Eccentricity Heuristics through Sublinear Analysis Lenses

Tal Wagner*

Abstract

The eccentricity of a node in a graph is its maximal shortest-path distance to any other node. Computing all eccentricities is a basic task in large-scale graph mining. Shun (KDD 2015) empirically studied two simple heuristics for this task: k-$\mathbf{BFS_1}$, based on parallel BFS from a small sample of nodes, was shown to work well on a variety of graphs; k-$\mathbf{BFS_2}$, a two-phase version, was shown to outperform state-of-the-art algorithms by up to orders of magnitude. This empirical success stands in apparent contrast to recent theoretical hardness results on approximating all eccentricities (Backurs et al., STOC 2018).

This paper aims to formally explain the performance of these heuristics, by studying them through computational models designed for sublinear time or sublinear space algorithms. We use the proposed framework to derive improved variants, which retain their practicality while having better performance and formal guarantees.

1. For k-$\mathbf{BFS_1}$, we draw a connection to diameter property testing (Parnas and Ron, Random Struct Alg. 2002). It is not hard to observe that k-$\mathbf{BFS_1}$ essentially tests the values of all eccentricities simultaneously, in the classical property testing sense. We show that the same guarantee is achieved by a more efficient algorithm, whose work is nearly linear in the number of nodes and independent of the number of edges. By utilizing the connection in the opposite direction, we also obtain some results on classical testing of the graph radius and diameter.
2. For k-$\mathbf{BFS_2}$, we draw a connection to the streaming Set Cover algorithm of Demaine et al. (DISC 2014). We use it to suggest a variant of k-$\mathbf{BFS_2}$ with similar work and depth bounds, which is guaranteed to compute almost all eccentricities exactly, if the graph satisfies a condition we call *small eccentric cover*. The condition can be ascertained for all real-world graph used in Shun (KDD 2015) and in our experiments.

Our experimental results on real-world graphs demonstrate the validity of our analysis and the empirical advantage of the proposed algorithms.

1 Introduction

The eccentricity of a node in a graph is the longest shortest-path distance to any other node reachable from it. This notion encompasses much useful information on the graph structure. The graph diameter is the maximal eccentricity, and the radius is the minimal one. A node is considered as central if its eccentricity is small, or as peripheral if it is large. Eccentricities play a key role in topology analysis of computer, social and biological networks [24, 20, 31, 27], and also arise in hardware verification [25] and sparse linear system solving [16].

A simple approach to computing all eccentricities in a graph is to perform a breadth-first search (BFS) from each node in a graph. This requires $O(nm)$ work on a graph with n node and m edges, which is prohibitively costly for real-world large-scale graphs. As a result, a large body of research has been dedicated to developing approximate or heuristic algorithms, both in theory [28, 9, 7, 2] and in practice [20, 30, 19].

In a recent work, Shun [30] conducted an empirical study of state-of-the-art algorithms for computing all eccentricities in undirected graphs. The study also included two simple heuristics, referred to henceforth as k-$\mathbf{BFS_1}$ and k-$\mathbf{BFS_2}$.[1] They are based on computing a full BFS from a small sample of nodes, chosen at random in k-$\mathbf{BFS_1}$ and somewhat more strategically in k-$\mathbf{BFS_2}$. We describe these methods in more detail later. The experimental results showed that both algorithms perform surprisingly well, with k-$\mathbf{BFS_1}$ outperforming some of the algorithms in the literature, and k-$\mathbf{BFS_2}$ outperforming all of them by large margins. These findings naturally raise questions about a possible formal analysis of these methods. This is the motivation for the present work.

Devising such an analysis is faced with certain notable limitations. Arguably the most standard approach to analyzing approximation algorithms is by proving worst-cast multiplicative approximation bounds on the estimates they produce. However, for graph eccentricities, a recent line of work known as *fine-grained complexity* [35] was able to establish that any substantial improvement over the currently known bounds would require an extreme breakthrough in complexity theory. Specifically, any such improvement would refute the Strong Exponential Time Hypothesis (SETH) [18],[2] whose resolution is widely thought to be well out

*CSAIL, MIT. Email: `talw@mit.edu`.

[1]In [30] they are called *k-BFS-1Phase* and *k-BFS* respectively.

[2]SETH postulates that for any $\delta \in (0, 1)$, there is a suffciently large k such that deciding if an input n-variable k-CNF formula is satisfiable (the k-SAT problem) cannot be solved in time $2^{(1-\delta)n}$.

of present theoretical reach (see eg. [34]). Most recently, this limitation was shown to apply directly to the state-of-the-art multiplicative approximation algorithms tested in [30], and it is now known that any better approximation factor would require asymptotically more work, unless SETH is false [1, 2].

There are a number of analytic ways to circumvent this barrier. One is to use a relaxed notion of approximation guarantee instead of multiplicative approximation. One such popular notion is *ε-closeness.* Loosely speaking, instead of requiring the estimate to be close to the true value, it requires the graph to be close to a graph where the estimate is correct. Graphs are considered close if only a small fraction of their edges need to be changed (added or removed) in order to obtain one from the other. This notion forms much of the basis of Property Testing, a research field largely motivated by computational settings that require extreme efficiency, where the input is too large to even read in full. It is by now an established field, with a large body of literature and recognized theoretical depth (see the textbook [17]). This makes it an appealing framework of study. This is the approach we will take for k-$\mathbf{BFS_1}$.

Another way around the aforementioned limitations on worst-case analysis is to introduce structural assumptions on the input graph. Real-world graphs are well-known to exhibit distinct structural properties, and many empirically successful heuristic algorithms capitalize exactly on those properties, which means their behavior cannot be fully captured by analyses that apply to arbitrary graphs. Nonetheless, a formal definition of these properties tends to be elusive, and one has to be careful to rely on assumptions which are actually plausible for real-world graphs, or better yet, can be ascertained for graphs "in the wild". This is the approach we will take for k-$\mathbf{BFS_2}$.

