Exacting Eccentricity for Small-World Networks

Wentao Li[♮], Miao Qiao[#], Lu Qin[♮], Ying Zhang[♮], Lijun Chang[§], and Xuemin Lin[‡]

[♮]*CAI, FEIT, University of Technology Sydney, Australia* [#]*Massey University, New Zealand*
[‡]*The University of New South Wales, Australia* [§]*The University of Sydney, Australia*
[♮]wentao.li@student.uts.edu.au; {lu.qin, ying.zhang}@uts.edu.au;
[#]m.qiao@massey.ac.nz; [§]lijun.chang@sydney.edu.au; [‡]lxue@cse.unsw.edu.au;

***Abstract*—This paper studies the efficiency issue on computing the exact eccentricity-distribution of a small-world network. Eccentricity-distribution reflects the importance of each node in a graph, which is beneficial for graph analysis. Moreover, it is key to computing two fundamental graph characters, diameter and radius. Existing eccentricity computation algorithms, however, are either inefficient in handling large-scale networks emerging nowadays in practice or approximate algorithms that are inappropriate to small-world networks. We propose an efficient approach for exact eccentricity computation. Our approach is based on a plethora of insights on the bottleneck of the existing algorithms — one-node eccentricity computation and the upper/lower bounds update. Extensive experiments demonstrate that our approach outperforms the state-of-the-art up to three orders of magnitude on real large small-world networks.**

I. Introduction

Shortest distances characterize the pair-wise relationships among nodes in a graph. Given a graph with a vertex/node set and an edge set, the shortest distance $dist(u, v)$ between two nodes u, v is defined as the least length of a path from u to v. Similarly, the longest shortest distance from one node u to all the other nodes of the graph, defined as the **eccentricity** of u, constitutes two fundamental [1] characters of the entire graph — diameter and radius. Diameter is the maximum eccentricity while radius is the minimum eccentricity, over all the nodes in a graph. For big-graph analysis nowadays, the computation of a graph's diameter and radius is inevitable.

The eccentricity, apart from its usage in computing the diameter and radius, measures the centrality of a node in the graph. The eccentricity-distribution, namely, the eccentricities of all the nodes in a graph, thus helps in identifying important vertices in a graph, which could be influential people in a social network, critical nodes in an epidemic contact network, or important sites in a web graph. With all the applications listed above, an efficient approach for computing the eccentricity-distribution of a graph is highly demanded.

Unfortunately, the computation of the eccentricities is ultra-expensive, especially in this big data era. Currently, all algorithms for computing even the diameter require $\Omega(\frac{n^2}{\log n})$ time (see [2], [3] and the reference therein). Here n denotes the total number of nodes in the graph, which is 1.32 billion on Facebook[1]. The size of n causes an efficiency issue; to resolve which, a natural solution is to introduce approximation.

Indeed, algorithms like [4], [5] have been proposed to compute an estimated eccentricity $\widetilde{ecc}(v)$ instead of the exact eccentricity $ecc(v)$ to gain the computation efficiency. For example, on an undirected and unweighted graph, Chechik et al. [5] computes $\widetilde{ecc}(v)$ in $O((m \log m)^{\frac{3}{2}})$ time, where m denotes the total number of edges in the graph. More efficient approximate algorithms are still expected; however, even if a linear-timed approximate algorithm has been found, it may not be suitable to certain unweighted graphs — small-world networks, as long as an error is allowed in computing $ecc(v)$.

Small-world networks, a term first proposed by Watts and Strogatz [6], describes a group of graphs that feature a highly clustered topology and short path-length. The phenomena of short path-length has been observed much earlier in the book of "Six Degrees of Separation" [7], and has been confirmed later on recent data of biological networks, neural networks, collaboration networks, communication networks, and social networks. For example[2], Slashdot, a social network, has a diameter of 11 while wiki-talk, a communication network, has a diameter of 9. On unweighted networks, any additive positive error $\delta = \widetilde{ecc}(\cdot) - ecc(\cdot)$ will have $\delta \geq 1$. Note that $\delta = 1$ is already significant to the short radius/diameter of a small-world network: if the radius $r = 5$, then $\frac{1}{r} = 20\%$, let alone the fact that δ can hardly be bounded by 1.

For small-world networks, the state-of-the-art exact eccentricity computation (see [8] and the reference therein) follows the same paradigm, which i) associates each node with an upper and a lower bound on its eccentricity; ii) for each node v, if the upper and lower bounds of v does not meet, compute the eccentricity $ecc(v)$ using a Breadth-First-Search (BFS) and then update the bounds globally for all other nodes. The performance is, therefore, largely dependent on the node-order of v traversed in Step ii).

Our approach revises the paradigm and demonstrates a superior efficiency. Specifically, we provide a spectrum of insights to *avoid* the exhaustive BFS and global update — the bottleneck of the existing approaches. Instead, we make the most out of each computation by leveraging a myriad of techniques in lowering the gap between the upper and lower bounds. Our contributions are summarized as below.

- In the eccentricity computation of a node v, our algorithm determines $ecc(v)$ at an early-stage by visiting first from nodes that are distant to v. Furthermore, our algorithm inherits the global bounds on the eccentricity distribution which can terminate the search even earlier. In contrast, BFS uses $\Omega(n)$ time to compute $ecc(v)$ regardless how close the upper and lower bounds are.
- In updating the eccentricity bounds, we show that it suffices for our algorithm to update only a connected area of, in expectation, $O(d)$ nodes while achieving the same

[1]https://newsroom.fb.com/company-info/

[2]https://snap.stanford.edu/data/index.html.

effect as performing a global update. Here d is the graph diameter — a small integer for a small-world network.
- Empirical studies show that our approach outperforms the state-of-the-art by up to three orders of magnitude. In particular, our approach is the only one that completed the computation within 24 hours on all graphs.

The paper is organized as follows. Section II formally introduces the problem definition and the state-of-the-art approach. Section III describes our algorithm in computing the eccentricity of one node. Section IV depicts an efficient update algorithm. Section V summarizes the related work. Section VI demonstrates the experimental results while Section VII concludes the paper.

II. Preliminary

This paper focuses on the eccentricity on an unweighted and undirected graph. Let $G(V, E)$ be a graph with a set V of nodes and a set E of edges. Each edge $e(u, v)$ in E connects two nodes u, v in V. Denote $|V|$ as n, $|E|$ as m. Given two nodes s and t in V, a *path* $p(s, t)$ from s to t is a sequence of distinct nodes $\langle u_0, u_1, \cdots, u_k \rangle$ starts from $u_0 = s$ and ends at $u_k = t$ with neighboring nodes connected by edges, that is, $(u_{i-1}, u_i) \in E$, for $\forall i \in [1, k]$. The *length* $|p(s, t)|$ of a path is the total number of edges on the path. The *shortest distance* $dist(s, t)$ between s and t is the length of the shortest path from s to t. The shortest distances on V hold the *triangle inequality*, that is, for three nodes $s, u, t \in V$, $dist(s, t) \leq dist(s, u) + dist(u, t)$.

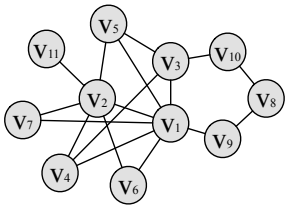

Fig. 1. An Example Graph G

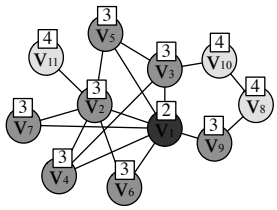

Fig. 2. Eccentricity of Nodes in G

Example 1. *Fig. 1 shows a running example of graph G with 11 nodes and 17 edges. The shortest path from v_7 to v_8 is $\langle v_7, v_1, v_9, v_8 \rangle$ with $dist(v_7, v_8) = 3$. For nodes v_7, v_8, v_3, the triangle inequality $dist(v_7, v_8) \leq dist(v_3, v_7) + dist(v_7, v_8)$ holds since $dist(v_7, v_3) = 2$ and $dist(v_3, v_8) = 2$.*

Definition 1 (Eccentricity)**.** *Given a node u of a graph $G(V, E)$, the eccentricity of u is defined as*

$$ecc(u) = \max_{v \in V} dist(u, v).$$

Example 2. *Fig. 2 labels, on the graph G of the running example, the eccentricity of each node. The color gray scale indicates, for each node, the eccentricity value: a node with a darker color means that it has a smaller eccentricity. For example, the eccentricity of node v_1 is calculated as $ecc(v_1) = \max_{v \in V} dist(v_1, v) = 2$, and $ecc(v_{11}) = \max_{v \in V} dist(v_{11}, v) = 4$. Intuitively, a node at the center has a smaller eccentricity than the node at the border.*

Trivially, if a graph is disconnected, that is, there exist two nodes u, v such that there is no path from u to v, then the eccentricities of all the nodes in V become $+\infty$. We, therefore, assume that $G(V, E)$ is connected.

Problem 1 (Eccentricity Computation)**.** *Given a connected graph $G(V, E)$, compute the eccentricity-distribution, namely, the eccentricity $ecc(u)$ for all the nodes $u \in V$.*

A. Pair-Wise Shortest Distance

The building block for the eccentricity computation is shortest distance computation. On unweighted graphs, the pair-wise shortest-distance problem (PWSD), that is, the computation of the distance between two given nodes u and v, can be resolved by performing a Breadth-First-Search (BFS) from node u. However, a BFS takes $O(m)$ time which can be more than 1 second for large real graphs.

2-hop labeling methods are proposed to efficiently answer PWSD queries. 2-hop labeling methods label each node w in V with the distances from w to every node in a set $S(w) \subseteq V$. The set $S(\cdot)$ is selected for each node such that for any two nodes u, v in V, $S(u) \cap S(v)$ contains at least one node on a shortest path from u to v. In such a way, the shortest distance can be computed with triangle inequalities:

$$dist(u, v) = \min_{x \in S(u) \cap S(v)} dist(u, x) + dist(x, v).$$

Pruned landmark labeling. General 2-hop labeling methods suffer from a large label set. A *2-hop labeling* method called Pruned Landmark Labeling (PLL) approach [9] has been proposed specifically for the PWSD problem on social networks. Specifically, it provides a label pruning technique that is especially effective on social networks: a PWSD query can be answered in *1 microsecond* even for a large social network.

