Algorithms for Analyzing and Mining Real-World Graphs

Takes, F.W.

Part I

Graph Algorithms

2

Determining the Diameter of Small-World Networks

This chapter presents a novel approach to determine the exact diameter (longest shortest path length) of large graphs, in particular of the nowadays frequently studied small-world networks. Typical examples include social networks, biological networks, webgraphs and internet topology networks. Due to complexity issues, the diameter is often computed based on a sample of only a fraction of the nodes in the graph, or some approximation algorithm is applied. Instead, we propose an exact algorithm that uses various lower and upper bounds as well as effective node selection and pruning strategies in order to evaluate only the critical nodes which ultimately determine the diameter. The proposed algorithm is able to quickly determine the exact diameter of various large small-world networks with millions of nodes and hundreds of millions of links, whereas before only approximations could be given. This chapter is based on:

- F. W. Takes and W. A. Kosters. Determining the diameter of small world networks. In *Proceedings of the 20th ACM International Conference on Information and Knowledge Management (CIKM 2011)*, pages 1191–1196, 2011

2.1 Introduction

With the rapidly increasing amount of graph data that is being generated and academically studied, researchers are often interested in quickly deriving various global properties of their graphs. While several trivial static properties of the graph such as the graph density (number of edges of the graph vs. the maximum number of edges that could possibly exist) can easily be computed, determining other properties of large graphs using straightforward algorithms may require a lot more computation time. One of these more "expensive" properties is the *diameter* of a graph, which is defined as the maximal distance (length of a longest shortest path) between any two nodes in the graph. Exact algorithms for computing the diameter traditionally require running an All Pairs Shortest Path (APSP) algorithm for each node in the graph, ultimately returning the length of one of the longest shortest paths that was found. While this will indeed return the exact diameter of the graph, complexity for a graph with n vertices and m edges is in the order $O(n^3)$ for weighted graphs and $O(mn)$ for sparse unweighted graphs. This naive method for obtaining the diameter is clearly not feasible in extremely large graphs with for example millions of vertices and a billion edges.

We will study the diameter of *small-world networks*: sparse networks that are most typically characterized by an average distance between two random nodes that grows only proportionally to the logarithm of the total number of nodes in the network [71]. Examples of small-world networks that are frequently studied are web-graphs [8, 28], internet topology networks [66] and biological networks [6, 67], but perhaps nowadays most well-known are social networks [120, 139]. With the introduction of *online* social networks such as Facebook, LinkedIn and Orkut, even more than before, the study of social networks has become interesting for computer scientists, as the data behind these networks can easily be gathered in a digital format. Other (implicit) social networks are telephone call graphs, e-mail networks [73] and scientific collaboration networks [13].

The diameter is a relevant property of a network for many reasons. For example in social networks, the diameter could be an indication of how quickly information reaches literally everyone in the network. Within a scientific collaboration network, a high diameter may indicate that there are groups of researchers that are not working together very closely. In an internet routing network, the diameter could reveal something about the worst-case response time between any two machines in the network. In a way, the diameter can be seen as a measure of how data or information spreads over the network in the worst case.

The diameter is not just a static property of a graph, but it is also used in various algorithms in which it serves as the maximum depth of a search procedure, for

example in a Depth Limited Search algorithm [117]. Work is also being done on studying how a graph evolves over time [42]. There, knowing the exact point in time when the diameter changes can be interesting, which may favor an exact answer over an approximation. Another important advantage of studying the exact diameter is that we can observe the actual path that realizes the diameter, a piece of information that we do not get when for example an approximation algorithm is used, or when the diameter is estimated by looking at a sample.

The main contribution of this chapter consists of a new algorithm for determining the exact diameter of small-world networks. Based on various lower and upper bounds and critical node selection strategies, we improve upon the straightforward APSP algorithm as well as upon existing approximation algorithms, obtaining the exact diameter of networks with millions of nodes in a matter of seconds or minutes. The performance is empirically verified on various large small-world networks.

The rest of the chapter is structured as follows. Section 2.2 introduces various definitions and a short analysis of the problem's complexity, after which we will cover relevant related work in Section 2.3. We will use Section 2.4 to outline our algorithm for deriving the diameter of a graph, and discuss experimental results in Section 2.5. In Section 2.6 we look at a parallel version of the proposed algorithm, and finally Section 2.7 concludes.

2.2 Preliminaries

In this section we will first consider some basic definitions related to graphs, distances, eccentricity, graph diameter and shortest path problems, and then give some insight in the complexity of determining these measures. After that we will briefly discuss small-world networks.

2.2.1 Definitions

A graph $G = (V, E)$ consists of a set of vertices (or *nodes*) V and a set of links (or *edges*) $E \subseteq V \times V$. Throughout the chapter we will use n to denote the number of nodes $|V|$, and for the number of links $|E|$ we use m. 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. This chapter deals with *unweighted* graphs, and we will assume that graphs are *undirected*, meaning that $(v, w) \in E$ iff $(w, v) \in E$ and thus $d(v, w) = d(w, v)$. Our definition of an edge does not allow parallel edges, and we furthermore disallow self-loops. Thus note that m is the number of (directed) links between distinct nodes and $m/2$ is the number of (undirected) edges. The *degree* of a node v in an undirected graph is simply defined as the number of (undirected) edges connected to that node. Finally,

we assume that the graph is *connected*, implying that for each $v, w \in V$, $d(v, w)$ is a finite number. We are now ready to define the two most important concepts used in this chapter, node *eccentricity* and graph *diameter*:

Eccentricity The *eccentricity* $e(v)$ of a node $v \in V$ is defined as $\max_{w \in V} d(v, w)$: the length of a longest shortest path starting at node v.

Diameter The *diameter* $D(G)$ of a graph G is defined as $\max_{v,w \in V} d(v, w)$: the longest shortest path length between any pair of nodes, or equivalently as the maximum eccentricity over all nodes: $\max_{v \in V} e(v)$.

