On Computing the Diameter of Real-World Directed (Weighted) Graphs

Pierluigi Crescenzi[1], Roberto Grossi[2], Leonardo Lanzi[1], and Andrea Marino[1]

[1] Dipartimento di Sistemi e Informatica, Università degli Studi di Firenze, Italy
[2] Dipartimento di Informatica, Università degli Studi di Pisa, Italy

Abstract. In this paper we propose a new algorithm for computing the diameter of directed 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 that in practice our method works in $O(m)$ time. Moreover, we show how to extend our algorithm to the case of directed weighted graphs and, even in this case, we present some preliminary very positive experimental results.

1 Introduction

The analysis of real-world networks such as biological, collaboration, communication, road, social, and web networks has attracted a lot of attention in the last two decades, and many properties of these networks have been studied (see, for example, [21,5,12]). Since the size of real-world networks has been increasing rapidly, in order to study these properties, we need algorithms that can handle huge amount of data. In this paper we will focus our attention on a very basic property of real-world networks, that is, their diameter. Given a directed graph $G = (V, E)$, the diameter of G is the minimum D such that, for any pair of nodes $u, v \in V$, the *distance* $d(u, v)$ between them is at most D, where $d(u, v)$ is the length of the shortest path from u to v (whenever the graph includes a pair of nodes u, v such that $d(u, v) = \infty$, we will study the diameter of its largest strongly connected component).

The diameter is a relevant measure whose meaning depends on the semantics of the real-world network. In the case of social networks, in which every node is an individual and the edges represent their social relationships, the diameter can indicate how quickly information reaches every individual in the worst case, and it has been studied for several social networks (see, for example, [26,20,18]). In the case of web networks, in which every node corresponds to a web page and the edges correspond to hyper-links, the diameter indicates how quickly (in terms of mouse clicks) any page can be reached in the worst case: for several web networks, the diameter has been considered, for example, in [6,18,15]. In the case of communication networks, in which every node is a device and the edges represent communication links, the diameter indicates, for example, the completion time of broadcast protocols based on network flooding. In the case of biological networks, finally, the cellular metabolism is represented by a network

R. Klasing (Ed.): SEA 2012, LNCS 7276, pp. 99–110, 2012.

of metabolites linked by biochemical reactions, and the diameter indicates how many reactions have to be performed, in the worst case, in order to produce any metabolite from any other metabolite [13]: for several such biological networks, the diameter has been studied, for example, in [2,18].

Because of the huge size of real-world networks, in almost all the above cases, the diameter of the strongly connected components has been only estimated. Indeed, most algorithms for finding the exact diameter solve the *all pair shortest path* problem, that is the problem of finding the shortest path between all pairs of nodes of the graph: this can be done either by applying *text-book algorithms* (such as breadth-first searches) for solving, for any node, the *single source shortest path* problem, or by applying fast matrix multiplication algorithms with subcubic complexity (see, for example, [20]). However in the context of real-world networks, these approaches are not practical and usually just estimations or bounds can be provided.

To this aim, some algorithms have been proposed in order to estimate the cumulative distribution of the shortest path lengths of a graph and to thus obtain an estimation of the diameter with a small additive error: this is the case, for example, of the algorithms proposed in [22,4,14]. In other works lower bounds for the diameter have been provided by using a *sample* of the nodes and returning their maximum eccentricity, where the eccentricity of a node u is defined as $\max_{v \in V} d(u, v)$ (see, for example, [17]). In the case of *undirected* graphs a lower bound can also be provided by using the so called *double sweep* algorithm, in short 2-Sweep: pick the farthest node from a random node and return its eccentricity. This idea can be iterated by picking at each step the farthest node from the previous one and maintaining the highest found eccentricity (see, for example, [18]). In real-world networks, this lower bound is very good and, in order to prove its effectiveness, several works, like [16,10], propose strategies to find a matching or close upper bound. Recent advances [9,24] have experimentally shown that in real-world networks a matching between a lower and upper bound for the diameter can be found by applying a very small number of computations of breadth-first searches. The most striking result, along this line of research, has been obtained in [1], where the algorithm proposed in [9] has been applied in order to compute the diameter of the biggest connected component of the Facebook network (approximately 721.1M of nodes and 68.7G of edges). In the case of *directed* graphs, in order to obtain a lower bound for the diameter, the idea of the *double sweep* has been adapted by [6]: pick the farthest node from a random node and return its backward eccentricity, i.e. its eccentricity in the graph with reversed arcs (we will make use of this adaptation in this paper).

