# An Exact Algorithm for Diameters of Large Real Directed Graphs

Takuya Akiba, Yoichi Iwata, and Yuki Kawata(✉)

The University of Tokyo, Bunkyo 113-0033, Japan
{t.akiba,y.iwata}@is.s.u-tokyo.ac.jp,
kawatea@lager.is.s.u-tokyo.ac.jp

**Abstract.** We propose a new algorithm to compute the diameters of large real directed graphs. In contrast to recent algorithms, the proposed algorithm is designed for general directed graphs, i.e., it does not assume that given graphs are undirected or strongly connected. Experimental results on large real graphs show that the proposed algorithm is several orders of magnitude faster than the naive approach, and it reveals the exact diameters of large real directed graphs, for which only lower bounds have been known.

## 1 Introduction

The *diameter* of a graph is the distance between the most distant pairs of vertices. Because of the connection to the *small-world phenomenon*, the diameters of real-world graphs have been of interest in sociology and network science [1,3,11,14].

However, even with the recent availability of large real graph data, the diameters of such graphs have remained unclear because computing diameters has been too computationally expensive. In theory, currently no algorithm has better time complexity than all-pairs shortest path algorithms. Our target graphs are sparse unweighted graphs; thus, this approach corresponds to a naive algorithm that conducts a breadth first search (BFS) from every vertex, which is obviously too slow for large graphs with millions of vertices.

Therefore, practical heuristic algorithms that are tailored to real graphs have been studied [2,5–7,10,12]. Table 1 summarizes such previous methods. Experiments on real-world networks have shown that these methods work surprisingly well; they can successfully give accurate bounds or exact values of diameters, especially for undirected or strongly connected graphs. Nevertheless, the table highlights a lack of practical exact algorithms for general (i.e., not necessarily strongly connected) directed graphs. On the other hand, from a practical perspective, it is of great interest to reveal the exact values of the diameters of large real directed graphs, such as web graphs and some online social networks.

In this study, we propose a new algorithm to compute the diameters of large real directed graphs. The proposed algorithm is designed for general directed graphs, i.e., it does not assume that given graphs are undirected or strongly


E. Bampis (Ed.): SEA 2015, LNCS 9125, pp. 56–67, 2015.
DOI: 10.1007/978-3-319-20086-6_5

**Table 1.** Previous empirical algorithms for graph diameters

| Method | Graph | Output |
|---|---|---|
| ***Classic Methods:*** | | |
| Naive (all-pairs) | any | exact value |
| Random sampling | any | lower bound |
| Double sweep [4,8] | any | lower bound |
| ***Recent Methods:*** | | |
| Magnien, et al., 2009[10] | undirected | upper and lower bound |
| Crescenzi, et al., 2010 [6] | undirected | upper and lower bound |
| Takes and Kosters, 2011 [12] | undirected | exact value |
| Crescenzi, et al., 2012 [7] | strongly connected | exact value |
| Crescenzi, et al., 2013 [5] | undirected | exact value |
| Borassi, et al., 2014 [2] | strongly connected | exact value |
| This work | any | exact value |

connected. The main technical challenge is to efficiently deal with unreachable vertex pairs, which do not exist in undirected or strongly connected graphs. As demonstrated by our experimental results, the proposed algorithm works quite well on large real directed graphs, and, to the best of our knowledge, it has yielded a list of precise diameters of famous large real graph datasets for the first time.

**Contributions.** Our technical contributions are as follows:

1. We propose the first practical algorithm that can compute the exact diameters of large real directed graphs with millions of vertices and edges.
2. Using the proposed algorithm, we present the first precise list of the diameters of famous large real directed graph datasets from SNAP [9].
3. Using the exact diameter values, the accuracy of classic approximation heuristics is evaluated on large real directed graphs for the first time.
4. As mentioned in Section 7, our implementation is publicly available online.

**Organization.** In Section 2, we discuss the definitions and notations used in this paper. In Section 3, we examine inequalities on vertex eccentricities, which are key components of the proposed algorithm. Section 4 details the proposed algorithm. We present experimental results in Section 6 and conclude the paper in Section 7.

