Journal of Universal Computer Science, vol. 22, no. 3 (2016), 302-318
submitted: 31/8/15, accepted: 23/1/16, appeared: 1/3/16 © J.UCS

Calculating Exact Diameter Metric of Large Static Graphs

Masoud Sagharichian
(Iran University of Science and Technology, Tehran, Iran
sagharichian@iust.ac.ir)

Morteza Alipour Langouri
(Iran University of Science and Technology, Tehran, Iran
m_alipour@comp.iust.ac.ir)

Hassan Naderi
(Iran University of Science and Technology, Tehran, Iran
naderi@iust.ac.ir)

Abstract: The variety of applications requiring graph analysis is growing rapidly. Diameter is one of the most important metrics of a graph. The diameter is important in both designing algorithms for graphs and understanding the nature and evolution of graphs. So, detecting diameter of large graphs is very important. We propose an algorithm to calculate the diameter of such graphs. The main goal of this algorithm is to reduce the number of breadth-first searches required to determine the diameter of the graph by finding a better upper bound for the eccentricity of vertices. Based on experimental results, our proposed algorithm can quickly detect the exact diameter of the large-scale real world graphs with a few number of breadth-first searches.
Keywords: diameter, static graphs, graph mining, social networks
Categories: F.2, J.1, J.4, K.4.2

1 Introduction

Due to the spread use of networks, including web graphs [Reka et al., 1999], internet topology networks [Shudong and Azer, 2006], road networks [Jure and Krevl, 2014], peer-to-peer networks [Bawa et al., 2003], complex networks [Del Mondo et al., 2010] and social networks ([Wasserman, 1994]; [Victor et al., 2012]), the size of these networks has been growing in recent years, so the graph processing becomes more and more important and many graph processing systems have been introduce recent years ([Malewicz et al., 2010]; [Sagharichian et al., 2015]; [Yan et al., 2014]). Since networks have a variety of applications in different areas such as economic, sociology, etc., extracting the characteristics of these networks has been an important research field [Newman, 2003]. One of the most important and basic feature of these networks is the diameter. Diameter of the graph is the longest shortest path between all vertices in the graph. Finding diameter of graphs is one of the fundamental issues in the graph theory that has many applications. For example, in the social networks, diameter of the graph indicates that how fast the information will be propagated in the network in the worst-case [Kumar et al., 2004]. Diameter is an important metric for the evolution of structure within social networks [Kumar et al., 2010] and analysing the user interactions for different applications [Wilson et al., 2009] like spam email mitigation [Garriss et al., 2006] or Sybil attacks [Yu et al., 2006]. In an internet routing network, the diameter represents the maximum response time between any two machines in the network [Kumar et al., 2004].

One way to calculate the diameter is All Pairs Shortest Path (APSP) algorithm, which is based on applying Breadth-First searches (BFS) on all vertices and selecting the longest path as the diameter. This algorithm has the complexity of $O(n^3)$ for weighted graphs and $O(mn)$ for unweighted ones. The main problem of this algorithm is its large number of BFS searches. Many other algorithms have been introduced based on this algorithm which aimed at reducing the number of BFS searches to find the diameter ([Fujiwara et al., 2011]; [Crescenzi et al., 2010]; [Crescenzi et al., 2013]; [Takes and Kosters, 2011]).

In this paper, we propose an algorithm named $iDiameter$ to determine the diameter of the real world large-scale graphs. The main goal of this algorithm is to calculate a better upper bound for the maximum distance of all vertices to reduce the number of searches in APSP. The proposed algorithm has been tested over certain real world graphs and the results show that $iDiameter$ reduces the number of BFSes needed to determine the diameter more than 80 percent than the well-known existing algorithms in most cases while improves the time by almost 30 percent in average.

The rest of the paper is structured as follows. In section 2, the necessary definitions will be introduced. In section 3, the related works will be presented and in section 4 we will outline our proposed algorithm to determine the diameter of real world large-scale graphs. In section 5, an experimental evaluation of the proposed algorithm will be presented and finally section 6 contains the conclusions and future works.

2 Preliminaries

A graph will be shown as $G = (V, E)$ that V is the set of all vertices, which $|V| = n$ and $E \subseteq V \times V$ contains all edges of G, which $|E| = m$. The shortest path between two vertices u and v is denoted by $d\ (u,\ v)$. The value of eccentricity for each vertex u of the graph is represented by $ecc(u)$, which is obtained by $max_{v \in V} d(u, v)$ as the longest shortest path from vertex u. The diameter of the graph is denoted by D, which is obtained by $max_{u \in V} ecc(u)$, which is the longest shortest path between all pairs of vertices of the graph [Takes and Kosters, 2011].