Note that when used as a variable, we simply use D to denote the diameter. For convenience, we define two combinatorial problems that are frequently addressed in this chapter and are tightly related to eccentricity. First, the *Single-source Shortest Path* (*SSP*) problem is the problem that deals with finding all shortest paths from a single source node $v \in V$ to all other nodes in the graph. For non-sparse graphs this problem has the traditional time complexity of $O(n^2)$ (Dijkstra's shortest path algorithm). When the graph is sparse, it can more efficiently be stored using an adjacency list instead of an adjacency matrix. Then in our unweighted case, time complexity can even be reduced to $O(m)$, as a Breadth First Search (BFS) from the starting node is sufficient to find all shortest paths starting at that node. In essence, solving the SSP problem for a node means that we have found the eccentricity of that particular node.

Next, we can define the *All Pairs Shortest Paths* (*APSP*) problem as the problem of finding the shortest paths between all pairs of nodes of the graph, which increases the previous time complexity by a factor n to $O(n^3)$ for weighted graphs, and $O(mn)$ for the considered sparse unweighted graphs. The maximum distance value that the APSP algorithm obtains, is then the maximum eccentricity (which is computed for a node v using the function ECCENTRICITY() in Algorithm 2.1) over all nodes and thus equal to the diameter of the graph. So if we solve the APSP problem, we have also found the diameter of the graph.

Algorithm 2.1 DIAMETERAPSP

```
1: Input: Graph G
2: Output: Diameter of G
3: D ← −∞
4: for v ∈ V do
5:     D ← max(D,ECCENTRICITY(v))        // one BFS
6: end for
7: return D
```

2.2.2 Small-world networks

In this chapter we will specifically look at the diameter of *small-world networks*. A good overview of algorithmic properties of these networks, of which we will discuss a few, is given in [71]. First of all, small-world networks are generally *sparse*: the total number of links m is very small compared to the maximum number of links $n(n-1)$. This may cause the reader to believe that nodes have a rather long shortest paths between them, as there are very few links in general. However, a second interesting characteristic is that even though the network is very sparse, the average distance between two nodes is very small. More specifically, this distance is typically somewhat proportional to the logarithm of the total number of nodes. The node degree distribution of a small-world network usually follows a power law: there are only a few nodes with a very large number of connections, the so-called *hubs*, and there are many nodes with relatively few connections. Hubs are in turn responsible for realizing very low average shortest path lengths. So even though many nodes are not direct neighbors of one another, most nodes can be reached from every other node via only a small number of steps. A last property of small-world networks, is that they generally contain one very large connected component (the *giant component*) which contains the vast majority of the nodes.

2.3 Related work

A lot of work has been done on devising algorithms for the *estimation* of the diameter [44, 115]. Such estimation algorithms typically determine the diameter of any type of graph (sparse or dense) with some very small additive error, but using significantly less computation time than the APSP algorithm. For example in [3], a method is suggested which finds the diameter in $O(n^{2.5}\sqrt{\log n})$ time with an additive error of 2. Work has also been been done on testing if the diameter is (with some small margin of error) equal to a certain value [110].

A popular method which is used in many graph analysis toolkits, is the Approximate Neighborhood Function (ANF) by Palmer et al. [109]. This technique approximates the size of the neighborhood of (sets of) nodes, and is thus also able to approximate the diameter. Based on this technique, a variant of the diameter called the *effective diameter* was introduced, which is defined as the 90-th percentile of the cumulative distribution of shortest path lengths. Though this measure may appear more robust to outliers, it is claimed that the diameter and the effective diameter "tend to exhibit qualitatively similar behavior" [86]. Another measure closely related to the diameter that is sometimes mistakenly spoken of as if it were the real diameter, is what some call the average diameter, which is actually the average shortest path length: the aver-

age distance between any pair of nodes, i.e., $1/(n(n-1)) \sum_{v \neq w \in V} d(v,w)$. This value is often approximated by selecting a few thousand random pairs of nodes from the graph, and determining the average of their pairwise distances.

In work which does not focus on actually determining the diameter, but where it is found only as a static property of the dataset, a *sample* of the graph is frequently used to determine the diameter [84, 103]. There are at least two directions to determining the diameter when using a sample. The first option would be to select a sample of the nodes in the original network using a suitable sampling approach [84], and then determining the diameter of this sample using the straightforward APSP algorithm. The second option would be to assess the diameter based on selecting a few nodes from the original graph that are likely to have a high eccentricity value, which is somewhat the idea behind the method described in [88]. In this work, the diameter is determined by, starting from a random node, repeatedly selecting the farthest node, meanwhile keeping track of the highest distance so far. If this value no longer increases after a certain number of iterations, then this value is a lower bound on the diameter has been found. Similar techniques are employed in [17, 34, 95], and it is argued that using a handful of Breadth First Searches, empirically tight bounds on the diameter can be obtained.

Most *exact* algorithms for finding the diameter are actually implementations of matrix multiplication that solve the APSP problem and thus also find the diameter. While these algorithms work well and have time complexity $O(n^{2.376})$ [9], they usually suffer from large hidden constants, and are often very unpractical due to large memory requirements. To the best of our knowledge, the exact approach suggested by Crescenzi et al. [36, 37] is the only other exact algorithm for determining the diameter of large graphs. The suggested approach uses a strategy somewhat similar to the algorithm that we propose in this chapter. A comparison is provided in [36, 99].

2.4 BoundingDiameters

In this section we describe our approach for computing the diameter. We will start with some observations about the eccentricity of neighboring nodes and how they influence the diameter. Then we describe the actual algorithm called BOUNDINGDIAMETERS, which makes use of these observations to improve upon the APSP algorithm. Next we discuss the algorithm's complexity and some simple optimization techniques.