In this paper we generalize the idea of the algorithm proposed in [9], by presenting the directed *i*FUB (in short, D*i*FUB) algorithm, in order to calculate the diameter of the strongly connected components of directed graphs. As far as we know, D*i*FUB is the first algorithm which is able to compute exactly the diameter of the strongly connected components of huge real-world directed graphs. The D*i*FUB algorithm can also return a pair of nodes whose distance is exactly equal to the diameter, and a natural adaptation of it works also for weighted graphs.

The algorithm uses very basic operations that are provided basically by all graph software libraries. Indeed, the main operation is the breadth-first search (in short, BFS): since BFS has a good external-memory implementation [19] and works on graphs stored in compressed format [3], we have been able to analyze very large graphs. As a matter of fact, we show the effectiveness of the DiFUB algorithm with a wide set of experiments, by using real-world directed networks which have been chosen in order to cover a large set of network typologies, and have been already used to validate other popular tools, such as [22,14,3,4]. As we already said, in the case of several of these networks, the exact value of the diameter of the largest strongly connected component was still unknown: in almost all the graphs with more than 10000 nodes, the number of BFSes executed by the algorithm is less than 0.01% of the total number of nodes in the component.

Structure of the Paper. In Section 2 we present and analyze the DiFUB algorithm in the case of directed unweighted strongly connected graphs, while in Section 3 we present our dataset and we show the results of our experiments. In Section 4 we extend the DiFUB algorithm to the case of weighted graphs and we briefly describe the corresponding experimental results. We conclude in Section 5 by proposing some interesting directions for future research.

2 The DiFUB Algorithm

Let $G = (V, E)$ be a directed strongly connected graph and let u be any node in V. We denote by T_u^F (respectively, T_u^B) a forward (respectively, backward) breadth-first search tree rooted at node u, and by $\mathrm{ecc}_F(u)$ (respectively, $\mathrm{ecc}_B(u)$) its height. Let $F_i^F(u)$ be the *forward fringe* of u, that is, the set of nodes x such that $d(u, x) = i$. Similarly, let $F_i^B(u)$ be the *backward fringe*, that is, the set of nodes x such that $d(x, u) = i$. In other words, $F_i^F(u)$ (respectively, $F_i^B(u)$) includes all nodes at level i of T_u^F (respectively, T_u^B).

Remark 1. For any two integers i, j with $1 \leq i \leq ecc_B(u)$ and $1 \leq j \leq ecc_F(u)$, for any two nodes x, y such that $x \in F_i^B(u)$ and $y \in F_j^F(u)$, $d(x, y) \leq i + j \leq 2\max\{i, j\}$.

Theorem 1. *For any integer i with $1 < i \leq \mathrm{ecc}_B(u)$, for any integer k with $1 \leq k < i$, and for any node $x \in F_{i-k}^B(u)$ such that $\mathrm{ecc}_F(x) > 2(i-1)$, there exists $y \in F_j^F(u)$, for some $j \geq i$, such that $d(x, y) = \mathrm{ecc}_F(x)$.*

Proof. Since $\mathrm{ecc}_F(x) > 2(i-1)$, then there exists y such that $d(x, y) > 2(i-1)$. If y was in $F_j^F(u)$ with $j < i$, then from Remark 1 it would follow that $d(x, y) \leq 2\max\{i-k, j\} \leq 2\max\{i-k, i-1\} = 2(i-1)$, which is a contradiction. Hence, y must be in $F_j^F(u)$ with $j \geq i$.

Similarly to the proof of Theorem 1, we can also prove the following symmetrical result.

Theorem 2. *For any integer i with $1 < i \leq \mathrm{ecc}_F(u)$, for any integer k with $1 \leq k < i$, and for any node $x \in F^F_{i-k}(u)$ such that $\mathrm{ecc}_B(x) > 2(i-1)$, there exists $y \in F^B_j(u)$, for some $j \geq i$, such that $d(y,x) = \mathrm{ecc}_B(x)$.*

In order to describe the DiFUB algorithm, we also need the following definitions. Let

$$B^F_j(u) = \begin{cases} \max_{x \in F^F_j(u)} \mathrm{ecc}_B(x) & \text{if } j \leq \mathrm{ecc}_F(u), \\ 0 & \text{otherwise} \end{cases}$$

and

$$B^B_j(u) = \begin{cases} \max_{x \in F^B_j(u)} \mathrm{ecc}_F(x) & \text{if } j \leq \mathrm{ecc}_B(u), \\ 0 & \text{otherwise.} \end{cases}$$

