Algorithms for Analyzing and Mining Real-World Graphs
Takes, F.W.

Citation

Version: Corrected Publisher’s Version
License: Licence agreement concerning inclusion of doctoral thesis in the Institutional Repository of the University of Leiden
Downloaded from: https://hdl.handle.net/1887/29764

Note: To cite this publication please use the final published version (if applicable).
The handle http://hdl.handle.net/1887/29764 holds various files of this Leiden University dissertation.

Author: Takes, Frank Willem
Title: Algorithms for analyzing and mining real-world graphs
Issue Date: 2014-11-19
The eccentricity of a node in a graph is defined as the length of a longest shortest path starting at that node. The eccentricity distribution over all nodes is a relevant descriptive property of the graph, and its extreme values allow the derivation of measures such as the radius and diameter of the graph. This chapter describes two new methods for computing the eccentricity distribution of large graphs such as social networks, biological networks, webgraphs and routing networks. We first propose an exact algorithm based on eccentricity lower and upper bounds that is significantly faster than the straightforward algorithm when computing both the extreme values of the distribution as well as the eccentricity distribution as a whole. The second algorithm that we describe is a hybrid strategy that combines the exact approach with an efficient sampling technique in order to obtain an even larger speedup on the computation of the entire eccentricity distribution. We perform a set of experiments on a number of large graphs in order to measure and compare the performance of the proposed algorithms, and demonstrate how we can efficiently compute the eccentricity distribution of various large real-world graphs. This chapter is based on:

3.1 Introduction

There exist all kinds of interesting properties that better describe the relationships between the objects in a dataset modeled by a graph. One of these properties is the eccentricity distribution, which indicates the distribution of the eccentricity over all nodes, where the eccentricity of a node refers to the length of a longest shortest path starting at that node. This distribution differs from properties such as the distance distribution in the sense that eccentricity can be seen as a more “extreme” measure of distance. It also differs from indicators such as the degree distribution in the sense that determining the eccentricity of every node in the graph is computationally expensive: the traditional method computes the eccentricity of each node by running an All Pairs Shortest Path (APSP) algorithm, requiring $O(mn)$ time for a graph with $n$ nodes and $m$ edges. Unfortunately, this approach is too time-consuming if we consider large graphs with possibly millions of nodes.

The aforementioned complexity issues are frequently solved by determining the eccentricity of a random subset of the nodes in the original graph, and then deriving the eccentricity distribution from the obtained values [118]. While such an estimate may seem reasonable when the goal is to determine the overall average eccentricity value, we will show that this technique does not perform well when the actual extreme values of the distribution are of relevance. The nodes with the highest eccentricity values realize the diameter of the graph and form the so-called graph periphery, whereas the nodes with the lowest values realize the radius and form the center of the graph. Finding exactly these nodes can be useful within various application areas.

In routing networks, for example, it is interesting to know exactly which nodes form the periphery of the network and thus have the highest worst-case response time to any other device [96]. Also, when (routing) networks are modeled, for example for research purposes, it is important to measure the exact eccentricity distribution so that the model can be evaluated by comparing it with the distribution of real routing networks [98]. The eccentricity also plays a role in biological networks [67], for example in networks that model some biological system. There, proteins (nodes in the network) that have a low eccentricity value are easily functionally reachable by other components of the network [111]. The diameter, defined as the length of a longest shortest path, is the most frequently studied eccentricity-based measure, and efficient algorithms for its computation have been discussed in Chapter 2 of this thesis.

Generally speaking, eccentricity can be seen as an extreme measure of centrality, i.e., the relative importance of a node in a graph. Eccentricity centrality [15], centroid centrality [23] and graph centrality [26] have been suggested as centrality measures based on the eccentricity of a node. Compared to other measures such as closeness
centrality, the main difference is that a node with a very low eccentricity value is relatively close to every other node, whereas a node with a low closeness centrality value is close to all the other nodes on average.

In this chapter we first discuss an algorithm based on eccentricity lower and upper bounds for determining the exact eccentricity of every node of a graph. We also present a useful pruning strategy, and show how the proposed method significantly improves upon the traditional APSP-based algorithm. To realize an even larger speed-up, we propose to incorporate a sampling technique on a specific set of nodes in the graph which allows us to obtain the eccentricity distribution much faster, while still ensuring a low error on the full eccentricity distribution and an exact result for the eccentricity-based graph properties such as the radius and diameter.

The rest of this chapter is organized as follows. We consider some notation and formulate the main problems addressed in this chapter in Section 3.2, after which we cover related work in Section 3.3. Section 3.4 describes an exact algorithm for determining the eccentricity distribution. In Section 3.5, we explain how sampling can be incorporated. Results of applying the methods to various large graphs are presented in Section 3.6, and finally Section 3.7 summarizes the chapter and offers suggestions for future work.

### 3.2 Preliminaries

We consider a graph $G = (V, E)$, where $V$ is the set of $|V| = n$ vertices (nodes) and $E \subseteq V \times V$ is the set of $|E| = m$ edges (also called links). The distance $d(v, w)$ between two nodes $v, w \in V$ is defined as the length of a shortest path from $v$ to $w$, i.e., the minimum number of edges that have to be traversed to get from $v$ to $w$. We assume that graphs are undirected, meaning that $(v, w) \in E$ iff $(w, v) \in E$ and thus $d(v, w) = d(w, v)$ for all $v, w \in V$. Note that each edge $(v, w)$ is thus included twice in $E$: once as a link from $v$ to $w$ and once as a link from $w$ to $v$. We will also assume that $G$ is connected, meaning that $d(v, w)$ is always finite. Furthermore it is assumed that there are no parallel edges and no loops linking a node to itself. The neighborhood $N(v)$ of a node $v$ is defined as the set of all nodes connected to $v$ via an edge: $N(v) = \{w \in V \mid (v, w) \in E\}$. The degree $\text{deg}(v)$ of a node $v$ can then be defined as the number of nodes connected to that node, i.e., its neighborhood size: $\text{deg}(v) = |N(v)|$.