## 2 Preliminaries

Let $G = (V, E)$ be a graph, where $V$ is the vertex set and $E$ is the edge set. The number of vertices and number of edges are denoted $n$ and $m$, respectively. We assume that graph $G$ is weakly connected and unweighted. We define $N_{in}(v)$ and $N_{out}(v)$ as the in- and out-neighbors of vertex $v$. The subgraph induced

by a vertex set $S$ is denoted $G[S]$, and $G^T$ describes the transpose graph of a graph $G$. In addition, let $d(v, w)$ denote the distance from a vertex $v$ to a vertex $w$. When $w$ is not reachable from $v$, for simplicity, we define $d(v, w) = \infty$, where $\infty$ is considered a sufficiently large number. The eccentricity $ecc(v)$ of vertex $v$ is defined as the distance from $v$ to the farthest reachable vertex, i.e., $ecc(v) = \max\{d(v, w) \mid w \in V, d(v, w) \neq \infty\}$. The diameter $D$ of a graph $G$ is the maximum eccentricity, i.e., $D = \max\{ecc(v) \mid v \in V\}$.

**Strongly Connected Components.** A strongly connected component (SCC) of a graph is defined as a strongly connected maximal subgraph. Each SCC does not overlap; therefore, we can partition the vertex set $V$ into SCCs. The set of SCCs is denoted $\mathcal{V}_{\mathrm{SCC}}$, and the vertex set of the SCC that a vertex $v$ belongs to is denoted $V_{\mathrm{SCC}}(v)$. Note that this decomposition can be obtained in linear time [13].

**Double Sweep Algorithm.** The double sweep algorithm [4,8] works as follows. First, we randomly select a vertex $v$ and conduct a breadth first search (BFS) from this vertex. Then, we select a vertex $w$ such that $d(v, w) = ecc(v)$ and conduct a backward BFS from $w$. Finally, the double sweep algorithm outputs $\max\{d(u, w) \mid u \in V, d(u, w) \neq \infty\}$ as the lower bound of the diameter. Note that this runs in $O(n + m)$ time. The precision can be improved by conducting the algorithm repeatedly with a different initial vertex. Typically, a few iterations of double sweep give quite precise approximation for real graphs [6,7,10]. In the proposed algorithm, we use the double sweep algorithm to initialize the diameter lower bound.

## 3 Upper Bounds

In this section, we study the upper bounds on the eccentricities. In previous work that focused on undirected or strongly connected graphs, upper bounds based on the triangle inequality turned out to be quite effective [2,12]. However, as discussed in Section 3.1, this does not hold between vertices in different SCCs. Therefore, we propose new inequalities (Sections 3.2 and 3.3) that can propagate the upper bounds of vertices in other SCCs.

### 3.1 Triangle Inequality

Due to its effectiveness, the following inequality is often solely used as the upper bound of the eccentricity in undirected or strongly connected graphs [2,12]. This inequality can be obtained easily from the triangle inequality.

**Lemma 1 (Borassi, et al. [2]).** *If the graph is strongly connected, for any pair of vertices $v$ and $u$, $ecc(v) \leq d(v, u) + ecc(u)$.*

Unfortunately, on general directed graphs, this inequality does not hold between any pair of vertices. Here, we illustrate this point with an example.

In Figure 1, $ecc(2)$ is 2. However, the sum of the distance $d(2,1)$ and $ecc(1)$ is $1+0=1$, which is less than $ecc(2)$. This is because vertex 1 cannot reach vertices 3 and 4.

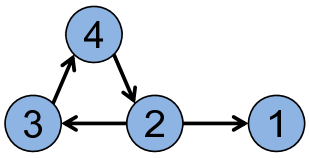


**Fig. 1.** An example of a general directed graph, on which the upper bound based on the triangle inequality does not necessarily hold

Therefore, we use this upper bound only inside an SCC to form the following corollary.

**Corollary 1.** *If vertices $v$ and $u$ are in the same SCC, $ecc(v) \leq d(v,u)+ecc(u)$.*

### 3.2 Inequality for Propagation

For vertices in large SCCs, similar to those in undirected or strongly connected graphs, good eccentricity upper bounds can be obtained using the above inequality. However, as there are many small SCCs in real-world networks, using the aforementioned inequality results in the eccentricities of vertices in those small SCCs remaining unbounded.

Therefore, we propose new inequalities that can propagate the eccentricity bounds of vertices in different SCCs. We begin with the following simple inequalities.