3 Related works

There are a number of research works have been developed to determine the diameter of graphs. We can categorize these works based on three aspects. The first categorization is based on the accuracy of finding the diameter, which can be exact and approximate. This article will focus on calculating the exact diameter. However, a lot of approximate methods have been introduced which obtain the diameter of the graph with a certain approximation ratio ([Roditty and Vassilevska Williams, 2013]; [Chechik et al., 2014]; [Zwick, 2001]). The second categorization is based on calculating the diameter for directed or undirected graphs. In this paper, we focus on finding diameter of undirected graphs, but for directed graphs we can refer the readers to ([Crescenzi et al., 2012]; [Borassi et al., 2015]). The last category is based on determining the diameter of static or dynamic graphs. In this paper, we focus on finding diameter of static graphs. There are several methods that determine the diameter of static graphs. One of these methods is matrix multiplication, which has the time complexity of $O(n^{2.376})$ and requires a lot of space, which is not suitable for large graphs [Yuster, 2010]. Another algorithm is $APSP$ that applies many BFSes starting from all vertices to obtain the longest shortest path as diameter, which is not feasible for large graphs. There are some algorithms which try to minimize the number of BFSes in APSP by pruning vertices that cannot effect on diameter calculation. One of these algorithms is $BoundingDiameters$ [Takes and Kosters, 2011]. This algorithm determines upper and lower bounds for diameter and eccentricity of all vertices. At each stage, it selects a vertex (u) with the maximum upper bound or the minimum lower bound and perform BFS on u. Then, it tries to limit diameter boundaries with $ecc\ (u)$ and $2 * ecc\ (u)$. Besides, boundaries of the eccentricity of all vertices will be updated by selecting the minimum value between the previous upper bound and $ecc\ (u) + d(\ u, v)$ as new upper bound and selecting the maximum value between the previous lower bound and $max(\ ecc\ (u) - d\ (u, v), d(u, v))$ as new lower bound. Vertices that their eccentricity upper bound is less than the diameter lower bound and their eccentricity lower bound is greater than half of the diameter upper bound will be pruned after this stage. This algorithm will be completed when lower and upper bounds of the diameter become equal or there are no vertices remained to perform BFS. It acts like APSP in the worst-case scenario. The author extend this algorithm to compute the eccentricity distribution of graphs. We refer the readers to [Takes and Kosters, 2013] for details.

Fujiwara [Fujiwara et al., 2011] introduced an algorithm named $filtering$, which uses an upper bound for eccentricity of all vertices. Along each stage it selects a vertex (u) with the highest degree and computes $ecc(u)$. Then, the upper bound of all vertices will be updated with $ecc(u) + d(u, v)$. Vertices that their eccentricity upper bound is less than the diameter will be pruned. This algorithm completes when either all vertices have pruned or there are no vertices remained from which already a BFS was started. In practice, this method has lower performance than $Bounding\ Diameters$.

Another algorithm to find the diameter of static graphs is $fringe$ [Crescenzi et al., 2010]. This algorithm also determines lower and upper bounds for the diameter, and continues until these boundaries become equal. It uses $double - sweep$ ([Corneil et al., 2001]; [Magnien et al., 2009]) to determine a lower bound and then by exploiting the $fringe$ algorithm, acquires the upper bound such that it is equal to the lower bound. In addition to the lower bound, $double - sweep$ method returns a vertex as a centre of the graph, which is the starting point of $fringe$. The biggest problem of this algorithm is that if this vertex is not chosen correctly, the algorithm may fail to find the answer. In this case, we should use another algorithm to determine the diameter.

$fringe$ has been improved in $iFUB$ [Crescenzi et al., 2013]. This algorithm uses $4 - sweep$ ([Corneil et al., 2001]; [Magnien et al., 2009]) instead of $double - sweep$ to find the center and lower bound of the diameter of the graph. In this method, $fringe$ has been implemented iteratively, so it acts like APSP in the worst-case scenario. But still start point selection influences the running time, seriously.

Our proposed algorithm for diameter calculation of static graphs is based on finding a better upper bound for the eccentricity of the vertices and pruning unlikely vertices that are not influential in diameter determination. The upper bound used in $BoundingDiameters$ and $filtering$ methods is $ecc\,(u) + d\,(u,v)$, but we try to find a better upper bound that leads us to better pruning and faster diameter detection.

4 The Proposed Algorithm

In this section, we introduce an algorithm to obtain the diameter of real world large-scale static graphs. As mentioned before, the APSP algorithm is considered the foundation of numerous graph diameter identification algorithms. In this algorithm, we should perform BFS from all vertices to find the diameter. The proposed algorithm, $iDiameter$, applies BFS from a few vertices, instead of all vertices to obtain the diameter. In this algorithm, vertices which their eccentricity will not certainly become greater than the diameter will be pruned after each BFS.

4.1 Diameter of static graphs

The main problem of APSP is the large number of BFSes required to obtain the diameter. Therefore, the aim of the proposed method is to reduce the number of BFSes required to calculate the diameter of the graph. What previous works have done to this aim is to calculate an upper bound for eccentricity of all vertices ($\widehat{ecc}$) and use it to prune unlikely vertices. For example, by selecting a vertex like v_x and computing ecc_{v_x}, we can obtain $\widehat{ecc}$ for all vertices using Equation 1.

$$\forall\, v \in V \;\rightarrow\; \widehat{ecc}_v = ecc_{v_x} + d(v_x, v)$$

Equation 1: Upper bound for eccentricity

By using this upper bound, we can prune vertices that their upper bound is less than or equal to the current diameter ($\widehat{ecc}_v \leq d$) (since $ecc_v \leq \widehat{ecc}_v$, and it can be concluded that $ecc_v \leq d$). It means that if we perform BFS from these vertices, the maximum height of the resulting tree will be d. But the Equation 1 does not have good precision in obtaining the upper bound and therefore less pruning may be done in the graph.

For example, consider the graph of Figure 1. If we perform BFS from vertex 2, Figure 2(a) will be produced, where $v_x = 2$ and $ecc_{v_x} = 5$. In this figure, along with the vertex number, there is another number that represents $\widehat{ecc}$ which is obtained by Equation 1. According to this figure, there is no vertex which can be pruned after this step because $\widehat{ecc}$ of all vertices is greater than 5.

Now, if we perform the second BFS from vertex 13, Figure 2(b) will be produced. If the new $\widehat{ecc}$ is smaller than the previous one, calculated with Equation 1, $\widehat{ecc}$ will be updated. The height of this tree is 7. Therefore, vertices that their $\widehat{ecc}$

are smaller than 7 will be pruned. As this example clearly illustrates, there are 4 vertices have not pruned yet. So, one more BFS is needed to prune them.