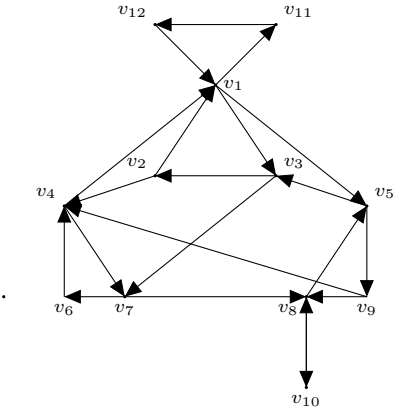
By using these two definitions, we are now ready to introduce the DiFUB algorithm, which is shown in Pseudocode 1. Intuitively, Theorems 1 and 2 suggest to perform a forward and a backward BFS from a node u, and to visit T^F_u and T^B_u in a bottom-up fashion, starting from the nodes in the last fringes. For each level i, we compute the eccentricities of all the nodes in the corresponding fringes: if the maximum eccentricity is greater than $2(i-1)$ then we can discard visiting the remaining levels, since the eccentricities of all their nodes cannot be greater.

Pseudocode 1. DiFUB

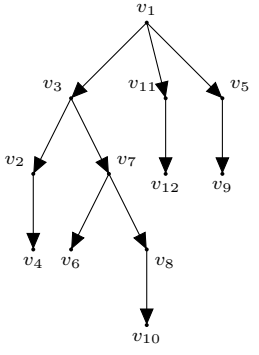
Input: A strongly connected di-graph G, a node u, a lower bound l for the diameter
Output: The diameter D

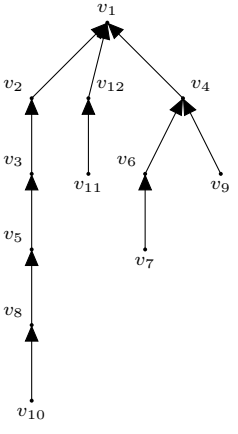
$i \leftarrow \max\{\mathrm{ecc}_F(u), ecc_B(u)\}$; $lb \leftarrow \max\{\mathrm{ecc}_F(u), ecc_B(u), l\}$; $ub \leftarrow 2i$;
while $ub - lb > 0$ **do**
  if $\max\{lb, B^B_i(u), B^F_i(u)\} > 2(i-1)$ **then**
    return $\max\{lb, B^B_i(u), B^F_i(u)\}$;
  else
    $lb \leftarrow \max\{lb, B^B_i(u), B^F_i(u)\}$; $ub \leftarrow 2(i-1)$;
  end
  $i \leftarrow i - 1$;
end
return lb;

An Example. Let us consider the graph shown in the top left part of Figure 1. All pairwise distances, and the forward and backward eccentricities of all its nodes are shown in the top right part of the figure. If we choose $u = v_1$, the corresponding two breadth-first search trees T^F_u and T^B_u are shown in the bottom left part of the figure. From these two trees we can easily derive the forward and backward fringe sets, which are shown in the bottom right part of the figure. If we choose $i = 2$, $j = 3$, $x = v_6$, and $y = v_8$, then it is easy to verify, by inspecting the two BFSes trees, that we can go from v_6 to v_8 by first going up in $T^B_{v_1}$ (by means of two edges) and then by going down in $T^F_{v_1}$ (by means of three edges). Hence,



	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	v_9	v_{10}	v_{11}	v_{12}	ecc_F
v_1	0	2	1	3	1	3	2	3	2	4	1	2	4
v_2	1	0	2	1	2	3	2	3	3	4	2	3	4
v_3	2	1	0	2	3	2	1	2	4	3	3	4	4
v_4	1	3	2	0	2	2	1	2	3	3	2	3	3
v_5	3	2	1	2	0	3	2	2	1	3	4	5	5
v_6	2	4	3	1	3	0	2	3	4	4	3	4	4
v_7	3	4	3	2	2	1	0	1	3	2	4	5	5
v_8	4	3	2	3	1	4	3	0	2	1	5	6	6
v_9	2	4	3	1	2	3	2	1	0	2	3	4	4
v_{10}	5	4	3	4	2	5	4	1	3	0	6	7	7
v_{11}	2	4	3	5	3	5	4	5	4	6	0	1	6
v_{12}	1	3	2	4	2	4	3	4	3	5	2	0	5
ecc_B	5	4	3	5	3	5	4	5	4	6	6	7	





i	$F_i^F(v_1)$	$F_i^B(v_1)$
1	v_3, v_5, v_{11}	v_2, v_4, v_{12}
2	v_2, v_7, v_9, v_{12}	v_3, v_6, v_9, v_{11}
3	v_4, v_6, v_8	v_5, v_7
4	v_{10}	v_8
5		v_{10}

