Fast diameter and radius BFS-based computation in (weakly connected) real-world graphs

Michele Borassi, Pierluigi Crescenzi, Michel Habib, Walter A. Kosters, Andrea Marino, Frank W. Takes

Fast Diameter and Radius BFS-based Computation in (Weakly Connected) Real-World Graphs

With an Application to the Six Degrees of Separation Games[☆]

Michele Borassi[a], Pierluigi Crescenzi[b], Michel Habib[c], Walter A. Kosters[d], Andrea Marino[e,1], Frank W. Takes[d]

[a] *IMT Institute of Advanced Studies, Lucca*
[b] *Dipartimento di Ingegneria dell'Informazione, Università di Firenze*
[c] *LIAFA, UMR 7089 CNRS & Université Paris Diderot - Paris 7*
[d] *LIACS, Leiden University*
[e] *Dipartimento di Informatica, Università di Milano*

Abstract

In this paper, we propose a new algorithm that computes the radius and the diameter of a weakly connected digraph $G = (V, E)$, by finding bounds through heuristics and improving them until they are validated. Although the worst-case running time is $\mathcal{O}(|V||E|)$, we will experimentally show that it performs much better in the case of real-world networks, finding the radius and diameter values after 10–100 BFSs instead of $|V|$ BFSs (independently of the value of $|V|$), and thus having running time $\mathcal{O}(|E|)$ in practice. As far as we know, this is the first algorithm able to compute the diameter of weakly connected digraphs, apart from the naive algorithm, which runs in time $\Omega(|V||E|)$ performing a BFS from each node. In the particular cases of strongly connected directed or connected undirected graphs, we will compare our algorithm with known approaches by performing experiments on a dataset composed by several real-world networks of different kinds. These experiments will show that, despite its generality, the new algorithm outperforms all previous methods, both in the radius and in the diameter computation, both in the directed and in the undirected case, both in average running time and in robustness. Finally, as an application example, we will use the new algorithm to determine the solvability over time of the "Six Degrees of Kevin Bacon" game, and of the "Six Degrees of Wikipedia" game. As a consequence, we will compute for the first time the exact value of the radius and the diameter of the whole Wikipedia digraph.

Keywords: Graph Diameter, Graph Radius, Actors Graph, Wikipedia Graph

1. Introduction

The diameter and the radius of a network are relevant measures (whose meaning depends on the semantics of the network itself), which have been almost always considered while analyzing real-world networks such as biological, collaboration, communication, road, social, and web networks (see, for example, [7]). Informally (we will see later the formal definition), the diameter is the maximum distance between two connected vertices, and the radius is the distance from a center (that is, a vertex that minimizes the maximum distance to all other vertices) to the vertex farthest from it (in this paper we will always refer to unweighted graphs). Many algorithmic results have been presented in the last few decades concerning the computation of the

[☆] This paper is an extension and a generalization of the results presented in [6].

Email addresses: michele.borassi@imtlucca.it (Michele Borassi), pierluigi.crescenzi@unifi.it (Pierluigi Crescenzi), habib@liafa.univ-paris-diderot.fr (Michel Habib), w.a.kosters@liacs.leidenuniv.nl (Walter A. Kosters), marino@di.unimi.it (Andrea Marino), ftakes@liacs.nl (Frank W. Takes)

[1] Supported by the EU-FET grant NADINE (GA 288956).

diameter and of the radius. As far as we know, in the general case, the best known solution is still, more or less, based on the computation of all-pairs shortest paths. The time complexity of this solution is $O(n^\omega)$, where $\omega < 2.38$, in the case of dense graphs [35] (by using efficient algorihtms for matrix multiplication), and $O(mn)$ in the case of sparse graphs (by simply performing a breadth-first search from each node of the graph).[2] This complexity is not feasible whenever we deal with real-world networks, since these networks may contain several millions of nodes and several billions of edges. For this reason, two different line of research have been also followed. On the one hand, more efficient algorithms have been proposed for special classes of graphs (see, for example, [11, 5]), while, on the other hand, more efficient approximation algorithms have been designed for the general case (see, for example, [1, 28, 9]). Observe that in [28, 9], the authors do not only propose a better approximation algorithm for sparse graphs, but they also show that, in this case, no sub-quadratic algorithm exists for computing the diameter unless the Strong Exponential-Time Hypothesis is false [18].

A third line of research has, instead, deeply explored the power of breadth-first search (in short, BFS) in order to exactly compute the diameter and the radius. Clearly, the height of any BFS tree provides us with both a lower bound on the diameter and an upper bound on the radius. Thus, a simple heuristic to estimate these values consists of executing a fixed number of random BFSs, and reporting the best bound found for each of them (see, for example, [25, 29]): unfortunately, no useful bound on the performed error can be provided and even experimentally this heuristic turns out to be not always precise. For this reason, several papers dealt with the problem of appropriately choosing the vertices from which the BFSs have to be performed. For example, the so-called 2Sweep heuristic picks one of the farthest vertices x from a random vertex r and returns the distance of the farthest vertex from x [22], while the 4Sweep picks the vertex in the middle of the longest path computed by a 2Sweep execution and performs another 2Sweep from that vertex [13]. Both methods work quite well and very often provide tight bounds. Indeed, in the case of special classes of graphs, they can even be (almost) exact: for example, the 2Sweep method gives the exact value of the diameter for trees [16], yields an approximation with additive error 1 for chordal graphs and interval graphs, and within 2 for AT-free graphs and hole-free graphs (see [12] for a survey and for the definitions of these graph classes). Adaptations of these methods to directed graphs have been proposed in [8, 14], and, even in this case, these techniques are very efficient and provide very good bounds on real-world networks.

However, in general, heuristics cannot guarantee the correctness of the results obtained. For this reason, a major further step in the diameter computation was the design of bound-refinement algorithms. These methods apply a heuristic and try to validate the result found or improve it until they successfully validate it. Even if in the worst case their time complexity is $\mathcal{O}(mn)$, they turn out to be linear in practice. The main algorithms developed until now are BoundingDiameters [31] and iFub [13]. While the first works only on undirected graphs, the second is also able to deal with the directed strongly connected case (the adaptation is called diFub [14]). For the radius computation, instead, the current best algorithm for undirected graphs is a modification of the BoundingDiameters algorithm [32] while for directed graphs it is possible to use the method in [23]. However, all these bound-refinement algorithms cannot deal with directed graphs that are not strongly connected. In the latter case, as far as we know, the only exact method to compute the radius and the diameter is the naive one, that is, the computation of all-pairs shortest paths.

1.1. Our results

In this paper, we will propose the first bound-refinement algorithm that is able to find diameter and radius in directed, not necessarily strongly connected graphs. It should be noticed that for non strongly connected digraphs, in order to avoid inifinte values, eccentricities are only computed between reachable vertices. This algorithm will not only be more general than all previous counterparts, but it will also outperform them on directed, strongly connected graphs or undirected graphs. It relates the *sweep* approach (i.e., a new visit of the graph depends on the previous one, as in [14, 13, 22, 23]) with the techniques developed in [31, 32]. It is based on a new heuristic, named SumSweep, which is able to compute very efficiently lower bounds on the diameter and upper bounds on the radius of a given graph. This heuristic computes the

[2] Given a graph $G = (V, E)$, we let $n = |V|$ and $m = |E|$.

eccentricities of "meaningful" vertices, where the forward eccentricity of a vertex v is the distance from v to the farthest reachable vertex and, conversely, the backward eccentricity of a vertex v is the maximum distance from another vertex to v. After each meaningful vertex is chosen, a BFS is performed from this vertex, providing its forward or backward eccentricity, together with some values that are used in the choice of the next vertices to be analyzed. After some steps, the maximum eccentricity found is a lower bound D_L for the diameter, while the minimum eccentricity found is an upper bound R_U for the radius. Then, these bounds are used as starting point for a validation algorithm, that generalizes the approach proposed in [31] for undirected graphs. The validation works as follows. We perform some more BFSs, with two goals: improving D_L and R_U, and finding upper bounds D_U on the diameter and lower bounds R_L on the radius. These latter values are obtained by upper and lower bounding the forward and backward eccentricities of all vertices, and setting D_U to the maximum (forward or backward) upper bound found and R_L to the minimum forward lower bound. Since $D_L \leq D \leq D_U$, as soon as $D_L = D_U$ we have computed the value of D; symmetrically, as soon as $R_L = R_U$, we have computed the value of R. After explaining this algorithm in its full generality, we will specialize it to directed strongly connected graphs and undirected graphs, so that it will be possible to compare it with other existing algorithms (as already said, as far as we know, this is the first algorithm able to efficiently compute the diameter and the radius of any directed real-world graph). The comparison will show that, despite its generality, the new algorithm will significantly outperform all previous algorithms, both in average number of BFSs and in robustness, both in directed and in undirected graphs.

Finally, we will use our new algorithm in order to analyze the solvability of the six degrees of separation game, which is a trivia game inspired by the well-known social experiment of Stanley Milgram [24], which was in turn a continuation of the empirical study of the structure of social networks by Michael Gurevich [15]. Indeed, the notion of six degrees of separation has been formulated for the first time by Frigyes Karinthy in 1929, who conjectured that any two individuals can be connected through at most five acquaintances. This conjecture has somehow been experimentally verified by Milgram and extremely popularized by a theater play of John Guare, successively adapted to the cinema by Fred Schepisi. The corresponding game refers to a social network, such as the (movie) actor collaboration network, and can be played according to two main different variants. In the first variant, given two vertices x, i.e. the source, and y, i.e. the target, of the network, the player is asked to find a path of length at most six between x and y: for instance, in the case of the actor collaboration network, the player is asked to list at most five actors $x_1, \ldots, x_5$ and at most six movies $m_1, \ldots, m_6$ such that x and x_1 played in m_1, x_5 and y played in m_6, and x_i and x_{i+1} played in m_{i+1}, for $1 \leq i \leq 4$. In the second variant of the game, the vertex x is fixed and only the target vertex y is chosen during the game: for instance, in the case of the actor collaboration network, one very popular instance of this variant is the so-called "Six Degrees of Kevin Bacon" game, where the vertex x is the actor Kevin Bacon, who is considered one of the centers of the Hollywood universe [27]. Many other examples of both variants of the six degrees of separation game are now available on the web: one of the most popular games is the so-called "Six Degrees of Wikipedia" game [10], in which the vertices of the network are the Wikipedia articles and the edges are the links between these articles (here, the network is directed).

In this paper we address the following question: is a given instance of a six degrees of separation game solvable? More generally, is a given instance of a k degrees of separation game solvable? In the case of the second variant of the game, an additional question is the following: which is the choice of vertex x that makes the game solvable? In particular, we will analyze the actor collaboration network, in order to answer to these questions, and we will consider the evolution of this network over time, from 1940 to 2014. It will turn out that neither variant of the six degrees of separation game has ever been solvable, since there have always been actors at distance 13 (that is, in order to be solvable the first variant of the game has to choose $k = 13$) and no actor ever existed who could reach all other vertices in less than 7 steps. It will turn out that, for the vast majority of the analyzed period, Kevin Bacon has never been the right choice of vertex x (indeed, this happened only in the last two/three years). Moreover, we will also analyze the Wikipedia graph: in this case, it will turn out that the six degree of separations game is solvable for instance by fixing, as a target, the page `United_States_of_America`.

1.2. Structure of the paper

The paper is structured as follows. After providing the main definitions in Section 2, in Section 3 we will show how the diameter and radius can be bounded by using the SumSweep heuristic, while in Section 4 we will describe how the SumSweep algorithm works. Section 5 aims to experimentally show the effectiveness of our techniques in the case of several real-world networks. In Section 6, a case study on the actor collaboration network is provided, while in Section 7 another case study concerns the Wikipedia graph. Finally, in Section 8 we conclude the paper.

2. Notations and preliminary definitions

Given a directed graph (in short, digraph) $G = (V, E)$, we will say that a vertex w is reachable from a vertex v if there is a path from v to w. The set of vertices reachable from a given vertex v will be denoted by $\texttt{ReachF}(v)$, and the set of vertices w such that $v \in \texttt{ReachF}(w)$ is denoted by $\texttt{ReachB}(v)$. Note that if G is undirected, then $\texttt{ReachF}(v) = \texttt{ReachB}(v)$, for any vertex v.