The eccentricity $e(v)$ of a node $v \in V$ is defined as the length of a longest shortest path from $v$ to any other node: $e(v) = \max_{w \in V} d(v, w)$. The eccentricity distribution counts the frequency $f(x)$ of each eccentricity value $x$, and can easily be derived when the eccentricity of each node in the graph is known. The relative eccentricity
distribution lists for each eccentricity value $x$ its relative frequency $F(x) = f(x)/n$, normalizing for the number of nodes in the graph. Figure 3.1 shows the relative eccentricity distributions of a number of large graphs (for a detailed description of these datasets, see Section 3.6.1).

Various other measures can be derived using the notion of eccentricity, such as the average eccentricity $\overline{e}(G)$ of a graph $G$, defined as $\overline{e}(G) = \frac{1}{n} \sum_{v \in V} e(v)$. Another related measure is the diameter $D(G)$ of a graph, which is defined as the maximum eccentricity over all nodes in the graph: $D(G) = \max_{v \in V} e(v)$. Similarly, we define the radius $R(G)$ of a graph as the minimum eccentricity over all nodes in the graph: $R(G) = \min_{v \in V} e(v)$. The graph center $C(G)$ refers to the set of nodes with an eccentricity equal to the radius, $C(G) = \{v \in V \mid e(v) = R(G)\}$. Similarly, the graph periphery $P(G)$ is defined as the set of nodes with an eccentricity value equal to the diameter of the graph: $P(G) = \{v \in V \mid e(v) = D(G)\}$. For ease of notation, we will often denote these measures as variables $\overline{e}$, $D$, $R$, $C$ and $P$. Each of the metrics explained above can be derived when the eccentricity of all nodes is known. Figure 3.3 shows an example of a graph that explains the different measures covered in this section. In Figure 3.2, a larger graph in which the node color corresponds to the eccentricity value is shown.

Computing the eccentricity of one node can be done by running Dijkstra’s algorithm, and returning the largest distance found (i.e., the distance to a node fur-

![Figure 3.1: Relative eccentricity distributions of various large graphs.](image-url)
Figure 3.2: The YEAST graph, consisting of 1458 nodes and 1948 undirected edges. The color of a node represents its eccentricity value in the range from 11 to 19 from low (lighter) to high (darker) and the size of a node is proportional to its degree. Visualized using the ForceAtlas2 algorithm in Gephi (http://gephi.org).
3.2. Preliminaries

Figure 3.3: Toy graph consisting of 12 nodes and 14 edges. The number next to a node denotes its eccentricity value. The graph has average eccentricity 4.67, radius 3 realized by center node $G$, and diameter 6 realized by periphery nodes $D$, $E$ and $L$.

The eccentricity of one node $v$ is denoted by $\text{ECCENTRICITY}(v)$ in Algorithm 3.1, representing one Breadth First Search (BFS) starting at node $v$.

Algorithm 3.1 simply computes the eccentricity for each of the $n$ nodes, resulting in an overall complexity of $O(mn)$ to determine the eccentricity of every node in the graph. Clearly, in graphs with millions of nodes and possibly hundreds of millions of edges, this approach is too time-consuming. The rest of this chapter describes more efficient approaches for determining the eccentricity distribution, where we are interested in two things:

- The (relative) eccentricity distribution as a whole.
- Finding the extreme values of the eccentricity distribution, i.e., the radius and diameter, as well as derived measures such as the center and periphery.

We will address these issues by answering the following two questions:

Algorithm 3.1 NaiveEccentricities

1: Input: Graph $G$
2: Output: List $e$, containing $e(v)$ for all $v \in V$
3: for $v \in V$ do
4: $e[v] \leftarrow \text{ECCENTRICITY}(v)$
5: end for
6: return $e$
• How can we obtain the exact eccentricity distribution by efficiently computing the exact value of \( e(v) \) for all nodes \( v \in V \)? (Section 3.4)

• How can we obtain an accurate approximation of the eccentricity distribution by using a sampling technique? (Section 3.5)

3.3 Related work

Work on the eccentricity distribution dates back to at least 1975, when the term “eccentric sequence” was used to denote the sequence that counts the frequency of each eccentricity value [90]. The facility location problem was suggested as an example of the usefulness of eccentricity as a measure of centrality. When considering the placement of emergency facilities such as a hospital or fire station, assuming the map is modeled as a graph, the node with the lowest eccentricity value might be a good location for such a facility. In other situations, for example for the placement of a shopping center, a related measure called closeness centrality, defined as the average distance from a particular node \( v \) to every other node \( \left( \frac{1}{n-1} \sum_{w \in V} d(v, w) \right) \), is more suitable. Generally speaking, the eccentricity is a relevant measure when some strict criterion (the firetruck has to be able to reach every location within ten minutes) has to be met [56]. An application of eccentricity as a measure on a larger scale is the network routing graph, where the eccentricity of a node says something about the worst-case response time between one machine and all other machines [98].

The most well-known eccentricity-based measure is the diameter, which has been extensively investigated in Chapter 2 and in [37, 95]. Several measures related to the eccentricity and diameter have also been considered. Kang et al. [68] study the effective radius, which they define as the 90th-percentile of all the shortest distances from a node. In a similar way, the effective diameter can be defined, which is shown to be decreasing over time for many large real-world graphs [86]. Each of these measures is computed by using an approximation algorithm [109] to determine the neighborhood of a node, a technique on which we will elaborate in Section 3.5.3. An overview of algorithms for approximating the radius and diameter is given in [115].