Fig. 1. A strongly connected di-graph with the corresponding all pairwise distances, forward and backward eccentricities and BFSes trees rooted at v_1, and fringe sets

as observed in Remark 1, $d(v_6, v_8) \leq 5$: indeed, $d(v_6, v_8) = 3$ (passing through v_4 and v_7). Moreover, if we choose $i = 2$, $k = 1$, and $x = v_4 \in F_1^B(v_1)$, then we have that $\text{ecc}_F(v_4) = 3 > 2 = 2(i-1)$: Theorem 1 is in this case witnessed by node $y = v_2 \in F_2^F(v_1)$ (indeed, $d(v_4, v_2) = 3$). If we choose $j = 1$, we have that $B_1^F(u) = \max\{\text{ecc}_B(v_3), \text{ecc}_B(v_5), \text{ecc}_B(v_{11})\} = \max\{3, 6\} = 6$. On the other hand, $B_1^B(u) = \max\{\text{ecc}_F(v_2), \text{ecc}_F(v_4), \text{ecc}_F(v_{12})\} = \max\{3, 4, 5\} = 5$. Finally, suppose we invoke the algorithm shown in Pseudocode 1 with $u = v_1$ and $l = 0$. Before the execution of the **while** loop starts, the two variables i and lb are both set equal to $\max\{\text{ecc}_F(v_1), \text{ecc}_B(v_1)\} = \max\{4, 5\} = 5$, while variable ub is set equal to $2i = 10$. Since $ub - lb = 5 > 0$, the algorithm enters the **while** loop with $i = 5$. Since $5 > \text{ecc}_F(u)$, $B_5^F(u) = 0$. On the other hand, $B_5^B(u) = \text{ecc}_F(v_{10}) = 7$: since, $7 < 8 = 2(i-1)$, the algorithm enters the **else** branch and set lb equal to 7 and ub equal to 8. Once again, $ub - lb = 1 > 0$

and the algorithm continues the execution of the **while** loop with $i = 4$. This time we have that $B_4^F(u) = \mathrm{ecc}_B(v_{10}) = 6$ and $B_4^B(u) = \mathrm{ecc}_F(v_8) = 6$: hence, $\max\{lb, B_4^B(u), B_4^F(u)\} = 7 > 6 = 2(i-1)$. The algorithm thus enters the **if** branch and returns the value 7 which is the correct diameter value. In other words, the diameter has been computed by exploring only three nodes (apart from v_1): note that we did not really need to compute $B_4^B(u)$ and $B_4^F(u)$, since l was already greater than $2(i-1)$.

Theorem 3. DiFUB *Algorithm correctly computes the value of the diameter of* G.

Proof (Sketch). Let us prove that if, at the iteration corresponding to a given value i, $\max\{lb, B_i^B(u), B_i^F(u)\} > 2(i-1)$ then the diameter of G is equal to $\max\{lb, B_i^B(u), B_i^F(u)\}$. By contradiction, assume that the diameter is greater than $\max\{lb, B_i^B(u), B_i^F(u)\}$ (note that the diameter cannot be smaller than $\max\{lb, B_i^B(u), B_i^F(u)\}$ since this value is the length of a shortest path). This implies that there exists $x \in F_{i-k}^B(u) \cup F_{i-k}^F(u)$ such that $\max\{\mathrm{ecc}_F(x), \mathrm{ecc}_B(x)\} = D > 2(i-1)$. From the previous two theorems, it follows that there exists $y \in F_j^F(u) \cup F_j^B(u)$ such that $\max\{d(x,y), d(y,x)\} = \max\{\mathrm{ecc}_F(x), \mathrm{ecc}_B(x)\} = D$ with $j \geq i$, thus contradicting the fact that $D > \max\{lb, B_i^B(u), B_i^F(u)\}$ (note that $lb \geq \max\{B_j^B(u), B_j^F(u)\}$ for any $j > i$).

The time complexity of DiFUB can be in the worst case $O(nm)$ where n denotes the number of nodes and m denotes the number of edges. However, the practical performance of the algorithm depends on the chosen node u. 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 $\max\{\lceil \mathrm{ecc}_B(u)/2 \rceil, \lceil \mathrm{ecc}_F(u)/2 \rceil\}$ iterations (note that we have that the number of iterations is bounded by $D/2$). A hopefully good starting point u (and a corresponding lower bound l) can be obtained by applying the following heuristics, called 2-*d*SWEEP, which is a natural extension to directed graphs of the 2-SWEEP method (in the following, the *middle* node between two nodes s and t is defined as the node belonging to the shortest path from s to t, whose distance from s is $\lceil d(s,t)/2 \rceil$).