2.4.1 Observations

If we compute the eccentricity $e(v)$ for some node v, we know that for all nodes w with $d(v,w) = k$, their eccentricity $e(w)$ lies between $e(v) - k$ and $e(v) + k$. The upper

bound follows because any node w at distance k of v can get to v in exactly k steps, and then reach any other node in at most $e(v)$ steps. The lower bound can be derived in the same way, by interchanging v and w in the previous statement. In the "best" case, w is on some path that realizes the eccentricity of v, and has an eccentricity $e(w)$ of only $e(v) - k$. This lower bound can of course never be less than k itself: if the shortest path between v and w has length k, then $e(w)$ is at least equal to k. In essence, we are making use of the triangle inequality in graphs. So we have:

Observation 2.1 ***Node eccentricity bounds***
If a node $v \in V$ has eccentricity $e(v)$, then for all nodes $w \in V$ we have:

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

The proof of this observation is simple: we know that the diameter of a graph is equal to the maximum eccentricity over all nodes. Therefore, the maximum lower bound on the eccentricity over all nodes, is also a lower bound for the diameter. Similarly, the maximum upper bound on the eccentricity over all nodes can be seen as an upper bound on the diameter. The upper bound can even be made more tight by observing that this bound can be at most twice as big as the smallest eccentricity upper bound over all nodes, as also observed in [95]. These observations can be formalized as follows to form lower and upper bounds on the diameter:

Observation 2.2 ***Diameter bounds***
Let $e_\ell(v)$ and $e_u(v)$ denote currently known lower and upper bounds for the eccentricity of node $v \in V$. For the diameter $D(G)$ of a graph G it holds that:

$$\max_{v \in V} e_\ell(v) \leq D(G) \leq \min(\max_{v \in V} e_u(v), 2 \cdot \min_{v \in V} e_u(v))$$

We will denote these lower and upper bounds on the diameter by D_ℓ and D_u, respectively. Note that as opposed to the even upper bound of $e(v) \leq D(G) \leq 2 \cdot e(v)$ suggested in [95], the proposed algorithm is able to derive an odd upper bound, which is obviously necessary for finding the exact diameter when the diameter itself has an odd value.

2.4.2 Algorithm

The bounds mentioned above can be used to improve the original APSP algorithm from Algorithm 2.1 by reducing the number of eccentricity computations, as only nodes that can actually contribute to the diameter bounds are considered. Pseudocode for the BOUNDINGDIAMETERS algorithm is given in Algorithm 2.2.

After setting some initial values (lines 3–7), in the main while-loop, the algorithm repeatedly selects a node v (line 9) from the candidate set W, which initially contains all the nodes. The various mechanisms for selecting the next node to be examined are outlined in Section 2.4.4. The algorithm then computes the eccentricity of that node v (line 10), and uses the result to update the lower and upper bound of the graph diameter (D_ℓ and D_u, lines 11–12), cf. Observation 2.2. Note that we use $e[v]$ when we reference to the (array) variable containing the eccentricity $e(v)$ of node v. Next, the eccentricity bounds of all nodes in the candidate set W are updated (lines 14–15) according to Observation 2.1. Then we determine which nodes (including v) can be removed from the set of candidates (line 17). This can happen either because the eccentricity of the node is already known since the lower and upper bounds are identical (which is always the case for the current node v), or because a node can no longer "contribute" to the diameter of the graph by increasing the lower bound or decreasing the upper bound (line 16). Note that because we performed a BFS to compute the eccentricity of v, we know for each node w the distance $d(v, w)$ which is needed to apply Observation 2.1. After adjusting the node eccentricity bounds, the diameter upper bound is further tightened using the largest node eccentricity upper bound, again cf. Observation 2.2 (line 20). Finally, the algorithm stops when all nodes have been examined, or when the lower bound is equal to the upper bound (line 8). It then returns the lower bound of the diameter, which at that point contains the real value of the diameter (line 22).

A proof of the correctness of this algorithm can be constructed by considering the fact that D_ℓ, which is returned on line 22, contains the largest computed eccentricity value (line 12). So, given Observation 2.1 and the assumption that in line 16 only nodes that can not potentially increase the value of D_ℓ are removed, the algorithm returns the correct value of the diameter.

2.4.3 Complexity

Computing the eccentricity of a node using the ECCENTRICITY() function (line 10) is the critical operation of the algorithm, as this requires running a SSP algorithm (one BFS), taking $O(m)$ time. In the best case, we only have to compute the eccentricity of two nodes v and w, only to find that $e(w) = 2 \cdot e(v)$ (or vice versa), which means that the diameter is equal to $e(w)$. In the worst case the algorithm needs to investigate the eccentricity of every single node, not improving the traditional APSP time complexity of $O(mn)$. An example of a graph in which all nodes have to be investigated in order to determine the diameter, is a graph where the nodes are connected through exactly one circle of edges, meaning that all nodes have identical eccentricity. Of course, graphs are generally not shaped as a circle, neither is the diameter always equal to two times

Algorithm 2.2 BOUNDINGDIAMETERS

```
1: Input: Graph G
2: Output: Diameter of G
3: W ← V    D_ℓ ← −∞    D_u ← +∞
4: for w ∈ W do
5:    e_ℓ[w] ← −∞
6:    e_u[w] ← +∞
7: end for
8: while D_ℓ ≠ D_u and W ≠ ∅ do
9:    v ← SELECTFROM(W)
10:   e[v] ← ECCENTRICITY(v)
11:   D_ℓ ← max(D_ℓ, e[v])
12:   D_u ← min(D_u, 2 · e[v])
13:   for w ∈ W do
14:      e_ℓ[w] = max(e_ℓ[w], max(e[v] − d(v, w), d(v, w)))
15:      e_u[w] = min(e_u[w], e[v] + d(v, w))
16:      if (e_u[w] ≤ D_ℓ and e_ℓ[w] ≥ D_u/2) or
            (e_ℓ[w] = e_u[w]) then
17:         W ← W − {w}
18:      end if
19:   end for
20:   D_u ← min (D_u, max_{w∈V}(e_u[w]))
21: end while
22: return D_ℓ;
```

the eccentricity of some node in the graph (if we are even able to quickly find two such nodes). In general, the eccentricity values of nodes in a network differ, and how much they differ will likely influence the number of iterations that is required, as larger differences in eccentricity values will result in tighter eccentricity bounds on surrounding nodes.