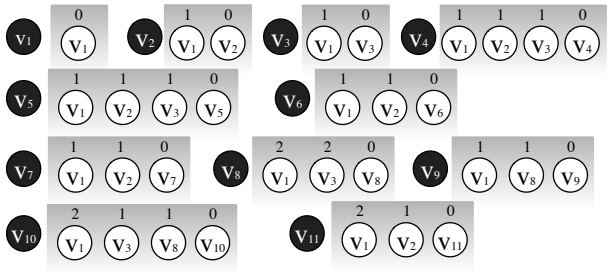

Fig. 3. Pruned Landmark Labeling for All Nodes in G

Example 3. *Fig. 3 shows the pruned landmark labeling for all nodes in graph G Fig. 1 of the running example. For example, the label of v_5 is $S(v_5) = \{v_1 : 1, v_2 : 1, v_3 : 1, v_5 : 0\}$ and the label of v_8 is $S(v_8) = \{v_1 : 1, v_3 : 1, v_8 : 0\}$. We have $S(v_5) \cap S(v_8) = \{v_1, v_3\}$. Therefore, we can calculate $dist(v_5, v_8) = \min\{dist(v_5, v_1) + dist(v_1, v_8), dist(v_5, v_3) + dist(v_3, v_8)\} = \min\{1 + 2, 1 + 2\} = 3$.*

Average label length. The technique of PLL can be used as a black box with a parameter b of graph G defined as below.

Definition 2 (Average Label Length)**.** *Given a graph G, denote by $S(u)$ the set of nodes selected by the pruned landmark labeling (PLL) approach for a node u in G. Define the Average Label Length as $b = average_{u \in V} |S(u)| = \Sigma_{u \in V} \frac{|S(u)|}{n}$.*

The parameter of average label length is introduced to quantify the query time of PLL. Fig. 1 and Fig. 3 show that the label length of a node is not proportional to its degree.

Lemma 1. *Expectedly, the shortest distance between two nodes can be computed in $O(b)$ time by leveraging PLL.*

Algorithm 1: BoundEcc

Input: Graph $G(V, E)$
Output: $ecc(u)$ for each $u \in V$

```
1  W ← a priority queue of V;
2  Initialize: ecc̄(u) ← +∞, ecc̲(u) ← 0, for ∀u ∈ V;
3  while W is not empty do
4      u ← W.pop();
5      Compute dist(u, v), for ∀v ∈ V, by calling a BFS;
6      ecc(u) ← max_{v∈V} dist(u, v);
7      for each node w ∈ W do
8          ecc̄(w) ← min{ecc̄(w), ecc(u) + d(u, w)};
9          ecc̲(w) ← max{ecc̲(w), d(u, w), ecc(u) − d(u, w)};
10         if ecc̄(w) = ecc̲(w) then remove w from W;
11 return ecc(u), ∀u ∈ V
```

Proof: For two nodes $u, v \in V$, it costs $O(|S(u)| + |S(v)|)$ to compute $\min_{w \in S(u) \cap S(v)} dist(u, w) + dist(w, v)$.

$$Exp_{u,v \in V}(|S(u)| + |S(v)|) = 2Exp_{u \in V}|S(u)| = 2b. \quad \blacksquare$$

As we shall see in our experiment (Table II), the average label lengths of most social networks are less than 10^2.

B. Eccentricity Computation: The State-Of-The-Art

The method BoundEcc [8] for computing the eccentricity of each node in V is shown in Algorithm 1. It follows the general framework of many eccentricity, radius, diameter computation methods. It associates each node $u \in V$ with an upper bound $\overline{ecc}(u)$ of the eccentricity $ecc(u)$ and a lower bound $\underline{ecc}(u)$ (Line 2), update them (Line 7-9) until either the bounds meet (Line 11) or a BFS is performed to determine the exact $ecc(u)$ (Line 5-6). The upper and lower bounds are generally updated with triangle inequalities (Line 8-9). Specifically, Lemma 2 shows how the bounds are derived.

Lemma 2 (Update Bounds [8])**.** *Let u be a node of graph $G(V, E)$ with eccentricity $ecc(u)$. Given a node v and its distance $dist(v, u)$ to u,*

$$ecc(v) \leq ecc(u) + dist(u, v) \quad (1)$$
$$ecc(v) \geq ecc(u) - dist(u, v) \quad (2)$$
$$ecc(v) \geq dist(u, v) \quad (3)$$

Theoretically, as long as one adopts this framework, the complexity in the worst-case will be at least *quadratic* to n. BoundEcc allows one to adopt different heuristics on the setting of the priorities in W (Line 2). However, the invariant action of BFS taken for computing $ecc(u)$ (Line 5-6) has two prominent drawbacks:

- As long as $\underline{ecc}(u) < \overline{ecc}(u)$, even when they differ by only 1, $ecc(u)$ will be computed *from scratch*. This essentially wastes the previous efforts in narrowing the gap between $\underline{ecc}(u)$ and $\overline{ecc}(u)$ entirely.
- Since $ecc(u)$ is the distance from u to the "farthest" node, BFS has to traverse **all** the nodes in V which leaves no chance for an early-stop.

Besides, BoundEcc updates the eccentricity bounds of **every** node in W (Line 7), which, together with the BFS (Line 5), renders a heavy burden to the performance of the eccentricity computation on real-graphs in practice.

	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	v_9	v_{10}	v_{11}
1	2	1, 3	1, 3	1, 3	1, 3	1, 3	1, 3	2, 4	1, 3	2, 4	2, 4
2	-	3	2, 3	2, 3	2, 3	2, 3	2, 3	3, 4	2, 3	3, 4	2, 4
3	-	-	3	2, 3	2, 3	2, 3	2, 3	3, 4	2, 3	3, 4	3, 4
4	-	-	-	3	2, 3	2, 3	2, 3	3, 4	3, 3	3, 4	3, 4
5	-	-	-	-	3	2, 3	2, 3	3, 4	-	3, 4	3, 4
6	-	-	-	-	-	3	2, 3	3, 4	-	3, 4	3, 4
7	-	-	-	-	-	-	3	3, 4	-	3, 4	3, 4
8	-	-	-	-	-	-	-	4	-	3, 4	4, 4
9	-	-	-	-	-	-	-	-	-	4	-

TABLE I
THE EXECUTION OF BoundEcc

Example 4. *The execution process of* BoundEcc *is shown in Table I. The two numbers in each cell are the upper and lower bounds of eccentricity for the corresponding node. Using* BoundEcc*, we need to recalculate the eccentricity for* 9 *nodes. The dark gray color indicates a recalculation while the light gray color means an update of lower/upper bound after computing the exact eccentricity for a certain node. For example, after computing $ecc(v_2) = 3$, we can update the lower/upper bounds of v_3 from $1, 3$ to $2, 3$. Obviously, the* BoundEcc *algorithm involves a large number of exact shortest path calculations.*

C. Problem Definition

We study the eccentricity computation by focusing on two specific problems, as shall be defined below.

Problem 2 (Exacting Eccentricity for a Node)**.** *Given a node x in graph $G(V, E)$, associated with $\underline{ecc}(x)$ and $\overline{ecc}(x)$, determine the eccentricity $ecc(x)$ of x in a way that is faster than BFS in practice.*

Let each node $u \in V$ in $G(V, E)$ bear an upper bound $\overline{ecc}(u)$ and a lower bound $\underline{ecc}(u)$ on the eccentricity $ecc(u)$ of u. The bounds $\{\overline{ecc}(u), \underline{ecc}(u)\}, \forall u \in V$, are called the **eccentricity-bounds**.

Problem 3 (Efficient Update)**.** *For a node x called* ***trigger node****, a solution to Problem 2 gains exact values of $ecc(x)$. Based on $ecc(x)$, update the eccentricity-bounds of all nodes in V without traversing the whole graph.*

To speed up the computation, we allow the system to pre-compute the following auxiliary structures:

1) for a **reference node z** that is pre-determined, the distance $dist(z, u)$ from z to all the nodes u in V can be accessed monotonically: list L_z stores $\{u_1, u_2, \cdots, u_n\}$ with $dist(z, u_1) \leq dist(z, u_2) \leq \cdots \leq dist(z, u_n)$.
2) any 2-hop labeling method for the pair-wise shortest-distance (PWSD) computation for the graph G; in particular, the **pruned landmark labeling** (PLL) structure with a parameter **b** (See Section II-A) answers the shortest distance of any two nodes in $O(b)$ time.

Note that, directly applying PLL to the eccentricity computation necessitates n^2 PWSD queries — impractical for large graphs. In this paper, we consider PLL as a black box for answering PWSD queries and propose novel techniques to compute the eccentricities efficiently. Our techniques are independent from the techniques in PLL.

In the following sections, we will address Problems 2 and 3 in Section III and IV, respectively.

III. Exacting Eccentricity for a Node

This section considers the node x with $\underline{ecc}(x) < \overline{ecc}(x)$ in Problem 2, aiming at computing $ecc(x)$ efficiently. We first introduce our techniques for a fixed reference node z in Section III-A and then discuss how to practically tailor the reference node z to the node of x in Section III-B.

A. Computing ecc(x) Under a Fixed Reference Node

The very reason that BFS has to traverse the whole graph to get the eccentricity of x is the conflict between

- the non-decreasing order of the distances from nodes to x in which BFS follows, and
- the max nature in $ecc(x)$ among the distances of all nodes to x.

If the order of BFS traversal can be reversed, then we can use only $O(1)$ time to access $ecc(x)$. However, this ideal case will not take place since the distance information for node x is unavailable unless v is exactly the reference node z.

*Utilize the pruned landmark labeling (*PLL*) structure.* With the tool of the PLL structure, one can get pair-wise shortest-distance efficiently. This allows us to probe the distance from x to a subset V' of nodes in V. By performing a max aggregation over the exact distances from x to nodes in V', we can partially evaluate the eccentricity of v.

Definition 3 (partial-eccentricity)**.** *Given a subset $V' \subseteq V$, define the partial-eccentricity of x on V' as the eccentricity of x on the set of V', denoted as $pecc(x|V') = \max_{u \in V'} dist(x,u)$.*

Lemma 3. *Let V' be a subset of V. $pecc(x|V') \leq ecc(x)$, that is, partial-eccentricity provides a lower bound for the eccentricity. Besides, $pecc(x|V) = ecc(x)$.*

Utilize the reference node z. To transfer the knowledge gained on the reference node z to the computation of $ecc(v)$, we first introduce the definition of a bounded set with bounded eccentricities.