**Lemma 2.** *For any vertex $v$, $ecc(v) \leq \max\{ecc(w)+1 \mid w \in N_{out}(v)\}$.*

*Proof.* Let $u$ be a vertex with $ecc(v) = d(v,u)$. Since there exists a vertex $w^* \in N_{out}(v)$ such that $d(v,u) = d(w^*,u)+1$, we obtain $ecc(v) = d(v,u) = d(w^*,u)+1 \leq ecc(w^*)+1 \leq \max\{ecc(w)+1 \mid w \in N_{out}(v)\}$.

Although this inequality is quite simple, it enables us to obtain the upper bounds of vertices in small SCCs effectively. Moreover, it becomes significantly more effective when combined with vertex ordering strategies that consider the topological order of SCCs. Note that if we contract each SCC to a single vertex, we obtain a directed acyclic graph, which defines the topological order of SCCs. Therefore, by visiting each SCC in reverse topological order, this inequality works more effectively because, when we examine a vertex, we have already visited all of its out-neighbors. Thus, their eccentricity bounds are already tightened with the inequality using the bounds of their out-neighbors. In other words, we can propagate the upper bounds from lower SCCs to upper SCCs efficiently.

### 3.3 Tight Inequality for Propagation

By considering SCCs, we can obtain the following inequality, which is tighter than the above simple inequality. Note that we use this inequality in the proposed algorithm.

**Lemma 3.** *For any vertex $v$,*
$ecc(v) \leq \max\{\min\{ecc(w)+1 \mid w \in N_{out}(v) \cap C\} \mid C \in \mathcal{V}_{\mathrm{SCC}}\}$.

*Proof.* Let $u$ be a vertex with $ecc(v) = d(v,u)$ and $C^*$ be an SCC with $d(v,u) = \min(d(w,u)+1 \mid w \in N_{out}(v) \cap C^*)$. As any vertex in $C^*$ can reach $u$, it holds that $d(w,u) \leq ecc(w)$ for any vertex $w \in C^*$. Thus, we obtain $ecc(v) = d(v,u) = \min(d(w,u)+1 \mid w \in N_{out}(v) \cap C^*) \leq \min(ecc(w)+1 \mid w \in N_{out}(v) \cap C^*) \leq \max(\min(ecc(w)+1 \mid w \in N_{out}(v) \cap C) \mid C \in \mathcal{V}_{\mathrm{SCC}})$

## 4 Algorithm Description

Here, we present the proposed algorithm for computing diameters. The proposed algorithm and some of the previous algorithms share a common approach that maintains and gradually tightens a diameter lower bound and eccentricity upper bounds [2,12]. The main technical challenge in designing the proposed algorithm is handling pairs in different SCCs efficiently using SCC decomposition and the new upper bound inequalities.

### 4.1 Overall Algorithm

Algorithm 1 shows an overview of the proposed algorithm. The variable $\overline{ecc}[v]$ maintains an upper bound of the eccentricity of each vertex $v$ (i.e., $\overline{ecc}[v] \geq ecc(v)$), and variable $\underline{D}$ maintains a lower bound of the diameter of the given graph (i.e., $\underline{D} \leq D$). At the end of Algorithm 1, $\overline{ecc}[v]$ becomes less than or equal to $\underline{D}$ for any vertex $v$. From the inequality $\underline{D} \leq D = \max(ecc(v) \mid v \in V) \leq \max(\overline{ecc}(v) \mid v \in V) \leq \underline{D}$, $\underline{D}$ matches the exact value of the diameter $D$.

**Algorithm 1.** Proposed algorithm

```
1: procedure Diameter(G)
2:     V_SCC ← StronglyConnectedComponents(G)
3:     ecc̄[v] ← ∞ for all v ∈ V
4:     D ← DoubleSweep(G)
5:     for all v ∈ V do
6:         e' ← max{min{ecc̄[w] + 1 | w ∈ N_out(v) ∩ C} | C ∈ V_SCC}
7:         ecc̄[v] ← min(ecc̄[v], e')
8:         if ecc̄[v] > D then
9:             SearchAndBound(G, v)
10:    return D
```

First, we decompose the graph into SCCs. The SCCs are later used for vertex ordering and eccentricity upper bounds. Next, we initialize the diameter lower bound $\underline{D}$ using the double sweep algorithm. We also set the initial eccentricity upper bound $\overline{ecc}[v]$ to $\infty$ for each vertex $v$.

