 4 meshes the error is between 3 and 9, and in the case of one road network the error is 14.

5.2. *Computing the diameter via* iFUB

For each of the 196 graphs, we executed the *i*FUB algorithm to calculate the exact value of the diameter, by setting $k = 0$ and by considering the four different strategies to select the starting node u described in Section 2.2.

*i*FUB *by using 4-Sweepr (for short i*FUB*+4-Sweepr).* For each graph we executed the *i*FUB algorithm ten times by using Algorithm 2, with r_1 chosen uniformly at random, to select the starting node u. In Table 2(a) we consider the average number v of BFSes executed over the ten experiments and in Table 3(a) we consider the corresponding ratio v/n.[2] Observe that given

[1] No significant variance was observed by running a larger number of experiments because central nodes are easily detected by the 4-Sweep algorithm.
[2] The inverse of the ratio v/n can be considered as the *gain* of the *i*FUB algorithm with respect to the textbook method. It is worth noting that this gain seems to increase for increasing values of n in almost all the network categories apart from very few ones.

Table 2
Computing the diameter via *i*FUB.

(a) Results obtained by ten executions of *i*FUB+4-SWEEP*r*, i.e. *i*FUB by using 4-SWEEP*r*.

size of |v|

#BFS done

v	Number of graphs in which *i*FUB has performed v BFSes on the average					
		Number n of nodes				
	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
$5 < v \leq 100$	121	17	44	43	$\boxed{11}$	6
$100 < v \leq 1000$	21	1	3	10	4	3
$1000 < v \leq 10^4$	17	–	4	12	1	0
$10^4 < v$	8	–	–	3	3	2

(b) Results obtained by the execution of *i*FUB+4-SWEEP*hd*, i.e. *i*FUB by using 4-SWEEP*hd*.

v	Number of graphs in which *i*FUB has performed v BFSes					
		Number n of nodes				
	Total	$n \leq 10^3$	$10^3 < n \leq 10^4$	$10^4 < n \leq 10^5$	$10^5 < n \leq 10^6$	$10^6 < n$
$v = 5$	35	11	4	11	9	0
$5 < v \leq 100$	115	25	29	42	14	5
$100 < v \leq 1000$	20	2	4	8	3	3
$1000 < v \leq 10^4$	19	–	4	13	2	0
$10^4 < v$	7	–	–	3	2	2

(c) Results obtained by the execution of *i*FUB+*hd*, i.e. *i*FUB with starting node of highest degree.

v	Number of graphs in which *i*FUB has performed v BFSes					
		Number n of nodes				
	Total	$n \leq 10^3$	$10^3 < n \leq 10^4$	$10^4 < n \leq 10^5$	$10^5 < n \leq 10^6$	$10^6 < n$
$v = 2$	22	10	3	7	2	0
$2 < v \leq 5$	38	14	13	6	5	0
$5 < v \leq 100$	74	12	13	28	14	7
$100 < v \leq 1000$	17	2	4	9	2	0
$1000 < v \leq 10^4$	28	–	8	19	1	0
$10^4 < v$	14	–	–	8	6	0

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 the diameter without entering the **while** loop), and $O(n)$ (that is, the case in which *i*FUB degenerates to the textbook algorithm). As shown in Table 3(a), only in the case of 22 graphs, *i*FUB is forced to perform more than 10% of the BFSes of the textbook method: these networks are mostly meshes and electronic circuit simulation networks, and synthetic graphs. For the great majority of the networks, instead, *i*FUB performs less than 1% of the BFSes executed by the textbook method.

*i*FUB *by using* 4-SWEEP*hd (for short i*FUB+4-SWEEP*hd).* For each graph we executed the *i*FUB algorithm by using again Algorithm 2 to select the starting node, with r_1 chosen as a node with the highest degree. In Table 2(b) we report the number v of BFSes executed by *i*FUB+4-SWEEP*hd* to calculate the exact value of diameter, and in Table 3(b) we consider the ratio v/n. As shown by Table 3(b), just for 19 graphs, *i*FUB is forced to perform more than 10% of the BFSes of the textbook method.