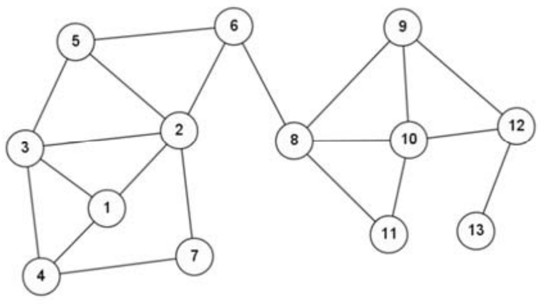

Figure 1: A sample graph with 13 vertices and 20 edges

Our main idea is to obtain $\widehat{ecc}$ closer to its real value as much as possible. This will result in more pruning, reducing number of BFSes, and reducing the running time. For this purpose, we should find the maximum distance of all vertices in the BFS tree. To do so, we introduce an algorithm that finds the maximum distance of all vertices in an accurate and fast manner.

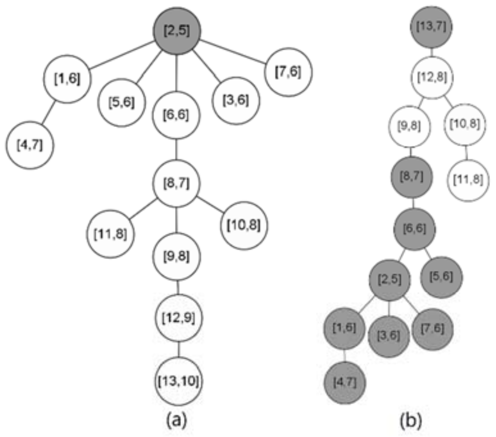

Figure 2: $\widehat{ecc}$ are calculated by Equation 1. (a) BFS from vertex 2 (b) BFS from vertex 13

If we ignore the root (v_r) of the BFS tree (T) and its edges, we would have k sub-trees where v_r has k children. Each sub-tree is named as r_i and the sub-tree with the maximum height is called r_{max}. The maximum height of T is h_t^1, that is certainly located in r_{max} and the second height of T is h_t^2, which should not belong to r_{max}. For all vertices that are not in r_{max}, the value of $\widehat{ecc}$ will be calculated with Equation 1.

But for vertices which are in r_{max}, this equation does not always hold. To calculate $\widehat{ecc}$ of any vertex (v_x), we use the following method.

- Case 1: if v_x is not in r_{max}, $\widehat{ecc}$ is obtained from Equation 1. As an example, consider vertex 4 in Figure 3(a), which it's $\widehat{ecc}$ is 7 from vertex 13.

If v_x belongs to r_{max}, we should investigate the following three cases and take the maximum of them as $\widehat{ecc}$.

- Case 2.1: $\widehat{ecc}$ of v_x is the distance to the farthest leaf in r_{max}, like. For example vertex 6, which it's $\widehat{ecc}$ is 4 in Figure 3(b).
- Case 2.2: $\widehat{ecc}$ of v_x is the sum of distances to the root, and h_t^2, e.g. Vertex 11 in Figure 3(c), which it's $\widehat{ecc}$ is 5 from vertex 4.
- Case 2.3: starting from v_x and moving towards the root. If any of its ancestors (v_a) has more than one child, we check these two conditions for them:
- Case 2.3.1: $\widehat{ecc}$ of v_x is the sum of the distance to v_a ($d(v_r, v_x) - d(v_r, v_a)$) and the height of the highest sub-tree of v_a (if v_x does not belong to the highest sub-tree of v_a). As an example, consider vertex 11 in Figure 3(d), which it's $\widehat{ecc}$ is 8. Vertex 11 does not belong to the highest sub-tree of 12.
- Case 2.3.2: $\widehat{ecc}$ of v_x is the sum of the distance to v_a ($d(v_r, v_x) - d(v_r, v_a)$) and the height of the second highest sub-tree of v_a (if v_x belongs to the highest sub-tree of v_a). For instance vertex 5 in Figure 3(e), which it's $\widehat{ecc}$ is 6. Vertex 11 belongs to the highest sub-tree of 12.

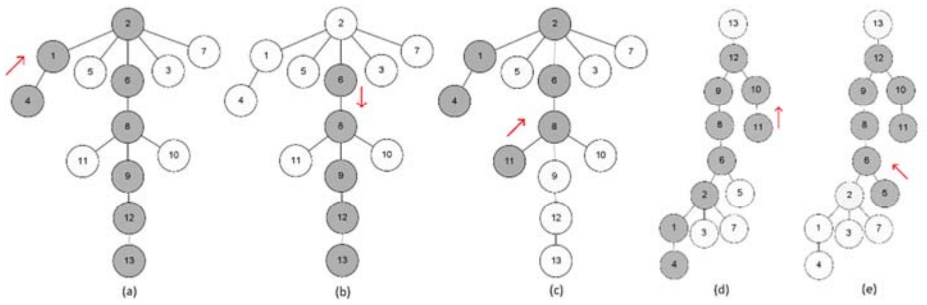

Figure 3: Five Different cases to obtain the maximum distance of a vertex in a BFS tree

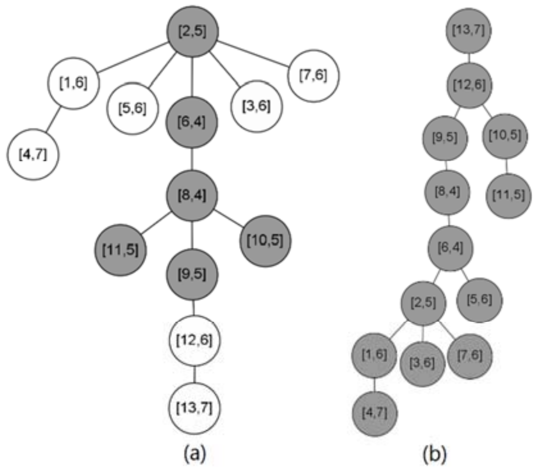

Figure 4: $\widehat{ecc}$'s are calculated by proposed procedure. (a) BFS from vertex 2 (b) BFS from vertex 13

To check whether v_x belongs to the highest sub-tree of v_a or not, we only have to compare the sum of distances from v_x to v_a, and the height of the maximum sub-tree of v_x with the height of the maximum sub-tree of v_a. If these two values became equal, v_x belongs to the highest sub-tree of v_a, otherwise it does not.