We then examine every vertex $v \in V$ in sequence. First, we refine the eccentricity upper bound $\overline{ecc}[v]$ by Lemma 3. We then compare $\overline{ecc}[v]$ and $\underline{D}$, and skip vertex $v$ if $\overline{ecc}[v] \leq \underline{D}$ because the lengths of paths from $v$ are bound by $\overline{ecc}[v]$; thus, they never update the diameter lower bound $\underline{D}$. Otherwise, we conduct breadth first searches (BFSs) from $v$ to update the bounds.

### 4.2 Updating Bounds by BFSs

The process of updating bounds by BFSs from vertex $u$ is described in Algorithm 2. We conduct two BFSs, a forward BFS from $u$ to the whole graph (lines 1–4) and a backward BFS from $u$ only to the SCC to which $u$ belongs (lines 5–7). In Algorithm 2, we assume that the BreadthFirstSearch function performs a BFS and returns an array that represents distances, i.e., $d_f[v] = d(u, v)$ for any $v \in V$ and $d_b[v] = d(v, u)$ for any $v \in V_{\mathrm{SCC}}(u)$.

**Algorithm 2.** Breadth first searches from vertex $u$ to update bounds

1: **procedure** SearchAndBound($G, u$)
2: $d_f \leftarrow$ BreadthFirstSearch($G, u$)
3: $ecc[u] \leftarrow \max\{d_f[v] \mid v \in V, d_f[v] \neq \infty\}$
4: $\overline{ecc}[u] \leftarrow ecc[u]$
5: $\underline{D} \leftarrow \max(\underline{D}, ecc[u])$
6: $d_b \leftarrow$ BreadthFirstSearch($G[V_{\mathrm{SCC}}(u)]^T, u$)
7: $\overline{ecc}[v] \leftarrow \min(\overline{ecc}[v], d_b[v] + ecc[u])$ **for all** $v \in V_{\mathrm{SCC}}(u)$

The forward BFS computes the exact eccentricity of $u$. Once the eccentricity is obtained, the diameter lower bound is compared to the eccentricity and updated if necessary. We then perform a reverse BFS to compute the distances to vertex $u$. This reverse BFS updates the eccentricity upper bounds of other vertices. Here we only visit vertices in the SCC to which $u$ belongs because we are dealing with general (not necessarily strongly connected) directed graphs. As discussed in Section 3, this upper bound holds only within an SCC (Corollary 1).

## 5 Vertex Ordering Strategies

Here, we discuss vertex ordering strategies. The proposed algorithm examines each vertex sequentially and conducts BFSs if necessary. While its correctness is not dependent on the order of vertices, the order has considerable effect on the performance of the proposed algorithm.

**Ordering SCCs.** First, we consider the overall graph. We update the upper bound of the eccentricity by checking all out-neighbors and using the upper bound for the out-neighbors. Therefore, we check each SCC in reverse topological order. This enables us to obtain a better upper bound because, when we check a neighbor in a different SCC, the neighbor has been visited and the upper bound of the neighbor has been tightened.

**Ordering inside an SCC.** Next, we consider each SCC. In each SCC, we use the triangle inequality to update the upper bound of the eccentricity. When we examine a central vertex, distances to the vertex are small; therefore, by examining such vertices first, we can tighten the upper bounds. We order vertices based on the degree of vertices to select central vertices. However, we can consider various strategies based on the degree in directed graphs. We propose and empirically compare the following five vertex ordering strategies.

- Random: A random permutation of vertices is used as the vertex order.
- In-Degree: We order vertices based on the in-degree of vertices, i.e., we use $|N_{in}(v)|$ as the index.
- Out-Degree: We order vertices based on the out-degree of vertices, i.e., we use $|N_{out}(v)|$ as the index.
- Degree-Product: We order vertices based on the product of the in-degree and out-degree of vertices, i.e., we use $|N_{in}(v)| \cdot |N_{out}(v)|$ as the index.
- Degree-Component-Product: We order vertices based on the product of the in-degree and out-degree for vertices in the same SCC, i.e., we use $|N_{in}(v) \cap V_{\mathrm{SCC}}(v)| \cdot |N_{out}(v) \cap V_{\mathrm{SCC}}(v)|$ as the index. We consider that vertices in different SCCs have little to do with the centrality in the SCC.

## 6 Experiments

Here, we present our experimental results. In Section 6.1, we discuss an experimental evaluation of the practicality of the proposed method and compare it to previous methods. In Section 6.3, we analyze the behavior of the proposed method empirically.