To the best of our knowledge, there are no efficient techniques that have been specifically designed to determine the exact eccentricity distribution of a graph. Obviously, the naive approach, for example as suggested in [29], is too time-consuming. Efficient approaches for solving the APSP problem (which makes deriving the eccentricities trivial) have been developed, for example using matrix multiplication [147]. Unfortunately, such approaches are still too complex in terms of time and memory requirements. The remainder of this chapter describes both a new exact algorithm and a new approximation algorithm to efficiently compute the eccentricity distribution.
3.4 Exact algorithm

In order to obtain an algorithm that can compute the eccentricity of all \( n \) nodes in a graph faster than simply recomputing the eccentricity \( n \) times (once for each node in the graph), we have two options:

1. Reduce the size of the graph to speed up one eccentricity computation.
2. Reduce the total number of eccentricity computations.

In this section we propose to use lower and upper bounds on the eccentricity, a strategy that accommodates the second type of speedup. We will also discuss a pruning strategy that helps to reduce both the number of nodes that have to be investigated, as well as the size of the graph.

3.4.1 Eccentricity bounds

We propose to use the following bounds on the eccentricity of all nodes \( w \in V \):

**Observation 3.1 Node eccentricity bounds**

If a node \( v \in V \) has eccentricity \( e(v) \), then for all nodes \( w \in V \) we have:

\[
\max(e(v) - d(v, w), d(v, w)) \leq e(w) \leq e(v) + d(v, w)
\]

For an explanation of these bounds, we refer the reader to Section 2.2, in which the same observation was used to determine which nodes can contribute to the process of computing the diameter of a graph. We employ an algorithm similar to what is proposed in Chapter 2, but this time using the eccentricity bounds to compute the full eccentricity distribution of the graph. The approach is outlined in Algorithm 3.2.

First, the candidate set \( W \) and the lower and upper eccentricity bounds are initialized (lines 3–7). In the main loop of the algorithm, a node \( v \) is repeatedly selected (line 9) from \( W \), its eccentricity is determined (line 10), and finally all candidate nodes are updated (lines 11–18) according to Observation 3.1. Note that the value of \( d(v, w) \) which is used in the updating process does not have to be computed, as it is already known because it was computed for all \( w \) during the computation of the eccentricity of \( v \). If the lower and upper eccentricity bounds for a node have become equal, then the eccentricity of that node has been derived and it is removed from \( W \) (lines 14–17). Algorithm 3.2 returns a list containing the exact eccentricity value of each node. Counting the number of occurrences of each eccentricity value results in the eccentricity distribution.

An overview of possible selection strategies for the function \texttt{SELECTFROM} can be found in Section 2.4.4. In line with results presented in Chapter 2, we found that
Algorithm 3.2 BoundingEccentricities

1: **Input:** Graph $G$
2: **Output:** List $e$, containing $e(v)$ for all $v \in V$
3: $W \leftarrow V$
4: for $w \in W$ do
5: $e_l[w] \leftarrow -\infty$
6: $e_u[w] \leftarrow +\infty$
7: end for
8: while $W \neq \emptyset$ do
9: $v \leftarrow \text{SELECTFROM}(W)$
10: $e[v] \leftarrow \text{ECCENTRICITY}(v)$
11: for $w \in W$ do
12: $e_l[w] \leftarrow \max(e_l[w], \max(e[v] - d(v, w), d(v, w)))$
13: $e_u[w] \leftarrow \min(e_u[w], e[v] + d(v, w))$
14: if $(e_l[w] = e_u[w])$ then
15: $e[w] \leftarrow e_l[w]$
16: $W \leftarrow W - \{w\}$
17: end if
18: end for
19: end while
20: return $e$

when determining the eccentricity distribution, interchanging the selection of a node with a small lower bound and a node with a large upper bound, breaking ties by taking a node with the highest degree, yielded by far the best results. As described in Section 2.4.3, examples of graphs in which this algorithm would definitely not work are complete graphs and circle-shaped graphs. However, most real-world graphs adhere to the small-world property [71], and in these graphs the eccentricity values are sufficiently diverse so that the proposed eccentricity lower and upper bounds can effectively be utilized.

3.4.2 Example run

For an example run of the proposed algorithm, consider the problem of determining the eccentricities of the nodes of the toy graph from Figure 3.3. We will denote the lower and upper eccentricity bounds $e_l(v)$ and $e_u(v)$ of a node $v$ by $[e_l(v); e_u(v)]$. If we compute the eccentricity of node $G$, which is 3, then we can derive bounds $[2; 4]$
3.4. Exact algorithm

Figure 3.4: Eccentricity bounds (lower and upper bound respectively below and above the node) of the toy graph in Figure 3.3 after computing the eccentricity of node $G$.

Figure 3.5: Eccentricity bounds of the toy graph in Figure 3.3 after subsequently computing the eccentricity of node $G$ and $E$.

Figure 3.6: Eccentricity bounds of the toy graph in Figure 3.3 after subsequently computing the eccentricity of node $G$, $E$ and $A$. 
for the nodes at distance 1 \((B, C, F, J\) and \(K)\), \([2; 5]\) for the nodes at distance 2 \((A, H\) and \(I)\) and \([3; 6]\) for the nodes at distance 3 \((D, E\) and \(L)\), as depicted in Figure 3.4. If we then compute the eccentricity of node \(E\), which is 6, we derive bounds \([5; 7]\) for node \(I\), \([4; 8]\) for node \(F\), \([3; 9]\) for nodes \(A, B\) and \(G\), \([4; 10]\) for nodes \(C, J\) and \(K\), \([5; 11]\) for node \(H\), and \([6; 12]\) for nodes \(D\) and \(L\). If we combine these bounds for each of the nodes cf. lines 12-13 of Algorithm 3.2, then we find that lower and upper bounds for a large number of nodes have become equal: \([4; 4]\) for \(C, F, J\) and \(K\), \([5; 5]\) for \(H\) and \(I\), and \([6; 6]\) for \(D\) and \(L\), as shown in Figure 3.5. Finally, computing the eccentricity of nodes \(A\) and \(B\) results in a total of 4 BFSes to compute the complete eccentricity distribution (Figure 3.6), which is a speedup of 3 compared to the naive algorithm, which would simply compute the eccentricity for all 12 nodes in the graph.