Definition 4 (Bounded set)**.** *Given $\lambda \geq 0$, the bounded set*

$$V_{\leq \lambda} = \{u \in V | dist(u,z) \leq \lambda\}.$$

Lemma 4 (Bounded Eccentricity)**.** *Given $\lambda \geq 0$, the partial-eccentricity of a bounded set of λ is also bounded:*

$$pecc(x|V_{\leq\lambda}) \leq dist(x,z) + \lambda.$$

Proof: According to Definition 3, $ecc(x|V_{\leq\lambda}) = \max_{u \in V_{\leq\lambda}} dist(x,u)$. For $\forall u \in V_{\leq\lambda}$, $dist(x,u) \leq dist(x,z) + dist(z,u) \leq dist(x,z) + \lambda$. Therefore,

$$ecc(x|V_{\leq\lambda}) = \max_{u \in V_{\leq\lambda}} dist(x,u) \leq \max_{u \in V_{\leq\lambda}} (dist(x,z) + \lambda) = dist(x,z) + \lambda. \quad \blacksquare$$

Definition 5 (Partial-set)**.** *Given $\lambda \geq 0$, a set V' is called a partial-set of λ, if $V' \cup V_{\leq\lambda} = V$, namely, V' is a super set of $V \setminus V_{\leq\lambda}$.*

Example 5. *Fig. 4 shows L_z for $z = v_2$ of the running graph in Fig. 1. Obviously, if $x = z$, we can determine $ecc(x) = 3$ directly from L_z. We also show $V_{\leq 1}$ and $V_{\leq 2}$ where $V_{\leq 1}$ contains the first 7 nodes in L_z while $V_{\leq 2}$ contains the first 9 nodes in L_z. The partial set V' with respect to $\lambda = 2$ can be any subset of V that contains $\{v_8, v_{10}\}$; V' w.r.t. $\lambda = 1$ can be any subset containing $\{v_3, v_9, v_8, v_{10}\}$. For $x = v_4$ and $\lambda = 2$, $pecc(x|V_{\leq\lambda}) = \max_{v \in V_{\leq\lambda}} dist(x,v) = 2$. Obviously, we have $pecc(x|V_{\leq\lambda}) \leq dist(x,z) + \lambda = dist(v_4, v_2) + 2 = 3$.*

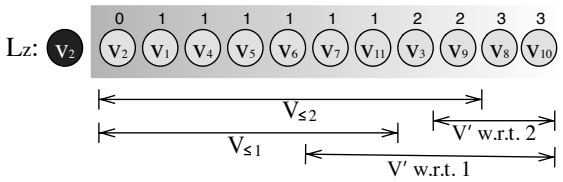

Fig. 4. Illustration of Bounded Set and Partial Set ($z = v_2$)

Utilize both the reference node z and the PLL*.* For a given λ, we combine the information we get from the reference node z and the partial-eccentricity on a partial-set of λ.

Lemma 5. *Given a partial-set V' with parameter λ, the eccentricity of x is:*

$$ecc(x) = \begin{cases} pecc(x|V'), & \text{if } pecc(x|V_{\leq\lambda}) \leq pecc(x|V') \\ pecc(x|V_{\leq\lambda}), & \text{otherwise} \end{cases}$$

Proof: According to the definition of a partial-set, $V' \cup V_{\leq\lambda} = V$, and the definition of the eccentricity $ecc(x) = max_{u \in V} dist(u,x)$, $ecc(x) = \max\{pecc(x|V'), pecc(x|V_{\leq\lambda}\}$. $\blacksquare$

Assume that at a time, for a given λ and a corresponding partial-set V' of λ, $pecc(x|V')$ is obtained using PLL while $pecc(x|V_{\leq\lambda})$ is bounded by Lemma 4. Is it possible that we can determine the eccentricity $ecc(x)$? The following theorem provides a positive answer.

Theorem 1. *Given a partial-set V' of parameter λ, if $pecc(x|V') \geq dist(x,z) + \lambda$, then $ecc(x) = pecc(x|V')$.*

Proof: From Lemma 4, $pecc(x|V_{\leq\lambda}) \leq dist(x,z) + \lambda$. If $pecc(x|V') \geq dist(x,z) + \lambda$, then $pecc(x|V') \geq pecc(x|V_{\leq\lambda})$. According to Lemma 5, we have $ecc(x) = pecc(x|V')$. $\blacksquare$

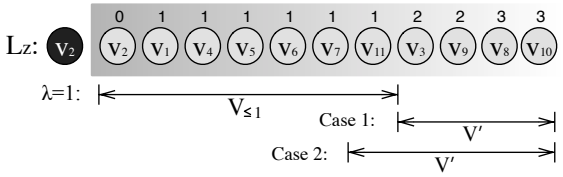

Fig. 5. Illustration of $V_{\leq\lambda}$ and V' for $\lambda = 1$ ($z = v_2$)

Example 6. *Suppose we would like to calculate $ecc(v_9)$. Given parameter $\lambda = 1$ and reference node $z = v_2$, $pecc(v_9|V_{\leq\lambda}) = 3$. Fig. 5 shows two cases to select the partial-set V'. In the first case, we have $prcc(v_9|V_{\leq\lambda}) > pecc(v_9|V') = 2$. We can thus compute $ecc(v_9) = pecc(v_9|V_{\leq\lambda})$. In the second case, we have $pecc(v_9|V_{\leq\lambda}) \leq pecc(v_9|V') = 3$. We can thus compute $ecc(v_9) = pecc(v_9|V')$. For case 2, we also have $pecc(v_9|V') = 3 \geq dist(v_9,z) + \lambda = 3$. Therefore, we can compute $ecc(v_9) = pecc(v_9|V')$ without knowing $pecc(v_9|V_{\leq\lambda})$.*

Theorem 1 leads to upper and lower bounds of $ecc(x)$.

Lemma 6. *Let V' be a partial-set of parameter λ. $\max\{pecc(x|V'), dist(x,z) + \lambda\}$ provides an upper bound on $ecc(x)$ while $pecc(x|V')$ provides a lower bound on $ecc(x)$.*

When the upper bound meets the lower bound, we can determine $ecc(u)$ straightly. In this way, we can even leverage

Algorithm 2: EccentricityOneNode

Input: Node x, z, $\overline{ecc}(x)$, $\underline{ecc}(x)$, L_z, the PLL structure
// $L_z = \{u_1, u_2, \cdots, u_n\}$ (Section II-C).
Output: $ecc(x)$

1 $p \leftarrow 0$; // $p = pecc(x|V')$ where V' is initially $\emptyset$
2 **for** i *takes from* n *down to* 1 **do**
3 $\quad \lambda \leftarrow dist(z, u_{i-1})$;
4 $\quad$ Obtain $dist(x, u_i)$ by inquiring PLL;
5 $\quad p \leftarrow \max\{p, dist(x, u_i)\}$; // By absorbing u_i, V' is still a partial-set of λ, $p = pecc(x|V')$.
6 $\quad \underline{ecc}(x) \leftarrow \max\{\underline{ecc}(x), p\}$;
7 $\quad \overline{ecc}(x) \leftarrow \min\{\overline{ecc}(x), \max\{p, dist(x, z) + \lambda\}\}$;
8 $\quad$ **if** $\underline{ecc}(x) = \overline{ecc}(x)$ **then return** $\underline{ecc}(x)$;
9 **return** $\underline{ecc}(x)$

the original upper and lower bound on u to narrow the gap. Now we are ready to introduce our approach.

Exact eccentricity computation. Theorem 1 implies a way in determining $ecc(v)$: traverse nodes of V in a non-increasing order of their distances to the reference node z; in this process, update the upper bound and lower bound using Lemma 6; terminate once the upper and lower bounds meet. We describe such an approach in Algorithm 2.

Algorithm 2 decreases λ from $dist(z, u_n)$ to 0 and grows a conceptual partial-set V' of λ from $\emptyset$ to V accordingly. Initially, $pecc(v|V') = 0$ since $\lambda = dist(z, u_n)$ and thus $V' = \emptyset$ is a partial set of λ (Line 1). Then nodes in V are examined in a reverse order of L_z (Line 2). For each node u_i, λ is set to be the distance from the reference node z to u_{i-1} (Line 3). Obviously, $\{u_1, u_2, \cdots, u_{i-1}\}$ is a subset of $V_{\leq\lambda}$. Conceptually, V' should be augmented with u_i such that it remains a partial-set of the newly updated λ. The partial-eccentricity $p = pecc(x|V')$ on V' is updated accordingly (Line 4-5). The upper bound and lower bound of $ecc(x)$ are then updated Line 6-7. The loop will be terminated immediately when the gap between the two bounds become 0 (Line 8). The entire loop transforms $\underline{ecc}(x)$ to $ecc(x)$ (Line 9).

Theorem 2. *Algorithm 2 reports the eccentricity of node x.*

Proof: Lemma 6 ensures that the upper and lower bounds of the eccentricity of x are correctly updated (Line 6-7). Therefore, if the two bounds match, they match on $ecc(x)$ (Line 8). If the two bounds have not agreed by the end of the loop when $V' = V$, then $\underline{ecc}(x) = pecc(x|V') = pecc(x|V) = ecc(x)$ can be safely reported (Line 9) due to Lemma 3. ■

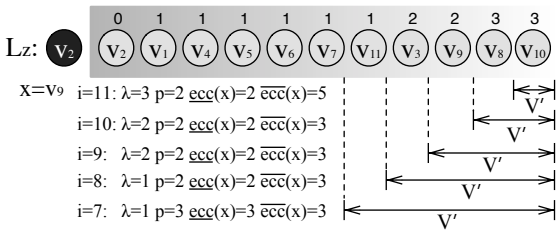

Fig. 6. The Process to Compute $ecc(v_9)$ ($z = v_2$)

Example 7. *Fig. 6 illustrates the process to compute $ecc(x)$ for $x = v_9$ for the running graph shown in Fig. 1. Suppose $z = v_2$, for $i = 11$ ($\lambda = 3$), the algorithm first computes $dist(x, v_{10}) = 2$ by inquiring* PLL*, updates p to be 2 and updates $\underline{ecc}(x)$ and $\overline{ecc}(x)$ to be 2 and 5 respectively. Then for $i = 10$ ($\lambda = 2$), the algorithm computes $dist(x, v_8) = 1$, and update $\overline{ecc}(x)$ to be $dist(x, z) + \lambda = 4$ The process continues until $i = 7$ ($\lambda = 1$), where the algorithm computes $dist(x, v_{11}) = 3$ and update $p = 3$ and $\underline{ecc}(x) = 3$. At this time we have $\underline{ecc}(x) = \overline{ecc}(x)$, and therefore the algorithm terminates by returning 3 as $ecc(x)$.*

Theorem 3. *In the worst case, Algorithms 2 reports the eccentricity of v in $O(bn)$ time.*

Proof: For node u_i, $i \in [1, n]$, Algorithms 2 computes the exact distances from x to u_i using the structure of PLL in $O(b)$ time. All other operations including the computation of the distance from a node to the reference node z, can be completed in $O(1)$ time. ■