**Setup.** We conducted experiments on a Linux server with an Intel Xeon X5675 processor (3.06 GHz) and 288 GB of main memory. All algorithms were implemented in C++ and compiled with gcc 4.1.2 using optimization level 3. We did not parallelize the proposed algorithm and used only a single core. We used various real-world directed networks from the Stanford Large Network Dataset Collection [9]. We treated all graphs as unweighted. In addition, we used the largest weakly connected component of the graphs. Unless otherwise stated, we used the Degree-Component-Product strategy for vertex ordering, and the initial lower bound was obtained by performing the double sweep algorithm from 10 randomly selected vertices.

### 6.1 Evaluating the Proposed Method

We evaluate the performance of the proposed method with real graphs and virtually compare performance to the naive method. Table 2 shows the results. We counted each call of function SearchAndBound (Algorithm 2) as a single BFS; SearchAndBound conducts one whole BFS and one small BFS that is limited

**Table 2.** Performance of the proposed algorithm on real graphs. #BFS denotes the number of BFSs to compute the diameter and Speed-up denotes the improvement factor of the number of BFSs against the naive approach.

| Dataset | | | | Proposed method | |
|---|---|---|---|---|---|
| **Name** | $n$ | $m$ | $D$ | **#BFS** | **Speed-up** |
| ***Social Networks:*** | | | | | |
| ego-Twitter | 81,306 | 1,768,149 | 15 | 353 | 230.3 |
| soc-Epinions1 | 75,878 | 508,837 | 16 | 1,066 | 71.2 |
| soc-LiveJournal1 | 4,843,953 | 68,983,820 | 22 | 53,819 | 90.0 |
| soc-Pokec | 1,632,803 | 30,622,564 | 18 | 2,110 | 773.8 |
| soc-Slashdot0811 | 77,361 | 905,469 | 12 | 6,744 | 11.5 |
| soc-Slashdot0922 | 82,169 | 948,465 | 13 | 6,891 | 11.9 |
| soc-sign-Slashdot081106 | 77,350 | 516,575 | 15 | 970 | 79.7 |
| soc-sign-Slashdot090216 | 81,867 | 545,671 | 15 | 983 | 83.3 |
| soc-sign-Slashdot092221 | 82,140 | 549,202 | 15 | 975 | 84.2 |
| soc-sign-epinions | 119,130 | 833,390 | 16 | 1,269 | 93.9 |
| wiki-Vote | 7,067 | 103,664 | 10 | 2 | 3,533.5 |
| ***Communication Networks:*** | | | | | |
| email-EuAll | 224,833 | 395,271 | 11 | 206 | 1,091.4 |
| wiki-Talk | 2,388,954 | 5,018,446 | 11 | 935 | 2,555.0 |
| ***Citation Networks:*** | | | | | |
| cit-HepPh | 34,401 | 421,485 | 49 | 915 | 37.6 |
| cit-HepTh | 27,400 | 352,542 | 37 | 2,412 | 11.4 |
| ***Web Graphs:*** | | | | | |
| web-BerkStan | 654,783 | 7,499,426 | 694 | 5,977 | 109.6 |
| web-Google | 855,803 | 5,066,843 | 51 | 12,347 | 69.3 |
| web-NotreDame | 325,730 | 1,497,135 | 93 | 3,626 | 89.8 |
| web-Stanford | 255,266 | 2,234,573 | 580 | 3,027 | 84.3 |
| ***Product Co-Purchasing Networks:*** | | | | | |
| amazon0302 | 262,112 | 1,234,878 | 88 | 1,666 | 157.3 |
| amazon0312 | 400,728 | 3,200,441 | 53 | 593 | 675.8 |
| amazon0505 | 410,237 | 3,356,825 | 55 | 388 | 1,057.3 |
| amazon0601 | 403,365 | 3,387,225 | 54 | 450 | 896.4 |
| ***Internet Peer-to-Peer Networks:*** | | | | | |
| p2p-Gnutella04 | 10,877 | 39,995 | 26 | 30 | 362.6 |
| p2p-Gnutella05 | 8,843 | 31,838 | 22 | 24 | 368.5 |
| p2p-Gnutella06 | 8,718 | 31,526 | 21 | 50 | 174.4 |
| p2p-Gnutella08 | 6,300 | 20,777 | 20 | 51 | 123.5 |
| p2p-Gnutella09 | 8,105 | 26,009 | 20 | 110 | 73.7 |
| p2p-Gnutella24 | 26,499 | 65,360 | 29 | 45 | 588.9 |
| p2p-Gnutella25 | 22,664 | 54,694 | 22 | 138 | 164.2 |
| p2p-Gnutella30 | 36,647 | 88,304 | 24 | 165 | 222.1 |
| p2p-Gnutella31 | 62,562 | 147,879 | 31 | 210 | 297.9 |

to an SCC. Speed-up represents the improvement factor of the number of BFSs against the naive approach, which requires $n$ BFSs (i.e., $n$ divided by #BFS).