*i*FUB *by starting from the highest degree node (for short i*FUB+*hd).* For each graph we executed the *i*FUB algorithm by selecting as starting node one with the highest degree (without using Algorithm 2). Again, Table 2(c) reports the corresponding number v of BFSes employed by *i*FUB+*hd* to calculate the exact value of diameter, and Table 3(c) shows the corresponding ratio v/n. Note that in this case the minimum number of BFSes employed by *i*FUB can be 2. By looking at Table 3, *i*FUB+*hd* seems to be less effective than *i*FUB+4-SWEEP*hd*: however, we will see in the next section that if we restrict our attention only to the naturally undirected networks of the dataset, this is not true any more.

*i*FUB *by starting from a random node (for short i*FUB+*r).* For each graph we executed the *i*FUB algorithm by selecting a randomly chosen starting node. We do not report the corresponding entries in Tables 2 and 3 since the results are very unstable. For the sake of completeness and in order to underline here the importance of a good starting strategy, we observe that, even if sometimes the number of BFSes is relatively small with respect to the number of nodes, from time to time this version of *i*FUB degenerates to the textbook algorithm so that, especially in the case of networks with more 100,000 nodes, the computation can require a huge amount of time and sometimes even days. If we restrict ourselves to the 156 networks having no more than 100,000 nodes, in the case of about 60 networks the average number of BFSes executed over ten experiments is more than 10% of n, and in the case of about 80 networks the maximum number of BFSes executed over ten experiments is more than 20% of n.

Table 3
Performance ratio: *i*FUB with respect to the textbook algorithm. For each kind of networks is reported how many graphs have a ratio v/n, i.e. number of BFSes over number of nodes, less than 0.001, in between 0.001 and 0.01, in between 0.01 and 0.1, and greater than 0.1. In the case of (a), v is the average number of BFSes.

(a) Results obtained by ten executions of *i*FUB+4-SWEEP*r*, i.e. *i*FUB by using 4-SWEEP*r*.

		Number of networks having performance ratio v/n			
Networks	Total	$v/n \le 0.001$	$0.001 < v/n \le 0.01$	$0.01 < v/n \le 0.1$	$0.1 < v/n \le 1$
Autonomous systems graphs	2	2	0	0	0
Biological networks	48	4	22	22	0
Citations networks	5	4	0	1	0
Collaboration networks	13	4	7	2	0
Communication networks	38	26	7	4	1
Internet peer-to-peer networks	1	0	0	0	1
Meshes and electronic circuits	34	5	8	8	13
Product co-purchasing networks	4	4	0	0	0
Road networks	3	1	0	2	0
Social networks	11	9	0	2	0
Synthetic graphs	18	3	5	5	5
Web graphs	9	8	1	0	0
Words adjacency networks	4	1	3	0	0
Others	6	2	1	1	2
Total	196	73	54	47	22

(b) Results obtained by the execution of *i*FUB+4-SWEEP*hd*, i.e. *i*FUB by using 4-SWEEP*hd*.

		Number of networks having performance ratio v/n			
Networks	Total	$v/n \le 0.001$	$0.001 < v/n \le 0.01$	$0.01 < v/n \le 0.1$	$0.1 < v/n \le 1$
Autonomous systems graphs	2	2	0	0	0
Biological networks	48	5	31	12	0
Citations networks	5	3	2	0	0
Collaboration networks	13	5	6	1	1
Communication networks	38	26	8	3	1
Internet peer-to-peer networks	1	0	0	0	1
Meshes and electronic circuits	34	8	7	9	10
Product co-purchasing networks	4	3	1	0	0
Road networks	3	1	0	2	0
Social networks	11	9	0	2	0
Synthetic graphs	18	7	5	2	4
Web graphs	9	9	0	0	0
Words adjacency networks	4	2	2	0	0
Others	6	1	2	2	1
Total	196	81	64	33	18

(c) Results obtained by the execution of *i*FUB+*hd*, i.e. *i*FUB by starting from the node with highest degree.