By using this procedure, we can determine $\widehat{ecc}$ of all vertices in the BFS tree. Figure 4 will be achieved if we compute $\widehat{ecc}$ for the previous example based on this procedure. As it can be seen from Figure 4(a), after the first BFS from vertex 2, $\widehat{ecc}$ of vertices 6, 8, 9, 10, 11 will be smaller or equal to 5. This means that these vertices can be pruned after this step. If we perform BFS from vertex 13, Figure 4(a) will be achieved. Since the height of the tree is 7 and the $\widehat{ecc}$ of all vertices is smaller or equal to 7, all vertices can be pruned and the diameter of the graph will be 7. By using this procedure, we can find the diameter of this graph with only 2 BFSes.

4.2 Implementation

We check 4 aforementioned cases for all vertices in our implementation. To check cases 2.3.1 and 2.3.2, we have to determine the highest and the second highest height of sub-tree of all vertices located in r_{max}. To this end, we consider a special data structure for the BFS tree. For each vertex in the tree, we use the following attributes:

- $Parent$: Pointer to the father.
- $NumberOfChildren$: number of children.
- $FirstSH$: height of the highest sub-tree.
- $SecondSH$: height of the second highest sub-tree.
- $CheckedBranch$: Number of checked branches of each vertex which is initially 0.
- $GrandParent$: Ancestor of each vertex in the first level of T. Each vertex inherits this value from its father.

The values of $Parent$, $Grandparent$ and $NumberOfChildren$ of all vertices will be computed after each BFS. Also, the vertex in the first level of T, which its sub-tree is the highest one among the others will be specified as v_{mh}. Then, we have to find $FirstSH$ and $SecondSH$ of all vertices in r_{max}. For this purpose, we introduce the $AttributingTree$ algorithm. This algorithm starts from the leaves of r_{max} and moves upward to the root, and checks every branch of all middle vertices only once. The input of this algorithm is the BFS tree and the leaves of the sub-tree which its vertices' $Grandparent$ is v_{mh} (located in r_{max}). The output of this algorithm is an $Attributedtree$, which values of $FirstSh$ and $SecondSH$ of all vertices of r_{max} are specified.

Algorithm1: AttributingTree

```
 1: **Input :** $T$ , $a\ bfs\ tree\ with\ root\ v_r$
 2:          $Leaves$ , $Leaves\ of\ r_{max}$
 3: **Output :** $T$ , $Attributed\ Tree$
 4: **foreach** $node$ **in** $T$ **do**
 5:     $node.CheckedBranch \leftarrow 0$;
 6:     $node.FirstSH \leftarrow 0$;   $node.SecondSH \leftarrow 0$;
 7: **end for**
 8: **foreach** $node$ **in** $Leaves$ **do**
 9:     $Queue.Add(node)$;
10: **end for**
11: **while** $!\,Queue.IsEmpty()$ **do**
12:     $node \leftarrow Queue.Remove()$;
13:     $distance \leftarrow node.FirstSH$;
14:     $node \leftarrow node.Parent$;
15:     $distance++$;
16:     **while** $node.NumberOfChildren = 1$ **and** $node\ != v_r$ **do**
17:         $node.FirtsSH \leftarrow distance$;
18:         $node \leftarrow node.parent$;
19:         $distance++$;
20:     **end while**
21:     **if** $node\ != v_r$ **then**
22:         **if** $node.FirstSH <= distance$ **then**
23:             $node.SecondSH \leftarrow node.FirstSH$;
24:             $node.FirstSH \leftarrow distance$;
25:         **else if** $node.SecondSH < distance$ **then**
26:             $node.SecondSH \leftarrow distance$
27:         **end if**
28:         $node.CheckedBranch++$;
29:         **if** $node.CheckedBranch = node.NumberOfChildren$ **then**
30:             $Queue.add(node)$;
31:         **end if**
32:     **end if**
33: **end while**
```

The complexity of this algorithm in the worst case (the root has only one child) is $O(n)$. After running this algorithm, we can calculate $\widehat{ecc}$ of all vertices based on the $Attributedtree$.

We introduce the $FindEccUpperBound$ algorithm to find $\widehat{ecc}$ of all vertices in the BFS tree. This algorithm checks 4 cases are introduced above to find $\widehat{ecc}$. The input of this algorithm is the $Attributedtree$ along with v_r, v_{mh}, h_t^1, h_t^2, and v_x, the vertex we want to find its $\widehat{ecc}$. The algorithm returns MD_x as $\widehat{ecc}$ of v_x.

If v_x does not belong to r_{max}, the value of MD_x will be simply calculated by Equation 1(lines 8-9). Otherwise, cases 2.1, 2.2, 2.3.1 and 2.3.2 will be checked. Cases 2.1 and 2.2 are checked in line 11. Then, cases 2.3.1 and 2.3.2 are checked for each vertex with more than one child as moving upward to the root. This operation continues until it reaches the root. Finally, the $\widehat{ecc}$ for the given vertex can be specified. The complexity of the algorithm for each vertex at level h of r_{max} is $O(h)$ and for all vertices that are not located in r_{max}, is O (1). The complexity of this algorithm in the worst case is O (nd) where d is the height of the tree.