An undirected graph is *connected* if, for any vertex v, $\texttt{ReachF}(v) = V$ (equivalently, $\texttt{ReachB}(v) = V$). A directed graph (in short, digraph) $G = (V, E)$ is said to be *weakly connected* if the undirected graph resulting from removing the orientation of the edges is connected (for example, the graph shown in the upper left part of Figure 1 is weakly connected). In this paper, we will always assume that graphs are weakly connected (otherwise we can apply our algorithms to each "weakly connected component"). A weakly connected digraph is *strongly connected* if, for any vertex v, $\texttt{ReachF}(v) = \texttt{ReachB}(v) = V$.

Given a digraph $G = (V, E)$, a *strongly connected component* (in short, SCC) of G is a subgraph that is strongly connected, and is maximal with respect to this property. The *strong component graph* of a digraph G is the directed acyclic graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V} = \{C_1, \ldots, C_k\}$ is the set of SCCs of G and an edge (C_i, C_j) exists if there is at least one edge in G from a vertex in the component C_i to a vertex in the component C_j. In the following, for each pair $(C_i, C_j) \in \mathcal{E}$, we will fix an edge $e_{ij} = (v_{ij}, w_{ij})$ of G such that $v_{ij} \in C_i, w_{ij} \in C_j$ (see the right part of Figure 1). Observe that this edge can be arbitrarily chosen during the construction of the strong component graph. Observe also that $\mathcal{G}$ is an acyclic digraph: hence, we may assume that a *topological order* is specified for its vertices, that is, $\mathcal{V}$ is ordered such that, for each $(C_i, C_j) \in \mathcal{E}$, $i < j$. For more background on these concepts, we refer to [3].

The forward and backward eccentricities of a vertex v are usually defined as:

$$\varepsilon^F(v) := \max_{w \in V} d(v, w),$$
$$\varepsilon^B(v) := \max_{w \in V} d(w, v),$$

where $d(v, w)$ denotes the length of a shortest path from node v to node w if $w \in \texttt{ReachF}(v)$, and $+\infty$ otherwise. Note that if the graph is undirected, $\varepsilon^F(v) = \varepsilon^B(v)$. Since most of the real-world graphs are not strongly connected, we prefer to ignore infinite distances and we define the forward and backward eccentricities as:

$$e^F(v) := \max_{w \in \texttt{ReachF}(v)} d(v, w),$$
$$e^B(v) := \max_{w \in \texttt{ReachB}(v)} d(w, v).$$

Moreover, it is worth observing that most real-world networks contain several vertices w with in-degree 0 or out-degree 0, and consequently both $\varepsilon^F(v)$ and $\varepsilon^B(v)$ are infinite for each vertex v, while $e^F(v)$ and $e^B(v)$ are always finite. From now on, by "eccentricity" we will always mean e as defined above.

The diameter is the maximum eccentricity of a vertex, that is, $D = \max_{v \in V} e^F(v) = \max_{v \in V} e^B(v)$: in other words, this is the length of "a longest shortest path" [17]. Note that in [3, 34] the diameter is defined through the original definition of eccentricity: however, this implies that the diameter of any disconnected graph is $+\infty$.

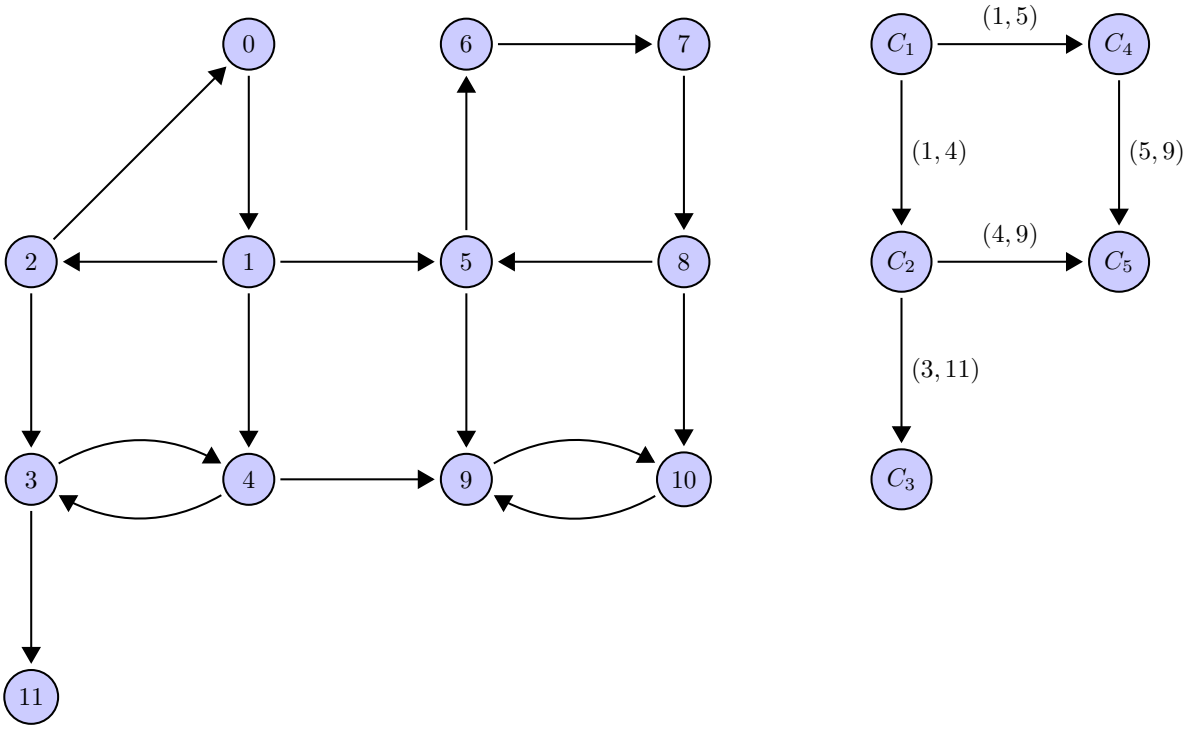

Figure 1: A weakly connected graph and the corresponding strong component graph. Each edge $(C_i, C_j) \in \mathcal{E}$ is labeled with a possible choice of e_{ij}.

The radius is usually defined as the minimum forward eccentricity of a vertex [3, 34]: in most real-world networks, using the old definition of eccentricity we have that the radius is $+\infty$, while, by using the new definition of eccentricity, the radius is 0. Both these definitions are just related to reachability: in particular, the first is affected by vertices that cannot be reached, while the second is affected by the existence of vertices with out-degree 0. In order to be able to exclude such vertices, we will consider a set V' of "meaningful" vertices, and define the radius as the minimum eccentricity of a vertex in V': if we choose V' to be the set of vertices v such that $\varepsilon^F(v) < +\infty$, we simulate the old definition. In this paper, if n_1 is the maximum size of a strongly connected component, we will choose $V' = V'_1 \cup V'_2$, where V'_1 is the set of vertices in a component of size n_1, and V'_2 is the set of vertices that are able to reach a vertex in V'_1 (in almost all real-world networks there is only a component of size n_1, the so-called *giant component* [26]). For example, by referring to the graph in Figure 1, we have that $V' = \{0, 1, 2, 5, 6, 7, 8\}$. Note that, with this choice of V', the radius is always finite. In any case, our algorithms work for any choice of V': in particular, they are also able to compute the radius according to the aforementioned definitions by choosing a suitable V'.

3. The SumSweep heuristic

The SumSweep is a heuristic that finds a lower bound for the diameter D_L and an upper bound on the radius R_U. The lower bound D_L on the diameter is obtained by finding several "peripheral" vertices and setting D_L as the maximum eccentricity of one of these vertices (it is a lower bound for D because D is the maximum eccentricity among all vertices). Then, the heuristic finds a very "central" vertex in V' and sets R_U as the eccentricity of this vertex. The pseudo-code is shown in Algorithm 1.

Like other heuristics [8, 22, 14, 13], SumSweep is based on performing alternatively forward and backward BFSs from some vertices. By forward BFS or FBFS (resp., backward BFS or BBFS) we mean a visit

Algorithm 1: the SumSweep heuristic.

Input: a graph $G = (V, E)$, a set $V' \subseteq V$, a node s, and an integer k
Output: an upper bound of the radius R_U and a lower bound of the diameter D_L of G
for $i \in V$ **do** $S^F(i) \leftarrow 0$; $S^B(i) \leftarrow 0$;
$F \leftarrow \{s\}$; $B \leftarrow \emptyset$;
FBFS(s);
$D_L \leftarrow e^F(s)$;
for $x \in V$ **do**
  if $d(s, x) < +\infty$ **then** $S^B(x) \leftarrow d(s, x)$;
end
for $i \in \{2, \ldots, k-1\}$ **do**
  if *i is odd* **then**
    $s \leftarrow argmax_{x \in V-F} S^F(x)$;
    $F \leftarrow F \cup \{s\}$;
    FBFS(s);
    $D_L \leftarrow \max(D_L, e^F(s))$;
    for $x \in V$ **do**
      if $d(s, x) < +\infty$ **then** $S^B(x) \leftarrow S^B(x) + d(s, x)$;
    end
  else
    $s \leftarrow argmax_{x \in V-B} S^B(x)$;
    $B \leftarrow B \cup \{s\}$;
    BBFS(s);
    $D_L \leftarrow \max(D_L, e^B(s))$;
    for $x \in V$ **do**
      if $d(x, s) < +\infty$ **then** $S^F(x) \leftarrow S^F(x) + d(x, s)$;
    end
  end
end
$s \leftarrow argmin_{x \in V'} S^F(x)$;
FBFS(s);
$R_U \leftarrow e^F(s)$;
return *D_L and R_U*

in BFS order in which a directed edge (v, w) is traversed from v to w (resp., from w to v). The new feature with respect to the previous heuristics is the choice of the starting vertices of the BFSs, which is based on the two following quantities, that try to distinguish "central" and "peripheral" vertices:

$$S^F(x) := \sum_{s \in B \cap \texttt{ReachF}(x)} d(x, s)$$
$$S^B(x) := \sum_{s \in F \cap \texttt{ReachB}(x)} d(s, x)$$

where F (resp. B) is the set of starting vertices of the forward (resp. backward) BFSs already performed. These quantities resemble $\frac{1}{c_x}$, where c_x is the *closeness centrality* of vertex x, that is, $\frac{1}{\sum_{v \in V} d(x,v)}$. The closeness centrality is a well-known centrality measure defined for the first time in 1950 [4] and more recently reconsidered when analyzing real-world networks (for more details, see [20] and the references therein). In particular, since a high closeness centrality value means that a vertex is central, if $S^F(x)$ or $S^B(x)$ is big, it means that x is "peripheral" and hence a good candidate to have a big eccentricity. For this reason, we

maximize $S^F(x)$ (resp. $S^B(x)$) when choosing the starting vertex of a forward (resp. backward) BFS. It is worth observing that the starting vertex of the first BFS plays a slightly different role: it should not be a vertex with high forward eccentricity, but a vertex which helps us finding high-eccentricity vertices in the next steps. To do so, for instance, we suggest to choose the maximum out-degree vertex. At the end of the procedure, we approximate the diameter with the maximum eccentricity found, and the radius with the eccentricity of the vertex $x \in V'$ minimizing $S^F(x)$. Observe that if the graph is undirected, the forward and backward eccentricities coincide: this means that a single BFS is enough to perform a forward and backward step of the SumSweep heuristic.

As an example, we will show the results of a SumSweep on the graph in Figure 1 with $k = 4$, $V' = \{0, 1, 2, 5, 6, 7, 8\}$, and $s = 1$ (which is the maximum out-degree vertex). The first forward BFS visits the whole graph and finds a lower bound $D_L = 4$ and an upper bound $R_U = 4$. The second step performs a backward BFS from vertex 8 (the only vertex at distance 4 from 1) and sets $D_L = 6$, since the distance from 2 to 8 is 6. This BFS is followed by a forward BFS from vertex 2, which has eccentricity 6 and consequently does not improve the previous bounds. Finally, a BFS from 8 is performed: since 8 has forward eccentricity 3, R_U is set to 3. Then, SumSweep returns $D_L = 6$ and $R_U = 3$ (note that these are the correct values of radius and diameter).

The effectiveness of this approach will be experimentally shown in Section 5.1.