It should be noted that when fitting an analytic framework to algorithms that have been devised with a practical mindset, it is often necessary to introduce some modifications to the algorithm. This is on one hand an advantage, as it points to improvements to the algorithms that may otherwise not be apparent. On the other hand, it is important to not lose sight of the practical aspects. In our context, k-$\mathbf{BFS_1}$ and k-$\mathbf{BFS_2}$ are appealing in practice not only due to their asymptotic complexity and perceived simplicity, but also due to certain implementational aspects which are typically abstracted-away in theoretical studies. These include bit-level optimizations [33, 30, 22], GPU-based implementations [29, 23], and distributed implementations [8, 15]. Therefore, an additional goal is to analyze variants which are as close as possible to the empirically tested methods, and retain their practical plausibility.

1.1 Overview of [30]

To better motivate our results, we first describe the results of [30] in more detail.

Algorithms. The tested methods in [30] included three simple heuristics:

- k-$\mathbf{BFS_1}$ samples k uniformly random nodes as *sources*, and runs a full BFS from each source, thereby computing the distance from every source to every node in the graph. The eccentricity of every node is estimated as its largest distance to a source.
- k-$\mathbf{BFS_2}$ uses the estimates of k-$\mathbf{BFS_1}$ to select the k nodes with the largest estimates, and performs a second phase of BFS with those k nodes as sources. The eccentricity of every node is estimated as its largest distance to any source from either the first or the second phase.
- SimpleApx is another notably simple heuristic. It runs a BFS from a single arbitrary node v^*, thereby computing its true eccentricity $e(v^*)$, and uses that value as the estimate for all other eccentricities. The triangle inequality on the graph shortest-path metric implies that $\frac{1}{2}e(v) \le e(v^*) \le 2e(v)$ for every node v, yielding a constant-factor approximation. Note that the estimated eccentricities of all nodes are the same.

If the input graph has n nodes and m edges, then k-$\mathbf{BFS_1}$ and k-$\mathbf{BFS_2}$ require $O(km)$ work each, while SimpleApx is faster and requires only $O(m)$ work. In addition to these, the literature-based methods tested in [30] can be divided into three categories:

- Exact algorithms: The naïve algorithm that runs a BFS from each node, and a different one due to [32].
- Constant-factor approximations: These algorithms have a guarantee of the form $\alpha \cdot e(v) \le \hat{e}(v) \le \beta \cdot e(v)$ for every node v, where $\hat{e}(v)$ is their output eccentricity estimate and $\beta > \alpha > 0$ are constants. They include SimpleApx and two additional methods [28, 9] with better approximation factors. Both are substantially more involved and require $\tilde{O}(m\sqrt{n})$ work.
- Counter-based algorithms: These are two methods arising from the empirical literature [20], without formal guarantees on their estimates. They are based on the probabilistic counters of [14, 13].

The depth of all of the above algorithms is roughly proportional to the graph diameter.

Experimental results. Generally speaking, the results in [30] showed that in terms of average relative error ($\frac{1}{n}\sum_v \frac{|e(v)-\hat{e}(v)|}{e(v)}$), k-$\mathbf{BFS_1}$ is superior to the counter-based algorithms,[3] but inferior to SimpleApx. The

[3]It should be noted that the algorithm of [20] was designed to

costlier constant-factor approximation algorithms and the exact algorithms were found to be accurate on the smaller graphs but infeasible on the larger ones, effectively affirming their theoretical merit but also their unsuitability for large-scale graph mining. k-$\mathbf{BFS_2}$ outperformed all other methods by up to orders of magnitude in running time and/or accuracy.

These results naturally raise several questions. Most immediately, what is the reason for the striking performance of k-$\mathbf{BFS_2}$? Shun offers the intuition that the first BFS phase identifies "periphery" nodes in the graph, meaning ones that are far from many other nodes, and the second phase capitalizes on them to produce accurate eccentricity estimates. We will aim to formalize this intuition.

The situation regarding k-$\mathbf{BFS_1}$ is perhaps more subtle. It is empirically and analytically inferior to SimpleApx, in terms of both quality and running time, which may give the impression that it is not interesting on its own right. Indeed, it is included in [30] primarily to substantiate the importance of the second phase in k-$\mathbf{BFS_2}$. Nonetheless, it does produce meaningful results: for example, its median average relative error is 7.55% over the 10 tested graphs, outperforming some more sophisticated algorithms. It also somewhat outperforms k-$\mathbf{BFS_2}$ on two synthetic graphs, attaining similar accuracy in less time due to the absence of the second phase. The reason is apparently that these graphs are highly symmetric and have no distinct periphery nodes for k-$\mathbf{BFS_2}$ to exploit. Lastly, it routinely shows up in empirical studies, which tend to heavily prioritize simple algorithms due to various implementational constraints (e.g. [8, 15]; see also Section 6). Combined with its fundamental nature, it calls for some deliberation.

1.2 Our Results

k-$\mathbf{BFS_1}$. For this algorithm we take a property testing view, motivated by [26] who analyzed a related algorithm for testing the graph diameter (which is the maximal eccentricity). It is not hard to see that the estimates $\hat{e}(v)$ of k-$\mathbf{BFS_1}$ satisfy $\hat{e}(v) \le e(v) \preceq_\varepsilon \hat{e}(v)$ for every node v, where "$\preceq_\varepsilon$" denotes that the input graph is ε-close to satisfying the inequality. Our focus is in attaining the same guarantee with better work and depth bounds, and in particular better than SimpleApx which has an arguably stronger guarantee. We show a variant, called k-$\mathbf{BFS_{TST}}$, that attains it with $\tilde{O}(n/\varepsilon^2)$ work and $\tilde{O}(\varepsilon^{-1}\log n)$ depth. The work is sublinear in the graph size, and asymptotically better than k-$\mathbf{BFS_1}$ and SimpleApx, for non-trivially dense graphs.

estimate a somewhat different notion of eccentricity than the one it is used to compute in [30].

k-$\mathbf{BFS_2}$. Here we cast the problem as a Set Cover instance with limited access to the input. With this view, we draw a close connection between k-$\mathbf{BFS_2}$ and the streaming Set Cover algorithm of [10]. As a result, we suggest a variant called k-$\mathbf{BFS_{SC}}$, with similar work and depth bound, which is guaranteed to compute the *exact* eccentricities of almost all nodes, as long as the input graph satisfies a property we call *small eccentric cover*. This property can be ascertained with good parameters for all real-world graphs used in the experiments of [30] as well as in ours. We also give a robust variant that computes approximate eccentricities for almost all nodes under a relaxed condition.