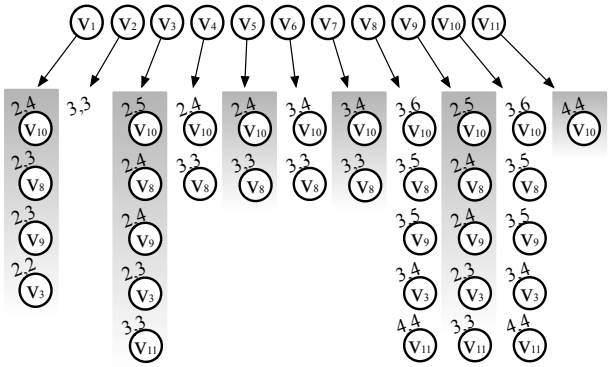

Fig. 7. Computing Eccentricity for All Nodes ($z = v_2$)

Example 8. *Fig. 7 demonstrates the process to compute the eccentricity for all nodes in the running graph in Fig. 1 by setting the reference node as $z = v_2$. For example, for node v_1, we need to inquiry* PLL *for 4 times. The $\underline{ecc}$ and $\overline{ecc}$ values after each* PLL *inquiry are also labelled beside each node. In total, we have 33* PLL *inquiries.*

Remarks. Algorithm 2 is superior to BFS in practice:

1) Algorithm 2 boosts the computation of $ecc(x)$ by leveraging the previously computed upper and lower bounds for $ecc(x)$. For example, if $\underline{ecc}(x)$ is smaller than $\overline{ecc}(x)$ by a tiny margin, say 1, before Algorithm 2 starts, then one only needs 1 effective update on either the upper or the lower bound (in Line 6-7) to terminate Algorithm 2.
2) Algorithm 2 traverses the nodes in the reversing order of their distances to the reference node z. If z is close to x, then the farthest node to x will not be close to z. Thus, the algorithm can terminate at an early-stage.

B. Reference-Node Pool

As observed from Section III-A (Remarks 2) and Lemma 2, a reference node z in the vicinity of x can greatly boost the computation of $ecc(x)$ in Algorithm 2 through an early-stop. Specifically, consider the case when $dist(x, z) = 1$. From Lemma 2, $ecc(z) - 1 \leq ecc(x) \leq 1 + ecc(z)$. Besides, by triangle

Algorithm 3: RefPool

Input: Graph $G(V, E)$, k
Output: *pool*; L_z, $\forall z \in pool$; $\overline{ecc}(u)$ and $\underline{ecc}(u)$, $\forall u \in V$

1 Let *pool* be the k nodes in G with the highest degrees;
2 **for** *each node* $z \in pool$ **do**
3 $\quad L_z \leftarrow$ a list of nodes in V in non-decreasing order of their distances to z;
4 $\quad$ **for** *each node* $u \in V$ **do**
5 $\quad\quad$ update $\overline{ecc}(u)$ and $\underline{ecc}(u)$ using Lemma 2;
6 **return** as required.

inequality, $ecc(z) - 1 \leq dist(x, u_n) \leq ecc(z) + 1$. Here u_n is the last element of L_z with $ecc(z) = dist(z, u_n)$.

- If $dist(x, u_n) = dist(z, u_n) + 1$, we can immediately terminate Algorithm 2. Note that the chance of this case is considerable since $dist(z, u_n) + 1$ is *one of the three* values that $dist(x, u_n)$ can possibly take.
- Otherwise, $dist(x, u_n)$ provides a strong lower bound for $ecc(x)$: $ecc(x) \geq dist(x, u_n) \geq ecc(z) - 1$. Let u be the node with $dist(u, x) = ecc(x)$. Then $dist(u, z) \geq dist(u, x) - dist(x, z) \geq ecc(x) - 1 \geq ecc(z) - 2$. That is, once the $\lambda = dist(u_{i-1}, z)$ in Line 3, Algorithm 2, drops below $ecc(z) - 2$, we can safely terminate the algorithm.

Example 9. *Suppose we would like to compute* $ecc(x)$ *for* $x = v_4$ *with reference node* $z = v_2$. *Note that* $dist(x, z) = 1$. *Therefore, in the worst case, we only need to visit those nodes* y *with* $dist(y, z) \geq ecc(z) - 2$, *which are* $\{v_{10}, v_8, v_9, v_3\}$. *As shown in Fig. 7, we only visit* $\{v_{10}, v_8\}$ *to compute* $ecc(v_4)$.

The theorem below generalizes the analysis on the case of $dist(x, z) = 1$.

Theorem 4. *Denote* $dist(x, z)$ *as* λ_0. *Let* y *be the most remote node to* x, *that is,* $ecc(x) = dist(y, x)$. *Then*

$$dist(y, z) \geq ecc(z) - 2\lambda_0.$$

Therefore, it suffices for Algorithm 2 to visit all nodes in $\{v \in V | dist(v, z) \geq ecc(z) - 2\lambda_0\}$ *to determine* $ecc(x)$.

Proof: From Lemma 2, $ecc(z) - \lambda_0 \leq ecc(x) = dist(y, x) \leq dist(y, z) + dist(x, z) = dist(y, z) + \lambda_0$ ∎

Interestingly, this observation combined with the properties of a small-world network provides a simple yet effective solution for reference node selection — reference-node pool.

In Algorithm 3, we set a parameter k as the size of the reference-node pool, and then let the reference nodes be the k nodes in $G(V, E)$ with the highest degrees (Line 1). This simple setting does not burden the entire computation. In addition, reference-node pool facilitates an easy way to setup the initial upper and lower bounds for the eccentricity of each node (Line 4-5).

As we shall see in Algorithm 5 (Line 4), at running time, each node will choose the closest node from the reference node pool as its reference node.

Example 10. *In the running example, graph* G *is shown in Fig. 1. Let* $k = 2$. *Select* 2 *nodes with the highest degree in* G *as the reference-node pool:* $pool = \{v_1, v_2\}$. *The process of computing the eccentricity for all nodes in* G *is then shown in Fig. 8. The reference node selected for each node is colored in black. For example, for node* $x = v_3$, v_1 *instead of* v_2 *is selected as the reference node since* $dist(v_2, v_3) > dist(v_1, v_3)$. *Using* v_1 *as the reference node, one only needs to obtain* $dist(v_3, v_{11}) = 3$ *to terminate the algorithm with* $ecc(v_3) = 3$ *by inquiring* PLL. *Compared to Example 8, by using the reference-node pool, the number of* PLL *inquiries is brought down from* 33 *to* 13.

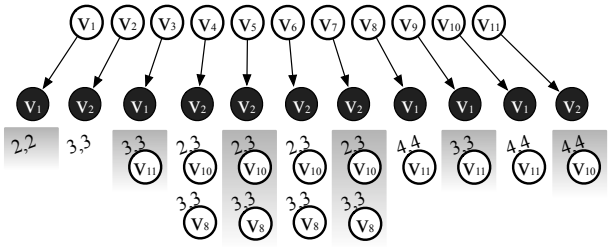

Fig. 8. Eccentricity Computation ($pool = \{v_1, v_2\}$)

In a small-world network, some nodes have much more connections than the other nodes. By choosing a small number of prominent hubs as reference-nodes, a node x is highly likely to find a reference node in the pool in its vicinity, that is, on average, the distance $d(x, z)$ from a node x to its reference node z is relatively small, as will be demonstrated in the experiment (*Exp-5*, Section VI). To determine $ecc(x)$, it suffices to scan only the nodes whose distances fall in to this narrow range, as indicated by Theorem 4.

IV. Update Optimization

This section dedicates to Problem 3 in Section II-C, that is, to efficiently update the eccentricity-bounds based on the eccentricity $ecc(x)$ of the trigger node x.

To avoid an exhaustive enumeration of nodes in V, consider the following rules on neighboring nodes in the graph. Applying Lemma 2 to u, v with $dist(u, v) = 1$, we have Lemma 7.

Lemma 7. *For each* $(u, v) \in E$ *(or equivalently* $(v, u) \in E$*):*

$$ecc(u) - 1 \leq ecc(v) \leq ecc(u) + 1.$$

Lemma 7 indicates that the eccentricity-bounds on neighboring nodes differ by at most one. We call the eccentricity-bounds stable if it satisfies Lemma 7.

Definition 6 (Stable State)**.** *The eccentricity-bounds are* ***stable****, if for each edge* $(u, v) \in E$*:*

$$\overline{ecc}(u) \leq \overline{ecc}(v) + 1, \text{ and} \quad (4)$$

$$\underline{ecc}(u) \geq \underline{ecc}(v) - 1. \quad (5)$$

Example 11. *The eccentricity-bounds shown in Fig. 9 (a) are stable. For example, for adjacent nodes* v_3 *and* v_{10}, *we have* $\overline{ecc}(v_{10}) \leq \overline{ecc}(v_3) + 1$ *and* $\underline{ecc}(v_3) \geq \underline{ecc}(v_{10}) - 1$.

Lemma 7 can be applied globally to iteratively update the eccentricity-bounds. The process is called Iterative-Update and will be shown in Section IV-A. Magically, the iterative-update is effectively applying the first two rules of Lemma 2, as will be proved in the weak bounds (Lemma 9). More importantly, the iterative-update can be dramatically optimized

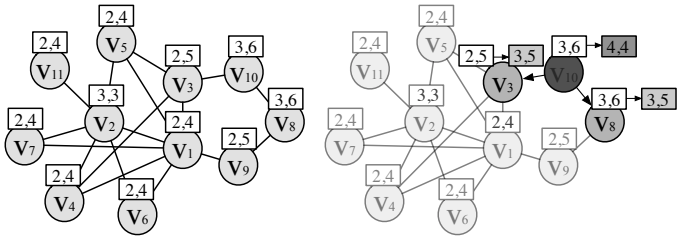

Fig. 9. Local Spread: Before and After Updating the Bounds of v_{10}

(Theorem 5) with an efficient local-spread update algorithm, as shall be introduced in Section IV-B.

A. Iterative-Update.

This subsection describes an iterative update process of the eccentricity-bounds from one stable state to another stable state in accordance with the update of the eccentricity-bounds of the trigger node x.

Assume that the eccentricity-bounds are **stable** (Definition 6) before the update. For the simplicity of the presentation, we conceptually take a snapshot of the eccentricity-bounds as $\{\overline{ecc}_{old}(u), \underline{ecc}_{old}(u)\}$, for $\forall u \in V$.

Let the new upper and lower eccentricity-bounds for the trigger node x be ub_x and lb_x, respectively. If $ecc(x)$ is available, then $ub_x = lb_x = ecc(x)$. The process has two steps.