4. The ExactSumSweep Algorithm

In this section we will show how to compute the exact values of the diameter and the radius of a graph, by using the bounds given by SumSweep. The general framework has been proposed in [31, 32] for undirected graphs: in this paper, we will adapt it for any directed graph. The general schema of our algorithm is shown in Algorithm 2.

Algorithm 2: the ExactSumSweep algorithm.

Input: a graph $G = (V, E)$, a set $V' \subseteq V$, a node s, and an integer k
Output: the radius R and the diameter D of G
for $i \in \{1, \ldots, |V|\}$ **do** $L^F(i) \leftarrow 0$; $L^B(i) \leftarrow 0$; $U^F(i) \leftarrow |V|$; $U^B(i) \leftarrow |V|$;
$D_L, R_U \leftarrow$ SumSweep(G, V', s, k);
while $(D_L < \max_{i \in V}\{U^F(i)\} \wedge D_L < \max_{i \in V}\{U^B(i)\}) \vee R_U > \min_{i \in V'}\{L^F(i)\}$ **do**
  Choose a technique from {StepForward, StepBackward, SingleCCUpperBound};
  Use it to update D_L, R_U, L^F, L^B, U^F, U^B;
end
return D_L, R_U;

After performing the SumSweep heuristic, the algorithm tries to prove that the computed bounds are the exact values of the radius and of the diameter, or to improve them. This can be done by bounding the eccentricities of all the vertices in the graph. More specifically, for each vertex v, we store these values:

- $L^F(v)$ is a lower bound on the forward eccentricity of v;
- $U^F(v)$ is an upper bound on the forward eccentricity of v;
- $L^B(v)$ is a lower bound on the backward eccentricity of v;
- $U^B(v)$ is an upper bound on the backward eccentricity of v.

As soon as, for a vertex v, $L^F(v) = U^F(v)$ (resp. $L^B(v) = U^B(v)$), the forward (resp. backward) eccentricity of v is exactly computed: in this case, the algorithm might improve the values of D_L and R_U. The value of D is established as soon as $D_L \geq \max_{v \in V} U^F(v)$ or $D_L \geq \max_{v \in V} U^B(v)$, and the value of R is established as soon as $R_U \leq \max_{v \in V'} L^F(v)$. This is because if $D_L \geq \max_{v \in V} U^F(v)$ or $D_L \geq \max_{v \in V} U^B(v)$, then this lower bound cannot be improved anymore, since the forward eccentricity of each other vertex is smaller

than D_L. Symmetrically, when $R_U \leq \min_{v \in V'} L^F(v)$, this upper bound cannot be improved anymore, and we can conclude that it is the actual value of the radius. Note that these inequalities are satisfied when all the eccentricities are known: since we will ensure that at each iteration a forward or a backward eccentricity is exactly computed, we need at most $\mathcal{O}(n)$ iterations, so that the worst-case running time is $\mathcal{O}(mn)$ as in the naive algorithm.

The computation of new lower bounds is based on performing a BFS from a vertex w and bounding the eccentricity of a visited vertex v with $d(v, w)$ or $d(w, v)$ (the techniques are named StepForward and StepBackward, depending on the direction of the BFS). The upper bound techniques are a bit more complicated: they choose a pivot vertex for each strongly connected component, they bound the eccentricities of pivot vertices, and they propagate these bounds within each strongly connected component. The hardest part is bounding the eccentricities of pivot vertices: the simplest technique, AllCCUpperBound, uses a dynamic programming approach, based on the topological ordering of the SCCs. This technique will not be used on its own, but it will be a significant part of SingleCCUpperBound, a more sophisticated technique, which is more time-consuming and provides better bounds. In particular, this technique performs a further forward and backward BFS from a given pivot q, allowing to improve the previous bounds when analyzing vertices reachable "by passing through q", while all other vertices are processed using the AllCCUpperBound technique. In the following three subsections, we will analyze each technique, we will provide more details, and we will prove the corresponding bounds. In Section 4.4 we will then analyze the running-time of each technique, in Section 4.5 we will show how these techniques apply to the special cases of strongly connected digraphs and of undirected graphs, and, finally, in Section 4.6 we will discuss how to select the technique to use at each step.

4.1. StepForward *and* StepBackward

The simplest technique performs a forward BFS from a "cleverly chosen" vertex w, setting $U^F(w) = L^F(w) = e^F(w)$ and, for each visited vertex v, $L^B(v) = \max(L^B(v), d(w, v))$ (StepForward). A similar technique can be applied by performing a backward BFS (StepBackward). Note that, since the algorithm starts by running the SumSweep heuristic, these bounds can also be computed during the first BFSs performed by the heuristic. For example, after the aforementioned SumSweep heuristic performed on the graph in Figure 1, the algorithm has already obtained the bounds in Table 1.

Table 1: Bounds obtained after the initial SumSweep heuristic, with $k = 4$. The sum of vertices whose eccentricity has already been computed exactly is set to -1 (in order to avoid these vertices to be chosen in subsequent BFSs).

Vertex	L^F	L^B	U^F	U^B	S^F	S^B
0	5	2	∞	∞	5	3
1	4	2	4	∞	-1	2
2	6	1	6	∞	-1	1
3	0	2	∞	∞	0	3
4	0	2	∞	∞	0	3
5	3	3	∞	∞	3	5
6	2	4	∞	∞	2	8
7	1	5	∞	∞	1	11
8	3	6	3	6	-1	-1
9	0	3	∞	∞	0	7
10	0	4	∞	∞	0	8
11	0	3	∞	∞	0	5

4.2. AllCCUpperBound

In order to compute upper bounds on the eccentricity of all vertices, we have to use more complicated techniques, based on the strong component graph (see Section 2). This technique chooses a "pivot vertex"

p_i for each SCC C_i of G and bounds the eccentricities of these vertices. Finally, it propagates these bounds by making use of the following inequalities, which are a simple consequence of the triangular inequality:

$$e^F(v) \leq d(v, p_i) + e^F(p_i) \tag{1}$$

$$e^B(v) \leq d(p_i, v) + e^B(p_i) \tag{2}$$

where v belongs to the SCC C_i having pivot p_i. In order to apply these inequalities, we need to compute upper bounds on the forward and backward eccentricity of each pivot in the graph ($e^F(p_i)$ and $e^B(p_i)$): before explaining this in full detail, we will show the main ideas of these bounds through an example, based on the graph in Figure 1.

Suppose we have chosen the pivots in Table 2 (actually this is the choice performed by the algorithm after the execution of SumSweep): we start to bound forward eccentricities in reverse topological order, that is, p_5, p_4, p_3, p_2, and p_1.

- Since no edge exits the SCC C_5, we set $U^F(p_5) = U^F(9) = e^F_{scc}(9) = d(9, 10) = 1$, where $e^F_{scc}(9)$ denotes the eccentricity of 9 if restricted to C_5.
- In order to bound the forward eccentricity of $p_4 = 5$, we observe that either the longest path stays in C_4, having length $e^F_{scc}(5)$, or it passes through the SCC C_5, reachable in one step from C_4 through edge $(C_4, C_5) \in \mathcal{E}$, which corresponds to the edge $(5, 9) \in E$, according to Figure 1. We bound the eccentricity of 5 with the maximum between $e^F_{scc}(5)$ and the length of a path from 5 to 9 passing through edge $(5, 9)$, plus $U^F(9)$ (this latter bound has already been computed thanks to the topological order). We obtain $U^F(5) = \max(e^F_{scc}(5), d(5, 5) + 1 + d(9, 9) + U^F(9)) = \max(3, 2) = 3$.
- $U^F(11) = e^F_{scc}(11) = 0$.
- There are two outgoing edges from C_2: $(C_2, C_3) \in \mathcal{E}$, which corresponds to $(3, 11) \in E$, and $(C_2, C_5) \in \mathcal{E}$, which corresponds to $(4, 9) \in E$. We bound $U^F(3)$ by considering the maximum among these possibilities: $U^F(3) = \max(e^F_{scc}(3), d(3, 3) + 1 + d(11, 11) + e^F(11), d(3, 4) + 1 + d(9, 9) + e^F(9)) = \max(1, 1, 3) = 3$.
- Finally, $U^F(0) = \max(e^F_{scc}(0), d(0, 1)+1+d(4, 3)+U^F(3), d(0, 1)+1+d(5, 5)+U^F(5)) = \max(2, 6, 5) = 6$.

The backward eccentricities are bounded similarly, considering SCCs in topological order. The forward and backward bounds computed in this way are summarized in Table 2.

Table 2: The bounds computed on the pivots of the different SCCs.

Pivot	SCC	U^F	U^B
$p_1 = 0$	C_1	6	2
$p_2 = 3$	C_2	3	5
$p_3 = 11$	C_3	0	6
$p_4 = 5$	C_4	3	4
$p_5 = 9$	C_5	1	7

Finally, we extend these bounds using inequalities (1) and (2): for instance, $U^F(1) = d(1, 0) + U^F(0) = 2 + 6 = 8$.

After showing the main ideas through this example, we may now formalize this intuition. In particular, we can generalize the example through the following lemma.

Lemma 4.1. *Given a SCC C_i with corresponding pivot p_i, for any j such that $(C_i, C_j) \in \mathcal{E}$ and $e_{ij} = (v_{ij}, w_{ij})$, the following formulas hold:*

$$e^F(p_i) \leq \max(e^F_{scc}(p_i), \max_{(C_i, C_j) \in \mathcal{E}}(d(p_i, v_{ij}) + 1 + d(w_{ij}, p_j) + e^F(p_j))) \tag{3}$$

$$e^B(p_i) \leq \max(e^B_{scc}(p_i), \max_{(C_j, C_i) \in \mathcal{E}}(d(w_{ji}, p_i) + 1 + d(p_j, v_{ji}) + e^B(p_j))) \tag{4}$$

Proof. We will prove the first formula, since the second is symmetric. Let x be one of the farthest vertices from p_i, so that $e^F(p_i) = d(p_i, x)$. If $x \in C_i$, $e^F(p_i) = d(p_i, x) = e^F_{scc}(p_i)$ and the first formula holds. Otherwise, the shortest path from p_i to x passes through a component C_j such that $(C_i, C_j) \in \mathcal{E}$. Then, $e^F(p_i) = d(p_i, x) \leq d(p_i, v_{ij}) + 1 + d(w_{ij}, p_j) + d(p_j, x) \leq d(p_i, v_{ij}) + 1 + d(w_{ij}, p_j) + e^F(p_j)$, and the first formula holds again. □

At this point, we may observe that the inequalities (3) and (4) can be "solved" recursively by analyzing strongly connected components with their corresponding pivot vertices in topological (resp. reverse topological) order, as we did in the previous example. Note that, if this procedure provides bounds that are worse than the bounds already stored in $U^F(p_i)$ or $U^B(p_i)$, then these latter bounds should be used. The pseudo-code for the computation of the pivot upper bounds and for the update of the upper bounds for each vertex is shown in Algorithm 3.

Algorithm 3: computing upper bounds for all vertices. Note that the graph $\mathcal{G}$ can be precomputed in linear time as well as a topological order at the beginning of the ExactSumSweep algorithm.

```
Let G = (V, E) be the graph of the SCCs in G.
U_P^F ← computePivotBoundsF (G);
U_P^B ← computePivotBoundsB (G);
for i = 0 to |V| do
    for v ∈ C_i do
        U^F(v) ← min(U^F(v), d(v, p_i) + U_P^F(p_i));
        U^B(v) ← min(U^B(v), d(p_i, v) + U_P^B(p_i));
    end
end
Procedure computePivotBoundsF(G)
    for i = |V| to 1 do
        U_P^F(p_i) ← min(U^F(p_i), max(e^F_scc(p_i), max_{(C_i,C_j)∈E}(d(p_i, v_ij) + 1 + d(w_ij, p_j) + U_P^F(p_j))));
    end
    return U_P^F;
Procedure computePivotBoundsB(G)
    for j = 1 to |V| do
        U_P^B(p_j) ← min(U^B(p_j), max(e^B_scc(p_j), max_{(C_i,C_j)∈E}(d(w_ij, p_j) + 1 + d(p_i, v_ij) + U_P^B(p_i))));
    end
    return U_P^B;
```

Before running these procedures, we just need to perform a forward and a backward BFS starting from p_i and restricted to C_i, for each i. In this way, we compute the following values:

1. $d(p_i, v), d(v, p_i)$ for each SCC C_i and each vertex $v \in C_i$;
2. $e^F_{scc}(p_i), e^B_{scc}(p_i)$ for each pivot vertex p_i.