Experiments. We test our methods on publicly available real-world graphs. Our experimental results show that they can improve the performance of k-$\mathbf{BFS_1}$ and k-$\mathbf{BFS_2}$ by up to an order of magnitude.

Perspective. Much of the point in these results is that the proposed algorithms k-$\mathbf{BFS_{TST}}$ and k-$\mathbf{BFS_{SC}}$ are close variants of their respective counterparts k-$\mathbf{BFS_1}$ and k-$\mathbf{BFS_2}$. Namely, k-$\mathbf{BFS_{TST}}$ is derived from k-$\mathbf{BFS_1}$ by truncating each BFS by a certain rule, and k-$\mathbf{BFS_{SC}}$ is derived from k-$\mathbf{BFS_2}$ by a drop-in replacement of its top-k selection step with an off-the-shelf parallel greedy Set Cover algorithm (eg. [5, 6]), leaving the two BFS phases unchanged.

The implication is twofold: First, the fact that our modifications lead to variants that retain and improve the performance of the original heuristics may serve as evidence that our analytic framework indeed captures what makes them work in practice. Second, it implies that our analyzable variants are plausible for implementation and practical use. Both k-$\mathbf{BFS_1}$ and k-$\mathbf{BFS_2}$ interact with the input graph only via BFS – an operation which has been minutely optimized in various system settings as mentioned earlier [33, 30, 22, 29, 23] – and our modifications require no change to it. This is particularly relevant for k-$\mathbf{BFS_2}$, since it is highly successful in practice, and our experiments show that our variant can significantly improve its performance.

On a more general level, the aim of this paper is to highlight connections between these parallel heuristics and other models of resource-constrained computation – particularly streaming, sketching and property testing. Similar algorithms to k-$\mathbf{BFS_1}$ and k-$\mathbf{BFS_2}$ have been studied in [26] and [10] (respectively), albeit in quite different contexts, and this may justify attention.

Paper organization. Section 2 sets up preliminaries and notation. Section 3 deals with k-$\mathbf{BFS_2}$. Section 4 deals with k-$\mathbf{BFS_1}$. Section 5 presents experimental results. Section 6 reviews additional related work. Appendix A elaborates on additional aspects of testing eccentricities.

Algorithm 1 : k-$\mathbf{BFS_{SC}}$

```
Input: Graph $G(V,E)$, integer $k > 0$
Output: Eccentricity estimate $\hat{e}(v)$ for every $v \in V$
                                                      // Phase 1:
$U_1 \leftarrow k$ nodes chosen i.i.d. uniformly at random
foreach $u \in U_1$:
    Compute a BFS started at $u$
foreach $v \in V$:
    $\tilde{A}_v \leftarrow \{u \in U_1 : e(u) = \Delta(u,v)\}$
$\mathcal{I} \leftarrow$ Set Cover instance with elements $U_1$ and sets $\{\tilde{A}_v\}_{v \in V}$
$\mathcal{C} \leftarrow$ cover for $\mathcal{I}$   // using parallel greedy Set Cover
                                                      // Phase 2:
$U_2 \leftarrow$ the set of nodes such that $\mathcal{C} = \{\tilde{A}_u : u \in U_2\}$
foreach $u \in U_2$:
    Compute a BFS started at $u$
foreach $v \in V$:
    return $\hat{e}(v) = \max_{u \in U_1 \cup U_2} \Delta(v,u)$
```

2 Preliminaries and Notation

Throughout we assume that the input graph $G(V,E)$ has n labeled nodes and m edges, and is undirected, unweighted and connected. The connectivity assumption can be removed by applying any of the algorithms under discussion to each connected component separately. The shortest-path distance between two nodes $v, u \in V$ is denoted by $\Delta(v,u)$. For $v \in V$ and $r > 0$, the r-neighborhood of v is defined as $N_r(v) = \{u \in V : \Delta(v,u) \le r\}$. The eccentricity of v is defined as $e(v) = \max_{u \in V} \Delta(v,u)$. We use $\hat{e}(v)$ to denote an eccentricity estimate by an algorithm which would be understood by context. The diameter of the graph is the maximal eccentricity and will be denoted $\mathcal{D} = \mathcal{D}(G)$.

Complexity bounds on all algorithms are stated in the work/depth model. The graph representation is assumed to be such that given a node we can enumerate over its neighbors. More specifically, we will only require the ability to start a BFS at a node given its label. We assume that a standard parallel BFS takes $O(n+t)$ work and $O(d \log n)$ depth if it traverses up to t edges and d levels. We use $\tilde{O}(f)$ for $O(f \cdot \text{polylog} f)$, and $[\ell]$ for $\{1, \ldots, \ell\}$ for an integer $\ell > 0$. We say "with high probability" when the probability is at least $1 - 1/n$.

3 k-$\mathbf{BFS_2}$ by Set Covering

In this section we analyze a variant of k-$\mathbf{BFS_2}$, called k-$\mathbf{BFS_{SC}}$, given in Algorithm 1. It will be shown to return accurate eccentricity estimates for graph that satisfy a property we call *small eccentric cover*, defined next.

Definition 3.1. *We say that a graph $G(V,E)$ has* eccentric cover *of size κ if κ is the smallest integer that satisfies the following: there exists $U \subset V$ of size κ such that for every $v \in V$, $e(v) = \Delta(v,u)$ for some $u \in U$.*

Put simply, this property states that all node eccentricities can be realized as distances to a subset of κ nodes. As a warm-up, one may observe that a path, star, clique and perfect binary tree all have eccentric covers of size 2, regardless of their size; an n-node hypercube has eccentric cover of size n; and an n-node cycle has eccentric cover of size n if n is even, or $\frac{1}{2}(n+1)$ if n is odd.

Obviously if the input graph has eccentric cover of size κ, then there is a realization of k-$\mathbf{BFS_1}$ and k-$\mathbf{BFS_2}$ with $k \ge \kappa$ that computes all eccentricies exactly, given the right choice of BFS sources. The question is how to identify those sources, and more specifically, how does k-$\mathbf{BFS_2}$ apparently succeed in finding them.