We can prune vertices that cannot affect the diameter by calculating $\widehat{ecc}$ of vertices in the BFS tree. To this end, an algorithm called $iDiameter$ has been introduced. The input of this algorithm is a graph. It returns the diameter.

Algorithm2: FindEccUpperBound

```
1:  Input : $T$ , $an\ Attributed\ tree\ with\ root\ v_r$
2:          $v_{mh}$ , $root\ of\ r_{max}$
3:          $h_t^1$ , $height\ of\ the\ tree$
4:          $h_t^2$ , $second\ height\ of\ the\ tree$
5:          $v_x$ , $node\ that\ we\ want\ to\ compute\ it's\ maximum\ distance$
6:  Output : $MD_x$ , $Maximum\ Distance\ of\ node\ v_x\ in\ T$
7:  $step \leftarrow 0$;   $MD_x \leftarrow 0$;   $distance \leftarrow 0$;
8:  if $v_x.GrandParent\ != v_{mh}$ then
9:      return $d(v_r, v_x) + h_t^1$ ;
10: else
11:     $MDx = max(v_x.FirstSH, d(v_r, v_x) + h_t^2)$;
12:     $distance = v_x.FirstSH$;
13:     while $v_x.Parent\ != v_r$ do
14:         $v_x = v_x.Parent$;
15:         $step++$;   $distance++$;
16:         If $(v_x.FirstSH > distance)$ then
17:             $MD_x = max(MD_x, v_x.FirstSH + step)$;
18:             $distance = v_x.FirstSH$;
19:         else
20:             $MD_x = max(MD_x, v_x.SecondSH + step)$;
21:         end if
22:     end while
23: end if
24: Retrun $MD_x$
```

This algorithm creates a set V', which is initially equal to V. At each step, it performs BFS by calling the $ComputeBFS$ function from a vertex, which is selected by the $NextNode$ function. If the height of the resulting BFS tree became larger than the current diameter, the diameter should be updated. Then, it calls the $Attributingtree$ function on the BFS tree to produce the $Attributedtree$. Then, it finds $\widehat{ecc}$ for all vertices in V' by calling the $FindEccUpperBound$ function. The algorithm compares $\widehat{ecc}$ of all vertices with the current diameter and prunes vertices that their $\widehat{ecc}$ are smaller than or equal to the current diameter. The algorithm completes when V' becomes empty. Otherwise, it selects another vertex to perform BFS.

4.3 Complexity

Performing BFS on a graph is the most time consuming part of this algorithm. BFS takes $O(m)$ time to complete. For each BFS, we should run $AttributingTree$ that takes $O\ (n)$ time in the worst case scenario. To calculate $\widehat{ecc}$ of each vertex, we have to run $FindEccUpperBound$ that takes $O\ (nd)$ time in the worst case for all vertices. As a result, the complexity of each BFS is $O\left(m + n + nd\right)$.

Algorithm3 : iDiameter

```
 1: Input : $Graph\ G(V,E)$
 2: Output : $Diameter\ of\ G$
 3: $D \leftarrow 0;\ \ V' \leftarrow V;$
 4: foreach $node$ in $V'$ do
 5:       $node.\widehat{ecc} \leftarrow +\infty;$
 6: end for
 7: while $V' \neq \emptyset$ do
 8:       $v_x \leftarrow NextNode(V');$
 9:       $V' \leftarrow V' - \{v_x\};$
10:       $\{T, v_{mh}, h_t^1, h_t^2, leaves\} \leftarrow ComputeBFS(G, v_x);$
11:       if $h_t^1 > D$ then
12:             $D \leftarrow h_t^1;$
13:       end if
14:       $AttributedTree \leftarrow AttributingTree(T, leaves);$
15:       foreach node in $V'$ do
16:             $MD_{node} \leftarrow FindEccUpperBound(AttributedTree, v_{mh}, h_t^1, h_t^2, node);$
17:             if $MD_{node} < node.\widehat{ecc}$ then
18:                   $node.\widehat{ecc} \leftarrow MD_{node};$
19:             end if
20:             if $node.\widehat{ecc} \leq D$ then
21:                   $V' \leftarrow V' - \{node\};$
22:             end if
23:       end for
24: end while
25: return $D$
```

4.4 Vertex selection strategy

To select a vertex, we can consider a variety of factors. Some of them like degree of vertices and their $\widehat{ecc}$ have been studied in [Takes and Kosters, 2011]. By selecting higher degree vertices, we probably cannot reach the diameter, but the chance of pruning will increase. The reason is that $\widehat{ecc}$ of vertices becomes closer to its real value in this case. On the other hand, choosing vertices with smaller degree causes larger-height BFS trees will be produced. Therefore, this may lead to reach the diameter faster. In this case, the chance of pruning will be less than before. Another factor that we have considered in this paper is selecting vertices based on their $\widehat{ecc}$. Like selecting vertices with higher degrees, if we select a vertex with smaller $\widehat{ecc}$, it usually results in more pruning, but smaller height than the diameter. In this case, we select vertices with larger degrees for breaking ties. In contrast, by selecting vertices with large $\widehat{ecc}$, we usually reach the diameter with less pruning. In this case, for breaking ties we select a vertex with smaller degree.

All subsequent breaks ties by selecting random vertex. The best way, which we used in this paper, is to select a vertex in an interchanging manner. In other words, we first select a vertex to reach the diameter and then select a vertex to prune more. Since $\widehat{ecc}$ of all vertices is $+\infty$ initially, we select a vertex with the largest degree.