4.3. SingleCCUpperBound

In order to further improve the upper bounds defined by Lemma 4.1, we introduce the SingleCCUpperBound technique. It requires two more BFSs from a pivot vertex q, that we call "main pivot": the technique works for any pivot, but a suitable choice provides better bounds. As in the previous section, we will first provide an example, then we will explain the technique and in Section 4.6 we will provide more details about the choice of the main pivot.

Our example deals again with the graph in Figure 1, choosing as main pivot vertex $q = 5$. We perform a forward and backward BFS from vertex 5, computing exactly its forward and backward eccentricities, thus setting $U^F(5) = 3$, and $U^B(5) = 3$. We will now analyze how to improve the previous forward

bounds (the backward case is completely analogous). First of all, we will use the previous technique to bound the forward eccentricities of all pivot vertices not visited in the backward BFS from 5, namely $U^F(3) = 5, U^F(9) = 1, U^F(11) = 0$. Then, we want to upper bound the eccentricity of 0, reached in the backward BFS from 5. We observe that $e^F(0) = d(0, x)$, for some vertex x: if x is reachable from the main pivot 5, $d(0, x) \leq d(0, 5) + d(5, x) \leq d(0, 5) + e^F(5)$. Otherwise, x is reachable from 0 by remaining in the graph G' obtained from G by removing all vertices in `ReachF`(5), that is, by removing all vertices visited in the forward BFS. We then set $U^F(0) = \max(d(0, 5) + U^F(5), U^F_{G'}(0))$, where $U^F_{G'}$ is the bound on the forward eccentricity of 0 obtained by running Procedure `computePivotBoundsF`($\mathcal{G}'$) in Algorithm 3, and $\mathcal{G}'$ is the strong component digraph of G' (note that $\mathcal{G}'$ can be computed without computing explicitly G'). We finally obtain $U^F(0) = \max(d(0, 5) + U^F(5), U^F_{G'}(0)) = \max(5, 3) = 5$. Forward and backward results are summarized in Table 3. Note that better forward bounds have been found for pivot 0, and better backward bounds have been found for pivot 9.

Table 3: The bounds computed on the pivots of the different SCCs by the SingleCCUpperBound technique.

Pivot	SCC	U^F	U^B
0	C_1	5	2
3	C_2	3	5
11	C_3	0	6
5	C_4	3	3
9	C_5	1	4

More formally, the SingleCCUpperBound technique is based on the following lemma.

Lemma 4.2. *Let p_i be a pivot, q be the main pivot, and suppose $p_i \in$ `ReachB`(q). If G' is the subgraph induced on G by removing* `ReachF`(q)*, the following inequality holds:*

$$e^F(p_i) \leq \max(d(p_i, q) + e^F(q), e^F_{G'}(p_i))$$

Proof. Let x be the farthest vertex from p_i: if $x \in$ `ReachF`(q), $e^F(p_i) = d(p_i, x) \leq d(p_i, q) + d(q, x) \leq d(p_i, q) + e^F(q)$. Otherwise, all paths from p_i to x are paths in G', so $e^F(p_i) = d(p_i, x) = d_{G'}(p_i, x) \leq e^F_{G'}(p_i)$, where by $d_{G'}$ we mean distances in the graph G'. In both cases, the lemma holds. □

The pseudo-code for the computation of the improved forward bounds is provided by Algorithm 4: it is enough to replace functions `computePivotBoundsF` and `computePivotBoundsB` in Algorithm 3 with their improved versions `computeImprovedPivotBoundsF` and `computeImprovedPivotBoundsB`. Note that, in order to compute forward and backward bounds, we need to perform a forward and a backward BFS from the main pivot, and a forward and backward BFS inside each SCC not containing the main pivot (for the SCC of the main pivot, the results of the first two BFSs can be used).

4.4. Running Time Analysis

In all previous papers that dealt with bound-refinement methods [14, 13, 23, 31, 32], the efficiency of an algorithm was defined in terms of the total number of BFSs needed before the values of D and R are found. However, if the graph is not strongly connected, the time needed to perform a BFS highly depends on the starting vertex: for instance, a forward BFS from a node with outdegree 0 takes very little time. As a consequence, we will consider as baseline the time needed to perform a BFS of the corresponding undirected graph, that is, the graph obtained by converting all directed edges into undirected edges. Since we deal with weakly connected graphs, this latter graph is connected, and the running time of a BFS on this graph does not depend on the starting vertex. The cost of the StepForward and StepBackward techniques is at most the cost of a BFS, because these techniques visit at most the same number of edges as a BFS of the corresponding undirected graph, and the operations performed when an edge is visited are the same. We will then consider the cost of these operations as 1 BFS. For the SingleCCUpperBound technique, we need to perform the following operations:

Algorithm 4: computing better upper bounds for some vertices.

Procedure `computeImprovedPivotBoundsF(`$\mathcal{G}$`)`
Let q be the main pivot.
Let $\mathcal{G}' = (\mathcal{V}', \mathcal{E}')$ be the subgraph of $\mathcal{G}$ induced by vertices not reachable from the SCC of q.
$U^F_{G'} \leftarrow$ `computePivotBoundsF(`$\mathcal{G}'$`)`;
$U^F_P \leftarrow$ `computePivotBoundsF(`$\mathcal{G}$`)`;
for $p_i \in$ `ReachB(`q`)` **do**
  $U^F_P(p_i) \leftarrow \min(U^F(p_i), \max(d(p_i, q) + e^F(q), U^F_{G'}(p_i)))$;
end
return U^F_P;

Procedure `computeImprovedPivotBoundsB(`$\mathcal{G}$`)`
Let q be the main pivot.
Let $\mathcal{G}' = (\mathcal{V}', \mathcal{E}')$ be the subgraph of $\mathcal{G}$ induced by vertices not reachable from the SCC of q.
$U^F_{G'} \leftarrow$ `computePivotBoundsB(`$\mathcal{G}'$`)`;
$U^F_P \leftarrow$ `computePivotBoundsB(`$\mathcal{G}$`)`;
for $p_i \in$ `ReachF(`q`)` **do**
  $U^B_P(p_i) \leftarrow \min(U^B(p_i), \max(d(p_i, q) + e^B(q), U^B_{G'}(p_i)))$;
end
return U^B_P;

1. a forward and a backward BFS from q and a BFS inside each SCC not containing q;
2. Algorithm 4, which does the following operations:
   (a) compute $\mathcal{G}'$;
   (b) perform twice `computePivotBoundsF` and `computePivotBoundsB` in Algorithm 3;
   (c) improve the bounds for each pivot vertex.
3. update all bounds using the inequalities (1), and (2), as in Algorithm 3.

It is clear that steps 2(a), 2(b) and 2(c) can be performed in time $\mathcal{O}(|\mathcal{G}|)$, and that step 3 can be performed in time $\mathcal{O}(|V|)$. Step 1 performs, for each visited edge, the same operations as a standard BFS, and each edge is visited at most three times in the whole operation. We will then consider the cost of running the SingleCCUpperBound technique as the cost of three BFSs, and we will ignore operations that are performed in $\mathcal{O}(|V| + |\mathcal{G}|)$ time, similarly to what has been done in previous papers for operations needing $\mathcal{O}(|V|)$ time [14, 13, 23, 31, 32]. This choice is justifiable also because, on average, in the dataset of our experiments, $|V| \approx 0.20 \cdot |E|$ and $|\mathcal{G}| \approx 0.17 \cdot |E|$.

4.5. Particular Cases

The Strongly Connected Case. In the strongly connected case, the aforementioned upper bound techniques AllCCUpperBound and SingleCCUpperBound collapse to a simpler technique, that performs a forward and backward BFS from the unique pivot vertex p, and bounds the eccentricity of any other vertex with $e^F(w) \leq d(w, p) + e^F(p)$ and $e^B(w) \leq d(p, w) + e^B(p)$. This technique costs two BFSs, differently from the SingleCCUpperBound technique that costs three BFSs. More specifically, Algorithm 3 becomes Algorithm 5.

The Undirected Case. In the undirected case, since we can deal with each connected component separately, again the two techniques are the same, and the cost reduces to one BFS, since the forward and backward BFSs coincide. Furthermore, we might improve the previous bounds by analyzing separately the first branch of the BFS tree and the other branches, as shown in the following lemma.

Algorithm 5: computing upper bounds for all vertices.

```
for v ∈ V do
    U^F(v) ← min(U^F(v), d(v, p) + U^F(p));
    U^B(v) ← min(U^B(v), d(p, v) + U^B(p));
end
```

Lemma 4.3. *Suppose we have performed a BFS from p, and we have obtained a tree T; let p' be the first vertex in T having more than one child. Let Φ be the set of vertices on the (only) path from p to p', let Ψ be the set of vertices in the subtree of T rooted at the first child of p', and let h be the maximum distance from p' to a vertex outside Ψ. Then, for each $v \in V$, $\epsilon(v) \leq U_p(v)$, where*

$$U_p(v) := \begin{cases} \max(d(p,v), \epsilon(p) - d(p,v)) & v \in \Phi \\ \max(d(p',v) + \epsilon(p') - 2, d(p',v) + h) & v \in \Psi \\ d(p',v) + \epsilon(p') & \textit{otherwise} \end{cases}$$

Proof. If $v \in \Phi$ or $v \notin \Phi \cup \Psi$, the conclusion follows easily by the triangle inequality. If $v \in \Psi$, let x be the farthest vertex from v: if $x \notin \Psi$, then $d(x,v) \leq d(x,p') + d(p',v) \leq h + d(p',v)$. If $x \in \Psi$ and r is the root of the subtree of T consisting of vertices in Ψ, $d(v,x) \leq d(v,r) + d(r,x) = d(v,p') + d(p',x) - 2 \leq d(v,p') + \epsilon(p') - 2$. □

In order to apply this lemma, after performing a BFS of an undirected graph, instead of bounding $e(v)$ with $d(v,p) + e(p)$, we may bound $e(v)$ with $U_p(v)$ (which is smaller or equal than $d(v,p) + e(p)$). This bound can be used everytime a BFS is performed, that is, not only during the SingleCCUpperBound technique, but also during a StepForward or a StepBackward. The pseudo-code of this procedure is provided in Algorithm 6, which replaces 3.

Algorithm 6: computing upper bounds for all vertices.

```
for v ∈ V do
    U(v) ← U_p(v);
end
```

4.6. Choosing the Technique to Use

In the previous subsections, several bound techniques have been defined. Here, we will explain how to put them together to be effective: in all previous papers, different techniques were alternated according to a fixed schema [31, 32]. In this paper, we will provide a different approach: a heuristic choosing the best technique will run at each step. The choice performed by this heuristic is based on the following definition, that quantifies how "close" we are to the solution.

Definition 4.4. *Let V_U be the smallest set among $\{v \in V : U^F(v) > D_L\}$, and $\{v \in V : U^B(v) > D_L\}$, and let V_L be $\{v \in V' : L^F(v) < R_U\}$. A vertex v is* open *if $v \in V_U \cup V_L$.*

At any point of the algorithm, the values of D_L and R_U can only be improved by the eccentricities of open vertices. For this reason, we will consider $N = |V_U| + |V_L|$ as a measure of the distance from the final solution. Thanks to this definition, we may now state how we are going to use the bounding techniques. In particular, this can be done as follows (see Section 3, for the definition of $S^F(v)$ and $S^B(v)$).

- StepForward from a vertex maximizing U^F (maximizing S^F is used to break ties);
- StepBackward from a vertex maximizing U^B (maximizing S^B is used to break ties);

- StepForward from a vertex in V' minimizing L^F (minimizing S^F is used to break ties);
- StepBackward from a vertex maximizing S^B (maximizing U^B is used to break ties);
- SingleCCUpperBound: the pivot of a SCC is chosen by minimizing $L^F(v) + L^B(v)$ among all vertices whose exact eccentricity has not been determined, yet; the main pivot is chosen as the pivot of the component C having more open vertices.

The last problem to address is which of these techniques should be chosen at any step, filling the gap in Algorithm 2.