1) Update the eccentricity-bounds of x with ub_x and lb_x:
   a) $\overline{ecc}(x) \leftarrow \min\{\overline{ecc}_{old}(x), ub_x\}$
   b) $\underline{ecc}(x) \leftarrow \max\{\underline{ecc}_{old}(x), lb_x\}$
2) The eccentricity-bounds are updated in an iterative manner based on Lemma 7, and terminates when eccentricity-bounds become stable. Specifically, for one update of an edge $(l, r) \in E$ or equivalently $(r, l) \in E$:
   a) $\overline{ecc}(l) \leftarrow \min\{\overline{ecc}(l), \overline{ecc}(r) + 1\}$
   b) $\underline{ecc}(l) \leftarrow \max\{\underline{ecc}(l), \underline{ecc}(r) - 1\}$

For the same reason, we conceptually take a snapshot of the eccentricity-bounds as $\{\overline{ecc}_{new}(u), \underline{ecc}_{new}(u)\}, \forall u \in V$, after the iterative-update process.

B. Local-spread.

Next, we will first introduce a favorable property of iterative-update. As will be proved by Lemmas 10 and 11, Theorem 5 shows that the iterative-update effectively tighten the eccentricity-bounds in a connected subgraph of G "centered" at the trigger node x, which enables a remarkable optimization over the iterative-update.

Theorem 5 (Update Locality)**.** *Let V' be $\{y \in V | \overline{ecc}_{old}(y) > \overline{ecc}_{new}(y)$ or $\underline{ecc}_{old}(y) < \underline{ecc}_{new}(y)\}$. If V' is not empty, then graph $G'(V', E')$ defined with $E' = \{(u, v) \in E | u, v \in V'\}$ is a connected graph. That is, for any two nodes a, b in V', there is a path of G' from a to b. Besides, for each node y in V',*

- $\overline{ecc}_{new}(y) = \min\{\overline{ecc}_{old}(y), ub_x + dist(x, y)\}$
- $\underline{ecc}_{new}(y) = \max\{\underline{ecc}_{old}(y), lb_x - dist(x, y)\}$.

Theorem 5 enables us to design a local-spread update approach in Algorithm 4. To update on the eccentricity-bounds triggered by node x, we visit the nodes in V in the style of Breadth-First-Search (Line 2,5). If one node u is not affected by the update on the trigger node x, then the node expansion on u is banned (Line 3,6,10). All the distances (Line 8-9) are obtained via BFS rather than pairwise distance queries. A state checking operation (Line 3,7,10,11) can be done in $O(1)$ time by associating proper labels to the bounds. The search terminates when the area affected by the update on the trigger node x is traversed (Line 4).

Algorithm 4: LocalSpread

Input: Node x, ub_x, lb_x, a stable eccentricity-bounds.
Output: A stable eccentricity-bounds
`// All eccentricity-bounds are clean. A bound` b `gets dirty once it is updated, which can be materialized by setting a field of` b `with` x.

```
1  ecc̄(x) ← min{ecc̄(x), ub_x}, ecc(x) ← max{ecc(x), lb_x};
2  Q ← a empty queue;
3  if either ecc̄(x) or ecc(x) is dirty then add x to Q;
4  while Q is not empty do
5      u ← Q.pop(); mark u as visited;
6      if either ecc̄(u) or ecc(u) is dirty then
7          for each unvisited neighbor v of u do
8              ecc̄(v) ← min{ecc̄(v), ub_x + dist(v, x)};
9              ecc(v) ←
                 max{ecc(v), lb_x − dist(v, x), dist(v, x)};
10             if either ecc̄(v) or ecc(v) is dirty then
11                 add v to Q if v is not in Q;
```

Lemma 8 (Complexity)**.** *Let set $V' = \{y \in V | \overline{ecc}_{old}(y) > \overline{ecc}_{new}(y)$ or $\underline{ecc}_{old}(y) < \underline{ecc}_{new}(y)\}$ be the set of nodes that has been affected by the update on the trigger node x. The complexity of Algorithm 4 is $O(\Sigma_{v \in V'} deg(v))$. Denote by $deg(v)$ the degree of node v in the graph $G(V, E)$.*

Example 12. *Fig. 9 (a) shows a stable state for the graph in Fig. 1 of the running example. Suppose we calculate $ecc(v_{10}) = 4$, we update $\underline{ecc}(v_{10}) = \overline{ecc}(v_{10}) = 4$. Using local spread, for the neighbor v_3 of v_{10}, we update $\underline{ecc}(v_3)$ to be 3. For the neighbor v_8 of v_{10}, we update $\overline{ecc}(v_8)$ to be 5. After local spread, the state becomes stable again as shown in Fig. 9 (b).*

Correctness of Theorem 5. Iterative-update tightens the eccentricity-bounds iteratively without specifying the convergence rate. To quantify the margin between the old and new snapshots, we found bounds on the margins: Lemma 9 (weak), 10 (strong), and 11 (strong), respectively.

We place the proofs of Lemma 9 and 10 in the Appendix. Lemma 11 can be proved by applying the proof on the upper bounds of Lemma 10 symmetrically to the lower bounds.

Lemma 9 (Weak bounds)**.** *For any node $v \in V$ other than u in the graph,*

$$\overline{ecc}_{new}(u) \leq \min\{\overline{ecc}_{old}(u), \overline{ecc}_{new}(x) + dist(x, u)\} \quad (6)$$

$$\underline{ecc}_{new}(u) \geq \max\{\underline{ecc}_{old}(u), \underline{ecc}_{new}(x) - dist(x, u)\} \quad (7)$$

Lemma 10 (Strong upper bounds)**.** *For each node $y \in V$ with $\overline{ecc}_{new}(y) < \overline{ecc}_{old}(y)$, there exists a shortest path*

$\langle u_0, u_1, \cdots, u_k \rangle$ *from* x *to* y *with* $k = dist(y, x)$ *such that*

1) *for each* $i \in [0, k]$, $\overline{ecc}_{new}(u_i) = ub_x + dist(u_i, x)$.
2) *for each* $i \in [0, k]$, $\overline{ecc}_{new}(u_i) < \overline{ecc}_{old}(u_i)$.
3) $\overline{ecc}_{new}(x) = ub_x$.

Lemma 11 (Strong lower bounds)**.** *For each node* $y \in V$ *with* $\underline{ecc}_{new}(y) > \underline{ecc}_{old}(y)$*, there exists a shortest path* $\langle u_0, u_1, \cdots, u_k \rangle$ *from* x *to* y *with* $k = dist(y, x)$ *such that*

1) *for* $\forall i \in [0, k]$, $\underline{ecc}_{new}(u_i) = \underline{ecc}_{new}(x) - dist(u_i, x)$.
2) *for* $\forall i \in [0, k]$, $\underline{ecc}_{new}(u_i) > \underline{ecc}_{old}(u_i)$.
3) $\underline{ecc}_{new} x = lb_x$.

C. Put All Parts Together

We now have all parts in the puzzle of exacting eccentricity completed in Algorithm 5. We pre-compute an auxiliary structure PLL for efficiently answer pair-wise shortest-distance queries (Line 1). Select the reference-node pool with k nodes (Line 2). For each node $v \in V$ (Line 3), we first find the reference node in the pool with the smallest distance to x (Line 4), then use the reference node to compute the exact eccentricity of x (Line 5). After that, we use $ecc(x)$ to update the eccentricity-bounds (Line 6). Finally, we are able to report the eccentricities of all nodes in V (Line 7). The correctness of Algorithm 3-5 can be easily guaranteed by the triangle inequality, Lemma 2 and Lemma 7.

Theorem 6. *In expectation, each call of* LocalSpread *updates the eccentricity of* $O(d)$ *nodes.*

Proof: According to Theorem 5, a node $y \in V$ is updated by LocalSpread only when $\overline{ecc}(y)$ is increased to $ecc(x) + dist(x, y)$ or $\underline{ecc}(y)$ is decreased to $ecc(x) - dist(x, y)$ upon $ecc(x)$ of the corresponding trigger node x. Note that $ecc(x) + dist(x, y) \leq 2d$, here d is the diameter of the graph. Thus, after the first update on $\overline{ecc}(y)$, $\overline{ecc}(y)$ will be decreasing within the range of $[0, 2d]$. Therefore, $\overline{ecc}(y)$ will be updated at most $2d + 1$ times. We can similarly prove that $\underline{ecc}(y)$ will be updated at most $d + 1$ times. In total, LocalSpread will update the eccentricity bounds of y at most $3d+1$ times. The total of $O((3d+1)n)$ updates over all nodes took place in n calls of LocalSpread, each call thus updates $O(d)$ nodes in expectation. ∎

Lemma 12. *The time complexity on updating the eccentricity-bounds in Algorithm 5 in total is* $O(dm)$.

Proof: According to Lemma 8, the adjacency list of a node $y \in V$ is visited by LocalSpread only when a bound of y is updated. Since each node is updated $O(d)$ times and each update reads through the corresponding node's adjacency list, therefore, the total time for LocalSpread is $O(d \cdot \sum_{v \in V} deg(v)) = O(dm)$. ∎

Remarks. The worst-case complexity of Algorithm 5 is quadratic, however,

- Algorithm 5 determines the eccentricity of a node x at an early-stage by i) searching from remote nodes of x guided by a reference node that is close to x; and ii) inheriting the eccentricity-bounds of x from the outer-loop which terminates the search whenever the bounds meet.
- Each call of LocalSpread updates the eccentricity-bounds of $O(d)$ nodes in $O(\frac{dm}{n})$ time in expectation (Lemma 12). Since a small-world network has a small d ($d < 40$ for all datasets in Table II), the update can be regarded as near-linear.

Algorithm 5: ECC-LS

Input: Graph $G(V, E)$, k
Output: $ecc(u)$ for each $u \in V$

```
1 PLL ← the PLL structure of G(V,E);
2 pool, L_z,eccentricity-bounds ← RefPool(G(V,E), k);
3 for each node x ∈ V do
4     z ← the node in the pool that is nearest to x;
5     ecc(x) ←
        EccentricityOneNode(x, z, ecc̄(x), ecc(x), L_z, PLL);
6     LocalSpread(x, ecc(x), ecc(x),eccentricity-bounds);
7 return ecc(u), ∀u ∈ V
```

Therefore, within the loop of a node in V (Line 3, Algorithm 5), the practical cost is far less than n. In this sense, our algorithm is more efficient than its counterparts.

V. Related work

Exact Eccentricity. A straightforward method to compute the exact eccentricity for all nodes is to apply all-pairs shortest path (APSP) algorithms or to pose pair-wise shortest distance (PWSD) queries quadratic times. These algorithms, however, require a high time complexity and thus are impractical to handle large graphs [10]. Although optimization strategies are proposed [11], [12], their approaches still cannot scale to handle large real-world graphs. An efficient approach to the PWSD problem is called Pruned Landmark Labeling (PLL) [9]; its detail has been introduced in Section II-A.