We claim that the algorithm specifically works well on small-world networks, which we believe is due to power law degree distribution within such networks, as discussed in Section 2.2.2. A small-world network has relatively few nodes with a very high degree (hubs), that will often (but not always) have a relatively low eccentricity value. The remainder of the nodes typically have a much lower degree, often (again, not always) resulting in a relatively high eccentricity value. Thus, due to the expec-

ted existence of a large diversity in eccentricity values of the nodes in a small-world network, the bounds on the diameter will typically converge very quickly. When we look at a degree-based selection strategy in Section 2.4.4.1, we will verify this claim empirically.

2.4.4 Selection strategies

This section describes the different strategies that can be used to select the next node for which we want to compute the eccentricity, outlining the possible functionality of the SELECTFROM() function that is called in line 9 of Algorithm 2.2. First notice how enumerating the nodes in some order, or selecting them at random, will in essence mean that we are executing the APSP algorithm, only now we discard nodes that can no longer contribute to the diameter bounds. While this will no doubt already improve upon the APSP algorithm, it mainly serves as a baseline of comparison, as the main objective is to tighten the bounds of the diameter as quickly as possible in order to efficiently reduce the size of the set of candidate nodes.

Any strategy that we come up with has to be easy to compute so that it does not influence the overall complexity of the diameter algorithm. More specifically, it should be possible to determine the next node by iterating over the set of nodes once, such that the selection function could even be done on-the-fly while updating the bounds in the previous iteration. We will test the performance of (combinations of) the strategies described below in Section 2.5.

2.4.4.1 Degree centrality

Perhaps the simplest strategy would be to select nodes based on their *degree*, hoping that high degree nodes have a low eccentricity value, and vice versa. This measure, known as *degree centrality*, is often suggested as a simple measure of the centrality of a node within a network, but is far from perfect for predicting the eccentricity. For example, in Figure 2.2 node F has the highest degree (6 links) and eccentricity $e(F) = 5$, whereas node J with lower degree 3 has eccentricity $e(J) = 4$. Also, a low degree is no guarantee for a low eccentricity value, as in small-world networks there are typically many nodes with a low degree, and these nodes may still be connected to the most central nodes, resulting in a low eccentricity value even though the degree is also very low. The problem here is that degree centrality is merely a local measure: it does not take into account any aspects of the graph beyond its own neighborhood. Indeed, as Figure 2.1 suggests, node degree and node eccentricity are not directly related: not all nodes with a high degree have a low eccentricity value, and not all nodes with a low degree have a high eccentricity value. Therefore we suggest using the degree as a secondary selection mechanism only, mainly to break ties in other

selection methods, or to select the very first node to be examined. We mention that although many more centrality measures exist [27], a downside is that they are often as hard to compute as the eccentricity or the diameter itself and therefore not suitable to serve as a selection mechanism.

2.4.4.2 Eccentricity bound difference

During the execution of the proposed algorithm, the *difference* $e_u(v) - e_\ell(v)$ between the lower and upper eccentricity bound could be an interesting feature, as it says something about how much we already know about the eccentricity of node v and its neighborhood. If for a certain node this difference is very big, determining its eccentricity may tighten the bounds of many nearby nodes. In essence, sorting by bound difference in decreasing order means that we are repeatedly taking a node in an area of the graph which has not been very thoroughly explored yet.

2.4.4.3 Interchanging eccentricity bounds

Inspired by traditional branch-and-bound algorithms that repeatedly select the nodes with the best bound value for expansion, we could choose to select nodes from the candidate set based on their their quality in terms of how well we expect them to

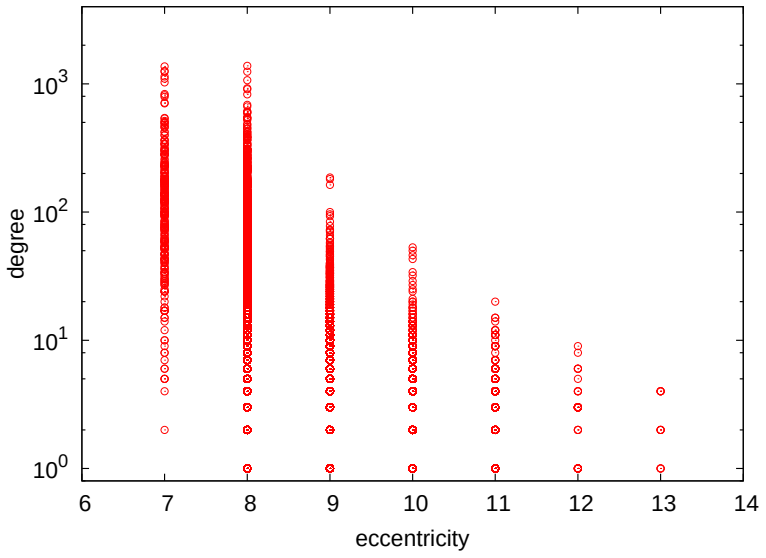

Figure 2.1: Degree (vertical axis) and eccentricity (horizontal axis) of the nodes in the ENRON graph.

contribute to tightening the diameter bounds. To find nodes with high eccentricity, we can select the node v with the largest upper bound $e_u(v)$, and similarly we choose a node with a small lower bound $e_\ell(v)$ to find nodes with a low eccentricity value. As the goal is to increase the lower bound and decrease the upper bound, we propose to *interchange* the selection of the node with the smallest lower bound and the node with the largest upper bound.