To give a first idea of the performance of the algorithm on larger real-world graphs, Figure 3.7 shows the number of iterations (vertical axis) that are needed to compute the eccentricity of all nodes with given eccentricity value (horizontal axis) for a number of large graphs (for a description of the datasets, see Section 3.6.1). We can clearly see that especially for the extreme values of the eccentricity distribution, very few iterations are needed to compute all of these eccentricity values, whereas many more iterations (though still much less than \(n\)) are needed to derive the values in between the extreme values.
3.4.3 Pruning

In this subsection we introduce a pruning strategy that is somewhat based on the pruning step introduced in Section 2.4.6. The pruning strategy is based on the following observation:

**Observation 3.2** Assume that $n > 2$. For a given $v \in V$, all nodes $w \in N(v)$ with $\text{deg}(w) = 1$ have $e(w) = e(v) + 1$.

Node $w$ is only connected to node $v$, and will thus need node $v$ to reach every other node in the graph. If node $v$ can do this in $e(v)$ steps, then node $w$ can do this in exactly $e(v) + 1$ steps. The restriction $n > 2$ on the graph size excludes the case in which the graph consists of $v$ and $w$ only.

All interesting real-world graphs have a lot more than two nodes, making Observation 3.2 applicable in the proposed algorithm. Observation 3.2 can be beneficial in two ways. First, when computing the eccentricity of a single node, the pruned nodes can be ignored in the shortest path algorithm. Second, when the eccentricity of a node $v$ has been computed or derived (line 10 or lines 14–17) of Algorithm 3.2), and this node has adjacent nodes $w$ with $\text{deg}(w) = 1$, then the eccentricity of these nodes $w$ can be set to $e(w) = e(v) + 1$.

In Figure 3.3, node $G$ has two neighbors with degree one, namely $J$ and $K$. According to Observation 3.2, the eccentricity values of these two nodes are equal to $e(J) = e(K) = e(G) + 1 = 4$. The same argument holds for nodes $D$ and $L$ with respect to node $H$. We expect that Observation 3.2 can be applied quite often, as many of the graphs that are nowadays studied have a power law degree distribution [46], meaning that there are many nodes with a very low degree (such as a degree of 1). Furthermore, there is no significant additional computation time involved in identifying prunable nodes.

The pruning strategy described in this section is conceptually similar to the pruning strategy used in Section 2.4.6 of Chapter 2. The major difference is that there, pruned nodes are not useful for computing the diameter and can be completely removed from the graph, whereas for computing the eccentricity distribution, the actual nodes that are pruned are relevant when the final eccentricity distribution is derived from the list of eccentricity values, and should thus not be removed from the node set.

3.5 Approximation algorithms

The use of sampling to determine the eccentricity distribution will be discussed in Section 3.5.1. Sampling is a technique in which a subset of the original dataset is
evaluated in order to estimate (the distribution of) characteristics of the (elements in the) complete dataset, with faster computation as one of the main advantages. We also take into account situations where we not only want to estimate the distribution of the eccentricity values, but also want to assess some parts of it (the extreme values) with high reliability. For that purpose we propose a hybrid technique to determine the eccentricity distribution that combines the exact approach from Section 3.4 with non-random sampling in Section 3.5.2. Finally in Section 3.5.3 we consider an adaptation of an existing approximation approach from literature.

### 3.5.1 Random node selection

If we want to apply sampling to the naive algorithm for obtaining the eccentricity distribution, we could choose to evaluate only a subset of size $n'$ of the original $n$ nodes (clearly, here $0 < n' < n$), and multiply the values of the sampled eccentricity distribution by a factor $n/n'$ to get an idea of the real eccentricity distribution, a process referred to as random node selection.

Indeed, taking only a subset of the nodes would clearly speed up the computation of the eccentricity distribution and realize the second type of speedup discussed in Section 3.4: reducing the total number of eccentricity computations. The trade-off here is that we no longer obtain the exact eccentricity distribution, but only get an approximation. The main question is then whether or not this approximation is representative of the original distribution. The effectiveness of sampling by random node selection with respect to various graph properties has been demonstrated in [84]. Using similar arguments as presented in [38, 45], the absolute error can be assessed; indeed, when the chosen sampling subset is sufficiently large, the error is effectively bounded.

However, there are situations where we are not interested in minimizing the absolute error, but in minimizing the relative standard deviation (i.e., the absolute standard deviation divided by the mean) of the distribution of a particular eccentricity value. This especially makes sense when each eccentricity value is of equal importance, which might be the case when the extreme values of the distribution that are realizing the radius and diameter are of relevance. Let us consider an example. The real exact eccentricity distribution over all nodes in the ENRON graph [73] is listed in Table 3.1.