Definition 4.5. *The utility U of a step is the difference between the value of N before the step and after the step.*

In order to speed up the computation, we want to perform steps with high utility. For this reason, for each technique, we keep an "expected utility" value U_E and at each step we choose the technique with the biggest expected utility. After a technique is applied, U_E is updated as follows: the expected utility of the technique used is set to U, while the expected utility of all other techniques is increased by $2/\texttt{iter}$, where `iter` is the number of BFSs already performed. This is done in order to alternate different techniques at the beginning, and to focus on a single technique when the best one becomes clear (even if, in the long run, every technique is applied).

5. Experimental Results

This section will experimentally show the effectiveness of the aforementioned techniques, by testing them on several real-world networks taken from the well-known datasets [29] and [19], which cover a large set of network types. In Section 5.1, we will show the effectiveness of the SumSweep heuristic compared to other similar heuristics. In Section 5.2, we will show that the SumSweep algorithm outperforms all previous algorithm in computing the diameter and radius of undirected and directed strongly connected graphs. Then, we will apply our algorithm to compute for the first time the diameter and radius of several directed, not strongly connected graphs: even in this case, our algorithm will have very good performances, even better than the strongly connected case (however, no comparison will be possible, since this is the first such algorithm). Finally, Section 5.3 will experimentally relate the performance of the SumSweep algorithm to various properties of the networks in the dataset. More detailed results of these experiments are provided in the appendix, and the code used is available on the website `http://piluc.dsi.unifi.it/lasagne/`.

5.1. Providing good lower bounds for the diameter

In this section, we will compare the SumSweep heuristic with the current most effective heuristics to compute lower bounds on the diameter [22, 14, 13], whose effectiveness has already been shown in [14, 13]. In the case of undirected graphs, we have compared the following heuristics:

k-SumSweep: performs a SumSweep from a random vertex, stopping after k BFSs;

4RandSamp: returns the maximum eccentricity among four random-chosen vertices;

4Sweep: the technique explained in [13];

2x2Sweep performs twice a 2Sweep [22] starting from two random vertices.

In the case of directed graphs, we have compared the following heuristics:

k-SumSweep: performs a SumSweep from a random vertex, stopping after k BFSs;

4RandSamp: returns the maximum eccentricity among four random-chosen vertices;

2-dSweep: the technique explained in [14].

Note that all the competitors above perform four BFSs, so they should be compared to the 4-SumSweep in order to obtain a fair comparison. We have run each heuristic ten times[3] for each graph, and we have considered the mean ratio r between the value returned and the diameter, among all ten experiments. In Table 4 we have reported the average r among all the graphs in the dataset, with the corresponding standard error. In Appendix B, the average r for each graph is provided.

As observed in [14, 13], the lower bound provided by the 4Sweep and 2-dSweep heuristics is usually tight, drastically outperforming the simple approach based on sampling, namely 4RandSamp. Table 4 shows that the SumSweep approaches are even improving these lower bounds: the 4-SumSweep is more effective both in the directed and the undirected case. Moreover, the SumSweep approach has another advantage: it is possible to further improve the bounds found by performing some more BFSs. This is very useful on directed graphs, while on undirected graphs the bounds in the 4-SumSweep are so good that they offer very little room for improvement.

Table 4: The average ratio r between the lower bound on the diameter returned by each heuristic and the diameter.

Method	r	Std Error
4-SumSweep	99.9830 %	0.0315 %
3-SumSweep	99.9671 %	0.0582 %
4Sweep	99.9353 %	0.1194 %
2x2Sweep	99.9295 %	0.1095 %
4RandSamp	76.9842 %	5.2841 %

(a) Undirected Graphs

Method	r	Std rror
8-SumSweep	97.2835 %	4.9030 %
7-SumSweep	97.2733 %	4.9010 %
6-SumSweep	97.2733 %	4.9010 %
5-SumSweep	96.8308 %	5.4324 %
4-SumSweep	96.6091 %	5.6564 %
3-SumSweep	95.5399 %	6.1901 %
2-dSweep	94.7607 %	6.5877 %
4RandomSamples	61.2688 %	15.1797 %

(b) Directed Graphs

5.2. Computing the Radius and the Diameter

The following set of experiments aims to show that the SumSweep algorithm improves the time bounds, the robustness, and the generality of all the existing methods, since they are outperformed for both radius and diameter computation, both in the directed and in the undirected case. Note that in the case of directed weakly connected graphs, there are no competitors (except the textbook algorithm), since SumSweep is the first algorithm able to deal with this case. Since any (weakly) connected component can be analyzed separately, we have restricted our attention to the biggest one, which usually contains most of the vertices in the graph (see Tables A.7–A.8 in the Appendix).

Undirected Graphs. In the undirected case, we compared our method with the state of the art: the iFub algorithm for the diameter and the BoundingDiameters (BD) algorithm both for the radius and for the diameter.

Indeed, this latter algorithm, used in [32] just to compute the diameter, can be easily adjusted to also compute the radius, using the same vertex selection strategy and updating rules for the eccentricity bounds. In particular, it bounds the eccentricity of vertices similarly to our method, by using the fact that, after a visit from a vertex v is performed, $d(v, w) \leq e(w) \leq d(v, w) + e(v)$. It does not perform the initial SumSweep and simply alternates between vertices v with the largest eccentricity upper bound and the smallest eccentricity lower bound.

For the diameter computation, we compared ExactSumSweep not only with BoundingDiameters, but also with two variations of iFub: iFubHd, starting from the vertex of highest degree, and iFub4S,

[3]Very low variance has been observed: even increasing the number of experiments does not change the shown results.

starting by performing a 4Sweep and choosing the central vertex of the second iteration (see [13] and related work for more details).

The results of the comparison are summarized in Table 5: for each method and for each graph in our dataset, we have computed the *performance ratio*, that is the percentage of the number of visits performed by the method with respect to the number of vertices of the network (i.e., the number of visits in the worst case). In Table 5 we report the average of these values on the whole dataset, together with the corresponding standard error.

In the diameter computation, the improvement is shown in Table 5(a). The new method is not only better than the previous ones on average, but it is even much more robust: the computation of the diameter for ExactSumSweep always ends in less than 180 BFSs, while the old methods need up to 1200 BFSs, as shown by Table C.11.

In the radius computation, the ExactSumSweep method is slightly more effective than the BoundingDiameters algorithm on average, as shown in Table 5(b). Again, we outline that the new method is much more robust: in our dataset, it never needs more than 18 BFSs, while the BoundingDiameters algorithm needs at most 156 BFSs. Moreover, in all the graphs in which the BoundingDiameters algorithm beats the ExactSumSweep algorithm, this happens always because of just one BFS. On the converse, when the ExactSumSweep algorithm beats the BoundingDiameters algorithm, the difference between the number of visits required can be much higher than 1: see Table C.11 for the detailed results.

Directed Strongly Connected Graphs. For the computation of the diameter of directed, strongly connected graphs, the best previous algorithms are four variations of the diFub method [14]:

diFubHdIn: starts from the vertex with highest in-degree;
diFubHdOut: starts from the vertex with highest out-degree;
diFub2In: starts from the central vertex of a 2Sweep performed from the vertex with highest in-degree;
diFub2Out: starts from the central vertex of a 2Sweep performed from the vertex with highest out-degree.

The results of the comparison with the new algorithm are shown in Table 5(c).

For the radius computation, the only efficient known method is explained in [23], which we will refer to as HR. Basically, it works as follows: given the farthest pair of vertices x and y found by the directed version of 2Sweep, order the vertices v according to $g(v) = \max\{d(v, x), d(v, y)\}$; scan the eccentricities of the vertices in this order and stop when the next vertex w has a value of $g(w)$ which is greater than the minimum eccentricity found. Since this method is the only algorithm to compute the radius, we compared our method just with this one. The results are shown in Table 5(d).

In the diameter computation, the best previous method is diFub2In: the new ExactSumSweep method performs more than 4 times better. We note again the robustness: the maximum number of BFSs is 59, against the maximum number for diFub2Out which is 482 (note that the maximum for the best competitor of ExactSumSweep, diFub2In, is 510).

In the radius computation, the ExactSumSweep algorithm performs about 31 times better than the old method. We also remark that the robustness of ExactSumSweep applies also to the directed case: at most 36 BFSs are needed to find the radius of any graph of our dataset.

The Directed General Case. Finally, we have analyzed the performances of ExactSumSweep in computing diameter and radius of directed, not strongly connected graphs. Due to the lack of similar algorithms, it is possible to compare the new method only with the textbook one, namely, performing a BFS from each vertex. The performances of the new method are on average about 1000 times better than the textbook algorithm, allowing us to compute the diameter and radius of several real-world networks for the first time. Moreover, it is worth observing that in the case of weakly connected directed graphs, ExactSumSweep seems to perform even better with respect to strongly connected graphs, if the performance ratio is considered.

Overall, we conclude that the new method is more general: it is the only method which is able to deal with both directed strongly connected and undirected graphs, both for the radius and the diameter

computation, with very small variations. Moreover, it is the only existing algorithm able to deal with weakly connected directed graphs, apart from the textbook one. Despite its generality, it is also faster than every existing algorithm, both on average running time and on robustness.

Table 5: The average performance ratio p, i.e., percentage of the number of BFSs used by the different methods, with respect to the number of vertices (number of visits using the textbook algorithm).

Undirected Connected Graphs

Method	p	Std Error
ExactSumSweep	0.0572 %	0.0567 %
BD	0.2097 %	0.2509 %
iFub4S	1.6937 %	2.5603 %
iFubHd	3.2550 %	5.4185 %

(a) Diameter

Method	p	Std Error
ExactSumSweep	0.0279 %	0.0276 %
BD	0.0427 %	0.0486 %

(b) Radius

Directed Strongly Connected Graphs

Method	p	Std Error
ExactSumSweep	0.1990 %	0.2245 %
diFub2In	0.8429 %	1.1937 %
diFub2Out	0.8458 %	1.2026 %
diFubHdOut	1.7454 %	2.6369 %
diFubHdIn	2.3249 %	3.3871 %

(c) Diameter

Method	p	Std Error
ExactSumSweep	0.1935 %	0.2203 %
HR	6.0136 %	8.0915 %

(d) Radius

Directed Weakly Connected Graphs

Method	p	Std Error
ExactSumSweep	0.1220 %	0.0969 %

(e) Diameter

Method	p	Std Error
ExactSumSweep	0.1367 %	0.1045 %

(f) Radius

5.3. Correlating ExactSumSweep *Effectiveness with Graph Properties*

In this section, we will attempt to quantify how the performance of the ExactSumSweep algorithm is related to specific properties of the graphs considered. It is clearly visible from the results shown in the previous section that a significant improvement is realized with respect to to the textbook algorithm for determining the radius and diameter which runs in $\mathcal{O}(|V| \cdot |E|)$. Other than expressing the number of iterations as a fraction of the number of vertices, it may be worth seeing if there are other graph-based properties that are related to the number of iterations needed by the algorithm.

Therefore, we will consider several graph properties and compare each of them with the number of iterations for computing both the directed radius and diameter of the SCC. To make the comparison, we will use the Pearson correlation coefficient, defined as the covariance of the two variables divided by the product of their standard deviations, essentially measuring the extent to which there is a linear correlation between the graph property and the number of iterations. The considered graph properties are shown in the different rows of Table 6 and include the number of vertices, number of edges, average degree, average distance, density, average local clustering coefficient (defined as the average (over all nodes) of the fraction of closed triangles amongst the direct neighbors of a node) and the (percentage of) vertices/edges in the SCC. We will also measure parameters that depend directly on the computed metrics, such as the average eccentricity, the radius and diameter itself, and the relation between the two.

Table 6 shows the respective correlations for the number of iterations needed by ExactSumSweep to compute the radius and the diameter. We say that a value close to zero indicates no significant correlation, whereas the correlation is higher as the value of the coefficient approaches 1 or -1. A numeric value greater than 0.5 (or smaller than -0.5) indicates that there is some stronger correlation (shown in bold) between the two variables.

Table 6: Correlation between different graph properties and the number of iterations used by SumSweep to compute the radius and diameter.