1. Run a forward BFS from a random node r: let a_1 be the farthest node.
2. Run a backward BFS from a_1: let b_1 be the farthest node.
3. Run a backward BFS from r: let a_2 be the farthest node.
4. Run a forward BFS from a_2: let b_2 be the farthest node.
5. If $\mathrm{ecc}_B(a_1) > \mathrm{ecc}_F(a_2)$, then set u equal to the middle node between a_1 and b_1 and l equal to $\mathrm{ecc}_B(a_1)$. Otherwise, set u equal to the middle node between a_2 and b_2 and l equal to $\mathrm{ecc}_F(a_2)$.

3 Experiments

We collected several real-world directed graphs, which have been chosen in order to cover the largest possible set of network typologies. In particular, we

Table 1. For any graph, a summary of 10 executions of 2-dSWEEP (3rd and 4th columns) and DiFUB (7th and 8th columns)

Network name	D	Numb. of runs (out of 10) in which $l = D$	Worst l found	n	m	Avg. Numb. of Visits	Visits in the worst run
Wiki-Vote	9	10	9	1300	39456	17	17
p2p-Gnutella08	19	9	18	2068	9313	45.9	64
p2p-Gnutella09	19	9	18	2624	10776	202.1	230
p2p-Gnutella06	19	10	19	3226	13589	236.6	279
p2p-Gnutella05	22	9	21	3234	13453	60.4	94
p2p-Gnutella04	25	7	22	4317	18742	36.7	38
p2p-Gnutella25	21	8	20	5153	17695	85.1	161
p2p-Gnutella24	28	10	28	6352	22928	13	13
p2p-Gnutella30	23	2	22	8490	31706	255.4	516
p2p-Gnutella31	30	9	29	14149	50916	208.7	255
s.s.Slashdot081106	15	10	15	26996	337351	22.3	25
s.s.Slashdot090216	15	10	15	27222	342747	21.5	26
s.s.Slashdot090221	15	10	15	27382	346652	22.8	26
soc-Epinions1	16	9	15	32223	443506	6.1	7
Email-EuAll	10	10	10	34203	151930	6	6
soc-sign-epinions	16	10	16	41441	693737	6	6
web-NotreDame	93	10	93	53968	304685	7	7
Slashdot0811	12	10	12	70355	888662	40	40
Slashdot0902	13	3	12	71307	912381	32.9	40
WikiTalk	10	9	9	111881	1477893	13.6	19
web-Stanford	210	10	210	150532	1576314	6	6
web-BerkStan	679	10	679	334857	4523232	7	7
web-Google	51	10	51	434818	3419124	9.4	10
wordassociation-2011	10	9	9	4845	61567	412.5	423
enron	10	10	10	8271	147353	19	22
uk-2007-05@100000	7	10	7	53856	1683102	14	14
cnr-2000	81	10	81	112023	1646332	17	17
uk-2007-05@1000000	40	10	40	480913	22057738	6	6
in-2004	56	10	56	593687	7827263	14	14
amazon-2008	47	10	47	627646	4706251	136.3	598
eu-2005	82	10	82	752725	17933415	6	6
indochina-2004	235	10	235	3806327	98815195	8	8
uk-2002	218	10	218	12090163	232137936	6	6
arabic-2005	133	10	133	15177163	473619298	58	58
uk-2005	166	10	166	25711307	704151756	170	170
it-2004	873	10	873	29855421	938694394	87	87

used web graphs, communication networks, product co-purchasing networks, autonomous systems graphs, Internet peer-to-peer networks, social networks, road networks, and words adjacency networks (see Table 1). All these networks have been downloaded either from [27] or from [23]. As it can be seen in the fifth and sixth columns of the table, an important feature is that almost all graphs in our dataset are sparse (that is, $m = O(n)$). Finally, note that, in the case of several of these graphs, the diameter value was still unknown.

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. We have performed 10 experiments on the biggest strongly connected component of each of the 36 networks, for a total of 360 experiments. The code and the data set are available at `http://piluc.dsi.unifi.it/lasagne/`.

3.1 Obtaining a Tight Lower Bound via 2-dSWEEP

For each of the analyzed graphs, we executed the 2-*d*SWEEP algorithm ten times.[1] A summary of the results obtained by ten executions of the 2-*d*SWEEP algorithm for the [23] dataset (upper part) and the [27] dataset (lower part) is shown in the middle part of Table 1. For each graph, the diameter (D), the number of runs in which the lower bound l returned by the 2-*d*SWEEP algorithm is equal to the diameter D, and the worst lower bound returned among the ten experiments are shown. It is worth noting that, in the case of web graphs and communication networks, the obtained lower bound is tight for any experiment, while in the case of the other networks, apart from `p2p-Gnutella04`, the absolute error, in all the ten experiments, is at most 1.