<table>
<thead>
<tr>
<th>$e(v)$</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
</tr>
</thead>
<tbody>
<tr>
<td>$f(e(v))$</td>
<td>248</td>
<td>12,210</td>
<td>17,051</td>
<td>3,647</td>
<td>485</td>
<td>44</td>
<td>11</td>
</tr>
</tbody>
</table>

Table 3.1: Eccentricity distribution of the ENRON graph.
Figure 3.8 shows the relative standard deviation for each eccentricity value for different sample sizes (1%, 2%, 5%, 10%, and 20%). For each sample size, we took 100 random samples to average the result. We note that low relative standard deviations can be observed for the more common eccentricity values (in this case, 8, 9 and 10). However, the standard deviation is quite large for the extreme values, meaning that on many occasions, the sample did not correctly reflect the frequency of that particular eccentricity value. Indeed, if only 11 out of 33,696 nodes (0.03%) have a particular eccentricity value (in this case, a value of 13), we need a sample size of at least $100\%/11 = 9\%$ if we want to expect just one node with that particular eccentricity value.

Figure 3.8 shows that even with a large sample size of 20%, the standard deviation for the extreme value of 13 is very high (0.63). For sample sizes of 1% or 2%, the eccentricity value of 13, that exists in the real distribution, was never even found. Similar events were observed for the other datasets, which is not surprising: the eccentricity distributions are tailed on the extremes, and these tails are likely to be left out in a sample. So clearly sampling does not suffice when extreme values of the eccentricity distribution have to be found, because such extreme values are simply too rare. However, for the more common eccentricity values, errors are very low, even for small sample sizes.

Figure 3.8: Relative standard deviation (vertical axis) of eccentricity values (horizontal axis) for different random sample sizes of the ENRON dataset.
Chapter 3. Computing the Eccentricity Distribution of Large Graphs

3.5.2 Hybrid algorithm

From Figure 3.7 and Figure 3.8 we can conclude that the exact approach is able to quickly derive the more extreme values of the eccentricity distribution, whereas a sampling technique is able to approximate the values in between the extremes with a very low error. Therefore we propose to combine the two approaches in order to quickly converge to an accurate estimation of the eccentricity distribution using a hybrid approach. We mention that the proposed technique assumes that the eccentricity distribution has a somewhat unimodal shape, which is the case for all real-world graphs that we have investigated.

The sampling window consists of bounding variables \( \ell \) and \( r \) that denote between which eccentricity values the algorithm is going to use the sampling technique. The sampling window obviously depends on the distribution itself, which is not known until the exact algorithm has finished. Therefore we set the value of \( \ell \) and \( r \) dynamically as outlined on lines 19–20 of Algorithm 3.3, which we will call BOUNDINGECCENTRICITIESLR. The main difference with respect to Algorithm 3.2 is a change in the stopping criterion in line 8. This means that the algorithm should now stop when there are no more candidates that are potentially part of the center or the periphery of the graph. Note that these bounds apply to the candidate set \( W \), but (via the values of \( \ell \) and \( r \)) we use information about all nodes \( V \), ensuring that at least the center and periphery are known before the exact phase is terminated. When the exact algorithm has done its job, it can tighten \( \ell \) and \( r \) even further, based on the eccentricity of the remaining candidate nodes, as outlined on lines 22–23. This can be beneficial when the center is much harder to find than the periphery (or the other way around), in which case the distribution of more eccentricity values may already have been computed exactly, resulting in the advantage that for these eccentricity values no sampling has to be done. After this, \( \ell \) and \( r \) become static, and the rest of the eccentricity distribution will be derived by using the sampling approach outlined in Algorithm 3.4.

Compared to Algorithm 3.2, the difference in terms of input is that the hybrid algorithm given in Algorithm 3.4 takes as input both the original graph and a sampling rate \( q \) between 0 and 1, where \( q = 0.10 \) means that 10% of the nodes have to be sampled. In Section 3.6 we will perform experiments to determine how \( q \) should be set. The resulting list \( f \) holds the eccentricity distribution and is initialized based on the exact eccentricity values \( e' \) for values outside the sampling window (line 7). Nodes for which the exact eccentricity is not yet known are added to set \( Z \) (line 9). In lines 12–17, the eccentricity of a total of \( n \cdot q \) random nodes are computed, and if this value lies within the specified window, the frequency of this eccentricity value is increased proportionally to the sample size. Finally, the eccentricity distribution \( f \) is returned.

To get a first idea of how the hybrid algorithm works, we look at Figure 3.9 of
Algorithm 3.3 \textsc{BoundingEccentricitiesLR} (adjustment of Algorithm 3.2)

1: \textbf{Input:} Graph $G$ and a reference to variables $\ell$ and $r$
2: \textbf{Output:} List $e$, containing $e(v)$ for values of $e(v)$ outside $[\ell; r]$
3: $W \leftarrow V$ \quad $\ell \leftarrow 0$ \quad $r \leftarrow \infty$
4: \ldots \quad // \text{original \textsc{BoundingEccentricities} algorithm}
8: while $\{w \in W \mid e_\ell[w] = \ell \lor e_u[w] = r\} \neq \emptyset$ do
9: \ldots \quad // \text{original \textsc{BoundingEccentricities} algorithm}
19: $\ell = \min_{v \in V} e_\ell[v]$
20: $r = \max_{v \in V} e_u[v]$
21: \textbf{end while}
22: $\ell = \min_{v \in W} e_\ell[v] - 1$
23: $r = \max_{v \in W} e_u[v] + 1$.
24: \textbf{return} $e$