In the literature, to compute the exact eccentricity, Henderson [13] speeds up the computation by making use of articulation points and eccentricity bounds. The state-of-the-art algorithm is proposed by Takes et al. [8], which has been introduced in Section II-B in details. Borassi et al. [14] focus on the "directed" aspect of the diameter/radius computation on directed graphs; their techniques fall into the framework of [8] when it comes to undirected scenarios.

As a related problem, graph diameter is defined as the maximum eccentricity among all nodes. A pruning based method to compute the graph diameter is introduced in [15] and the method is further improved by Akiba et al. [16] using eccentricity bounds propagation.

Approximate Eccentricity. In the literature, because of the huge computational cost for exact eccentricity, several approaches focus on approximate eccentricity computation. A straightforward approach is to adopt the approximate APSP [17]. However, this method does not consider the properties involved in eccentricity. Roditty et al. [4] presents an algorithm to estimate eccentricity $\widetilde{ecc}(v)$ using sampling. $ecc(v)$ is bounded by $[\frac{2}{3}\widetilde{ecc}(v), \frac{3}{2}\widetilde{ecc}(v)]$, for each node v in an undirected and unweighted graph. The time complexity is $O(m\sqrt{n \log n})$. The method is further improved by Chechik et al. [5] by transforming the graph to a bounded-degree graph. $ecc(v)$ is bounded by $[\widetilde{ecc}(v), \frac{5}{3}\widetilde{ecc}(v)]$. The complexity is $O((m \log m)^{\frac{3}{2}})$. Recently, Shun [18] parallelized existing approximate algorithms for eccentricity computation. Real world scale-free graphs such as social networks usually have a

very small diameter. Therefore, approximate algorithm may lead to undesirable errors for such networks.

Other Graph Centrality Measures. In addition to graph eccentricity, there are some other famous graph centrality measures. For example, closeness centrality, which is the inverse of the average shortest distance from the vertex to any other vertex in the graph [19], is useful to measure the efficiency of each vertex in spreading information to all other vertices. Betweenness centrality, which is the fraction of shortest paths between node pairs that pass through the target node [20], is used to measure the ability of a node to control the information flow between other nodes. A recent survey of graph centrality measures and their application in different domains can be found in [21].

VI. Experiments

Algorithms. We compare our proposed algorithms against the state-of-the-art algorithm BoundEcc [8] for exact eccentricity computation. Three node ordering strategies introduced in [8] are used as the following three baseline methods:

- Degree: The nodes are visited in non-increasing order of their degrees.
- MaxGap (and MinGap, resp.): The node with the maximum (and minimum, resp.) eccentricity upper bound and the node with minimum (and maximum, resp.) eccentricity lower bound are visited alternatively.

Our techniques include the following two methods:

- ECC: Invoke EccentricityOneNode for each node in the graph. (Algorithm 2 in Section III).
- ECC-LS: Update the eccentricity-bounds using the local-spread technique. (Algorithm 5 in Section IV).

Each of ECC and ECC-LS contains the following three phases. The first two phases are shared by ECC and ECC-LS.

- Labeling: compute the PLL structure for the graph [9].
- RefPool: compute the reference-node pool, the lists L_z for each z in the pool, and the initial $\underline{ecc}$ and $\overline{ecc}$ for each node of the graph(Algorithm 3 in Section III-B).
- Eccentricity: compute the eccentricity-distribution.

The cost were evaluated in the wall-clock time, the cut-off time was set to 24 hours. The costs of the three phases were evaluated respectively. By default, the number of reference nodes was set to be $k = 16$; otherwise, the varying number k of reference nodes ranged from 1 to 32. All algorithms were implemented in C++ and compiled with GNU GCC 4.4.7 and -O3 level optimization. All experiments were conducted on a machine with an Intel Xeon 3.1GHz CPU and 128 GB main memory running Linux (Red Hat Linux 4.4.7, 64bit).

Datasets. Our experiments were conducted on 20 real-world graphs with various properties. The first 9 graphs are online social networks. Grqc, Hepth, Hepph, Astroph, and Condmat are collaboration networks. Askubuntu and Superuser are interaction networks on the stack exchange website — nodes represent users and edges indicate the answer and comment relationships. Wiki-vote, Wiki-temporal and Wiki-talk are communication networks while Web-stanford is a web graph. All graphs are considered as undirected and connected graphs: if a graph is not connected, we used the largest connected component of the graph. The details, that are, the total number of nodes n, the total number of edges m, radius r and diameter d, of all graphs are presented in Table II. The largest diameter is 38, and the smallest diameter is 7. The average label size for PLL is no larger than 100 on most graphs. All graphs were downloaded from Stanford Large Network Dataset Collection[3] [22].

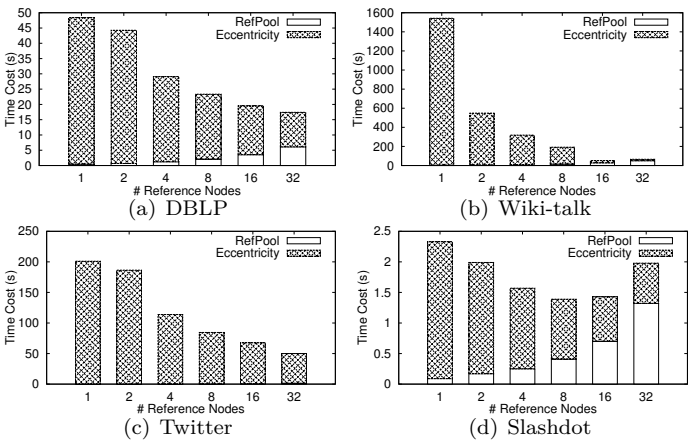

Fig. 10. Testing ECC (Varying # Reference Nodes)

Exp-1: Comparison with the State-of-the-art. This experiment compares our algorithm ECC with Degree, MaxGap, and MinGap. The results are shown in Table II.

Firstly, Table II shows that the performance of BoundEcc is sensitive to the node order. For instance, MinGap needs only 0.56 seconds to compute the results of Facebook while the other two methods need 13.05 seconds and 7.1 seconds, respectively. The performance of our proposed method is not sensitive to the node order.

Secondly, Table II indicates that our method ECC outperforms BoundEcc on all types of graphs by up to three orders of magnitude. ECC is, on average, 203.03 times faster than BoundEcc. For example, on Brightkite, ECC is more than 162 times faster than MinGap while MinGap is the best node order of BoundEcc among the three. On Wiki-temporal, ECC is 467 times faster than MaxGap.

For Youtube, our ECC algorithm had completed the eccentricity computation in 10 minutes while the state-of-the-art algorithms had not terminated within 24 hours.

Finally, we observe that, in the three phrases of our proposed algorithm ECC, the time for Labeling dominates the overall cost for most of the graphs. The actual time used for Eccentricity is no more than 100 seconds on all graphs. This means that ECC can be naturally scaled to handle larger graphs upon an accelerated labeling method.

In a nutshell, the results in Table II demonstrate that our proposed method is superior to the state-of-the-art methods.

Exp-2: Testing ECC.This experiment shows the performance of ECC under a varying number k of reference nodes. k ranges from 1 to 32. Since the time for Labeling is independent of the number of reference nodes, we only report the processing time for RefPool and Eccentricity. The experimental results are shown in Fig. 10. Due to the space limitation, we show the experimental results for four representative large graphs — DBLP, Wiki-talk, Twitter, and Slashdot.

Fig. 10 indicates that the time for RefPool increases with an increasing k; and the time for Eccentricity decreases with

[3]http://snap.stanford.edu/data/

TABLE II
DATA SET DESCRIPTION AND COMPARISON WITH THE STATE-OF-THE-ART METHODS

Statistical Information					Proposed Algorithm ECC (Sec)				BoundEcc (Sec)			PLL
Dataset	n	m	r	d	Labeling	RefPool	Eccentricity	Total	Degree	MaxGap	MinGap	Avg Label Size
Facebook	4 039	88 234	4	8	0.38	0.04	0.01	0.43	13.05	7.1	0.56	53.73
Brightkite	56 739	212 945	9	18	3.48	0.49	0.13	4.1	2222.23	614.38	664.78	64.87
Epinions	75 877	405 739	8	15	4.61	0.62	0.3	5.53	5052.3	175.66	163.87	59.18
Slashdot	77 360	469 180	6	12	7.18	0.73	0.65	8.56	3838.07	541.12	555.21	66.89
Twitter	81 360	1 342 296	4	7	14.95	0.85	68.29	84.09	6221.46	1268.25	1354.23	70.17
Gowalla	196 591	950 327	8	16	31.82	2.13	0.36	34.31	30764.32	2078.67	1060.48	91.57
DBLP	317 080	1 049 866	12	23	272.58	3.45	14.53	290.56	72798.26	2647.04	2945.94	263.57
Youtube	1 134 890	2 987 624	12	24	260.52	14.19	8.17	282.88	–	–	–	115.04
Flicker	1 624 992	15 476 835	12	24	3090.03	22.09	16.22	3128.34	–	7003.67	6165.31	353.29
Grqc	4 158	13 422	9	17	0.1	0.03	0.05	0.18	8.16	0.83	0.84	53.11
Hepth	8 638	24 806	10	18	0.28	0.07	0.04	0.39	35.81	2.56	2.72	60.56
Hepph	11 204	117 619	7	13	0.79	0.08	0.13	1	109.03	11.3	12.76	63.44
Astroph	17 903	196 972	8	14	2.06	0.15	0.21	2.42	279.23	29.84	30.56	75.35
Condmat	21 363	91 286	8	15	1.3	0.2	0.11	1.61	277.08	24.82	25.01	69.19
Askubuntu	152 599	453 221	7	13	6.45	1.58	0.29	8.32	17981.26	193.13	187.17	54.56
Superuser	189 191	712 870	6	12	9.73	1.98	0.74	12.45	29158.28	2051.17	2134.05	57.35
Wiki-vote	7 066	100 736	4	7	0.48	0.06	0.05	0.59	45.95	3.49	4.26	51.41
Wiki-temporal	1 091 742	2 786 764	5	9	54.16	12.14	23.43	89.73	–	41945.19	40614.19	56.02
Wiki-talk	2 388 953	4 656 682	6	11	132.34	27.87	22.17	182.38	–	9431.34	9281.13	61.68
Web-stanford	266 388	2 228 348	5	9	122.86	3.46	4.33	130.65	71121.36	3848.26	3917.98	143.89