3.2 Obtaining the Diameter via DiFUB

For each of the analyzed graphs, we executed the D*i*FUB algorithm ten times: the results are summarized in the rightmost part of Table 1. In particular, we report the average number of visits performed by D*i*FUB in order to obtain the diameter, and the largest number of visits performed by D*i*FUB among the ten experiments. Observe that given a graph with n nodes, the number of BFSes may range between 6 (that is, the case in which the algorithm shown in Pseudocode 1 returns the exact value of the diameter without entering the **while** loop), and $O(n)$ (that is, the case in which the algorithm degenerates in the text-book algorithm). The ratio between the number of BFSes performed by the algorithm and the number of nodes gives us the fraction of visits performed by our algorithm with respect to the worst case. This *performance ratio* in almost any graph with more than 10000 nodes is less than 0.01%. Moreover it is worth observing that this ratio seems to decrease with respect to n, as shown in Figure 2: we argue that the number of visits performed is asymptotically constant. In particular, in the figure the ratio between the average number of visits and the number of nodes is reported in log scale: from the figure is clear that our *gain*, that is the ratio between the number of nodes and the average number of visits, increases exponentially with the size of the graph.

4 The DiFUB Algorithm for Weighted Graphs

Theorem 1 and 2 can be easily extended to the case of directed weighted graphs. Indeed, let T_u^F (respectively, T_u^B) denote the forward (respectively, backward) lightest path tree rooted at node u, computed, for instance, by means of the Dijkstra algorithm [11] in G (respectively, in G by reversing the orientation of the arcs). Moreover, let $\mathrm{ecc}^F(u)$ (respectively, $\mathrm{ecc}^B(u)$) denote the weighted forward (respectively, backward) eccentricity of u, that is the weight of the longest

[1] No significant variance was observed also on more experiments because central nodes are easily detected by the 2-*d*SWEEP algorithm: they usually are the same in all the experiments, even if we perform random choices.

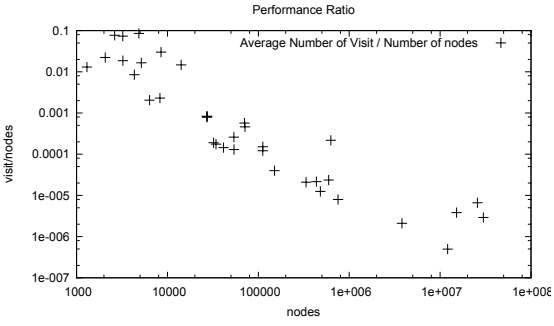


Fig. 2. For any number of nodes n, on the x-axis, the average number of visits performed by DiFUB normalized with respect to n, is reported on the y-axis in log scale (each point corresponds to a graph in Table 1)

path from (respectively, to) u to (respectively, from) one of the leaves of T_u^F (respectively, T_u^B). Finally, let $F_d^F(u)$ (respectively, $F_d^B(u)$) denote the set of nodes whose weighted distance from (respectively, to) u is equal to d: hence, $F_d^F(u) \neq \emptyset$ if and only if there exists at least one node x in T_u^F such that the weight of the path from u to x is equal to d, and $F_d^B(u) \neq \emptyset$ if and only if there exists at least one node x in T_u^B such that the weight of the path from x to u is equal to d.

Let $d_1, d_2, \ldots, d_h$ be the sequence of distinct values d such that $F_d^F(u) \neq \emptyset$ or $F_d^B(u) \neq \emptyset$ ordered in increasing order, that is, $d_1 < d_2 < \cdots < d_h$: note that $d_h = \max\{\mathrm{ecc}_F(u), \mathrm{ecc}_B(u)\}$. We then have the following two results, whose proofs are similar to the proofs of Theorems 1 and 2, respectively.

Theorem 4. *For any integer i with $1 < i \leq h$, for any integer k with $1 \leq k < i$, and for any node $x \in F_{d_{i-k}}^B(u)$ such that $\mathrm{ecc}_F(x) > 2d_{i-1}$, there exists $y \in F_{d_j}^F(u)$, for some $d_j \geq d_i$, such that $d(x, y) = \mathrm{ecc}_F(x)$.*

Theorem 5. *For any integer i with $1 < i \leq h$, for any integer k with $1 \leq k < i$, and for any node $x \in F_{d_{i-k}}^F(u)$ such that $\mathrm{ecc}_B(x) > 2d_{i-1}$, there exists $y \in F_{d_j}^B(u)$, for some $d_j \geq d_i$, such that $d(y, x) = \mathrm{ecc}_B(x)$.*