To answer this, we point out that if the all-eccentricities problem is viewed as a Set Cover instance, then k-$\mathbf{BFS_2}$ is seen to be closely related to the streaming Set Cover algorithm of [10]. By making this connection exact, we obtain k-$\mathbf{BFS_{SC}}$ with the following guarantee.

Theorem 3.1. *Suppose $G(V,E)$ has eccentric cover of size at most κ. Let $\varepsilon > 0$. Then for $k = \tilde{O}(\varepsilon^{-1}\kappa \log n)$, k-$\mathbf{BFS_{SC}}$ runs in $O(km)$ expected work and $O(\mathcal{D} \log n + \log^3(kn))$ expected depth, and with high probability computes the exact eccentricities of all but an ε-fraction of the nodes in V.*

In Section 3.4 we also give a variant of the theorem for approximate computation of the eccentricities, under a weaker notion of eccentric cover.

The applicability of the above result hinges on whether the small eccentric cover property occurs in real-world graph. Here it is worth noting that the property can be ascertained for all 8 real-world graphs considered in [30],[4] as a byproduct of the experiments therein. In particular, whenever k-$\mathbf{BFS_2}$ computes a $(1-\varepsilon)$-fraction of the eccentricities exactly, it certifies that the eccentric cover has size at most $\varepsilon n + 2k$. Thus, 7 of the graphs in [30], containing between $1M$ to $4M$ nodes, were shown to have eccentric cover size of only few thousands (between 0.1% to 0.6% of their nodes). For two of them the size is as small as 128 nodes. The 8th graph has eccentric cover containing 4.4% of its nodes. These are upper bounds obtained as a byproduct, and it remains possible that the true parameters are even smaller.

[4]This count does not include 4 additional graphs whose true eccentricities were not computed in [30] due to their large size.

We also remark one does not need to estimate κ in order to invoke the algorithm; it suffices to set k directly, as it acts as a smooth handle on the time to accuracy trade-off (similarly to how k-$\mathbf{BFS_2}$ is used in [30]).

3.1 Set Cover Formulation

Recall that in the Set Cover problem, we are given a set of elements $\mathcal{E}$ and a collection of subsets $\mathcal{S} \subset 2^{\mathcal{E}}$. We call $\mathcal{C} \subset \mathcal{S}$ a *cover* if $\mathcal{E} \subset \cup_{A \in \mathcal{C}} A$. The goal is to find a cover of minimum size. For $\rho \geq 1$, we say that a cover $\mathcal{C}$ is *ρ-approximate* if $|\mathcal{C}|$ is at most ρ times the size of an optimal cover.

Computing all eccentricites can be cast as a Set Cover instance as follows. For every $u \in V$ define

$$A_u = \{v \in V : e(v) = \Delta(v, u)\},$$

i.e., A_u contains the nodes whose eccentricity is attained by their distance to u. The Set Cover instance is formed by the elements $\mathcal{E} = V$ and sets $\mathcal{S} = \{A_u : u \in V\}$.

The connection to the BFS-based heuristics is seen if given $U \subset V$, we define for every $v \in V$,

$$e_U(v) = \max_{u \in U} \Delta(u, v).$$

Then, k-$\mathbf{BFS_1}$ estimates $e(v)$ as $\hat{e}(v) = e_{U_1}(v)$ where U_1 is a random sample of k nodes, and k-$\mathbf{BFS_2}$ uses $\hat{e}(v) = e_{U_1 \cup U_2}(v)$ where U_2 consists of the k nodes $v \in V$ that maximize $e_{U_1}(v)$. Denoting

$$\mathcal{C}_U = \{A_u : u \in U\},$$

we see that if $\mathcal{C}_U$ covers v then $e(v) = e_U(v)$. Therefore, computing all eccentricities (exactly) as $e_U(v)$ reduces to solving the above Set Cover instance with a cover $\mathcal{C}_U$. Furthermore, observe that the optimal cover size is precisely the eccentric cover size, as per Definition 3.1.

Let us highlight the non-standard computational constraints of this Set Cover setting, that arise if we consider it for computing all eccentricities. Given the index $u \in V$ of a set A_u, it is prohibitive to compute which elements are contained in A_u, as that already entails knowing all eccentrities. Given an element $v \in V$, it is expensive but non-prohibitive to compute which subsets contain it, as that requires a single full BFS started at v. Hence we can afford it for only a small number of elements.

3.2 Relation to DIMV

While we are not aware of any Set Cover algorithms that were explicitly designed for these constraints, there is in fact one that meets them: the streaming Set Cover algorithm of [10], referred to henceforth as DIMV. This is somewhat incidental, and indeed other streaming Set Cover algorithms do not meet these constraints. Another interesting fact is that k-$\mathbf{BFS_2}$ turns out to be closely related to DIMV, as we explain next.

DIMV is a combination of two modules: The *set sampling* module simply includes random sets in the output cover. The *element sampling* module chooses a small random sample of elements, and computes a cover only for the sample using an offline black-box algorithm (eg. greedy). Note that set sampling is a vanilla module that need not know anything about which sets cover which elements, while the more informed element sampling module only needs to know which sets cover the elements in the sample. Thus both of them meet the model constraints specified above. The key observation is that k-$\mathbf{BFS_1}$ is just set sampling, while k-$\mathbf{BFS_2}$ corresponds to a combination of set and element sampling, as follows:

- Phase 1 runs a BFS from each node u in a random sample U_1. This implicitly computes which subsets $\{A_v\}_{v \in V}$ cover u.
- Phase 2 computes U_2 as the k nodes that maximize $e_{U_1}(v)$. This is akin to computing a cover $\mathcal{C}_{U_2}$ of U_1.[5]

Thus we interpret the top-k selection step in k-$\mathbf{BFS_2}$ as a heuristic Set Cover step. Indeed, the node v^* with the largest estimate $e_{U_1}(v^*)$ satisfies $e(u^*) = \Delta(u^*, v^*)$ for some $u^* \in U$, and thus A_{v^*} covers u^*. However, the node in V with the second-largest $e_{U_1}(v)$ estimate might redundantly cover u^* again, and so on, ultimately leaving some elements uncovered.