We emphasize that the proposed algorithm can compute the diameters of large real directed graphs with up to tens of millions of edges. Obviously, this is too large for the naive method. However, the proposed algorithm can compute diameters several tens to several thousand times faster than the naive method. In addition, we do not necessarily visit all vertices in a BFS. Thus, the proposed algorithm can compute diameters faster than it appears.

### 6.2 Evaluating the Previous Approximation Heuristics

We then evaluate the classic lower bound heuristics. Note that the evaluation of these methods on large directed networks became possible for the first time using the proposed algorithm. We evaluated both random sampling and the double sweep algorithm. For random sampling, we sampled 10, 100, or 1,000 vertices and conducted BFSs from these vertices. We ran the double sweep algorithm from 10 randomly selected vertices. The results are given in Table 3.

Note that the lower bounds obtained by random sampling are far from exact diameters. For any graph, we could not compute exact diameters by sampling less than 100 vertices. Even if we sampled 1,000 vertices, we could compute exact diameters in only eight graphs. On the other hand, the lower bounds obtained by the double sweep algorithm were correct for many graphs. However, these bounds varied greatly from the exact diameters in a few graphs.

### 6.3 Analysis

**Upper Bounds.** Table 4 compares the effectiveness of inequalities for eccentricity upper bounds. In Table 4, (1), (2), and (3) represent Corollary 1, Lemma 2, and Lemma 3, respectively. We reduced the number of BFSs by approximately five percent on average using Lemma 3 rather than Lemma 2. This indicates that Lemma 3 is effective for obtaining tighter upper bounds. In addition, if we did not use Lemma 3, the number of BFSs increased drastically due to many small SCCs. On the other hand, we could compute diameters efficiently for many graphs using only Lemma 3. These results confirm the effectiveness of our new upper bounds by propagation.

**Vertex Ordering Strategies.** Table 5 shows the number of BFSs for each vertex ordering strategy. Essentially, we used the Degree-Component-Product strategy. However, all strategies demonstrated nearly the same performance, with exception of the Random strategy. The reason is as follows. We use two inequalities with Corollary 1 and Lemma 3. As mentioned previously, Lemma 3 is quite powerful. In addition, the ordering inside an SCC affects only the triangle inequality. Therefore, performance is insulated from the influence of ordering inside an SCC. As can be seen, the Random strategy is worse. However, the difference between the Random strategy and other strategies is small in many cases. This confirms that the order of vertices in each SCC has less effect on performance.

**Table 3.** Comparison between exact diameters and lower bounds using heuristics. Sampling denotes the lower bound computed by 10, 100, or 1,000 random samplings. Double Sweep denotes the lower bound computed by performing the double sweep algorithm 10 times.