Graph Property	Radius	Diameter
Vertices $\|V\|$	-0.246	-0.244
Edges $\|E\|$	-0.274	-0.224
Average degree $\|E\|/\|V\|$	-0.125	-0.065
Average distance $\overline{d}$	0.108	-0.095
Density $\|E\|/(\|V\|(\|V\|-1))$	0.134	0.082
Clustering coefficient	0.108	-0.095
Vertices in SCC	-0.188	-0.201
Percentage of vertices in SCC	0.142	0.204
Edges in SCC	-0.234	-0.182
Percentage of edges in SCC	0.040	0.120
Average eccentricity $\overline{e}$	0.128	-0.077
Radius R	0.147	-0.083
Diameter D	0.115	-0.107
$(D-R)/D$	-0.294	$\mathbf{-0.528}$

Most noteworthy in Table 6 is the relation between the diameter and radius $(D-R)/D$ and the number of iterations to compute the diameter: the higher this value is, the lower of number of iterations (the correlation coefficient is equal to -0.528). A similar observation was made in [30], where the relation between the radius and diameter was also shown to be of influence on the performance of the BoundingDiameters algorithm.

In general, we note that for most of the graph properties listed in Table 6, the correlation is low. A limitation of this comparison approach is obviously that the set of graphs never encompasses all possible combinations of different graph properties (see result tables in Appendix C). One can also immediately observe that some of the graph properties in Table 6 are related by definition, and therefore also show similar correlations with the number of iterations. Nevertheless, the fact that no significant correlation is found is actually a positive result, as it seems that ExactSumSweep works well in most graphs, independently of their properties.

6. Internet Movies Database Case Study

This section applies the ExactSumSweep algorithm to the Internet Movies Database, in particular to the so-called actor graph, in which two actors are linked if they played together in a movie (we ignore TV-series in this work). All data have been taken from the website `http://www.imdb.com`. In line with [27], we decided to exclude some genres from our database: awards-shows, documentaries, game-shows, news, realities and talk-shows. We analyzed snapshots of the actor graph, taken every 5 years from 1940 to 2010, and 2014.

6.1. Analysis of the Graph Stretch

Figure 2 shows the evolution of the diameter, the radius and the eccentricity of some actors. It shows that the stretch of the graph (in terms of radius and diameter) increased between 1940 and 1955, then it started decreasing, as it has been observed in [21]. The first increase might be explained by the fact that the years between the forties and the sixties are known as the golden age for Asian cinema, especially Indian and Japanese.[4] Examples of popular movies of that period include *Tokyo Story* and *Godzilla*. This trend is also confirmed by the names of the central actors during that period. In 1940, they are all Western, usually German, like Carl Auen. The only non-western central actor is the Birmanian Abraham Sofaer, but he

[4] All other historical data in this section is taken from [33].

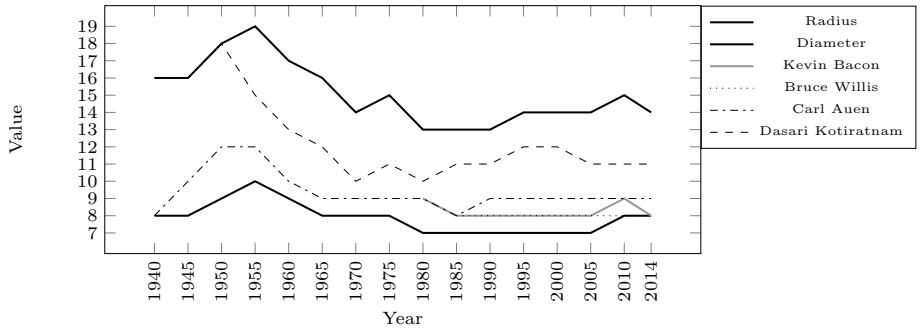

Figure 2: Actor graph evolution in terms of radius, diameter, and actor eccentricity.

worked in England and America. On the other hand, in 1955, we find both Western actors like James Bell and Eastern actors like Sjin Kamiyama.

Later, in the sixties, the increase in independent producers and growth of production companies led to an increase of power of individual actors. This can explain the decreasing size of the graph during those years: the number of contacts between actors from different countries increased. An example of this trend is the first James Bond movie, *Dr. No*, starring Sean Connery (from Scotland), Ursula Andress (who is Swiss-American), Joseph Wiseman (from Canada), etc.

The decrease of the graph size halted in the eighties, and there were little changes until the present. Now it seems that the size is slightly increasing again, but the number of central actors is increasing as well. It will be interesting to see if this trend will continue or there will soon be an actor who will obtain again an eccentricity of 7.

6.2. Analysis of the Eccentricity of Actors

All actors seem to decrease their eccentricity as time passes. Even Dasari Kotiratnam, an Indian actress active between 1935 and 1939, is a diametral vertex when she is active, then she becomes more and more central. Also Carl Auen, who started from the center in 1940, remains quite central, having an eccentricity of 9 in the present.

Instead, the periphery is usually composed by recent actors: for example, in the last graph, the actress Neeta Dhungana is a diametral vertex who played only in the Nepalese movie *Masaan*. Another example is Steveanna Roose, an American actress who played only in *Lost in the Woods*: in 2010 she is diametral, having eccentricity 15, while in 2014 she obtained an eccentricity of 12 (without playing any movie in that period).

We also remark that Kevin Bacon has not minimum eccentricity until the present, and he never gets eccentricity 6, as suggested by the "Six Degrees of Kevin Bacon" game. Hence, not all the actors can be linked to Kevin Bacon by using at most 6 edges.

7. Wikipedia Case Study

The Wikipedia graph consists of all pagelinks between (English) Wikipedia articles and can be downloaded at DBpedia [2]. In the "Six Degrees of Wikipedia" game one is asked to connect two given Wikipedia pages, i.e., a source and a target page, by using at most six links. In this section, we will analyze the Wikipedia directed graph, trying to understand whether the "Six Degrees of Wikipedia" game is always solvable whenever a path from the source to the target exists. We will compute for the first time the radius and diameter of the whole Wikipedia graph, we will further analyze the biggest SCC, and finally we will try to avoid the problems generated by "long paths" inside the graph.

First of all, we have computed the radius and the diameter of the whole Wikipedia graph (which is composed by 4,229,722 nodes and 102,165,856 edges, with 452,488 strongly connected components). The

diameter is 377 (254 iterations needed by using ExactSumSweep Algorithm) and the radius is 42 (203 iterations). Note that these values are extremely high if compared with diameter and radius of real-world networks: in order to explain this phenomenon, it is worth analyzing the paths involved. In particular, the diameter starts from page `List_of_minor_planets/108101108200` and remains inside this list until page `List_of_minor_planets/145701145800`, in order to reach the last page `(145795)_1998_RA16` (which is a minor planet). Clearly the diameter only depends on this list and does not give any information about the connectivity of the remaining part of the graph.

A similar phenomenon holds for the radius: a radial vertex is the page `1954_in_Ireland`, and a longest path from this vertex reaches in 6 steps the page `Papyrus_Oxyrhynchus_116`, where a long path starts until the page `Papyrus_Oxyrhynchus_158`, adding 36 steps.[5] Moreover, the first step after the page `1954_in_Ireland` is the page `England`, but this latter vertex has bigger eccentricity because of the existence of a long path of pages with title `All-Ireland_Minor_Hurling_Championship`: the reason why the page `1954_in_Ireland` is central is that it stays "in the middle" of the two lists `All-Ireland_Minor_Hurling_Championship` and `Papyrus_Oxyrhynchus`, not meaning that it is well connected to the rest of the graph.

Similar results are found if we restrict ourselves to the biggest SCC of the graph, composed by 3,763,632 nodes. The diameter is 49 (9 iterations) and the radius is 42 (14 iterations). A diametral path is again based on the `All-Ireland_Minor_Hurling_Championship` path, starting from `Kickxellales` (an order of fungus) and ending into `All-Ireland_Minor_Hurling_Championship_1928`. A radial vertex is `Play_it_Again_Des`, and again the longest path from this vertex reaches `All-Ireland_Minor_Hurling_Championship_1928`.

The last experiment tries to "eliminate" these long paths from the graph: to do so, we have modified all page titles by leaving only letters (case-sensitive), and we have collapsed all the pages having the same title. In this way, all numbers are deleted and the previous long paths of pages collapse to a single vertex: for instance, all pages like `All-Ireland_Minor_Hurling_Championship_YYYY` for years `YYYY` collapse to the single page `AllIrelandMinorHurlingChampionship`. After this procedure, the graph has 3,939,060 nodes, the diameter becomes 23 (14 iterations) and the radius becomes 17 (16 iterations). However, this is still not sufficient to analyze the structure of the graph, since many important different pages collapse together: for instance the collapsed page `s` (a radial vertex) corresponds to all pages like `1960s`, `1970s`, `1980s`, etc. Moreover, the problem of long lists is not completely solved yet, because the diameter starts from page `Advanced_Diabetes_Management_Certification` and ends to page `Osieki_Lborskie_railway_station` (a Polish railway station): 15 steps of this path pass from a Polish railway station to another (the list starts from `Lbork_railway_station`). The same issues hold if only the biggest SCC is considered, since all considered paths are contained in the biggest SCC (the iterations become 15 for the diameter and 13 for the radius by using ExactSumSweep Algorithm). We argue that dealing with these long paths in this graph is a difficult task that require more sophisticated techniques exploiting the content of the pages.

More interesting results can be obtained by reversing all the edges in the origina graph: in this way, a radial vertex is an "easily reachable vertex" and not a vertex that reaches easily all the others. In this case, the radius is very small, because "popular vertices" are cited by many other pages. Indeed, although the diameter remains obviously 377, the radius becomes 6 (7 iterations), and a radial vertex is the page `United_States_of_America`. Intuitively, this happens because it is much easier "to reach very popular nodes" than to reach unpopular nodes by starting from very popular ones. This means that if we fix the target of the "Six Degrees of Wikipedia" game to be the `United_States_of_America` the game becomes always solvable!

8. Conclusion and Open Problems

In this paper, we have presented a very efficient algorithm to compute radius and diameter of real-world networks. We have proved that this algorithm is more general than all previous ones, and it is

[5] In the path of the pages `Papyrus_Oxyrhynchus`, a jump is done from page `Papyrus_Oxyrhynchus_145` to page `Papyrus_Oxyrhynchus_152`

the first algorithm able to compute the radius and diameter of directed, not strongly connected graphs. Furthermore, in this latter case, we have generalized the definition of radius, in order to make it meaningful in the context of complex networks. However, the algorithm proposed does not only work with our new definition: with an appropriate choice of the set V', it can compute the radius according to any previously proposed definition. We have experimentally shown the effectiveness of the new approach, that outperforms all previous approaches in generality, average running time and robustness, using a dataset of several real-world networks of different types. Finally, we have applied our algorithm to two case-studies: the actors graph and the Wikipedia graph. In the first case, we have outlined that the evolution of the radius and the diameter mirrors events in the history of cinema, we have shown that the "Six Degrees of Separation" game has never been solvable for this graph, and that Kevin Bacon was not the center of the graph until the present day. In the second case, we have computed for the first time both the diameter and radius of the whole Wikipedia graph, and we have outlined that the presence of long lists heavily affects these values. After this, we have found an instance of the "Six Degrees of Wikipedia" game that is actually solvable, that is, fixing the target to be the page `United_States_of_America`.

Some significant problems are left open by this paper. First of all, it would be interesting to give an "explanation" for the surprisingly good results obtained by this algorithm. It is clear that these results heavily rely on the properties of real-world networks, since they do not hold, for instance, for random graphs generated with the Erdös-Renyi model (see [26] for more background on the model and [23] for the results of these algorithms). It would be interesting to analyze the performances of these algorithms on other models of random graphs, both with probabilistic methods and with testing. These results could help us outlining the key properties that make these algorithms work, also based on the analysis performed in Section 5.3: it would be nice to prove the lack of these correlations, or the existence of correlation with other properties, at least on graphs generated with a particular model. Another significant open problem is whether these algorithms work on particular classes of graphs, or more generally, if it is possible to beat the $\mathcal{O}(mn)$ worst-case running time in finding the diameter of any planar graph or any directed acyclic graph (it has already been proved in [28, 9] that it is highly improbable to improve this running time in split graphs and chordal graphs).

Acknowledgments

The authors would like to thank the anonymous reviewers for their helpful suggestions.