Algorithm 3.4 \textsc{HybridEccentricities}

1: \textbf{Input:} Graph $G$ and sampling rate $q$
2: \textbf{Output:} List $f$, containing the eccentricity distribution of $G$, initialized to 0
3: $e' \leftarrow \textsc{BoundingEccentricitiesLR}(G, \ell, r)$
4: $Z \leftarrow \emptyset$
5: \textbf{for} $v \in V$ \textbf{do}
6: \quad if $e'[v] \neq 0$ \textbf{and} ($e'[v] \leq \ell$ \textbf{or} $e'[v] \geq r$) then
7: \quad \quad $f[e'[v]] \leftarrow f[e'[v]] + 1$
8: \quad \quad else
9: \quad \quad $Z \leftarrow Z \cup \{v\}$
10: \quad \quad \textbf{end if}
11: \textbf{end for}
12: \textbf{for} $i \leftarrow 1$ \textbf{to} $n \cdot q$ \textbf{do}
13: \quad $v \leftarrow \textsc{RandomFrom}(Z)$
14: \quad $e[v] \leftarrow \textsc{Eccentricity}(v)$
15: \quad $f[e[v]] \leftarrow f[e[v]] + (1/q)$
16: \quad $Z \leftarrow Z - \{v\}$
17: \textbf{end for}
18: \textbf{return} $f$
the CA-HEP\(\text{PH}\) graph with radius \(R = 7\) and diameter \(D = 13\). The figure shows for each eccentricity value on the left axis the number of exact iterations to compute all eccentricities of that value (just like Figure 3.7) and the relative standard deviation when sampling (just like Figure 3.8) on the right axis. Clearly, if we exactly compute the eccentricity values 7, 12 and 13, and use sampling within the window \([8; 11]\), then the region below the horizontal line in Figure 3.9 indicates how we obtain the eccentricity distribution of the CA-HEP\(\text{PH}\) graph with very low errors, while not using many exact BFSes for the extreme values. Note that we could also have chosen \([8; 12]\) as a sampling window, as the sampling error for an eccentricity value of 12 is still rather low. We will further analyze the performance of the hybrid approach using a larger set of graphs in Section 3.6.

### 3.5.3 Neighborhood approximation

Algorithms for efficiently computing the neighborhood function have been introduced in [18, 68, 109]. These algorithms can be adjusted as follows to also approximate the eccentricities. For an integer \(h > 0\), the normalized size of the neighborhood \(N_h(u)\) of a node \(u\) can be defined as:

\[
N_h(u) = \frac{1}{n-1} | \{ v \in V \mid 0 < d(u,v) \leq h \} |
\]

![Figure 3.9: Number of exact iterations and sampling relative standard deviation for different eccentricity values of the CA-HEP\(\text{PH}\) dataset.](image)
If we determine the value of $N_h(u)$ for increasing values of $h$, then the eccentricity $e(u)$ is the smallest $h$ such that the approximated value of $N_h(u)$ is sufficiently close to 1 or does not change in successive iterations.

We have used the Approximate Neighborhood Function (ANF) [109] algorithm by Palmer et al. in order to approximate $N_h(u)$ (using the C source code available at the author’s website). Figure 3.10 shows the actual relative eccentricity distribution of the ENRON dataset, as well as the distribution that was approximated using the ANF-based approach. As ANF gives an approximate result, we averaged the result of 100 runs. We note that although the distribution shapes are very similar, ANF clearly underestimates the eccentricity values. We furthermore mention that the obtained distribution is clearly an approximation: a real eccentricity distribution would never have eccentricity values for which the smallest and largest value differ more than a factor of 2. Similar results were observed for other datasets, which leads us to believe that these approximation algorithms are not suitable for determining the eccentricity distribution, especially because they fail at assessing the width and the extreme values of the distribution.

![Figure 3.10: Exact relative eccentricity distribution of the ENRON dataset and an approximation using ANF.](image-url)
3.6 Experiments

In this section we will use a number of large real-world datasets to assess the performance of both the exact algorithm from Section 3.4 and the hybrid algorithm from Section 3.5. The performance of the algorithms is measured by comparing the number of iterations (actual eccentricity computations by means of a BFS) and is therefore independent of the hardware and software used to implement and run the algorithm.

3.6.1 Datasets

We use a variety of graph datasets, of which the name and source are listed in the first column of Table 3.2. The set of graphs covers a broad range of real-world graphs:

- Citation networks (CIT-HEPTh, CIT-HEPPh)
- Scientific collaboration networks (CA-HEPTh, CA-HEPPh, CA-CONDMat and DBLP20080824)
- Communication networks (ENRON, SLASHDOT)
- Peer-to-peer networks (P2P-GNUTELLA)
- Protein-protein interaction networks (YEAST, DIP20090126),
- Router topology networks (ITDK0304-RLINKS, AS-SKITTER)
- Social networks (FACEBOOK, EPINIONS, SOC-SLASHDOT, FLICKR)
- Webgraphs (WEB-STANFORD, WEB-NOTREDAME, EU-2005)

We will only consider the largest connected component of each graph, which is always the vast majority of the original graph. We also mention that some directed graphs have been interpreted as if they were undirected, and that self-edges and parallel edges are ignored. These factors may cause minor differences between the number of nodes and edges that we present here, and the numbers presented in the source papers. In Table 3.2 we also present for each dataset the exact average eccentricity $\bar{e}$, radius $R$, diameter $D$ and center and periphery sizes $|C|$ and $|P|$. For a more detailed description of these graphs, we refer the reader to the source papers in which the datasets were introduced.

Looking at the exact eccentricity distributions that we were able to compute for each of the graphs, we can conclude that the distribution is not perfectly Gaussian as [97] suggests, but does appear to be unimodal. The distribution appears to be somewhat “tailed”: a positive skew is noticeable in Figure 3.1, which shows the relative eccentricity distribution of a number of graphs. The tail also becomes clear when
comparing the average eccentricity value $\bar{e}$ with the radius and diameter of the graph: for every dataset, the average eccentricity is closer to the radius. For example, for the Cit-HepTh dataset with radius 8 and diameter 15, the average eccentricity is 10.14.