We can then appropriately modify the DiFUB algorithm in order to deal with directed weighted graphs. To this aim, we define

$$B_{d_i}^F(u) = \begin{cases} \max_{x \in F_{d_i}^F(u)} \mathrm{ecc}_B(x) & \text{if } F_{d_i}^F(u) \neq \emptyset \text{ and } d_i \leq \mathrm{ecc}_F(u), \\ 0 & \text{otherwise} \end{cases}$$

and

$$B_{d_j}^B(u) = \begin{cases} \max_{x \in F_{d_j}^B(u)} \mathrm{ecc}_F(x) & \text{if } F_{d_j}^B(u) \neq \emptyset \text{ and } d_j \leq \mathrm{ecc}_B(u), \\ 0 & \text{otherwise.} \end{cases}$$

The DiFUB algorithm for directed weighted graphs is then described in Pseudocode 2: observe that, in order to start the execution of the algorithm, we

Pseudocode 2. DiFUB for weighted directed graphs

Input: A weighted directed strongly connected graph G, a node u, a lower bound for the diameter l
Output: The diameter D
Let $d_1 < d_2 < \ldots < d_h$ be the sequence of values d such that $F_d^{\mathrm{F}}(u) \neq \emptyset$ or $F_d^{\mathrm{B}}(u) \neq \emptyset$
$i \leftarrow h$; $lb \leftarrow \max\{\mathrm{ecc}_F(u), ecc_B(u), l\}$; $ub \leftarrow 2d_i$;
while $ub - lb > 0$ **do**
  if $\max\{lb, B^B_{d_i}(u), B^F_{d_i}(u)\} > 2d_{i-1}$ **then**
    return $\max\{lb, B^B_{d_i}(u), B^F_{d_i}(u)\}$;
  else
    $lb \leftarrow \max\{lb, B^B_{d_i}(u), B^F_{d_i}(u)\}$; $ub \leftarrow 2d_{i-1}$;
  end
  $i \leftarrow i - 1$;
end
return lb;

can also modify the 2-dSWEEP algorithm by using single source lightest path algorithm executions instead of BFSes.

4.1 Dataset and Experiments

We have experimented the modification of the DiFUB and 2-dSWEEP algorithms on several directed weighted real-world graphs: these graphs have been downloaded either from the weighted network dataset available at [25] or from the web site of the 9th DIMACS Implementation Challenge on shortest paths [7]. For the sake of brevity, we do not fully report these experimental results, but we limit ourselves to observe that, in most of the cases, the performances are very similar to the experiments on unweighted graphs. The only significant exceptions concern some of the road networks taken from the [7] dataset: in these cases, the number of performed single source shortest path computations is a quite large fraction (more than 50%) of the total number of nodes. Apart from these exceptions, our results turn out to be very promising.

5 Conclusion and Open Questions

In this paper we have described and experimented a new algorithm for computing the diameter of directed (weighted) graphs. Even though the algorithms has $O(nm)$ time complexity in the worst case, our experiments suggests that its execution for real-world networks requires time $O(m)$.

The performance of our algorithm depends on the choice of the starting node u (indeed, it could be interesting to experimentally analyze its behavior depending on this choice). Ideally, u should be a "center" of the graph G, that is, the maximum between the forward and the backward eccentricity of u should be close to the radius R of the graph (which is defined as $R = \min_{v \in V}\{\max\{ecc_F(v), ecc_B(v)\}\}$). Surprisingly, we have observed that in

the case of real-world graphs, R is close to the minimum possible, that is $D/2$. This peculiar structural property affects the performance of our algorithm: since the minimum number of iterations performed by DiFUB is obtained whenever the starting node u is a center of the graph, we have that, in this case, the upper bound on the iterations is minimum and equal to $R - D/2 + 1$. Since the radius is approximately half the diameter and, as shown by our experiments, the double sweep seems to be very effective in order to find central nodes, peripheral nodes are discovered during the first iterations.

The main fundamental questions are now the following. Why the double sweep, both in the directed and in the undirected version, is so effective in finding tight lower bounds for the diameter and nodes with low eccentricity? Which is the topological underlying property that can lead us to these results? Why real world graphs exhibit this property? Some progress has been done by [8], but still a lot has to be done. Finally, it could be interesting to analyze a parallel implementation of the DiFUB algorithm. Indeed, the eccentricities of the nodes belonging to the same fringe set can be computed in parallel. Moreover, a variety of parallel BFS algorithms have been explored in the literature and can be integrated in the implementation of our algorithm.