| Dataset | | | | Sampling | | | Double |
|---|---|---|---|---|---|---|---|
| **Name** | $n$ | $m$ | $D$ | 10 | 100 | 1,000 | **Sweep** |
| ***Social Networks:*** | | | | | | | |
| ego-Twitter | 81,306 | 1,768,149 | 15 | 10 | 10 | 11 | **15** |
| soc-Epinions1 | 75,878 | 508,837 | 16 | 12 | 12 | 14 | **16** |
| soc-LiveJournal1 | 4,843,953 | 68,983,820 | 22 | 15 | 17 | 18 | **22** |
| soc-Pokec | 1,632,803 | 30,622,564 | 18 | 12 | 14 | 14 | **18** |
| soc-Slashdot0811 | 77,361 | 905,469 | 12 | 9 | 9 | **12** | **12** |
| soc-Slashdot0922 | 82,169 | 948,465 | 13 | 10 | 10 | **13** | **13** |
| soc-sign-Slashdot081106 | 77,350 | 516,575 | 15 | 13 | 13 | 13 | **15** |
| soc-sign-Slashdot090216 | 81,867 | 545,671 | 15 | 11 | 12 | 13 | **15** |
| soc-sign-Slashdot092221 | 82,140 | 549,202 | 15 | 10 | 13 | 13 | **15** |
| soc-sign-epinions | 119,130 | 833,390 | 16 | 12 | 12 | 13 | **16** |
| wiki-Vote | 7,067 | 103,664 | 10 | 6 | 8 | 8 | **10** |
| ***Communication Networks:*** | | | | | | | |
| email-EuAll | 224,833 | 395,271 | 11 | 9 | 9 | **11** | **11** |
| wiki-Talk | 2,388,954 | 5,018,446 | 11 | 7 | 7 | 8 | **11** |
| ***Citation Networks:*** | | | | | | | |
| cit-HepPh | 34,401 | 421,485 | 49 | 32 | 47 | **49** | **49** |
| cit-HepTh | 27,400 | 352,542 | 37 | 29 | 32 | 34 | **37** |
| ***Web Graphs:*** | | | | | | | |
| web-BerkStan | 654,783 | 7,499,426 | 694 | 580 | 585 | 645 | **694** |
| web-Google | 855,803 | 5,066,843 | 51 | 36 | 39 | 41 | **51** |
| web-NotreDame | 325,730 | 1,497,135 | 93 | 8 | 54 | 66 | 64 |
| web-Stanford | 255,266 | 2,234,573 | 580 | 148 | 149 | 557 | 244 |
| ***Product Co-Purchasing Networks:*** | | | | | | | |
| amazon0302 | 262,112 | 1,234,878 | 88 | 69 | 75 | 85 | **88** |
| amazon0312 | 400,728 | 3,200,441 | 53 | 38 | 40 | 48 | **53** |
| amazon0505 | 410,237 | 3,356,825 | 55 | 37 | 41 | **55** | **55** |
| amazon0601 | 403,365 | 3,387,225 | 54 | 36 | 48 | 48 | **54** |
| ***Internet Peer-to-Peer Networks:*** | | | | | | | |
| p2p-Gnutella04 | 10,877 | 39,995 | 26 | 19 | 22 | 23 | **26** |
| p2p-Gnutella05 | 8,843 | 31,838 | 22 | 18 | 20 | **22** | **22** |
| p2p-Gnutella06 | 8,718 | 31,526 | 21 | 16 | 19 | 20 | **21** |
| p2p-Gnutella08 | 6,300 | 20,777 | 20 | 16 | 17 | **20** | **20** |
| p2p-Gnutella09 | 8,105 | 26,009 | 20 | 18 | 19 | 19 | **20** |
| p2p-Gnutella24 | 26,499 | 65,360 | 29 | 25 | 25 | 27 | **29** |
| p2p-Gnutella25 | 22,664 | 54,694 | 22 | 17 | 18 | 19 | 21 |
| p2p-Gnutella30 | 36,647 | 88,304 | 24 | 20 | 22 | 22 | **24** |
| p2p-Gnutella31 | 62,562 | 147,879 | 31 | 24 | 28 | **31** | **31** |

**Table 4.** Comparison of the number of BFSs between different sets of eccentricity upper bound inequalities

| Dataset | | | Upper Bounds | | | |
|---|---|---|---|---|---|---|
| **Name** | $n$ | $m$ | (1)+(3) | (1)+(2) | (1) | (3) |
| ego-Twitter | 81,306 | 1,768,149 | 353 | 350 | 11,704 | 540 |
| soc-Pokec | 1,632,803 | 30,622,564 | 2,110 | 2,118 | 307,827 | 5,476 |
| soc-sign-epinions | 119,130 | 833,390 | 1,269 | 1,022 | 53,346 | 1,254 |
| wiki-Talk | 2,388,954 | 5,018,446 | 935 | 515 | 2,257,197 | 750 |
| cit-HepPh | 34,401 | 421,485 | 915 | 2,023 | 19,265 | 1,315 |
| web-Google | 855,803 | 5,066,843 | 12,347 | 12,550 | 304,516 | 25,302 |
| amazon0601 | 403,365 | 3,387,225 | 450 | 450 | 1,573 | 5,637 |
| p2p-Gnutella30 | 36,647 | 88,304 | 165 | 226 | 27,969 | 892 |
| p2p-Gnutella31 | 62,562 | 147,879 | 210 | 256 | 47,906 | 1,430 |