### 3.6.2 Exact algorithm

In order to determine the performance of the exact algorithm (Algorithm 3.2), we can count the number of shortest path computations that are needed to obtain the full distribution, and compare this value to $n$, the number of iterations performed by the naive algorithm (Algorithm 3.1). The number of iterations performed by the exact algorithm is displayed in the third column of Table 3.3, followed by the speedup factor compared to the naive algorithm. We note that significant speedups can be observed in terms of the number of iterations, especially for datasets for which the eccentricity distribution is wide, i.e., the difference between the radius and diameter is very large.

| Dataset            | Nodes $n$ | Links $m$ | $\bar{e}$ | $R$ | $D$ | $|C|$ | $|P|$ |
|--------------------|-----------|-----------|-----------|-----|-----|-------|------|
| YEAST [64]         | 1,458     | 3,896     | 13.28     | 11  | 19  | 48    | 4    |
| CA-HepTh [87]      | 8,638     | 49,612    | 12.53     | 10  | 18  | 74    | 4    |
| CA-HepPh [87]      | 11,204    | 235,238   | 9.40      | 7   | 13  | 12    | 17   |
| Dip20090126 [121]  | 19,928    | 82,404    | 22.01     | 15  | 30  | 1     | 2    |
| CA-CondMat [86]    | 21,363    | 182,572   | 10.58     | 8   | 15  | 6     | 11   |
| Cit-HepTh [86]     | 27,400    | 704,080   | 10.14     | 8   | 15  | 4     | 4    |
| Enron [73]         | 33,696    | 361,622   | 8.77      | 7   | 13  | 248   | 11   |
| Cit-HepPh [86]     | 34,401    | 841,612   | 9.18      | 7   | 14  | 1     | 2    |
| Slashdot [52]      | 51,083    | 243,780   | 11.66     | 9   | 17  | 7     | 3    |
| P2P-Gnutella [87]  | 62,561    | 295,754   | 8.94      | 7   | 11  | 55    | 118  |
| Facebook [103]     | 63,392    | 1,633,772 | 9.96      | 8   | 15  | 168   | 7    |
| Epinions [114]     | 75,877    | 811,476   | 9.74      | 8   | 15  | 614   | 6    |
| Soc-Slashdot [88]  | 82,168    | 1,086,761 | 8.91      | 7   | 13  | 484   | 3    |
| ITDK0304 [121]     | 190,914   | 1,215,220 | 17.09     | 14  | 26  | 155   | 7    |
| Web-Stanford [88]  | 255,265   | 3,883,852 | 106.49    | 82  | 164 | 1     | 3    |
| Web-Notredame [12] | 325,729   | 2,180,216 | 27.76     | 23  | 46  | 12    | 172  |
| DBLP20080824 [121] | 511,163   | 3,742,140 | 14.79     | 12  | 22  | 72    | 9    |
| EU-2005 [121]      | 862,664   | 32,276,936| 14.03     | 11  | 21  | 3     | 4    |
| Flickr [103]       | 1,624,992 | 30,953,670| 15.03     | 12  | 24  | 17    | 3    |
| AS-Skitter [86]    | 1,694,616 | 22,188,418| 21.22     | 16  | 31  | 5     | 2    |

Table 3.2: Number of nodes, links, average eccentricity, radius, diameter, center size and periphery size of the various graph datasets.
We believe that this is due to the fact that the nodes that form the periphery of the network are sufficiently eccentric to have a large influence on the more central nodes, so that their lower and upper eccentricity bounds can very quickly be fixed by the bounding technique. We mention that the performance of the exact algorithm does not appear to be directly related to the number of nodes.

The reduction in terms of the number of nodes as a result of the proposed pruning strategy (see Section 3.4.3) is displayed in the column labeled “Pruned”, and is diverse, yet sometimes very significant (anywhere between 1% and 34%). To get a better idea of the performance of the algorithm, we propose to look at the quality of both the lower and upper bounds as the exact algorithm iterates over the candidate nodes. For this we define the measure of bound accuracy as the percentage of bound values that are correct at that iteration:

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Exact algorithm</th>
<th>Hybrid algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Pruned</td>
<td>Iter.</td>
</tr>
<tr>
<td>YEAST</td>
<td>399</td>
<td>213</td>
</tr>
<tr>
<td>CA-HepTh</td>
<td>351</td>
<td>1,055</td>
</tr>
<tr>
<td>CA-HepPh</td>
<td>282</td>
<td>1,588</td>
</tr>
<tr>
<td>DIP20090126</td>
<td>3,032</td>
<td>224</td>
</tr>
<tr>
<td>CA-CondMat</td>
<td>353</td>
<td>3,339</td>
</tr>
<tr>
<td>Cit-HepTh</td>
<td>140</td>
<td>8,104</td>
</tr>
<tr>
<td>ENRON</td>
<td>8,715</td>
<td>678</td>
</tr>
<tr>
<td>Cit-HepPh</td>
<td>150</td>
<td>10,498</td>
</tr>
<tr>
<td>SLASHDOT</td>
<td>19,255</td>
<td>31</td>
</tr>
<tr>
<td>P2P-Gnutella</td>
<td>16,413</td>
<td>21,109</td>
</tr>
<tr>
<td>Facebook</td>
<td>1,075</td>
<td>11,185</td>
</tr>
<tr>
<td>Epinions</td>
<td>20,779</td>
<td>4,302</td>
</tr>
<tr>
<td>Soc-Slashdot</td>
<td>14,848</td>
<td>1,460</td>
</tr>
<tr>
<td>ITDK0304</td>
<td>16,434</td>
<td>10,830</td>
</tr>
<tr>
<td>Web-Stanford</td>
<td>10,350</td>
<td>9</td>
</tr>
<tr>
<td>Web-Notredame</td>
<td>141,178</td>
<td>143</td>
</tr>
<tr>
<td>DBLP20080824</td>
<td>22,579</td>
<td>42,273</td>
</tr>
<tr>
<td>EU-2005</td>
<td>26,507</td>
<td>59,751</td>
</tr>
<tr>
<td>Flicker</td>
<td>553,242</td>
<td>4,810</td>
</tr>
<tr>
<td>AS-Skitter</td>
<td>114,803</td>
<td>42,996</td>
</tr>
</tbody>
</table>