		Number of networks having performance ratio v/n			
Networks	Total	$v/n \le 0.001$	$0.001 < v/n \le 0.01$	$0.01 < v/n \le 0.1$	$0.1 < v/n \le 1$
Autonomous systems graphs	2	2	0	0	0
Biological networks	48	7	33	8	0
Citations networks	5	4	1	0	0
Collaboration networks	13	8	4	1	0
Communication networks	38	12	15	11	0
Internet peer-to-peer networks	1	0	0	1	0
Meshes and electronic circuits	34	1	2	6	25
Product co-purchasing networks	4	4	0	0	0
Road networks	3	0	0	0	3
Social networks	11	9	2	0	0
Synthetic graphs	18	7	3	0	8
Web graphs	9	7	1	1	0
Words adjacency networks	4	3	1	0	0
Others	6	2	2	0	2
Total	196	66	64	28	38

Table 4
Naturally undirected networks and their largest connected components.

Category	Name	Nodes	Edges	Diameter
Collaboration	`jazz`	198	5 484	6
Biological	`iPfam`	513	18 740	12
Biological	`psimap`	526	19 048	11
Biological	`Rattus_norvegicus`	1 415	3 570	19
Biological	`interdom`	1 654	157 832	8
Biological	`string`	2 575	53 514	9
Words adjacency	`japaneseBookInter_st`	2 698	15 990	8
Collaboration	`geom`	3 621	18 922	14
Biological	`Mus_musculus`	3 745	10 340	20
Biological	`ppi_dip_swiss`	3 766	23 844	12
Biological	`HC-BIOGRID`	4 039	20 642	23
Collaboration	`ca-GrQc`	4 158	26 844	17
Words adjacency	`darwinBookInter_st`	7 377	88 410	8
Words adjacency	`frenchBookInter_st`	8 308	47 664	9
Collaboration	`ca-HepTh`	8 638	49 612	18
Biological	`hprd_pp`	9 219	73 800	14
Collaboration	`ca-HepPh`	11 204	235 238	13
Words adjacency	`spanishBookInter_st`	11 558	86 100	10
Collaboration	`ca-AstroPh`	17 903	393 944	14
Biological	`dip20090126_MAX`	19 928	82 404	30
Collaboration	`ca-CondMat`	21 363	182 572	15
Collaboration	`Cond_mat_95-99`	22 015	117 156	12
Biological	`ppi_gcc`	37 333	271 236	27
Autonomous systems	`itdk0304_rlinks`	190 914	1 215 220	26
Collaboration	`dblp20080824_MAX`	511 163	3 742 140	22
Collaboration	`imdb`	880 455	74 989 272	14
Autonomous systems	`as-skitter`	1 694 616	22 188 418	31

6. Comparing *i*FUB with other algorithms

In this section we will focus our attention on the subset of naturally undirected real-world networks included in our dataset, in order to run a fair comparison with other algorithms. The categories involved are biological networks (in particular, protein interaction networks), collaboration networks, words adjacency networks, and autonomous systems graphs (see Section 4). For each graph, we considered its largest connected component: the restricted dataset is shown in Table 4, where we report the category and the name of the network, and the number of nodes, the number of edges, and the diameter of its largest connected component. We have chosen to use in the experiments the *i*FUB+4-SWEEP*hd* and the *i*FUB+*hd* strategies. Observe that all the networks included in this restricted dataset have one node of maximum degree, apart from `iPfam` (with 5 nodes of maximum degree) and `interdom` (with 8 nodes of maximum degree): in the case of these two networks, we have experimentally verified that the performances of *i*FUB+*hd* and of *i*FUB+4-SWEEP*hd* are independent of the choice of the starting node.

Comparing iFUB *with upper bound computation methods.* The lower bounds provided by the 2-SWEEP or the 4-SWEEP methods turn out to be, in practice, almost always tight: however, there is, in theory, no guarantee about the quality of the approximation. For this reason, some methods have been proposed in [6,8] in order to find an upper bound on the diameter which bounds the absolute error or even validates the tightness of the lower bound. All these methods return the diameter of the BFS tree of a node r, i.e. T_r. In particular, two methods, called RTUB and HTUB respectively, have been presented in [8]: RTUB selects r as a random node, while HTUB chooses r as one node with the highest degree. Thus, both RTUB and HTUB execute two BFSes: one BFS from r is used to create T_r and one BFS is used to compute the diameter of T_r. The other two methods, called MTUB and MTUBHD respectively, are evolutions of a method proposed in [6]: in the case of MTUB, r is the node returned by 4-SWEEP*r*, while in the case of MTUBHD r is the node returned by 4-SWEEP*hd*. Hence, both MTUB and MTUBHD execute six BFSes: four BFSes are used to execute 4-SWEEP and to select r, one BFS from r is used to create T_r, and one BFS is used to compute the diameter of T_r.