5 Experimental results

We performed experiments on a number of existing real world graphs. Our method is implemented using JAVA and all experiments are tested on a standalone machine with 3.1 GHz CPU and 8GB of RAM.

#	Name	#Nodes	#Edges	#D	#CC	Type
1	ca-GrQc	5,242	14,496	17	355	Collaboration networks
2	p2p-Gnutella08	6,301	20,777	9	2	peer-to-peer networks
3	Wiki-Vote	7,115	103,689	7	24	Wikipedia networks
4	p2p-Gnutella09	8,114	26,013	10	6	peer-to-peer networks
5	p2p-Gnutella06	8,717	31,525	10	1	peer-to-peer networks
6	bcsstk33	8,728	291,583	25	1	Meshes & electronic circuits
7	p2p-Gnutella05	8,846	31,839	9	3	peer-to-peer networks
8	ca-HepTh	9,877	25,998	18	429	Collaboration networks

Table 1: Test graphs

#	Name	#Nodes	#Edges	#D	#CC	Type
9	p2p-Gnutella04	10,876	39,994	10	1	peer-to-peer networks
10	CA-HepPh	12,008	118,521	13	278	Collaboration networks
11	p2p-Gnutella25	22,687	54,705	11	13	peer-to-peer networks
12	p2p-Gnutella24	26,518	65,369	11	11	peer-to-peer networks
13	Cit-HepTh	27,770	352,807	15	143	Citation networks
14	p2p-Gnutella30	36,682	88,328	11	12	peer-to-peer networks
15	Email-Enron	36,692	183,831	13	1,065	Communication networks
16	p2p-Gnutella31	62,586	147,892	11	12	peer-to-peer networks
17	brack2	62,631	366,559	73	2	Meshes & electronic circuits
18	soc-Slashdot0902	82,168	948,464	13	1	Social networks
19	amazon0302	262,111	1,234,877	32	1	Product co-purchasing networks
20	web-Stanford	281,903	2,312,497	753	365	Web graphs
21	amazon0601	403,364	3,387,388	21	1	Product co-purchasing networks
22	web-BerkStan	685,230	7,600,595	714	676	Web graphs
23	roadNet-PA	1,087,562	1,541,898	786	206	Road networks
24	roadNet-TX	1,379,917	1,921,660	1,054	424	Road networks
25	cit-patents	3,774,768	16,518,948	22	3,627	Citation networks
26	com-lj	3,997,962	34,681,189	17	1	Networks with communities
27	soc-livejournal	4,847,571	68,993,773	16	1,876	Social networks

Table 1 (continued): Test graphs

We select graphs from a variety of applications and sizes. Graphs which are experimented in this paper include social networks, communication networks, citation networks, collaboration networks, web graphs, p2p networks, road networks and signed networks. All graphs are available in ([Jure and Krevl, 2014]; [Walshaw, 2015]) and considered undirected in this study. They are listed in Table 1 along with their main properties such as number of vertices, number of edges, the diameter and number of connected components.

#	Name	BFS		Time (ms)	
		iDiameter	Bounding Diameters	iDiameter	Bounding Diameters
1	ca-GrQc	**13.9**	15.3	14.1	**8.7**
2	p2p-Gnutella08	**204.4**	1,076.9	**253.5**	483.8
3	Wiki-Vote	**7.6**	13.8	**17.8**	23.0
4	p2p-Gnutella09	**49.8**	290.3	**237.4**	338.4
5	p2p-Gnutella06	**29.3**	29.9	51.1	**30.8**
6	bcsstk33	**3.0**	229.3	**208.0**	760.6
7	p2p-Gnutella05	**206.6**	1,076.9	**390.2**	1,174.2
8	ca-HepTh	**10.0**	17.0	58.6	**26.0**
9	p2p-Gnutella04	**42.9**	325.3	**105.9**	465.7
10	CA-HepPh	**15.0**	20.0	72.0	**54.9**
11	p2p-Gnutella25	**89.1**	758.3	**443.8**	2,361.7
12	p2p-Gnutella24	10.6	**9.4**	68.2	**43.0**
13	Cit-HepTh	**6.0**	**6.0**	337.4	**91.1**
14	p2p-Gnutella30	**189.3**	2,290.7	**5,741.6**	20,800.7
15	Email-Enron	**8.0**	11.0	**65.5**	68.9
16	p2p-Gnutella31	**671.3**	7,149.8	**25,166.8**	108,442.6
17	brack2	**3.0**	42.0	**350.1**	498.4
18	soc-Slashdot0902	**5.0**	**5.0**	115.0	**95.5**
19	amazon0312	**11.0**	21.0	**1,972.4**	2,888.4
20	web-Stanford	**4.8**	5.6	522.3	**454.2**
21	amazon0601	**26.0**	28.0	4,700.3	**3,845.4**
22	web-BerkStan	**2.0**	5.0	**295.7**	587.5
23	roadNet-PA	**36.3**	60.2	14,893.7	**10,071.7**
24	roadNet-TX	**44.1**	75.4	20,255.4	**15,484.0**
25	cit-patents	**39.7**	67.7	**105,198.3**	133,083.1
26	com-lj	**8.0**	9.0	**21,605.1**	21,812.8
27	soc-livejournal	**6.2**	7.8	**20,206.2**	24,011.2

Table 2: Diameter of static graphs

The format of the raw data set is edge file, which each line in the file represents an edge between two vertices. Each graph has its own properties, for example road networks have large diameter while social networks follow power law and usually have small diameter.