2.4.4.4 Repeated farthest distance

Another option is to select a node based on its distance to the previously investigated node, and then select a node with the highest distance. So starting from some initial node v, we repeatedly select the farthest possible candidate node w with $d(v, w) = e(v)$. This is a variation of the heuristic for approximating the diameter suggested in [88]. The only difference is that in this context, the stopping criterion for the algorithm is exactly defined, namely when the diameter lower and upper bounds are equal.

2.4.5 Example

We will give an example of how BOUNDINGDIAMETERS would determine the diameter of the graph depicted in Figure 2.2. As a selection strategy we alternately choose the largest upper bound and smallest lower bound (cf. Section 2.4.4.3), breaking ties by choosing the nodes with the highest degree (cf. Section 2.4.4.1). Any remaining ties are broken by choosing a random node. In this example, we will denote the lower and upper eccentricity bounds $e_\ell(v)$ and $e_u(v)$ of a node v by $[e_\ell(v); e_u(v)]$.

Initially, all nodes form the candidate set, and all lower bounds and all upper bounds are equal. We start at node F which has the highest degree, and remove it from the candidate set. The situation of Figure 2.2 depicts the situation after the first iteration, where node F has been investigated. The diameter lower and upper bounds are now equal to $D_\ell = e(F) = 5$ and $D_u = 2 \cdot e(F) = 10$, respectively. Notice how for node M and N we set the bounds to $[3; 8]$ and not to $[2; 8]$, because $d(M, F) = d(N, F) = 3$ and the eccentricity is at least equal to 3. The current eccentricity bounds do not yet require us to remove any nodes from the candidate set.

In the second iteration, we determine the eccentricity of the node with the largest eccentricity upper bound, which could be T or S as they both have bounds $[5; 10]$. We choose T. The eccentricity of node T turns out to be 7, and the eccentricity bounds after the second iteration are depicted in Figure 2.3. Here, nodes that can no longer contribute to computing the diameter are green if the lower and upper bounds have become equal and red if they have bounds such that they cannot contribute to either increasing the lower bound or decreasing the upper bound. The graph diameter now

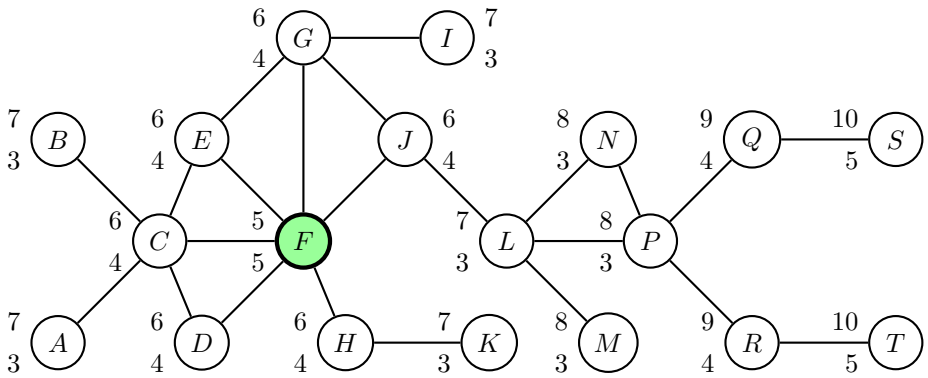

Figure 2.2: Example graph with eccentricity bound values after the first BFS from F.

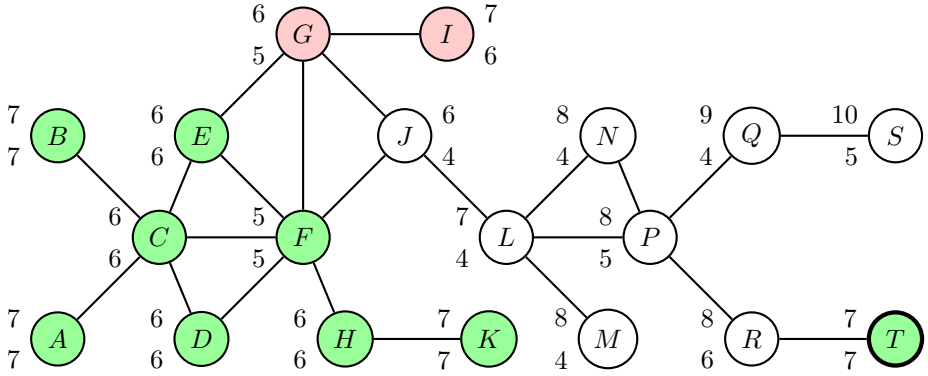

Figure 2.3: Example graph with eccentricity bounds after the second BFS from T.

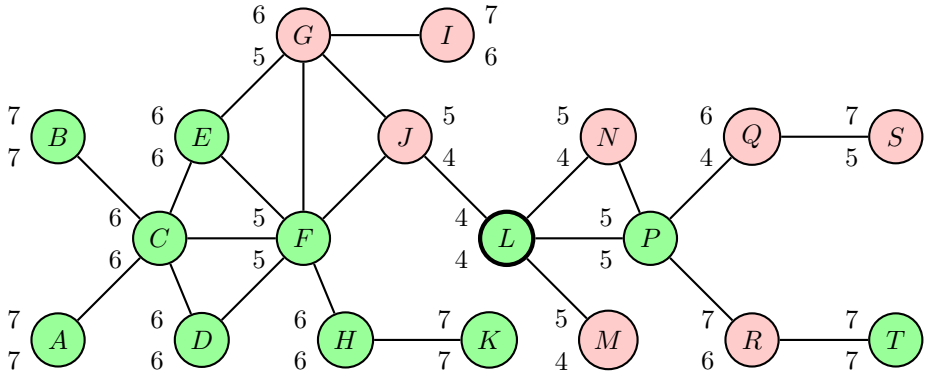

Figure 2.4: Example graph with eccentricity bound values after the third BFS from L.

lies between $D_\ell = 7$ and $D_u = 10$. We can now remove A, B, C, D, E, H and K from the candidate list, as we have found the exact eccentricity of these nodes (but without having computed it explicitly). We can also remove node I and node G with bounds $[6; 7]$ because they can no longer contribute to raising the lower bound or decreasing the upper bound.