To avoid such degeneracy, and make the connection to DIMV exact (which would allow us to leverage its formal analysis), all we need is to replace the covering heuristic by an actual Set Cover algorithm. Fortunately, this is a well-studied problem in parallel computing. In particular, we can use the parallel greedy algorithm of [5], which guarantees an approximation factor of $O(\log |\mathcal{E}|)$ (essentially optimal unless P = NP [12, 11]) and has been tested for implementation [5, 6]. This leads to k-$\mathbf{BFS_{SC}}$.

It is worth asking whether such degeneracy in the covering step of k-$\mathbf{BFS_2}$ in fact shows up in practice, or in other words, whether we expect k-$\mathbf{BFS_{SC}}$ to improve over k-$\mathbf{BFS_2}$ empirically. Here we note that this exact phenomenon has been recently discussed in [19], who report observing it in many large real-world graphs, and take heuristic measures to mitigate its adverse effect on

[5]Note that the final estimates of k-$\mathbf{BFS_2}$ are $e_{U_1 \cup U_2}(v)$, which correspond to the cover $\mathcal{C}_{U_1} \cup \mathcal{C}_{U_2}$ rather than just $\mathcal{C}_{U_2}$. Hence each $u \in U_1$ plays a dual role, of a sample element to cover in the second phase as well as a set A_u in the final cover. The first phase is thus seen to function as both element sampling and set sampling, though our analysis will only rely on the element sampling role.

the accuracy of k-**BFS**$_\mathbf{2}$. Equipped with the Set Cover formulation, our approach avoids the issue altogether.

3.3 Proof of Theorem 3.1

k-**BFS**$_\mathbf{SC}$ uses the BFS results of the first phase to compute the sets $\tilde{A}_v = A_v \cap U_1$ for every $v \in V$. This is possible since those sets are fully determined by the distances $\Delta(u, v)$ for all $u \in U_1$ and $v \in V$. It then approximately solves the Set Cover instance with elements U_1 and sets $\{\tilde{A}_v\}_{v \in V}$, say using [5], obtaining a cover $\mathcal{C}$. The BFS sources for the second phase are chosen as $U_2 = \{v \in V : \tilde{A}_v \in \mathcal{C}\}$. The final estimates returned by the algorithm are $\hat{e}(v) = e_{U_1 \cup U_2}(v)$. Noting that for every $U \subset U' \subset V$ and every $v \in V$ it holds that $e_U(v) \le e_{U'}(v)$, we have in particular,

$$e_{U_2}(v) \le \hat{e}(v) \le e(v). \tag{3.1}$$

DIMV uses the element sampling for Set Cover instances whose optimal cover is small. In the context of eccentricities, this translates to having small eccentric cover. Specifically, the Element Sampling Lemma of [10], when phrased for eccentricities as per Section 3.1, states the following.

Lemma 3.1. *Suppose $G(V, E)$ has eccentric cover of size κ. Let $\varepsilon > 0$ and $\rho \ge 1$. Let $k \ge 3\varepsilon^{-1}\rho\kappa \log n$. Let U_1 be a sample of k uniformly random nodes, and let $U_2 \subset V$ be such that $\mathcal{C}_{U_2}$ is a ρ-approximate cover of U_1. Then, with probability at least $1 - \frac{1}{n}$, $\mathcal{C}_{U_2}$ covers all but εn of the nodes in V.*

We include the short proof from [10] for completeness:

Proof. For $U \subset V$, call $\mathcal{C}_U$ "bad" if it covers less than $(1 - \varepsilon)n$ nodes in V. The probability that a bad $\mathcal{C}_U$ covers U_1 is at most $(1 - \varepsilon)^{|U_1|} \le (\frac{1}{n})^{3\rho\kappa}$. The number of subsets $U \subset V$ of size $|U| \le \rho\kappa$ is upper-bounded by $n^{\rho\kappa+1}$. Hence, by a union bound, the probability that any bad $\mathcal{C}_U$ with $|U| \le \rho\kappa$ covers U_1 is at most $(\frac{1}{n})^{3\rho\kappa} \cdot n^{\rho\kappa+1} \le \frac{1}{n}$.

The eccentric cover size is the optimal cover size of V, and an optimal cover of U_1 is no larger. Since C_{U_2} is a ρ-approximate cover of U_1, we have $|U_2| \le \rho\kappa$. Thus $\mathcal{C}_{U_2}$ is bad with probabiliy at most $1/n$. ◻

Since the Set Cover algorithm of [5] has an approximation factor of $\rho = O(\log k)$ for an instance with k elements, we can invoke the lemma if we choose k such that $k = \Omega(\varepsilon^{-1}\kappa \log k \cdot \log n)$, for which $k = \tilde{O}(\varepsilon^{-1}\kappa \log n)$ suffices. The lemma implies that $e(v) = e_{U_2}(v)$ for at least $(1 - \varepsilon)n$ of the nodes in V. Combining this with Equation (3.1), we get $e(v) = \hat{e}(v)$ for each such node, proving the correctness guarantee of Theorem 3.1.

As for the work and depth, since the Set Cover instance in k-**BFS**$_\mathbf{SC}$ has k elements and n sets, the algorithm of [5] runs in $O(kn)$ expected work and $O(\log^3(kn))$ expected depth.[6] This is added to the $O(km)$ work and $O(\mathcal{D} \log n)$ depth of the BFS phases.

3.4 Approximate Computation

So far the discussion has been restricted to exact computation of almost all eccentricities. Nonetheless, it extends to approximate computation in a natural way, by introducing a relaxed notion of eccentric cover.

Definition 3.2. *Let $\delta > 0$. We say that a graph $G(V, E)$ has δ-*approximate eccentric cover* of size κ if κ is the smallest integer that satisfies the following: there exists $U \subset V$ of size κ such that for every $v \in V$, $e(v) \le (1 + \delta)\Delta(v, u)$ for some $u \in U$.*

The definition of the Set Cover instance from Section 3.1 is modified accordingly. For every $v \in V$ we define the set $A_v^\delta = \{u \in V : e(u) \le (1 + \delta)\Delta(u, v)\}$. Note that A_v is contained in A_v^δ, and thus this Set Cover instance is "easier" than the one in Section 3.1. Furthermore, given $U \subset V$, if v is covered by $\mathcal{C}_U^\delta = \{A_u^\delta : u \in U\}$, then $\frac{1}{1+\delta}e(v) \le e_U(v) \le e(v)$.