References

1. Backstrom, L., Boldi, P., Rosa, M., Ugander, J., Vigna, S.: Four Degrees of Separation (2011) arXiv:1111.4570v1
2. Bansal, S., Khandelwal, S., Meyers, L.: Exploring biological network structure with clustered random networks. BMC Bioinformatics 10(1), 405+ (2009)
3. Boldi, P., Vigna, S.: The WebGraph Framework I: Compression Techniques. In: Proceedings of the 13th International World Wide Web Conference, pp. 595–601. ACM Press, Manhattan (2003)
4. Boldi, P., Rosa, M., Vigna, S.: Hyperanf: approximating the neighbourhood function of very large graphs on a budget. In: WWW, pp. 625–634 (2011)
5. Brandes, U., Erlebach, T.: Network Analysis: Methodological Foundations. Springer (2005)
6. Broder, A.Z., Kumar, R., Maghoul, F., Raghavan, P., Rajagopalan, S., Stata, R., Tomkins, A., Wiener, J.L.: Graph structure in the web. Computer Networks 33(1-6), 309–320 (2000)
7. 9th DIMACS Implementation Challenge - Shortest Paths (2006), `http://www.dis.uniroma1.it/~challenge9/`
8. Chepoi, V., Dragan, F., Estellon, B., Habib, M., Vaxès, Y.: Diameters, centers, and approximating trees of delta-hyperbolic geodesic spaces and graphs. In: Proceedings of the 24th Annual Symposium on Computational Geometry, SCG 2008, pp. 59–68. ACM, New York (2008)
9. Crescenzi, P., Grossi, R., Habib, M., Lanzi, L., Marino, A.: On Computing the Diameter of Real-World Undirected Graphs. Presented at Workshop on Graph Algorithms and Applications (Zurich–July 3, 2011) and selected for submission to the special issue of Theoretical Computer Science in honor of Giorgio Ausiello in the occasion of his 70th birthday (2011)
10. Crescenzi, P., Grossi, R., Imbrenda, C., Lanzi, L., Marino, A.: Finding the Diameter in Real-World Graphs. In: de Berg, M., Meyer, U. (eds.) ESA 2010, Part I. LNCS, vol. 6346, pp. 302–313. Springer, Heidelberg (2010)

11. Dijkstra, E.: A note on two problems in connexion with graphs. Numerische Mathematik 1, 269–271 (1959)
12. Havlin, S., Cohen, R.: Complex Networks: Structure, Robustness and Function. Cambridge University Press, Cambridge (2010)
13. Junker, B.O.H., Schreiber, F.: Analysis of Biological Networks. Wiley Series in Bioinformatics. Wiley Interscience (2008)
14. Kang, U., Tsourakakis, C.E., Appel, A.P., Faloutsos, C., Leskovec, J.: Hadi: Mining radii of large graphs. TKDD 5(2), 8 (2011)
15. Kang, U., Tsourakakis, C.E., Faloutsos, C.: PEGASUS: A Peta-Scale graph mining system implementation and observations. In: 2009 Ninth IEEE International Conference on Data Mining, pp. 229–238. IEEE (December 2009)
16. Latapy, M., Magnien, C.: Measuring Fundamental Properties of Real-World Complex Networks. CoRR abs/cs/0609115 (2006)
17. Leskovec, J., Faloutsos, C.: Sampling from large graphs. In: Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD 2006, pp. 631–636. ACM, New York (2006)
18. Leskovec, J., Lang, K.J., Dasgupta, A., Mahoney, M.W.: Community structure in large networks: Natural cluster sizes and the absence of large well-defined clusters. Internet Mathematics 6(1), 29–123 (2009)
19. Mehlhorn, K., Meyer, U.: External-Memory Breadth-First Search with Sublinear I/O. In: Möhring, R.H., Raman, R. (eds.) ESA 2002. LNCS, vol. 2461, pp. 723–735. Springer, Heidelberg (2002)
20. Mislove, A., Marcon, M., Gummadi, K.P., Druschel, P., Bhattacharjee, B.: Measurement and analysis of online social networks. In: Proceedings of the 7th ACM SIGCOMM Conference on Internet Measurement, IMC 2007, pp. 29–42. ACM, New York (2007)
21. Newman, M.E.J.: The structure and function of complex networks. SIAM Review 45, 167–256 (2003)
22. Palmer, C.R., Gibbons, P.B., Faloutsos, C.: 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, pp. 81–90 (2002)
23. SNAP: Stanford Network Analysis Package (SNAP) (2009), `http://snap.stanford.edu`
24. Takes, F.W., Kosters, W.A.: Determining the diameter of small world networks. In: CIKM, pp. 1191–1196 (2011)
25. Network datasets (2009), `http://toreopsahl.com/datasets/`
26. Wang, F., Moreno, Y., Sun, Y.: Structure of peer-to-peer social networks. Phys. Rev. E 73, 036123 (2006)
27. WebGraph: WebGraph (2001), `http://webgraph.dsi.unimi.it/`