For the third loop of the algorithm we compute the eccentricity of the node with the smallest lower bound (and as secondary selection, the highest degree), which is node L. It has an eccentricity of 4, meaning that we can now discard all nodes based on the same arguments as in the previous iteration, resulting in all nodes being visited (see Figure 2.4), terminating the algorithm after only 3 eccentricity computations, and returning the maximum over all lower bounds as the final value of the diameter: 7.

2.4.6 Pruning

The size of the graph can be reduced by applying the following pruning strategy beforehand. For every node we can determine if removing all of its adjacent edges would disconnect the graph. If this is the case, and multiple identically structured small subgraphs remain, we can remove each but one of them, and still obtain the correct diameter value, assuming of course that a path that realizes the diameter of the graph does not run from one subgraph to another (pruned) subgraph. Therefore, the diameter of the pruned subgraph has to be smaller than $D(G)/2$.

For example node C in Figure 2.2 is connected to two identical subgraphs, namely the subgraph consisting of node A and the subgraph consisting of node B. We could prune one of these subgraphs, as they will both have identical eccentricity (bound) values. Similarly, P is connected to identical subgraphs $Q - S$ and $R - T$, both with identical eccentricity values. Indeed, in the second iteration of the example run described in Section 2.4.5, we could have chosen either S or T, both resulting in the same adjustments to the bound values of the remaining nodes.

The proof of the validity of this pruning strategy can be constructed based on the concept of graph isomorphism, a bijection from one graph to another graph in which the connectedness of the graph is preserved. More precisely, a *graph isomorphism* $h : G \to G'$ of a graph $G = (V, E)$ to another graph $G' = (V', E')$ is a bijection $h : V \to V'$ from the set of nodes V in the original graph to the set of nodes in the projected graph V' such that $(u, v) \in E$ iff $(h(u), h(v)) \in E'$ [60]. In the example from the previous paragraph, node A and B map to each other, as do nodes Q and S to nodes R and T, respectively. Because graph isomorphism preserves connectedness, distance measures such as the eccentricity are also preserved. Although the general problem of deciding if there exists a isomorphism from one graph to another graph is NP-complete, the strategy described above is able to efficiently detect the simple type

of isomorphism, namely that of subgraphs of a very small size that arise when only one edge is removed.

Nodes that are pruned contribute to speeding up the algorithm in two ways: pruned nodes do not have to be considered during the eccentricity computation and also do not have to be included in the set of candidate nodes.

2.5 Experiments

This section starts with a brief description of the datasets (graphs), and then describes a measurement methodology for the different selection strategies described in Section 2.4.4. Next, the results of applying these strategies in the BOUNDINGDIAMETERS algorithm are discussed.

2.5.1 Datasets

We will verify the algorithm on various small-world networks. Characteristics of these graphs, such as the number of nodes, the number of links, the average degree $\overline{deg}$, average node-to-node distance $\overline{d}$ and the diameter $D(G)$, are given in Table 2.1. Numbers are based solely on the largest connected component of each graph, and originally directed graphs are interpreted as if they are undirected. Therefore, slight deviations from statistics presented in the original papers describing these graphs may be observed.

The CA-ASTROPH dataset is a an undirected network of scientific collaborations (co-authorship) in the field of astrophysics, which was obtained through arXiv and analyzed in [87]. ENRON [73] is a well-known network of e-mail contacts within a

Dataset	**Nodes** n	**Links** m	$\overline{deg}$	$\overline{d}$	$D(G)$
CA-ASTROPH [87]	17,903	393,944	21	4.15	14
ENRON [73]	33,696	361,622	10	4.07	**13**
WEB-GOOGLE [88]	855,802	8,582,704	10	6.30	24
YOUTUBE [103]	1,134,890	5,975,248	5	5.32	24
FLICKR [103]	1,624,992	30,953,670	18	5.38	24
AS-SKITTER [86]	1,694,616	22,188,418	13	5.08	**31**
WIKIPEDIA-NL [10]	2,213,236	23,520,520	11	4.81	18
ORKUT [103]	3,072,441	234,370,166	76	4.16	10
LIVEJOURNAL3 [103]	5,189,809	97,839,882	19	5.48	23
HYVES [128]	8,083,964	912,067,984	112	4.75	25

Table 2.1: Characteristics of the datasets: various small-world graphs.

company, in which a node represents an e-mail address and two nodes are connected if an e-mail has been sent between these two addresses. The WEB-GOOGLE dataset is a partial crawl of the world wide web [88]. The FLICKR, LIVEJOURNAL, ORKUT and YOUTUBE datasets are partial crawls of the respective online social networks, and are studied in detail in [103]. AS-SKITTER is an undirected internet topology graph created from network traceroutes, which is analyzed in detail in [86]. WIKIPEDIA-NL is a full crawl of the Dutch Wikipedia graph as present in DBpedia 3.5 [10], a community effort to extract structured information from Wikipedia. The dataset denoted by HYVES is the full friendship graph of a Dutch online social network (see Chapter 6).

2.5.2 Measurement methodology

In the following experiments we will compare the three different node selection strategies from Section 2.4.4:

- **Strategy 1**: Largest eccentricity bound difference (cf. Section 2.4.4.2)
- **Strategy 2**: Interchanging largest upper bound and smallest lower bound (cf. Section 2.4.4.3)
- **Strategy 3**: Farthest distance (see Section 2.4.4.4)

Ties are broken by taking the node with the highest degree (cf. Section 2.4.4.1). Any remaining ties are broken by picking the lexicographically first node, which is determined by the order in which the nodes are read from the input file. Thus, each of the selection strategies is deterministic.