Table 3.3: Performance of the exact algorithm (Algorithm 3.2) and the hybrid algorithm (Algorithm 3.4).
3.6. Experiments

\[
\text{bound accuracy} = \frac{1}{n} \left| \{v \in V \mid e_{\text{real}}(v) = e_{\text{bound}}(v) \} \right|
\]

Here, \(e_{\text{real}}(v)\) is the actual eccentricity value, and \(e_{\text{bound}}(v)\) is the (lower or upper) bound that we investigate. Figure 3.11 shows the lower and upper bound accuracy for a number of datasets. We can observe that for each of the datasets, after just a few iterations, the lower bound gives a very accurate indication of the actual eccentricity values. Apparently, the majority of the iterations are spent lowering the upper bound, whereas the lower bound quickly reflects the actual eccentricity distribution. Thus, in order to get an online estimate of the distribution, we could choose to consider only the lower bound, and obtain an accuracy of around 90% after a handful of iterations.

3.6.3 Hybrid algorithm

In the second set of experiments, we have evaluated the performance of the hybrid approach that incorporates sampling for the non-extreme values of the distribution (Algorithm 3.4). We will verify the quality of the obtained distribution by using the measure of distribution error, defined as the sum of the absolute difference between the eccentricity counts of real relative eccentricity distribution \(F(x)\) and the estimated distribution \(\hat{F}(x)\):

![Figure 3.11: Bound accuracy vs. number of iterations for Algorithm 3.2.](image-url)
distribution error = \sum_{x=\mathbb{R}}^{D} |F(x) - \hat{F}(x)|

Here a distribution error value of 0 indicates a perfect match between the actual and estimated relative eccentricity distributions. So for the hybrid approach, the error outside the sampling window is always equal to 0. Note that as a result of the exact approach, we have access to the real eccentricity distribution to make this error comparison. Therefore we can investigate the number of iterations that are needed to obtain an error of 0.05 (or lower) in order to determine how we should set the sampling rate \( q \).

The fifth column of Table 3.3 shows the number of iterations needed to (exactly) compute the extreme values, which together with the column titled “Sampling” (averaged over 10 runs) results in the value denoted in column “Total”, which shows the total number of iterations needed to obtain an error of 0.05 or lower. Apparently, for large graphs (over 100,000 nodes), a sampling rate of \( q = 0.10 \) is more than sufficient to accurately derive the eccentricity distribution. An alternative would be to change the number of samples to a constant number, which would ensure that the number of eccentricity computations does not exceed some fixed maximum.

An advantage of the exact approach is obviously that the nodes that have a certain eccentricity value can be pointed out exactly, whereas this is only possible for the extreme values in case of the hybrid approach. The hybrid approach does however improve upon the exact algorithm in terms of computation time (number of iterations) in many cases, especially for the larger graphs (over 100,000 nodes) with relatively tight eccentricity distributions, as can be seen in the rightmost column of Table 3.3.

A bold value indicates the largest of the two speedups when comparing the hybrid approach to the exact approach. The exact approach performs better than the hybrid approach in a few cases, which we believe is due to the fact that the sampling phase should not have a too large window when the number of nodes is very small, because then a large number of iterations would be needed to obtain a representative sample of the distribution. This is especially the case for Web-Stanford, which has a very long tail. For this dataset, the exact algorithm performs really well, whereas the hybrid approach needs quite a few more iterations. It would be possible to determine a priori whether the exact or the hybrid algorithm should be used, as after a few BFSes the expected width of the eccentricity distribution is already known. We mention that, analogously to the performance of the exact algorithm, the performance of the hybrid algorithm does not appear to be correlated with the number of nodes.

From the experiments in this section we can generally conclude that we have developed a flexible approach for determining the eccentricity distribution of a large
graph with high accuracy, while guaranteeing an exact result on the extreme values of the distribution.

3.7 Conclusion

In this chapter we have studied means of computing the eccentricity of all nodes in a graph, resulting in the (relative) eccentricity distribution of the graph. It turns out that the eccentricity distribution of real-world graphs has an unimodal shape and tends to have a positive tail. We have shown how large speedups compared to the traditional method can be achieved by considering lower and upper bounds on the eccentricity, and by applying a pruning strategy. Even when the exact algorithm does not immediately give a satisfying result, the lower bounds proposed in this chapter can serve as a reliable online estimate of the distribution as a whole.

We have also investigated the use of sampling as a means of computing the eccentricity distribution. The resulting hybrid algorithm uses the exact approach to derive the extreme values of the distribution and a sampling technique within a specific sampling window to accurately assess the values in between the extreme values. This results in an overall approach that is able to efficiently determine the eccentricity distribution with only a fraction of the number of computations required by the naive approach.

In future work we would like to investigate the extent to which eccentricity and other centrality measures are related, and when the eccentricity can be used as a meaningful centrality measure. We furthermore found that the radius, center and periphery of the graph can be derived using the exact algorithm proposed in this chapter. The question remains whether or not these measures can be computed more efficiently if the right stopping criterion is used, which we will discuss in Chapter 4. It may also be interesting to look at how the eccentricity distribution of a graph that is evolving through the addition and deletion of nodes and edges can be efficiently computed and monitored.