We compared $iDiameter$ to the best existing algorithm, $BoundingDiameters$ [Takes and Kosters, 2011] which is based on breadth-first search and pruning unlikely vertices. Because of selecting random vertices for breaking ties, there was a large deviation in the number of iterations required to find the diameter in some graphs. For example, in $bcsstk33$, the minimum number of iterations is 8 for $BoundingDiameters$, but the average iteration of running 100 times is 230. So, we evaluate each algorithm up to 100 times (not less than 20 times) and report the average of results. The results of Table 2 show that our algorithm can find the diameter with less BFS than $BoundingDiameters$. $iDiamter$ wecan find more accurate $\widehat{ecc}$ for all vertices than the other algorithm. This improved accuracy imposes additional costs, but we claimed that by using less BFSes, this extra time is negligible. To affirm this claim, we report the average running time of these two algorithms in Table 2, which shows that our algorithm finds the diameter faster than the other one in almost 60 percent of tested graphs.

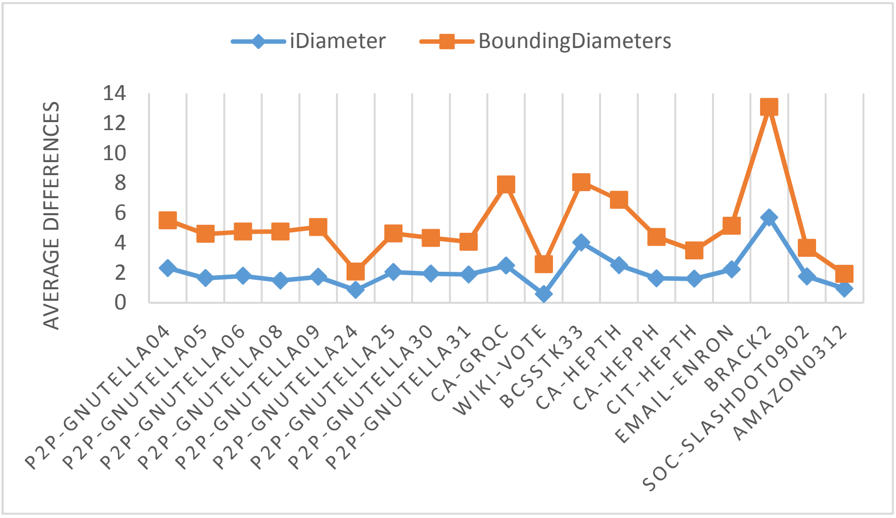

Figure 5: Average differences of $\widehat{ecc}$ and ecc

Another interesting result is the average of differences between $\widehat{ecc}$ and its actual value. We calculate the average of differences between $\widehat{ecc}$ and ecc in two algorithms which are reported in Figure 5. Since our method is more accurate than the state of the art methods, the average of differences is less than the other one in almost all tested graphs. For example, in graph $p2p - Gnutella04$, despite the fact that the number of BFSes in $iDiameter$ and $BoundingDiameters$ is 42 and 325, respectively, the average of differences between $\widehat{ecc}$ and ecc in our algorithm is about 2 while it is 6 in $BoundingDiameters$. This means that our algorithm can find better upper bound for the eccentricity of the vertices even with less BFSes.

6 Conclusion and future work

In this paper, $iDiameter$ is proposed in order to calculate the diameter of static graphs based on APSP approach. One of the fundamental problems of APSP is that it requires a large number of BFSes starting from different vertices. There are some extensions of this method using estimation of eccentricity idea to overcome this shortcoming. Based on this idea, BFS trees starting from vertices that their eccentricities are smaller than the current diameter have no effects on calculation of the diameter. Therefore, the more accurate estimation of eccentricity causes more pruning of vertices. As a result, diameter will be calculated with lesser number of BFSes. The main goal of the proposed method is to increase the accuracy of eccentricity estimations. Evaluation of the proposed method with the current well-known algorithm showed that $iDiameter$ can reduce estimation error by more than 50% in average. These improvements result more than 80% reduction of required BFSes to calculate the diameter of graphs in most cases. Another contribution of this paper is proposing a sophisticated vertex selection strategy for determining the starting point of each BFS. This strategy simultaneously increases the speed of diameter calculation as well as the number of pruned vertices. Performance evaluations showed that $iDiameter$ can reduce the running time more than 30% in average. It must be noticed that there are some cases where the amount of reductions are more than 70%.

The proposed algorithm can only calculate the diameter of static graphs, but the real world graphs are changing rapidly over the time. As our future works, we want to investigate whether the proposed algorithm can be extended to dynamic graphs or not. Moreover, we will consider developing a distributed version of $iDiameter$ to face with large-scale graphs.

References

[Bawa et al., 2003] Bawa, M., Cooper, B. F., Crespo, A., Daswani, N., Ganesan, P., Garcia-Molina, H., Kamvar, S., Marti, S., Schlosser, M. and Sun, Q.: “Peer-to-peer research at Stanford,” ACM SIGMOD Record, 32, 3 (2003), 23-28.

[Borassi et al., 2015] Borassi, M., Crescenzi, P., Michel, H., Kosters, W. A., Marino, A. and Takes, F. W.: “Fast diameter and radius BFS-based computation in (weakly connected) real-world graphs: With an application to the six degrees of separation games,” Theoretical Computer Science, 586 (2015), 59-80.

[Chechik et al., 2014] Chechik, S., Larkin, D. H., Roditty, L., Schoenebeck, G., Tarjan, R. E. and Williams, V.: “Better approximation algorithms for the graph diameter,” in Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SIAM (2014), 1041-1052.

[Corneil et al., 2001] Corneil, D. G., Dragan, F. F., Habib, M. and Paul, C.: “Diameter determination on restricted graph families,” Discrete Applied Mathematics, 113, 2 (2001), 143-166.

[Crescenzi et al., 2010] Crescenzi, P., Grossi, R., Imbrenda, C., Lanzi, L. and Marino, A.: “Finding the diameter in real-world graphs,” in Algorithms--ESA 2010, Springer (2010), 302-313.