The critical step of the algorithm is clearly one BFS, so the step of computing the actual eccentricity of one node. The number of times that a BFS is executed (which we will refer to as the number of iterations) will therefore serve as a basis of comparison of the three strategies. Note that the number of iterations that the traditional APSP algorithm from Algorithm 2.1 would perform, is one eccentricity computation for each node in the graph, so a total of n iterations. We have chosen to only implement the simple optimization strategy (see Section 2.4.6) of pruning duplicate connected subgraphs consisting of one node.

2.5.3 Results

The number of iterations for each of the three strategies is given in the second, third and fourth column of Table 2.2, where a bold value indicates the best result amongst the three strategies. The last column indicates the number of pruned nodes as well as the percentage of the total number of nodes that was pruned.

The results show that with only a few actual eccentricity computations, the algorithm is able to determine the exact diameter of the datasets, with Strategy 2 as the best-performing node selection strategy. We expect this to be because interchanging the search for low and high eccentricity nodes means that we are interchanging the selection of a node in the dense and the peripheral part of the small-world network, quickly lowering the upper bound and increasing the lower bound, respectively. We believe that Strategy 1 did not perform so well, because although this strategy gives a hint towards areas of the graph that have note been thoroughly explored, it does not give any guarantees on the size of this area (which could be very small) and the number of such areas (which could be very large). Strategy 3 appeared to perform worse because it was not able to find any low-eccentricity valued nodes that could lower the diameter upper bound.

We observed that even when no selection strategy is applied (so, by selecting candidate nodes at random), using the suggested lower and upper eccentricity bounds can help to converge on the diameter more quickly than using the APSP algorithm. For larger datasets, this number was well over $10,000$ and thus still far too time-consuming, but for CA-ASTROPH 260 ± 95, for ENRON 316.5 ± 142 and for WEB-GOOGLE a total of $5,975 \pm 2,249$ iterations were needed (the number of iterations is averaged over 10 runs, so we also report the standard deviation) to obtain the exact diameter. This may suggest that contrary to the various selection strategies, random candidate node selection does not scale as the size of the graph increases.

We mention that for the YOUTUBE dataset, Strategy 2 only needs 2 eccentricity computations to determine the diameter, demonstrating the best-case performance of the algorithm. It turned out that the node with the highest degree had eccentricity 12, causing nodes with bounds $[12; 24]$ to exist. One of these nodes apparently had

Dataset	**Strategy 1**	**Strategy 2**	**Strategy 3**	**Pruned nodes**
CA-ASTROPH	18	**9**	63	185 (1.0%)
ENRON	12	**11**	61	8,715 (25.8%)
WEB-GOOGLE	20	**4**	28	91,965 (10.7%)
YOUTUBE	**2**	**2**	**2**	399,553 (35.2%)
FLICKR	10	**3**	7	553,242 (34.0%)
AS-SKITTER	10	**4**	19	114,803 (6.8%)
WIKIPEDIA-NL	21	**3**	583	947,582 (30.8%)
ORKUT	357	**106**	389	27,429 (0.9%)
LIVEJOURNAL	6	**3**	14	318,378 (6.1%)
HYVES	40	**21**	44	446,258 (5.6%)

Table 2.2: Performance (number of iterations) of different node selection strategies.

an eccentricity of value 24, realizing bounds of $[24; 24]$ and terminating the algorithm after just 2 iterations.

For the ORKUT dataset, compared to the other graphs, a relatively large number of eccentricity computations was needed to determine the diameter. To further analyze this, we look at how quickly the number of candidate nodes decreases during the execution of the algorithm. Therefore we show the number of unvisited nodes and the lower and upper bounds on the diameter during the execution of the algorithm using Strategy 2 on the ORKUT dataset in Figure 2.5. After 16 computations, another 90 computations were needed to decide whether the diameter was equal to 9 or 10, and the number of nodes to be examined only decreases by 1 or 2 after each eccentricity computation. Apparently the remaining unvisited nodes are positioned in the graph in such a way that the computation of the actual eccentricity of these nodes can not be avoided using the neighboring bounds. Although in this case it takes a while to find the exact diameter, tight bounds on the diameter are quickly available, as we have narrowed down the value diameter down to either one of two values.

The last column of Table 2.2 shows how the pruning strategy is able to significantly reduce the size of the problem. This is not surprising as small-world networks typically have many low degree nodes, and it is quite likely that many of these nodes are linked to the same node, and can thus be pruned.

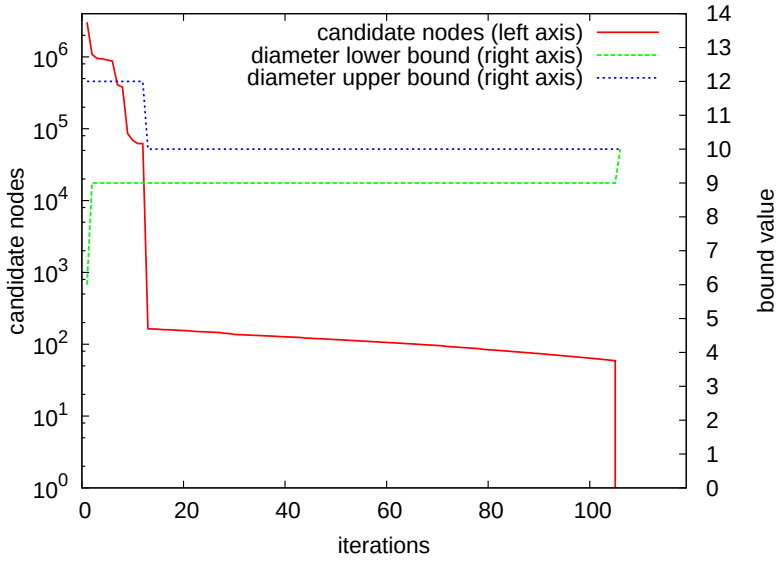

Figure 2.5: Candidate nodes (left vertical axis; logarithmic) and lower and upper bounds (right vertical axis) vs. iterations (horizontal axis) for the ORKUT dataset.