Let k-**BFS**$^\delta_\mathbf{SC}$ be the modified variant of k-**BFS**$_\mathbf{SC}$ that uses the results of the first phase to compute the sets $\tilde{A}_v^\delta = A_v^\delta \cap U_1$ for all $v \in V$, instead of $\tilde{A}_v = A_v \cap U_1$, and uses them for the Set Cover instance.

Theorem 3.2. *Let $\varepsilon, \delta > 0$. Suppose $G(V, E)$ has δ-approximate eccentric cover of size at most κ. Then for $k = \tilde{O}(\varepsilon^{-1}\kappa \log n)$, k-**BFS**$^\delta_\mathbf{SC}$ runs in $O(km)$ expected work and $O(\mathcal{D} \log n + \log^3(kn))$ expected depth, and with high probability returns estimates $\{\hat{e}(v)\}_{v \in V}$ such that $\frac{1}{1+\delta}e(v) \le \hat{e}(v) \le e(v)$ for all but an ε-fraction of V.*

The proof is the same as for k-**BFS**$_\mathbf{SC}$; the only difference is in relaxing the eccentric cover condition into a weaker one, which is met by more graphs.

4 k-BFS$_1$ by Property Testing

In this section we focus on k-**BFS**$_\mathbf{1}$. At first glance, the algorithm might seem too naïve for non-trivial analysis. However, in [26], a similar algorithm has been analyzed for the related purpose of property testing of the graph diameter. This motivates us to consider k-**BFS**$_\mathbf{1}$ from a property testing point of view, extending the approach of [26] from the diameter to all eccentricities.

We use $e(v) \preceq_\varepsilon r$ to denote that at most εn edges need to be added to the graph to decrease the eccentricity of v to at most r. The eccentricity

[6]In comparison, the top-k selection step of k-**BFS**$_\mathbf{2}$ is implemented in [30] with $O(n)$ work and $O(\log n)$ depth with high probability.

estimates of k-**BFS$_1$** are formally given by $\hat{e}(v) = \min\left\{r : v \in \bigcap_{u\in U} N_r(u)\right\}$, where U is a uniformly random subset of k nodes. The following claim is not hard to establish.

CLAIM 4.1. *For $k = O(\varepsilon^{-1}\log n)$, with high probability, the estimates $\hat{e}(v)$ of k-**BFS$_1$** satisfy $\hat{e}(v) \le e(v) \preceq_\varepsilon \hat{e}(v)$ for every $v \in V$.*

Proof. Let $R_v = \left\{r \in [n] : v \in \bigcap_{u\in U} N_r(u)\right\}$, so $\hat{e}(v) = \min R_v$. Note that $e(v) \in R_v$, hence $\hat{e}(v) \le e(v)$. If $r \in [n]$ does not satisfy $e(v) \preceq_\varepsilon r$, then more than εn nodes $u \in V$ satisfy $\Delta(v,u) > r$, and such u will appear in U with probability at least $1-(1-\varepsilon)^k \ge 1-\exp(-\varepsilon k)$, rendering $r \notin R_v$. Setting $k = \lceil 3\varepsilon^{-1}\log n\rceil$ allows for a union bound over all $v \in V$ and all $r \in [n]$. □

While this guarantee holds for arbitrary graphs, it seems rather weak for the computational cost of k-**BFS$_1$**, particularly in light of the better performance of k-**BFS$_2$** and SimpleApx. Thus our focus is obtaining the same guarantee by a significantly more efficient algorithm. This is achieved by k-**BFS$_{\text{TST}}$**, given in Algorithm 2. It is similar to k-**BFS$_1$** except that it truncates each BFS at the first level that contains at least $\tau = O(\varepsilon^{-1}\log(1/\varepsilon))$ nodes. (This level may still contain $\Omega(n)$ nodes, and we scan it fully, so each BFS takes up to linear work in n.[7]) Formally, it uses the estimates $\hat{e}(v) = \min\left\{r : v \in \bigcap_{u\in U:|N_{r-1}(u)|<\tau} N_r(u)\right\}$. Its guarantees are summarized in the next theorem.

THEOREM 4.1. *Let $G(V,E)$ be a graph, and $\varepsilon > 0$. For $k = O(\varepsilon^{-1}\log n)$, k-**BFS$_{\text{TST}}$** runs in $O(\varepsilon^{-2}\log(1/\varepsilon)\cdot n\log n)$ work and $O(\varepsilon^{-1}\log(1/\varepsilon)\log n)$ depth, and with high probability, returns estimates $\{\hat{e}(v)\}_{v\in V}$ such that $\hat{e}(v) \le e(v) \preceq_\varepsilon \hat{e}(v)$ for every $v \in V$.*

As an intermediate step, we start by presenting an algorithm for testing the eccentricity of a given node in the classical property testing setting.

4.1 Property Testing Model

We now define the property testing model that we will use for the eccentricity tester. It is known as the *general graph model* [17], and was introduced in [26, 21]. Its name reflects the fact that it was designed to handle graphs with arbitrary degree sequences, by interpolating between two historically preceding models, one for sparse bounded-degree graphs and one for dense graph with $\Omega(n^2)$ edges.

[7]If we were to halt the BFS once $\tau = O(\varepsilon^{-1}\log(1/\varepsilon))$ nodes had been seen, without finishing scanning the current level, then we would get the weaker guarantee $\hat{e}(v) \le e(v) \preceq_\varepsilon \hat{e}(v)+1$, with $\tilde{O}(\varepsilon^{-1}n + \varepsilon^{-3})$ instead of $\tilde{O}(\varepsilon^{-2}n)$ work. See Appendix A.1.