[Crescenzi et al., 2012] Crescenzi, P., Grossi, R., Lanzi, L. and Marino, A.: “On computing the diameter of real-world directed (weighted) graphs,” Experimental Algorithms, (2012), 99-110.

[Crescenzi et al., 2013] Crescenzi, P., Grossi, R., Habib, M., Lanzi, L. and Marino, A.: “On computing the diameter of real-world undirected graphs,” Theoretical Computer Science, 514 (2013), 84-95.

[Del Mondo et al., 2010] Del Mondo, G., Stell, J. G., Claramunt, C. and Thibaud, R.: “A Graph Model for Spatio-temporal Evolution,” Journal of Universal Computer Science, 16, 11 (2010), 1452-1477.

[Fujiwara et al., 2011] Fujiwara, Y., Onizuka M. and Kitsuregawa, M.: “Real-time diameter monitoring for time-evolving graphs,” in Database Systems for Advanced Applications, Springer (2011), 311-325.

[Garriss et al., 2006] Garriss, S., Kaminsky, M. , Freedman, M. J., Karp, B., Mazieres, D. and Yu, H.: “RE: Reliable Email.,” in NSDI (2006), 22-22.

[Jure and Krevl, 2014] Jure, L. and Krevl, A.: “SNAP Datasets: Stanford Large Network Dataset Collection,” (2014), [Online]. Available: http://snap.stanford.edu/data. [Accessed 2015].

[Kumar et al., 2004] Kumar, A., Merugu, S., Xu, J. J., Zegura, E. W. and Yu, X.: “Ulysses: a robust, low-diameter, low-latency peer-to-peer network,” European transactions on telecommunications, 15, 6 (2004), 571-587.

[Kumar et al., 2010] Kumar, R., Novak, J. and Tomkins, A.: “Structure and evolution of online social networks,” in Link mining: models, algorithms, and applications, Springer (2010), 337-357.

[Magnien et al., 2009] Magnien, C., Latapy, M. and Habib, M.: “Fast computation of empirically tight bounds for the diameter of massive graphs,” Journal of Experimental Algorithmics (JEA), 13 (2009), 10.

[Malewicz et al., 2010] Malewicz, G., Austern, M. H., Bik, A. J. C., Dehnert, J. C., Horn, I., Leiser, N. and Czajkowski, G.: “Pregel: A System for Large-Scale Graph Processing,” in Proceedings of the 2010 ACM SIGMOD International Conference on Management of data, ACM (2010), 135-146.

[Newman, 2003] Newman, M. E.: “The structure and function of complex networks,” SIAM review, 45, 2 (2003), 167-256.

[Reka et al., 1999] Reka, A., Hawoong, J. and Albert-Laszlo, B.: “Internet: Diameter of the world-wide web,” Nature, 401, 6749 (1999), 130-131.

[Roditty and Vassilevska Williams, 2013] Roditty, L. and Vassilevska Williams, V.: “Fast approximation algorithms for the diameter and radius of sparse graphs,” in Proceedings of the forty-fifth annual ACM symposium on Theory of computing, ACM (2013), 515-524.

[Sagharichian et al., 2015] Sagharichian, M., Naderi, H., and Haghjoo, M.: “ExPregel: a new computational model for large-scale graph processing,” Concurrency and Computation: Practice and Experience, 27, 17 (2015), 4954-4969.

[Shudong and Azer, 2006] Shudong, J. and Azer, B.: “Small-world characteristics of internet topologies and implications on multicast scaling,” Computer Networks, 50, 5 (2006), 648-666.

[Takes and Kosters, 2011] Takes, F. W. and Kosters, W. A.: “Determining the diameter of small world networks,” in Proceedings of the 20th ACM international conference on Information and knowledge management, ACM (2011), 1191-1196.

[Takes and Kosters, 2013] Takes, F. W. and Kosters, W. A.: “Computing the eccentricity distribution of large graphs,” Algorithms, 6, 1 (2013), 100-118.

[Victor et al., 2012] Victor, S., Zimbrão, G. and Souza, J. M.: “Modeling, Mining and Analysis of Multi-Relational Scientific Social Network,” Journal of Universal Computer Science, 18, 8 (2012), 1048-1068.

[Walshaw, 2015] Walshaw, C.: “The university of greenwich gaph partitioning archive,” (2000), [Online]. Available: http://staffweb.cms.gre.ac.uk/~c.walshaw/partition/. [Accessed 2015].

[Wasserman, 1994] Wasserman, S.: “Social network analysis Methods and applications”, Cambridge university press, (1994).

[Wilson et al., 2009] Wilson, C., Boe, B., Sala, A., Puttaswamy, K. P. and Zhao, B. Y.: “User interactions in social networks and their implications,” in Proceedings of the 4th ACM European conference on Computer systems, ACM (2009), 205-218.

[Yan et al., 2014] Yan, D., Cheng, J., Lu, Y. and Ng, W.: “Blogel: A block-centric framework for distributed computation on real-world graphs.” Proceedings of the VLDB Endowment, 7, 14 (2014), 1981-1992.

[Yu et al., 2006] Yu, H., Kaminsky, M., Gibbons, P. B. and Flaxman, A.: “Sybilguard: defending against sybil attacks via social networks,” ACM SIGCOMM Computer Communication Review, 36, 4 (2006), 267-278.

[Yuster, 2010] Yuster, R.: “Computing the diameter polynomially faster than APSP,” arXiv preprint arXiv:1011.6181, (2010).

[Zwick, 2001] Zwick, U.: “Exact and approximate distances in graphs—a survey,” in Algorithms—ESA 2001, Springer (2001), 33-48.