As an interesting side result it turned out that, very often, the actual eccentricity has been found for a large portion of the nodes in the graph when the algorithm has terminated, because the eccentricity lower and upper bounds of these nodes have become equal. For example, for the ORKUT dataset, $777,257$ actual eccentricity values (25%) were obtained, while only 106 values were explicitly computed. Similar numbers were observed for the other datasets. Also interesting to note is that the exact diameter that we computed for the ENRON and AS-SKITTER datasets deviates from the values of respectively 12 and 24 that were approximated in previous work.

Although not related to the performance measurement methodology, we mention that with a straightforward C++ implementation, one node eccentricity computation takes around six seconds for a dataset with 8 million nodes and 912 million edges on a standard 3.2GHz machine with 10GB of memory. This means that we are able to determine the exact diameter in a matter of seconds or minutes, which is a big improvement over the traditional APSP approach which would easily take over a year of computation time. More information on the datasets used in this chapter, the obtained diameter paths, and a simple implementation can be found at the supporting website: `www.liacs.nl/~ftakes/diameter/`. In later experiments, we ran the proposed diameter algorithm on a much larger set of graphs. The results can be found in Chapter 4. For a comparison of the proposed algorithm with related work, we refer the reader to two works that were published after the original article on which this chapter is based [36, 99].

2.6 GPU parallelism

A well-known technique to improve the performance of almost any algorithm, is to introduce parallelism in (parts of) the computation. With dual, quad and octa-core CPUs available in standard desktop machines, CPU parallelism can be an excellent way to reduce the runtime of an algorithm by performing certain operations in parallel. Nowadays, graphics processing units (GPUs) have even hundreds of cores, suggesting a much higher potential for exploiting parallelism compared to the CPU. However, whereas with a CPU algorithm it is often possible to obtain a speedup equal to the number of cores, using the GPU the speedup is usually much lower than the number of cores, as the architecture of the GPU is typically more "exotic" compared to what a regular CPU algorithm would expect. This section, which is largely based on work [41] together with Giso Dal, briefly summarizes to what extent GPU parallelism using the Nvidia CUDA framework can be applied to the BOUNDINGDIAMETERS algorithm.

Parallelizing an existing sequential algorithm is not always trivial, and involves

carefully selecting procedures within the algorithm that can be parallelized. In case of the BOUNDINGDIAMETERS algorithm discussed in this chapter, it turned out that 97% of the running time (the performance of GPU algorithms is usually measured in seconds instead of iterations) is spent on computing eccentricity values (line 10 of Algorithm 2.2). Therefore it makes sense to optimize the eccentricity computation (so, one SSP run or one BFS). In general the specific architecture of the considered GPU should be carefully taken into account when designing data structures and algorithms that are to be used on a GPU. Extensive research on parallelizing graph traversal on a GPU has been done, and it is clear that compared to sequential CPU algorithms, GPU parallelism introduces new challenges with respect to for example shared memory and synchronization [57, 101].

For the considered NVIDIA GPU (Fermi, compute 2.0), this means that the data structure used to store the frontier of the BFS should be optimized for the specific access pattern for which the GPU memory achieves the best performance. A straightforward GPU implementation of the eccentricity function, where a thread is assigned to each node in the frontier of the BFS, already results in a speedup factor up to $12\times$ on various large real-world graphs. The speedup appears to be somewhat dependent on the properties of the considered graph. Most notably, it appears to be influenced by the relation between the diameter and the effective diameter. A clear bottleneck of the standard (thread-based) GPU algorithm is the size of adjacency lists and the number of vertices in the frontier of the BFS. This problem can be overcome by looking at the number of nodes in the frontier at each step of the BFS. If this number is above a certain threshold, a different approach can be used, which utilizes a so-called "warp" of 32 parallel threads that processes the longer adjacency list more efficiently. Thus, a choice is made between either using one thread per adjacency list, or using multiple threads per list (but processing fewer lists in parallel). The resulting hybrid approach, which picks a different eccentricity algorithm (thread-based or warp-based) depending on the stage of the BFS, is able to realize a speedup of up to $21\times$ compared to the sequential CPU algorithm. For a more detailed description of these GPU algorithms and experimental results, the reader is referred to [41].

2.7 Conclusion

We have shown that the proposed algorithm, BOUNDINGDIAMETERS, is able to efficiently determine the exact diameter of small-world networks, making use of lower and upper bounds on the eccentricity of the nodes and on the diameter itself. A proper selection strategy allows the algorithm to exploit the characteristic properties of small-world networks. Moreover, we have outlined a pruning strategy which

reduces the size of the problem. We have also shown that even when the diameter is not found very quickly, very tight bounds on the diameter are available after only a few iterations.

In future work we will investigate if the proposed algorithm can be used to determine the radius (minimum eccentricity value over all nodes) of a graph. We furthermore want to see if we can obtain the eccentricity of all nodes in the network, allowing the study of the exact eccentricity distribution of a graph. These two issues will be addressed in Chapter 4 and Chapter 3, respectively.

It may also be interesting to look at the problem of determining the diameter of the strongly connected component of a directed graph, and that of weighted graphs. In line with results presented in related work [39], we expect that the bounding approach will also work well on weighted graphs, as the diversity in edge weights and thus path lengths will undoubtedly influence the difference in eccentricity values and speed up the convergence of lower and upper eccentricity bounds and therewith the diameter bounds. In preliminary experiments we see that by using only the lower diameter bounds, significant improvements over the APSP algorithm can already be observed. Following up on the GPU implementation of the proposed algorithm briefly discussed in Section 2.6, work can still be done on parallelization using the CPU or a combination of the CPU and GPU. Last but not least, we hope to investigate how the exact diameter of small-world networks behaves over time, and how the algorithm can be adjusted to adapt to changes in the network, i.e., the addition and deletion of nodes and links.