References

[1] Donald Aingworth, Chandra Chekuri, Piotr Indyk, and Rajeev Motwani. Fast Estimation of Diameter and Shortest Paths (without Matrix Multiplication). *SIAM Journal on Computing*, 28:1167–1181, 1999.

[2] Sören Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak, and Zachary Ives. DBpedia: A Nucleus for a Web of Open Data. In *The Semantic Web*, volume 4825 of *LNCS*, pages 722–735, 2007.

[3] Jørgen Bang-Jensen and Gregory Gutin. *Digraphs Theory, Algorithms and Applications*. 2nd edition, 2008.

[4] Alex Bavelas. Communication Patterns in Task-Oriented Groups. *Journal of the Acoustical Society of America*, 22:725–730, 1950.

[5] Boaz Ben-Moshe, Binay Bhattacharya, Qiaosheng Shi, and Arie Tamir. Efficient Algorithms for Center Problems in Cactus Networks. *Theoretical Computer Science*, 378:237–252, 2007.

[6] Michele Borassi, Pierluigi Crescenzi, Michel Habib, Walter Kosters, Andrea Marino, and Frank Takes. On the Solvability of the Six Degrees of Kevin Bacon Game — A Faster Graph Diameter and Radius Computation Method. In *Fun with Algorithms*, volume 8496 of *LNCS*, pages 57–68, 2014.

[7] Ulrik Brandes and Thomas Erlebach. *Network Analysis: Methodological Foundations*. Springer-Verlag, 2005.

[8] Andrei Z. Broder, Ravi Kumar, Farzin Maghoul, Prabhakar Raghavan, Sridhar Rajagopalan, Raymie Stata, Andrew Tomkins, and Janet L. Wiener. Graph Structure in the Web. *Computer Networks*, 33:309–320, 2000.

[9] Shiri Chechik, Daniel Larkin, Liam Roditty, Grant Schoenebeck, Robert E. Tarjan, and Virginia Vassilevska Williams. Better Approximation Algorithms for the Graph Diameter. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1041–1052, 2014.

[10] Alex Clemesha. The Wiki Game. `http://thewikigame.com`, 2013.

[11] Derek G. Corneil, Feodor F. Dragan, Michel Habib, and Christophe Paul. Diameter Determination on Restricted Graph Families. *Discrete Applied Mathematics*, 113:143–166, 2001.

[12] Derek G. Corneil, Feodor F. Dragan, and Ekkehard Köhler. On the power of BFS to determine a graph's diameter. *Networks*, 42(4):209–222, 2003.
[13] Pierluigi Crescenzi, Roberto Grossi, Michel Habib, Leonardo Lanzi, and Andrea Marino. On Computing the Diameter of Real-World Undirected Graphs. *Theoretical Computer Science*, 514:84–95, 2013.
[14] Pierluigi Crescenzi, Roberto Grossi, Leonardo Lanzi, and Andrea Marino. On Computing the Diameter of Real-World Directed (Weighted) Graphs. In *Experimental Algorithms (SEA)*, volume 7276 of *LNCS*, pages 99–110, 2012.
[15] Michael Gurevich. *The Social Structure of Acquaintanceship Networks*. PhD thesis, MIT, Boston, USA, 1961.
[16] G. Handler. Minimax location of a facility in an undirected tree graph. *Trans Sci*, 7:287–293, 1973.
[17] Frank Harary. *Graph Theory*. Addison-Wesley Series in Mathematics. Perseus Books, 1994.
[18] Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which Problems Have Strongly Exponential Complexity? *Journal of Computer and System Sciences*, 63:512–530, 2001.
[19] Jérôme Kunegis. KONECT — The Koblenz Network Collection. In *Proceedings of the International Conference on World Wide Web Companion*, pages 1343–1350, 2013.
[20] Vito Latora and Massimo Marchiori. A Measure of Centrality Based on Network Efficiency. *New Journal of Physics*, 9(188), 2007.
[21] Jure Leskovec, Jon Kleinberg, and Christos Faloutsos. Graph Evolution: Densification and Shrinking Diameters. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 1(1–2), 2007.
[22] Clémence Magnien, Matthieu Latapy, and Michel Habib. Fast Computation of Empirically Tight Bounds for the Diameter of Massive Graphs. *Journal of Experimental Algorithmics*, 13:1–7, 2009.
[23] Andrea Marino. *Algorithms for Biological Graphs: Analysis and Enumeration*. PhD thesis, Dipartimento di Sistemi e Informatica, University of Florence, Italy, 2013.
[24] Stanley Milgram. The Small World Problem. *Psychology Today*, 2:60–67, 1967.
[25] Alan Mislove, Massimiliano Marcon, Krishna Gummadi, Peter Druschel, and Samrat Bhattacharjee. Measurement and Analysis of Online Social Networks. In *Proceedings of the 7th Usenix/ACM SIGCOMM Internet Measurement Conference (IMC)*, pages 29–42, 2007.
[26] Mark E.J. Newman. The Structure and Function of Complex Networks. *SIAM Review*, 45:167–256, 2003.
[27] Patrick Reynolds. The Oracle of Kevin Bacon. `http://oracleofbacon.org`, 2013.
[28] Liam Roditty and Virginia Vassilevska Williams. Fast Approximation Algorithms for the Diameter and Radius of Sparse Graphs. In *Proceedings of the 45th ACM Symposium on the Theory of Computing (STOC)*, pages 515–524, 2013.
[29] SNAP. Stanford Network Analysis Package Website. `http://snap.stanford.edu`, 2009.
[30] Frank W. Takes. *Algorithms for Analyzing and Mining Real-World Graphs*. PhD thesis, LIACS, Universiteit Leiden, The Netherlands, 2014.
[31] Frank W. Takes and Walter A. Kosters. Determining the Diameter of Small World Networks. In *Proceedings of the 20th ACM Conference on Information and Knowledge Management (CIKM)*, pages 1191–1196, 2011.
[32] Frank W. Takes and Walter A. Kosters. Computing the Eccentricity Distribution of Large Graphs. *Algorithms*, 6:100–118, 2013.
[33] Kristin Thompson and David Bordwell. *Film History: An Introduction*. McGraw-Hill, 3rd edition, 2009.
[34] Douglas B. West. *Introduction to Graph Theory*. Prentice Hall, 2nd edition, 2000.
[35] Virginia Vassilevska Williams. Multiplying Matrices Faster than Coppersmith-Winograd. In *Proceedings of the 44th ACM Symposium on the Theory of Computing (STOC)*, pages 887–898, 2012.

Appendix A. Dataset

The following tables provide the graphs included in our dataset, showing for each network the number of vertices (n) and the number of edges (m). Moreover for each undirected network, we report the number of vertices (n_{cc}), the number of edges (m_{cc}), the diameter (D), and the radius (R) of the biggest connected component. Furthermore, for each directed network, we report the number of vertices (n_{wcc}), the number of edges (m_{wcc}), the diameter (D), and the radius (R) of the largest weakly connected component together with the number of vertices (n_{scc}), the number of edges (m_{scc}), the diameter (D_{scc}), and the radius (R_{scc}) of the biggest strongly connected component.

Table A.7: undirected Graphs: n and m indicate respectively the number of nodes and the number of edges, D and R indicate diameter and radius, and the subscript cc refers to the biggest connected component.

Network	n	m	n_{cc}	m_{cc}	D	R
as20000102	6474	12572	6474	12572	5	9
CA-AstroPh	18772	198050	17903	196972	8	14
CA-CondMat	23133	93439	21363	91286	8	15
ca-GrQc	5241	14484	4158	13422	9	17
ca-HepPh	12006	118489	11204	117619	7	13
ca-HepTh	9875	25973	8638	24806	10	18
com-amazon.all.cmty	134386	99433	7011	8955	20	39
com-amazon.ungraph	334863	925872	334863	925872	24	47
com-dblp.ungraph	317080	1049866	317080	1049866	12	23
com-lj.all.cmty	477998	530872	303526	427701	16	32
com-youtube.ungraph	1134890	2987624	1134890	2987624	12	24
email-Enron	36692	183831	33696	180811	7	13
facebook_combined	4039	88234	4039	88234	4	8
flickrEdges	105938	2316948	105722	2316668	5	9
gowalla_edges	196591	950327	196591	950327	8	16
loc-brightkite_edges	58228	214078	56739	212945	9	18
oregon1_010519	11051	22724	11051	22724	6	11
oregon1_010526	11174	23409	11174	23409	5	10
oregon2_010519	11375	32287	11375	32287	5	9
oregon2_010526	11461	32730	11461	32730	5	9
orkut-links	3072441	117185083	3072441	117185083	5	10
p2p-Gnutella09	8114	26013	8104	26008	6	10
roadNet-CA	1965206	2766607	1957027	2760388	494	865
roadNet-PA	1088092	1541898	1087562	1541514	402	794
roadNet-TX	1379917	1921660	1351137	1879201	540	1064
soc-pokec-relationships	1632803	44603928	1632803	44603928	14	7
youtube-u-growth	3223585	9375374	3216075	9369874	16	31

Table A.8: directed Graphs: n and m indicate respectively the number of nodes and the number of edges, D and R indicate diameter and radius, and subscripts wcc and scc refer respectively to the biggest weakly connected component and the biggest strongly connected component.

Network	n	m	n_{wcc}	m_{wcc}	n_{scc}	m_{scc}	D	R	D_{scc}	R_{scc}
amazon0302	262111	1234877	262111	1234877	241761	1131217	88	50	88	48
amazon0312	400727	3200440	400727	3200440	380167	3069889	53	28	52	26
amazon0505	410236	3356824	410236	3356824	390304	3255816	55	27	55	27
amazon0601	403394	3387388	403364	3387224	395234	3301092	54	29	52	25
as-caida20071105	26475	106762	26475	106762	26475	106762	17	9	17	9
ca-AstroPh	18771	396100	17903	393944	17903	393944	14	8	14	8
ca-CondMat	23133	186878	21363	182572	21363	182572	15	8	15	8
ca-GrQc	5241	28968	4158	26844	4158	26844	17	9	17	9
ca-HepPh	12006	236978	11204	235238	11204	235238	13	7	13	7
ca-HepTh	9875	51946	8638	49612	8638	49612	18	10	18	10
cit-HepPh	34546	421534	34401	421441	12711	139965	49	12	49	15
cit-HepTh	27769	352768	27400	352504	7464	116252	37	12	35	13
cit-Patents	3774768	16518947	3764117	16511740	1	0	24	0	0	0

Network	n	m	n_{wcc}	m_{wcc}	n_{scc}	m_{scc}	D	R	D_{scc}	R_{scc}
email-EuAll	265009	418956	224832	394400	34203	151132	11	5	10	5
flickr-growth	2302925	33140017	2173370	32948343	1605184	30338513	27	13	27	12
p2p-Gnutella04	10876	39994	10876	39994	4317	18742	26	15	25	15
p2p-Gnutella05	8846	31839	8842	31837	3234	13453	22	14	22	14
p2p-Gnutella06	8717	31525	8717	31525	3226	13589	21	13	19	12
p2p-Gnutella08	6301	20777	6299	20776	2068	9313	20	13	19	12
p2p-Gnutella09	8114	26013	8104	26008	2624	10776	20	14	19	13
p2p-Gnutella24	26518	65369	26498	65359	6352	22928	29	16	28	15
p2p-Gnutella25	22687	54705	22663	54693	5153	17695	22	14	21	13
p2p-Gnutella30	36682	88328	36646	88303	8490	31706	24	16	23	15
p2p-Gnutella31	62586	147892	62561	147878	14149	50916	31	20	30	19
soc-Epinions1	75879	508837	75877	508836	32223	443506	16	8	16	8
soc-sign-epinions	131828	840799	119130	833390	41441	693507	16	8	16	7
soc-sign-Slash081106	77350	516575	77350	516575	26996	337351	15	7	15	7
soc-sign-Slash090216	81867	545671	81867	545671	27222	342747	15	7	15	7
soc-sign-Slash090221	82140	549202	82140	549202	27382	346652	15	7	15	7
soc-Slashdot0811	77360	828161	77360	828161	70355	818310	12	7	12	7
soc-Slashdot0902	82168	870161	82168	870161	71307	841201	13	7	13	7
trec-wt10g	1601787	8063026	1458316	7487449	470441	3012375	351	79	130	37
web-BerkStan	685230	7600595	654782	7499425	334857	4523232	694	283	679	249
web-Google	875713	5105039	855802	5066842	434818	3419124	51	27	51	24
web-NotreDame	325729	1469679	325729	1469679	53968	296228	93	44	93	44
web-Stanford	281903	2312497	255265	2234572	150532	1576314	580	134	210	97
wiki-Talk	2394385	5021410	2388953	5018445	111881	1477893	11	5	10	5
wiki-Vote	7115	103689	7066	103663	1300	39456	10	4	9	3
youtube-links	1138494	4942297	1134885	4938950	509245	4269142	23	13	20	10
zhishi-baidu	2141300	17632190	2107689	17607140	609905	8300678	39	17	39	17
zhishi-hudong	1984484	14682258	1962418	14672183	365558	4689296	31	19	31	19

Appendix B. The SumSweep Heuristic

Table B.9: efficiency of different lower bound techniques on undirected graphs.