Table 5 shows the upper bounds found by using HTUB and MTUBHD, and the randomized methods MTUB and RTUB: in order to run a fair comparison, we have applied the following schema in the case of the randomized methods. Let v be the number of BFSes performed by *i*FUB in order to calculate the diameter: in each experiment, we have repeated at least $\lceil v/6 \rceil$ times the MTUB and RTUB methods. Specifically, we have executed ten experiments: we report the best upper bound over all these experiments in Table 5, along with the number of runs out of ten in which the returned upper bound is tight. Consistently with [6], MTUB and MTUBHD seem to be more effective in finding better upper bounds rather than RTUB and HTUB. However in the great majority of the networks, all these methods are not able to find a tight upper bound (see the rightmost part of Table 5). Finally, it is worth observing that the 2-SWEEP or 4-SWEEP methods return the height of a BFS tree and that, in any graph, there is always at least one node such that the height of its BFS tree is equal to the diameter. On the other hand, the upper bound based methods return the diameter of a BFS tree, and there are infinite graphs such that no nodes have a BFS tree whose diameter is equal to the diameter of the graph (note that this is the case of the graph shown in Fig. 2).

Table 5
Comparing *i*FUB with other methods.

		EXACT ALGORITHMS			UPPER BOUND ALGORITHMS					
		*i*FUB+ 4-SWEEP*hd*	*i*FUB+*hd*	TK	MTUB		MTUBHD	RTUB		HTUB
Name	D	BFSes	BFSes	BFSes	Best UB	# Runs (out of 10) s.t. UB=D	Best UB	Best UB	# Runs (out of 10) s.t. UB=D	Best UB
jazz	6	6	**3**	7	7	0	7	8	0	8
iPfam	12	5	**4**	**4**	12	10	12	12	3	13
psimap	11	8	13	**6**	12	0	12	12	0	12
Rattus_norvegicus	19	8	17	**4**	20	0	20	20	0	23
interdom	8	7	**3**	7	8	6	9	8	1	8
string	9	30	20	**15**	10	0	12	11	0	10
japaneseBookInter_st	8	7	**4**	13	9	0	10	10	0	9
geom	14	23	7	**6**	14	1	15	14	1	16
Mus_musculus	20	26	**3**	6	20	4	21	21	0	22
ppi_dip_swiss	12	7	**3**	4	13	0	13	13	0	14
HC-BIOGRID	23	17	**5**	**5**	24	0	25	23	1	25
ca-GrQc	17	26	**11**	14	20	0	20	19	0	19
darwinBookInter_st	8	5	**3**	4	9	0	8	9	0	8
frenchBookInter_st	9	21	**6**	12	10	0	11	10	0	11
ca-HepTh	18	10	**9**	17	20	0	20	20	0	21
hprd_pp	14	7	**3**	8	16	0	16	16	0	16
ca-HepPh	13	39	**10**	20	15	0	15	15	0	15
spanishBookInter_st	10	5	**2**	4	10	10	10	11	0	11
ca-AstroPh	14	**8**	12	19	15	0	15	17	0	16
dip20090126_MAX	30	8	33	**7**	30	10	31	30	3	34
ca-CondMat	15	31	**6**	14	16	0	18	17	0	17
Cond_mat_95-99	12	4485	**78**	577	15	0	16	15	0	15
ppi_gcc	27	**7**	24	**7**	27	10	27	28	0	30
itdk0304_rlinks	26	9	**11**	**11**	28	0	28	28	0	29
dblp20080824_MAX	22	17	**13**	30	25	0	26	24	0	25
imdb	14	20	**19**	33	16	0	16	16	0	16
as-skitter	31	7	12	**5**	32	0	32	34	0	40