Algorithm 2 : k-**BFS$_{\text{TST}}$**

```
**Input:** Graph $G(V,E)$, integer $k > 0$ and $\varepsilon \in (0,1)$
**Output:** Eccentricity estimate $\hat{e}(v)$ for every $v \in V$

$t \leftarrow \lceil 16/\varepsilon \rceil$
**foreach** $v \in V$:
  $\hat{e}(v) \leftarrow 0$
$U \leftarrow k$ nodes chosen i.i.d. uniformly at random
**foreach** $u \in U$:
  Start a BFS at $u$, until $t\ln t$ nodes have been
  reached. Let $\ell(u)$ be the BFS level in which this
  happened. Continue the BFS until $N_{\ell(u)}(u)$ has been
  fully scanned, then halt it.
  **foreach** $v \in V$:
    **if** $v \in N_{\ell(u)}(u)$:
      $\hat{e}(v) \leftarrow \max\{\hat{e}(v), \Delta(u,v)\}$
    **else**:
      $\hat{e}(v) \leftarrow \max\{\hat{e}(v), \ell(u)+1\}$
**foreach** $v \in V$:
  **return** $\hat{e}(v)$
```

In this model, the input is a graph $G(V,E)$ on a fixed set of n labeled nodes. A property $\mathcal{P}$ is a subset of all graphs on V. The graph satisfies the property if $G \in \mathcal{P}$. Given $\varepsilon > 0$ and an upper bound $\bar{m} = O(|E|)$ on the number of edges, G is said to be *ε-close* to satisfying $\mathcal{P}$ if there is a sequence of at most $\varepsilon\bar{m}$ edge insertions and edge deletions that results in a graph $G' \in \mathcal{P}$. Otherwise, G is *ε-far* from $\mathcal{P}$. For our needs, it will suffice to only consider edge insertions, and to simply set $\bar{m}$ to be n, which is indeed $O(|E|)$ as we assume that the graph is connected.

The goal of a testing algorithm for $\mathcal{P}$ is to return "accept" if G satisfies $\mathcal{P}$, and "reject" if G is ε-far from $\mathcal{P}$. It is allowed to err with probability at most 1/3 (over its internal randomness) in each case, and there is no requirement on its output for graphs which do not satisfy the property but are ε-close to it. The algorithm can access the graph by two types of queries:

- *Neighbor queries:* Given $u \in V$ and $i \in [n]$, the query returns the ith neighbor of u if its degree is at least i, or a null symbol "$\perp$" if its degree is smaller than i. Here we assume that the neighbors of each node are numbered arbitrarily from 1 to its degree.
- *Adjacency queries:* Given $u, u' \in V$, the query returns whether they are adjacent in G.

In practice, the ability to perform these queries efficiently depends on the representation of the input graph. We remark that while we use adjacency queries in **EccTester**, they will not be required for implementing k-**BFS$_{\text{TST}}$**, which relies only on the ability to start a BFS at a uniformly random node.

Algorithm 3 : EccTester

```
Input: $G(V,E)$; $v^* \in V$; integer $r > 0$; $\varepsilon, \eta \in (0,1)$
Output: Accept or Reject

$t \leftarrow \lceil 16/\varepsilon \rceil$
$k \leftarrow \lceil 2\ln(1/\eta)/\varepsilon \rceil$
$U \leftarrow k$ nodes chosen i.i.d. uniformly at random

foreach $u \in U$:
    Use neighbor queries to start a BFS from $u$, halted
    either when $N_{r-1}(u)$ has been fully scanned, or
    when $t\ln t$ nodes have been reached, whichever
    happens first.

    /* By now we have determined if $|N_{r-1}(u)| < t\ln t$.
    Furthermore, if this is the case, then we have fully
    scanned $N_{r-1}(u)$, thus determining whether $v^* \in N_{r-1}(u)$,
    and also computing the subset of all nodes
    which are at distance exactly $r-1$ from $u$. The
    remainder of the iterarion uses these intermediate
    computations to reject if both $|N_{r-1}(u)| < t\ln t$ and
    $v^* \notin N_r(u)$ hold, and proceeds to the next iteration
    otherwise. See Claim 4.2. */

    if $|N_{r-1}(u)| < t\ln t$ and $v^* \notin N_{r-1}(u)$:
        foreach $v' \in V$ such that $\Delta(u,v') = r-1$:
            Perform an adjancecy query for $v^*, v'$
        if $(v^*, v') \notin E$ for all such $v'$:
            return Reject

return Accept
```

4.2 Eccentricity Tester

For a fixed node $v^* \in V$, and $r \in [n]$, the property we are interested in is that the eccentricity of v^* is at most r. In this section we develop a testing algorithm for this property, given as **EccTester** in Algorithm 3. Its guarantees are summarized in the following theorem.

Theorem 4.2. *Let $G(V,E)$ be an input graph on n nodes, $v^* \in V$, $r > 0$ an integer, and $\varepsilon, \eta \in (0,1)$. **EccTester** makes $O(\varepsilon^{-3}\log^2(1/\varepsilon)\log(1/\eta))$ queries to the graph, and guarantees: If $e(v^*) \le r$ it accepts with probability 1, and if G is ε-far from satisfying $e(v^*) \le r$ it rejects with probability at least $1-\eta$.*

The algorithm builds on the graph diameter tester of [26]. Let us briefly describe it. They sample $\tilde{O}(1/\varepsilon)$ random BFS sources, and then accept if all source neighborhoods are large – namely, if each contains at least $\tilde{\Omega}(1/\varepsilon)$ nodes – or reject if any source neighborhood is smaller. The rationale is that large neighborhoods are easy to "hit" simultaneously with a small subset of nodes, which can be used to enhance the connectivity of the graph and reduce the diameter by adding only a small number of edges. Small neighborhoods, on the other hand, are difficult to hit, and if there are too many of these then we expect one to show up in a random sample, cueing us to reject the graph.

To adapt this approach for testing the eccentricity of a specific node v^*, we will also look into the content of the small neighborhoods. We accept a neighborhood either if it is large or if it contains v^*, since in the former case it can be easily "hit" (as above) and made closer to v^*, and in the latter case there is no need to "hit" it at all. We also make use of adjacency queries to remove an additive loss incurred in [26]. This is the technical readon why **EccTester** performs BFS only up to level $r-1$ and then partially queries level r, and this is the source of the BFS truncation rule of k-**BFS**$_{\mathbf{TST}}$.