Network	3-SumSweep	4-SumSweep	2x2Sweep	4Sweep	4RandSamp
as20000102	100.00%	100.00%	100.00%	100.00%	80.00%
CA-AstroPh	100.00%	100.00%	100.00%	100.00%	76.43%
CA-CondMat	100.00%	100.00%	100.00%	100.00%	74.00%
ca-GrQc	100.00%	100.00%	100.00%	100.00%	78.24%
ca-HepPh	100.00%	100.00%	100.00%	100.00%	79.23%
ca-HepTh	100.00%	100.00%	100.00%	100.00%	76.11%
com-amazon.all.cmty	100.00%	100.00%	100.00%	100.00%	77.18%
com-amazon.ungraph	100.00%	100.00%	100.00%	100.00%	75.74%
com-dblp.ungraph	100.00%	100.00%	100.00%	100.00%	71.74%
com-lj.all.cmty	100.00%	100.00%	100.00%	99.69%	70.31%
com-youtube.ungraph	100.00%	100.00%	100.00%	100.00%	65.42%
email-Enron	100.00%	100.00%	100.00%	100.00%	70.77%
facebook_combined	100.00%	100.00%	100.00%	100.00%	87.50%
flickrEdges	100.00%	100.00%	100.00%	100.00%	86.67%
gowalla_edges	100.00%	100.00%	100.00%	100.00%	71.88%
loc-brightkite_edges	100.00%	100.00%	100.00%	100.00%	67.78%
oregon1_010519	100.00%	100.00%	100.00%	100.00%	80.00%
oregon1_010526	100.00%	100.00%	100.00%	100.00%	76.00%
oregon2_010519	100.00%	100.00%	100.00%	100.00%	77.78%
oregon2_010526	100.00%	100.00%	100.00%	100.00%	76.67%
p2p-Gnutella09	100.00%	100.00%	100.00%	100.00%	85.00%
roadNet-CA	99.64%	99.88%	99.28%	99.76%	85.36%
roadNet-PA	99.67%	99.67%	99.67%	99.60%	90.43%
roadNet-TX	99.83%	100.00%	99.92%	99.28%	85.12%
soc-pokec-relationships	100.00%	100.00%	99.29%	100.00%	67.86%
youtube-u-growth	100.00%	100.00%	100.00%	100.00%	68.39%

Table B.10: efficiency of different lower bound techniques on directed graphs.

Network	3-SS	4-SS	5-SS	6-SS	7-SS	8-SS	2-dSweep	4RandSamp
amazon0302	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	79.43%
amazon0312	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	72.26%
amazon0505	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	64.36%
amazon0601	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	68.70%
as-caida20071105	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	88.82%
ca-AstroPh	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	75.00%
ca-CondMat	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	75.33%
ca-GrQc	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	76.47%
ca-HepPh	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	80.77%
ca-HepTh	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	77.22%
cit-HepPh	99.80%	100.00%	100.00%	100.00%	100.00%	100.00%	98.16%	71.02%
cit-HepTh	85.14%	86.76%	86.76%	86.76%	86.76%	86.76%	82.43%	72.97%
cit-Patents	67.08%	72.92%	72.92%	84.58%	84.58%	84.58%	59.58%	26.67%
email-EuAll	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	70.91%
flickr-growth	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	60.37%
p2p-Gnutella04	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	97.69%	73.08%
p2p-Gnutella05	94.55%	97.27%	97.27%	100.00%	100.00%	100.00%	97.73%	73.64%
p2p-Gnutella06	95.71%	100.00%	100.00%	100.00%	100.00%	100.00%	99.52%	50.95%
p2p-Gnutella08	88.00%	100.00%	100.00%	100.00%	100.00%	100.00%	95.50%	74.00%
p2p-Gnutella09	99.50%	100.00%	100.00%	100.00%	100.00%	100.00%	95.50%	86.50%
p2p-Gnutella24	94.48%	100.00%	100.00%	100.00%	100.00%	100.00%	93.79%	66.55%
p2p-Gnutella25	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	97.27%	32.27%
p2p-Gnutella30	95.83%	95.83%	95.83%	99.58%	99.58%	100.00%	95.42%	31.67%
p2p-Gnutella31	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	98.06%	48.06%
soc-Epinions1	98.13%	100.00%	100.00%	100.00%	100.00%	100.00%	92.50%	65.63%
soc-sign-epinions	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	90.63%	51.25%
Soc-sign-Slash081106	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	57.33%
Soc-sign-Slash090216	99.33%	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	59.33%
Soc-sign-Slash090221	97.33%	100.00%	100.00%	100.00%	100.00%	100.00%	96.67%	56.00%
soc-Slashdot0811	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	75.83%
soc-Slashdot0902	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	74.62%
trec-wt10g	75.21%	75.21%	75.21%	75.21%	75.21%	75.21%	75.21%	28.35%
web-BerkStan	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	83.37%
web-Google	96.08%	100.00%	100.00%	100.00%	100.00%	100.00%	89.80%	60.39%
web-NotreDame	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	46.13%
web-Stanford	42.05%	42.07%	42.07%	42.07%	42.07%	42.07%	42.05%	24.00%
wiki-Talk	90.91%	90.91%	100.00%	100.00%	100.00%	100.00%	99.09%	0.00%
wiki-Vote	98.00%	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	60.00%
youtube-links	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	49.13%
Zhishi-baidu	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	91.79%	49.74%
Zhishi-hudong	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	96.77%	73.87%

Appendix C. Computing the Radius and Diameter

The following tables provide the number of iterations needed to compute diameter and radius, using various algorithms, for different types of graphs.

Appendix C.1. Undirected Graphs

Table C.11: number of iterations needed by various algorithms on undirected graphs.

Network	Diameter				Radius	
	SS	BD	iFub4S	iFubHd	SS	BD
as20000102	10	14	34	29	3	2
CA-AstroPh	15	18	12	12	8	9
CA-CondMat	11	13	14	6	4	3
ca-GrQc	4	24	6	11	4	10
ca-HepPh	11	20	39	10	6	9
ca-HepTh	10	14	12	9	6	6

Network	Diameter				Radius	
	SS	BD	iFub4S	iFubHd	SS	BD
com-amazon.all.cmty	4	16	16	10	4	7
com-amazon.ungraph	5	7	7	52	4	3
com-dblp.ungraph	9	8	34	9	4	3
com-lj.all.cmty	4	3	31	9	4	3
com-youtube.ungraph	3	2	6	2	3	2
email-Enron	6	10	14	19	4	3
facebook_combined	4	9	64	143	4	9
flickrEdges	36	1255	32178	11	3	156
gowalla_edges	6	5	6	2	6	5
loc-brightkite_edges	3	2	8	2	3	2
oregon1_010519	4	25	6	3	3	3
oregon1_010526	4	3	14	2	4	3
oregon2_010519	8	14	13	16	3	2
oregon2_010526	8	10	29	12	3	2
orkut-links	23	144	18722	103	8	13
p2p-Gnutella09	35	178	219	15	7	7
roadNet-CA	180	181	122817	354382	18	29
roadNet-PA	55	60	497	263606	10	11
roadNet-TX	68	84	25996	544280	6	9
soc-pokec-relationships	6	3	74	2	6	3
youtube-u-growth	4	5	6	5	4	5

Appendix C.2. Directed Strongly Connected Graphs

Table C.12: number of iterations needed by various algorithms on directed strongly connected graphs.

Network	Diameter					Radius	
	SS	diFub2In	diFub2Out	diFubHdOut	diFubHdIn	SS	HR
amazon0302	17	482	482	264	146	20	284
amazon0312	36	68	68	2553	3198	29	242
amazon0505	17	41	24	173	194	22	2820
amazon0601	26	104	104	644	78	36	1021
as-caida20071105	15	8	8	10	10	15	2
ca-AstroPh	32	24	24	24	24	26	467
ca-CondMat	19	24	24	12	12	9	21
ca-GrQc	17	8	8	22	22	17	53
ca-HepPh	33	32	32	20	20	17	135
ca-HepTh	26	16	16	18	18	17	107
cit-HepPh	9	9	9	1719	5403	10	70
cit-HepTh	9	7	7	13	361	14	96
email-EuAll	18	10	10	11	11	19	4539
flickr-growth	15	18	18	17	17	12	4488
p2p-Gnutella04	15	38	38	44	117	18	38
p2p-Gnutella05	16	39	39	36	50	18	40
p2p-Gnutella06	21	172	172	113	117	17	167
p2p-Gnutella08	12	70	64	499	51	16	237
p2p-Gnutella09	33	307	307	137	187	32	28
p2p-Gnutella24	9	22	22	13	31	16	20
p2p-Gnutella25	59	152	152	88	1003	18	155
p2p-Gnutella30	38	246	288	1281	306	25	940
p2p-Gnutella31	26	255	255	132	212	22	306
soc-Epinions1	15	6	6	3	5	16	168
soc-pokec-relationships	9	14	14	4	4	9	163
soc-sign-epinions	15	6	6	10	3	21	23
soc-sign-Slashdot081106	11	22	22	9	11	9	232
soc-sign-Slashdot090216	11	21	21	9	11	9	98
soc-sign-Slashdot090221	11	22	22	9	11	9	98
soc-Slashdot0811	48	28	28	10	10	13	17326
soc-Slashdot0902	9	21	21	10	10	11	12727
trec-wt10g	9	21	21	34	39	14	165
web-BerkStan	15	7	7	244	235	21	250

Network	Diameter					Radius	
	SS	diFub2In	diFub2Out	diFubHdOut	diFubHdIn	SS	HR
web-Google	15	8	8	51	12	17	243
web-NotreDame	9	7	7	6	5	11	45
web-Stanford	15	6	6	39	4	17	13941
wiki-Talk	20	13	13	45	6	18	61379
wiki-Vote	9	17	17	9	9	20	878
youtube-links	9	13	13	4	4	7	19247
zhishi-baidu-internallink	9	7	7	6	5	7	4990
zhishi-hudong-internallink	15	510	201	29	35	15	169

Appendix C.3. Directed Weakly Connected Graphs

Table C.13: number of iterations needed by the new algorithm on directed graphs.

Network	Diameter	Radius
amazon0302	21	18
amazon0312	21	30
amazon0505	10	329
amazon0601	53	37
as-caida20071105	15	15
ca-AstroPh	33	26
ca-CondMat	19	9
ca-GrQc	17	17
ca-HepPh	34	17
ca-HepTh	26	17
cit-HepPh	10	128
cit-HepTh	327	72
cit-Patents	1510	7
email-EuAll	33	36
flickr-growth	16	7
p2p-Gnutella04	17	19
p2p-Gnutella05	27	26
p2p-Gnutella06	20	27
p2p-Gnutella08	10	21
p2p-Gnutella09	43	34
p2p-Gnutella24	10	17
p2p-Gnutella25	78	24
p2p-Gnutella30	48	28
p2p-Gnutella31	37	39
soc-Epinions1	13	12
soc-sign-epinions	13	10
Soc-sign-Slash081106	20	15
Soc-sign-Slash090216	16	19
Soc-sign-Slash090221	13	17
soc-Slashdot0811	52	21
soc-Slashdot0902	20	16
trec-wt10g	559	7
web-BerkStan	17	19
web-Google	24	22
web-NotreDame	10	12
web-Stanford	69	7
wiki-Talk	10	10
wiki-Vote	10	16
youtube-links	21	13
Zhishi-baidu	10	7
Zhishi-hudong	17	17