Comparing iFUB with Takes–Kosters algorithm. Recently and independently from this work, a new algorithm to compute the diameter of large real-world networks has been proposed in [38]. The algorithm, which we refer to as TK, maintains a lower bound Δ_L and an upper bound Δ_U of the diameter D and, for each node w, it maintains a lower bound $\varepsilon_L[w]$ and an upper bound $\varepsilon_U[w]$ of its eccentricity $\mathrm{ecc}(w)$. Moreover, it maintains a set W of nodes, at the beginning initialized with V, that are candidate extremes of a path whose length is the diameter. At the beginning all the lower and upper bounds are respectively initialized with 0 and n. At each step the node u, with minimum lower bound or with maximum upper bound, is selected from set W: Δ_L is updated with $\max\{\Delta_L, \mathrm{ecc}(v)\}$, Δ_U is updated with $\min\{\Delta_U, 2\mathrm{ecc}(v)\}$, and, for any node w, $\varepsilon_L[w]$ is updated with $\max\{\varepsilon_L[w], \mathrm{ecc}(v) - d(v, w), d(v, w)\}$ and $\varepsilon_U[w]$ is updated with $\min\{\varepsilon_U[w], \mathrm{ecc}(v) + d(v, w)\}$. Then, the nodes $v \in W$ such that $\varepsilon_U[v] \leq \Delta_L$ and $\varepsilon_L[v] \geq \Delta_U/2$ are removed from W, since a selection of v cannot improve the bounds Δ_L and Δ_U. The algorithm terminates and returns Δ_L when Δ_L is equal to Δ_U or the set W is empty. In Table 5, we report for each network the number of BFSes performed by TK to compute the diameter, by applying the implementation given by the authors. In the case of the 27 networks in our dataset, the more effective approach is *i*FUB+*hd*: it requires less BFSes than TK in the case of 17 networks, and more BFSes than TK in the case of just 7 networks (while the two methods require the same number of BFSes in the remaining 3 networks).[3]

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

[3] A comment is in order on the parallelization of *i*FUB and of TK to speed up the computation: indeed, while the selection process in TK seems to be inherently sequential, the BFSes of the nodes at the same level i required by *i*FUB to compute $B_i(u)$ can be performed in parallel.

executes in $O(m)$ time when applied to real-world networks (and, thus, in $O(n)$ time because of the sparsity of these networks). This notable performance is also, but not only, due to the high *centrality* of the highest-degree nodes in real-world networks and 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 selection of the highest-degree node and the usage of 4-Sweep, limit the variability of the performances of *i*FUB. As a result, the combination of *i*FUB, with the selection of the highest degree node or with 4-Sweep is a quite effective method to compute the diameter of huge real-world networks of unknown diameter at the time of our experiments.

The *i*FUB algorithm has been recently integrated into the WebGraph Java library [9], and it has been used 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 [39]. We also studied how to apply variants of the *i*FUB algorithm to weighted graphs and we refer the reader for a thorough discussion given in [40].

The good performance 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; 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 the 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 have been performed by [41]. 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. Moreover it could be interesting to find a generalized example with arbitrary diameter value in which the lower bound computed by 4-Sweep is about half the diameter for any computation.

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, Journal of Computer and System Sciences 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, 2009. http://snap.stanford.edu.
[6] 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, in: LNCS, vol. 6346, 2010, pp. 302–313.
[7] D. Corneil, F. Dragan, M. Habib, C. Paul, Diameter determination on restricted graph families, Discrete Applied Mathematics 113 (2–3) (2001) 143–166.
[8] C. Magnien, M. Latapy, M. Habib, Fast computation of empirically tight bounds for the diameter of massive graphs, Journal of Experimental Algorithmics 13 (2008).
[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) (2007).
[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, 2010, http://www-mat.upc.es/grup_de_grafs/.
[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, Publicationes Mathematicae Debrecen 6 (1959) 290–297.
[37] M. Penrose, Random Geometric Graphs, in: Oxford Studies in Probability, 2003.

[38] F.W. Takes, W.A. Kosters, Determining the diameter of small world networks, in: Proceedings of the 20th ACM International Conference on Information and Knowledge Management, CIKM'11, ACM, New York, NY, USA, 2011, pp. 1191–1196.
→ [39] L. Backstrom, P. Boldi, M. Rosa, J. Ugander, S. Vigna, Four degrees of separation, arXiv:1111.4570v1.
[40] P. Crescenzi, R. Grossi, L. Lanzi, A. Marino, On computing the diameter of real-world directed (weighted) graphs, in: SEA, 2012, pp. 99–110.
[41] 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.