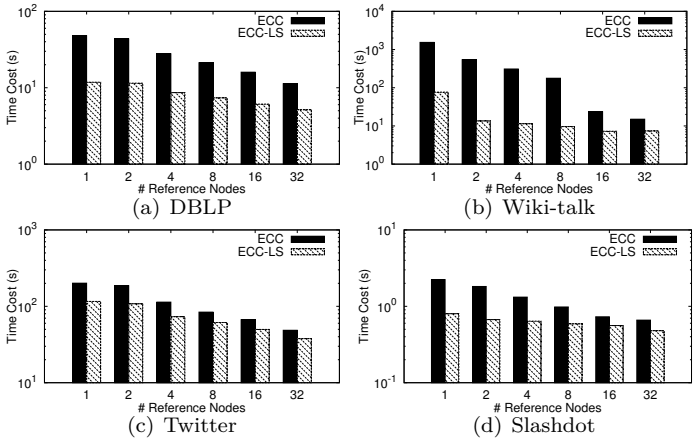

Fig. 11. Testing ECC-LS (Processing Time for Eccentricity)

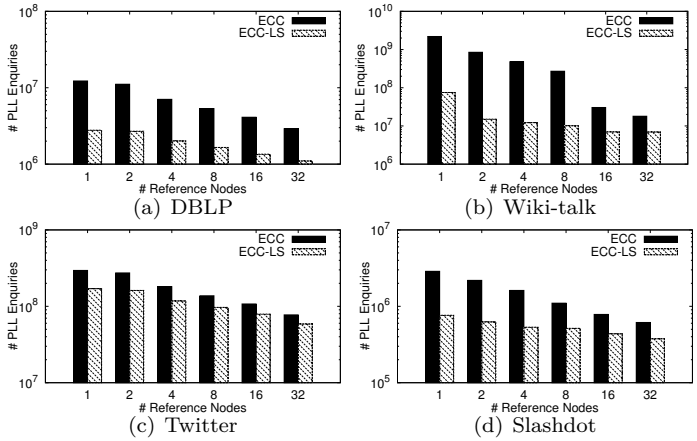

Fig. 12. Testing ECC-LS (# PLL Inquiries)

the increasing k. k reference nodes incur k BFSs, thus the time of RefPool increases with k. The efficiency of Eccentricity for one node u is dependent on the distance $dist(u, z)$ from u to its reference node z (Theorem 4) while an enlarged the reference-node pool decreases $dist(u, z)$. Therefore, increasing k reduces the cost of Eccentricity.

Fig. 10 also shows a trade-off between RefPool and Eccentricity in ECC. For graphs Wiki-talk and Slashdot, the running time first decreases and then increases with an increasing k; for graphs DBLP and Twitter, the running time drops with an increasing k. The results over all graphs suggest that $k = 16$ is a reasonable number of reference nodes to balance the cost for RefPool and Eccentricity.

Exp-3: Testing ECC-LS. This experiment examines the local-spread technique by comparing ECC with ECC-LS. Since ECC and ECC-LS have the same costs for Labeling and RefPool, we only report the cost of the third phase — Eccentricity.

Fig. 11 reports the processing time on four representative graphs DBLP, Wiki-talk, Twitter, and Slashdot when varying the number k of reference nodes. Firstly, the processing time for both ECC and ECC-LS increases with an increasing k. Secondly, local-spread speeds up the Eccentricity by tightening the eccentricity-bounds: ECC-LS outperforms ECC on Eccentricity by a factor of 2 to 10.

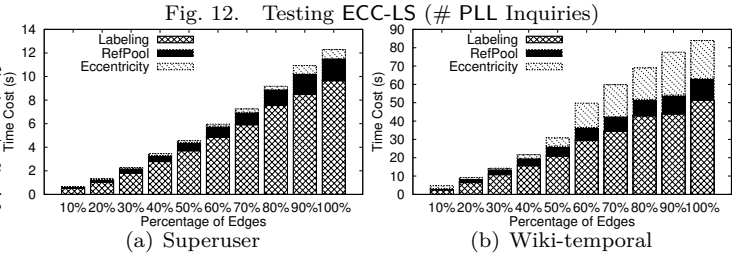

Fig. 13. Scalability Testing

We also report the number of PLL inquiries for ECC and ECC-LS in Fig. 12 when varying the number of reference nodes. The trend is similar to that for the processing time in Fig. 11. For example, for the DBLP dataset, when the number of reference nodes is 4, ECC is 3 times slower than ECC-LS while the number of PLL inquiries for ECC is 3 times larger than ECC-LS. This shows that the processing time for Eccentricity is proportional to the number of PLL inquiries.

Exp-4: Scalability Testing. Temporal graphs Superuser and Wiki-temporal were selected for the scalability test. Edges of a temporal graph are associated with timestamps. For each graph, edges were sorted in the ascending order of their

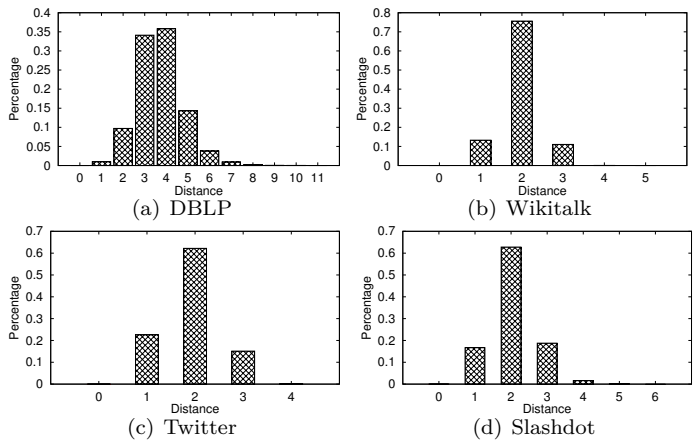

Fig. 14. Testing Distribution of Distance to Reference Nodes

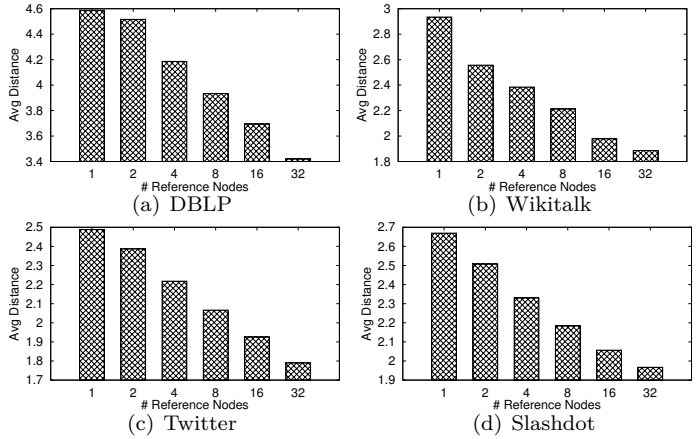

Fig. 15. Testing Average Distance to Reference Nodes

timestamps. The first 10%, 20%, ..., 100% of the sorted edges consisted a series of 10 generated-graphs, respectively, with increasing sizes. Each generated-graph is a real-world graph with the similar graph properties as the original graph.

Fig. 13 demonstrates the processing time for the three phases Labeling, RefPool, and Eccentricity, respectively, of our algorithm ECC, on the three graphs. When the graph size increases, the processing time for all three phases Labeling, RefPool, and Eccentricity increases. The three phrases Labeling, RefPool, and Eccentricity share a linear trend with an increasing graph size. This indicates that ECC has a high scalability. The curves for ECC-LS in scalability are similar to those of ECC, which are not shown in the paper due to the space limitation.

Exp-5: Testing Distance to Reference Nodes. Fig. 14 shows the distribution of the distance $\lambda_0(u)$ from a node u to its reference node z on four representative graphs DBLP, Wikitalk, Twitter, and Slashdot. The reference node z of u is the node that is nearest to u in the reference-node pool. The number k of reference nodes was set to be 16. For DBLP, most distances fall into the range of $[2, 5]$. For the other three graphs, more than 60% of nodes u have $\lambda_0(u) \leq 2$; for all the nodes u in the graph, $\lambda_0(u) \leq 6$.

Fig. 15 illustrates the average distance of a node to its nearest reference node when varying the number of reference nodes k from 1 to 32 on the four graphs. The average distance decreases with an increasing k on all graphs. When there is only one reference node ($k = 1$), the average distance is 4.6 for DBLP and less than 3 on the other three graphs. When $k = 32$, the average distance decreases to less than 3.5 for DBLP and less than 2 on the other three graphs. This result justifies the claim in Section III-B.

Exp-6: Comparison with an Approximate Method. The necessity of an exact eccentricity computation can be justified by comparing the approximate radius, diameter, and eccentricities obtained from a method suggested by the SNAP website[4] with the exact ones, respectively. This method samples 1000 random nodes and estimates the eccentricity of each node v in the graph as v's largest distance to the sample nodes, denoted as $\widetilde{ecc}(v)$. The approximate radius and diameter are derived from $\widetilde{r} = \min_{v \in V} \widetilde{ecc}(v)$ and $\widetilde{d} = \max_{v \in V} \widetilde{ecc}(v)$.

The approximation is measured in the five ratios below. Correct ratio CR is the percentage of the nodes in the graph whose eccentricities are correctly estimated. The average ratio and minimum ratio are defined, respectively, as $\mathsf{AR} = \frac{1}{n} \sum_{v \in V} \frac{\widetilde{ecc}(v)}{ecc(v)}$ and $\mathsf{MR} = \min_{v \in V} \frac{\widetilde{ecc}(v)}{ecc(v)}$. The radius ratio and diameter ratio are $\mathsf{RR} = \widetilde{r}/r$ and $\mathsf{DR} = \widetilde{d}/d$, respectively.

Fig. 16 shows the five ratios in percentage over the 20 datasets. It is observed that this approximation method only works well when the size of the sample, that is, 1000, is comparable to the size of the graph n; when n increases, the performance deteriorates severely. For example, the Facebook graph with 4039 nodes enjoys a precise estimation while the larger graphs such as Flicker, wiki-temporal, Wiki-talk, and Gowalla suffer an almost zero CR value: few nodes know their correct eccentricities. This observation applies to diameter and radius as well. On Youtube, the errors of radius and diameter reach 4 and 5, respectively, intolerable given the real radius 12 and diameter 24. This echoes our motivation: an efficient exact eccentricity computation is highly demanded.

VII. Conclusions

We provided, in this paper, a spectrum of insights into the bottleneck of existing approaches on the eccentricity computation of a graph. These insights led to a suite of techniques on eccentricity computation on undirected and unweighted graphs. The superior efficiency has been confirmed by a comprehensive experimental evaluation. Actually, all our techniques are adaptable to directed and unweighted graphs, the corresponding techniques are left to the extended version due to the space limit. We think that developing an efficient eccentricity computation approach is important to big graph processing and analysis and will look into this problem on dynamic graphs in future.