**Table 5.** Comparison of the number of BFSs between different vertex ordering strategies

| | Degree | | | | |
|---|---|---|---|---|---|
| **Dataset** | In | Out | Product | Component Product | Random |
| ego-Twitter | 364 | **350** | 355 | 353 | 406 |
| soc-Pokec | **2,110** | 2,111 | **2,110** | **2,110** | 3,799 |
| soc-sign-epinions | **1,027** | 1,040 | 1,103 | 1,269 | 1,050 |
| wiki-Talk | **925** | 1,272 | 978 | 935 | 983 |
| cit-HepPh | **856** | 1,573 | 859 | 915 | 977 |
| web-Google | 12,071 | 12,085 | **12,064** | 12,347 | 12,118 |
| amazon0601 | **450** | 466 | **450** | **450** | 508 |
| p2p-Gnutella30 | 213 | 269 | 244 | **165** | 327 |
| p2p-Gnutella31 | **187** | 273 | 264 | 210 | 241 |

## 7 Conclusion

In this paper, we have proposed an algorithm to exactly compute the diameters of large real graphs. The proposed algorithm gradually tightens the diameter lower bound and eccentricity upper bound, and they are guaranteed to match when the algorithm terminates. Note that the worst-case time complexity is still $\Theta(nm)$. However, as demonstrated by our experimental results, the proposed algorithm works surprisingly well in practice; it yielded the exact diameters of real directed graphs with millions of vertices and edges.

**Software Available Publicly.** Our implementation of the proposed algorithm is publicly available online at http://git.io/graph-diameter. It is our hope that further scientific findings will be enabled by our public code.

**Acknowledgments.** Takuya Akiba and Yoichi Iwata are supported by Grant-in-Aid for JSPS Fellows 256563 and 256487, respectively. Yuki Kawata is supported by JST, ERATO, Kawarabayashi Project.

## References


1. Albert, R., Jeong, H., Barabási, A.-L.: Diameter of the World-Wide web. Nature **401**(6749), 130–131 (1999)
2. Borassi, M., Crescenzi, P., Habib, M., Kosters, W., Marino, A., Takes, F.: On the solvability of the six degrees of kevin bacon game. In: Ferro, A., Luccio, F., Widmayer, P. (eds.) FUN 2014. LNCS, vol. 8496, pp. 52–63. Springer, Heidelberg (2014)
3. Broder, A., Kumar, R., Maghoul, F., Raghavan, P., Rajagopalan, S., Stata, R., Tomkins, A., Wiener, J.: Graph Structure in the Web. Computer Networks: The International Journal of Computer and Telecommunications Networking **33**(1–6), 309–320 (2000)
4. Corneil, D.G., Dragan, F.F., Habib, M., Paul, C.: Diameter Determination on Restricted Graph Families. Discrete Applied Mathematics **113**(2), 143–166 (2001)
5. Crescenzi, P., Grossi, R., Habib, M., Lanzi, L., Marino, A.: On Computing the Diameter of Real-World Undirected Graphs. Theoretical Computer Science **514**, 84–95 (2013)
6. 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)
7. Crescenzi, P., Grossi, R., Lanzi, L., Marino, A.: On computing the diameter of real-world directed (weighted) graphs. In: Klasing, R. (ed.) SEA 2012. LNCS, vol. 7276, pp. 99–110. Springer, Heidelberg (2012)
8. Handler, G.Y.: Minimax Location of a Facility in an Undirected Tree Graph. Transportation Science **7**(3), 287–293 (1973)
9. Leskovec, J., Krevl, A.: SNAP Datasets: Stanford Large Network Dataset Collection. http://snap.stanford.edu/data
10. Magnien, C., Latapy, M., Habib, M.: Fast Computation of Empirically Tight Bounds for the Diameter of Massive Graphs. Journal of Experimental Algorithmics **13**(10), 1.10–1.9 (2009)
11. Milgram, S.: The Small-World Problem. Psychology Today **2**(1), 60–67 (1967)
12. Takes, F.W., Kosters, W.A.: Determining the Diameter of Small World Networks. In: CIKM, pp. 1191–1196 (2011)
13. Tarjan, R.: Depth-First Search and Linear Graph Algorithms. SIAM Journal on Computing **1**(2), 146–160 (1972)
14. Watts, D.J., Strogatz, S.H.: Collective Dynamics of 'Small-World' Networks. Nature **393**(6684), 440–442 (1998)