Acknoledgement

Lu Qin is supported by ARC DP160101513. Ying Zhang is supported by ARC FT170100128 and DP180103096. Lijun Chang is supported by ARC DP160101513 and DE150100563. Xuemin Lin is supported by NSFC 61672235, DP170101628 and DP180103096.

References

[1] D. West, *Introduction to Graph Theory*. Prentice Hall, 2001.
[2] T. M. Chan, "All-pairs shortest paths for unweighted undirected graphs in *o(mn)* time," in *SODA*, 2006, pp. 514–523.

[4] https://snap.stanford.edu/data/index.html

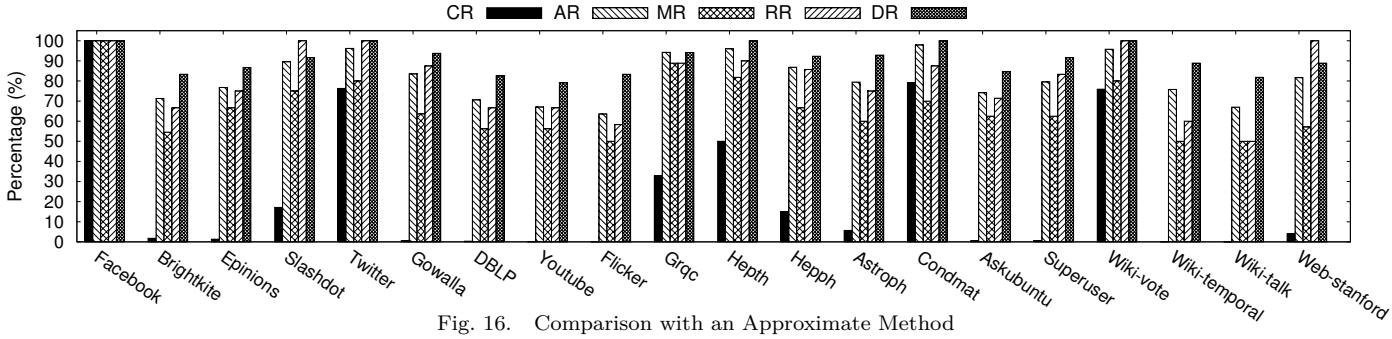

Fig. 16. Comparison with an Approximate Method

[3] R. Williams, "Faster all-pairs shortest paths via circuit complexity," in *STOC*, 2014, pp. 664–673.

[4] L. Roditty and V. V. Williams, "Fast approximation algorithms for the diameter and radius of sparse graphs," in *STOC*, 2013, pp. 515–524.

[5] S. Chechik, D. H. Larkin, L. Roditty, G. Schoenebeck, R. E. Tarjan, and V. V. Williams, "Better approximation algorithms for the graph diameter," in *SODA*, 2014, pp. 1041–1052.

[6] D. J. Watts and S. H. Strogatz, "Collective dynamics of'small-world'networks." *Nature*, vol. 393, no. 6684, pp. 409–10, 1998.

[7] J. Guare, *Six Degrees of Separation: A Play*, ser. Vintage Series. Vintage Books, 1990.

[8] F. W. Takes and W. A. Kosters, "Computing the eccentricity distribution of large graphs," *Algorithms*, vol. 6, no. 1, pp. 100–118, 2013.

[9] T. Akiba, Y. Iwata, and Y. Yoshida, "Fast exact shortest-path distance queries on large networks by pruned landmark labeling," in *SIGMOD*, 2013, pp. 349–360.

[10] D. B. Johnson, "Efficient algorithms for shortest paths in sparse networks," *JACM*, vol. 24, no. 1, pp. 1–13, 1977.

[11] P. S. Almeida, C. Baquero, and A. Cunha, "Fast distributed computation of distances in networks," in *Proc. of CDC'12*, 2012, pp. 5215–5220.

[12] M. Then, M. Kaufmann, F. Chirigati, T. Hoang-Vu, K. Pham, A. Kemper, T. Neumann, and H. T. Vo, "The more the merrier: Efficient multi-source graph traversal," *VLDB*, vol. 8, no. 4, pp. 449–460, 2014.

[13] K. Henderson, "Opex: Optimized eccentricity computation in graphs," Lawrence Livermore National Laboratory (LLNL), Livermore, CA, Tech. Rep., 2011.

[14] M. Borassi, P. Crescenzi, M. Habib, W. A. Kosters, A. Marino, and F. W. Takes, "Fast diameter and radius bfs-based computation in (weakly connected) real-world graphs: With an application to the six degrees of separation games," *Theor. Comput. Sci.*, vol. 586, pp. 59–80, 2015.

[15] F. W. Takes and W. A. Kosters, "Determining the diameter of small world networks," in *CIKM*, 2011, pp. 1191–1196.

[16] T. Akiba, Y. Iwata, and Y. Kawata, "An exact algorithm for diameters of large real directed graphs," in *ESA*, 2015, pp. 56–67.

[17] D. Aingworth, C. Chekuri, P. Indyk, and R. Motwani, "Fast estimation of diameter and shortest paths (without matrix multiplication)," *SIAM J. of Comp.*, vol. 28, no. 4, pp. 1167–1181, 1999.

[18] J. Shun, "An evaluation of parallel eccentricity estimation algorithms on undirected real-world graphs," in *SIGKDD*, 2015, pp. 1095–1104.

[19] K. Okamoto, W. Chen, and X. Y. Li, "Ranking of closeness centrality for large-scale social networks," *Lecture Notes in Computer Science*, vol. 5059, pp. 186–195, 2008.

[20] M. E. Newman, "A measure of betweenness centrality based on random walks," *Social networks*, vol. 27, no. 1, pp. 39–54, 2005.

[21] L. Lü, D. B. Chen, X. L. Ren, Q. M. Zhang, Y. C. Zhang, and T. Zhou, "Vital nodes identification in complex networks," *Physics Reports*, vol. 650, pp. 1–63, 2016.

[22] J. Leskovec and A. Krevl, "Snap datasets: Stanford large network dataset collection," 2015.

Appendix

Proof of Lemma 9. Let a shortest path from u to x be $\langle u_0, u_1, u_2, \cdots, u_k\rangle$ with $u_0 = u$, $u_k = x$ and $k = dist(x, u)$. Since the new snapshot is taken on a stable state, therefore, consider edges (u_0, u_1), (u_1, u_2), $\cdots$, (u_{k-1}, u_k), we have

$$\overline{ecc}_{new}(u) = \overline{ecc}_{new}(u_0) \leq \overline{ecc}_{new}(u_1) + 1$$
$$\leq \overline{ecc}_{new}(u_2) + 2 \leq \cdots ... \leq \overline{ecc}_{new}(u_k) + k = \overline{ecc}_{new}(x) + k.$$

Therefore, $\overline{ecc}_{new}(u) \leq \min\{\overline{ecc}_{old}(u), \overline{ecc}_{new}(x)+dist(x,u)\}$. Similar proof can be applied on showing that

$$\underline{ecc}_{new}(u_0) \geq \underline{ecc}_{new}(u_1) - 1 \geq \cdots \geq \underline{ecc}_{new}(u_k) - k.$$

By plugging $u_0 = u$, $u_k = x$ and $k = dist(x, u)$ in the above inequality, we complete the proof.

Proof of Lemma 10. Observe that in Step 2) of the iterative-update, $\overline{ecc}(l)$ of node l will be updated only if $\overline{ecc}(r)$ of its neighbor r is small enough such that $\overline{ecc}(l) > \overline{ecc}(r)+1$. Once the update takes place, we conceptually associate with $\overline{ecc}(l)$ a source $\overline{ecc}(l).s \leftarrow r$ to record the source of the bound. Note that this source field may be overwritten upon a subsequent update; however, it will not be removed once created.

$\overline{ecc}_{new}(y).s$ exists since $\overline{ecc}(y)$ must have been updated to let $\overline{ecc}_{new}(y) < \overline{ecc}_{old}(y)$. Now, we trace from y via the source link of $\overline{ecc}_{new}(\cdot).s$, generating a path $\langle u'_0, u'_1, \cdots, u'_{k'}\rangle$ with $u'_0 = y$, $u'_i = \overline{ecc}_{new}(u'_{i-1}).s$, for each $i \in [1, k']$ while $\overline{ecc}_{new}(u'_{k'})$ does not have a source. Note that in this sequence we have $\overline{ecc}_{new}(u'_{i-1}) = \overline{ecc}_{new}(u'_i) + 1$ for all $i \in [1, k']$, thus the sequence cannot contain a loop and thus $k' \leq n$. We have

$$\overline{ecc}_{new}(y) = \overline{ecc}_{new}(u'_0) = \overline{ecc}_{new}(u'_{k'}) + k'.$$

We argue that $u'_{k'}$ must be the trigger node x. Since if otherwise, $\overline{ecc}_{new}(u'_{k'})$ has no source means that $\overline{ecc}_{new}(u'_{k'}) = \overline{ecc}_{old}(u'_{k'})$. Therefore, $\overline{ecc}_{old}(y) > \overline{ecc}_{new}(y) = \overline{ecc}_{old}(u'_{k'}) + k'$. According to pigeon principle, there must be $\exists j \in [1, k']$ such that $\overline{ecc}_{old}u'_{j-1} > \overline{ecc}_{old}u'_j+1$—violating the assumption that the old snapshot is stable.

The fact that $u'_{k'} = x$ implies three important results:

1) For any node u'_j with $j \in [0, k')$, $\overline{ecc}_{new}(u'_j) < \overline{ecc}_{old}(u'_j)$. Since if otherwise, the path would have stopped at j rather than k'.
2) $\overline{ecc}_{new}(x) = ub_x < \overline{ecc}_{old}(x)$. Since if $\overline{ecc}_{new}(x) = \overline{ecc}_{old}(x)$, there will be a violation to the assumption that the old snapshot is stable. Besides, $\overline{ecc}_{new}(x)$ has no source, thus it has not been updated in Step 2) of the iterative-update. Therefore,
$$\overline{ecc}_{new}(x) = \min\{\overline{ecc}_{old}(x), ub_x\} = ub_x < \overline{ecc}_{old}(x).$$
3) $\langle u'_0, u'_1, \cdots, u'_{k'}\rangle$ is a shortest path from y to x. Since k' is the length of a path from x to y, thus $k' \geq dist(x, y)$. Based on Lemma 9, that is, $\overline{ecc}_{new}(y) \leq \overline{ecc}_{new}(x) + dist(x, y)$, it can be assured that $k' = dist(x, y)$ since $\overline{ecc}_{new}(y) = \overline{ecc}_{new}(x) + k'$.

From the above three results, we complete the proof.