Apart from yielding an eccentricity tester, these refinements allow us to slightly improve the bounds of [26] for testing the graph diameter, and to design a tester for the graph radius. They also lead to a natural model for sketching large graphs in sublinear time to support property testing queries. We discuss these ramifications in Appendix A.

4.3 Proof of Theorem 4.2

As in Algorithm 3, we let $t = \lceil 16/\varepsilon \rceil$ and $k = \lceil 2\ln(1/\eta)/\varepsilon \rceil$, and let U be a subset of k nodes chosen independently and uniformly at random from V. We start with the query complexity.

Lemma 4.1. ***EccTester** makes a total of at most $O(\varepsilon^{-3}\log^2(1/\varepsilon)\log(1/\eta))$ queries to the graph.*

Proof. For each $u \in U$, the BFS started at u is halted after at most $t\ln t$ nodes have been reached, and thus traverses at most $O((t\ln t)^2)$ edges with neighbor queries. If $|N_{r-1}(u)| < t\ln t$ then the tester also performs an adjacency query for every v' such that $\Delta(u,v') = r-1$. Since every such v' is contained in $N_{r-1}(u)$, this amounts to at most $t\ln t$ additional queries. In total, $O(k(t\ln t)^2) = O(\varepsilon^{-3}\log^2(1/\varepsilon)\log(1/\eta))$ queries. ☐

We turn to proving correctness. For brevity, we will simply say that a node $u \in U$ "rejects" if **EccTester** rejects during the iteration that starts a BFS at u. Otherwise we say that u "accepts". Note that **EccTester** returns Accept if and only if every $u \in U$ accepts.

Claim 4.2. *Each $u \in U$ rejects if and only if both $|N_{r-1}(u)| < t\ln t$ and $v^* \notin N_r(u)$ hold.*

Proof. In **EccTester**, u rejects if and only if all of the following conditions hold: (i) $|N_{r-1}(u)| < t\ln t$, (ii) $v^* \notin N_{r-1}(u)$, (iii) $(v^*, v') \notin E$ for every v' such that $\Delta(u,v') = r-1$. Conditions (ii) and (iii) together are equivalent to $v^* \notin N_r(u)$. ☐

Lemma 4.2. *If $e(v^*) \le r$, **EccTester** accepts with probability* 1.

Proof. $e(v^*) \le r$ implies that $v^* \in N_r(u)$ for every $u \in U$, thus by Claim 4.2, every $u \in U$ accepts. ∎

Lemma 4.3. *If G is ε-far from satisfying $e(v) \le r$, then **EccTester** rejects with probability at least $1 - \eta$.*

We prove this using the following technical claim.

Claim 4.3. *If every $u \in V$ satisfies either $|N_{r-1}(u)| \ge t \ln t$ or $v^* \in N_r(u)$, then the graph is $\frac{1}{2}\varepsilon$-close to satisfying $e(v^*) \le r$.*

Proof. We need to show the existence of a subset $Z \subset V$ of size at most $\frac{1}{2}\varepsilon n$, such that if we add an edge between v^* and every node in Z, the resulting graph satisfies $e(v^*) \le r$. We will use a probabilistic argument.

Let $X \subset V$ be a random subset of nodes, in which every node is included with independent probability $1/t$. Then $\mathbb{E}|X| = n/t$, and by Markov's inequality, $\Pr[|X| \ge 4n/t] \le 1/4$. Let Y be the subset of all nodes $v \in V$ that satisfy both:

- $|N_{r-1}(v)| > t \ln t$, and
- $\Delta(v, v') > r - 1$ for every $v' \in X$.

The latter event for $v \in V$ can be written as

$$\{u' \notin X \text{ for every } u' \in N_{r-1}(v)\},$$

and if $|N_{r-1}(v)| > t \ln t$, then it occurs with probability

$$(1 - 1/t)^{|N_{r-1}(v)|} \le (1 - 1/t)^{t \ln t} \le \exp(-\ln t) = 1/t.$$

Therefore $\mathbb{E}|Y| \le n/t$, and again by Markov's inequality, $\Pr[|Y| \ge 4n/t] \le 1/4$. By a union bound, with probability $1/2$ we have both $|X| \le 4n/t$ and $|Y| \le 4n/t$. We fix such X, Y and let $Z = X \cup Y$.

We add an edge from v to each node in Z. Recalling that $t = \lceil 16/\varepsilon \rceil$, we have $|Z| \le |X| + |Y| \le 8n/t \le \frac{1}{2}\varepsilon n$, and thus we add at most $\frac{1}{2}\varepsilon n$ edges. It remains to argue that after adding them, the graph satisfies $e(v^*) \le r$.

Let $u \in V$ be any node. If $|N_{r-1}(u)| \le t \ln t$ then by the claim's hypothesis, $v^* \in N_r(u)$. Otherwise, we consider two cases:

- If $N_{r-1}(u)$ intersects X, let u' be a node in the intersection. Since $u' \in X$, we have added an edge between v^* and u'. Since $u' \in N_{r-1}(u)$, we have $\Delta(u, u') \le r - 1$, thus $\Delta(v^*, u) \le r$.
- If $N_{r-1}(u)$ does not intersect X, then $u \in Y$ by construction of Y, hence we have added an edge between v^* and u.

In all cases we have $\Delta(v^*, u) \le r$ after adding the edges. As this holds for every $u \in V$, we have $e(v^*) \le r$. ∎

Claim 4.4. *If at most $8n/t$ nodes $u \in V$ satisfy both $|N_{r-1}(u)| < t \ln t$ and $v^* \notin N_r(u)$, then the graph is ε-close to satisfying $e(v^*) \le r$.*

Proof. For every $u \in V$ that satisfies the condition in the claim, we add an edge between u and v^*, thus adding at most $8n/t \le \frac{1}{2}\varepsilon n$ edges. Now we are in the setting of Claim 4.3, by which we can add $\frac{1}{2}\varepsilon n$ additional edges in order to make the graph satisfy $e(v^*) \le r$. ∎

Proof of Lemma 4.3. Suppose G is ε-far from satisfying $e(v) \le r$. By the contrapositive of Claim 4.4, there are more than $8n/t$ nodes $u \in V$ that satisfy both $|N_{r-1}(u)| \le t \ln t$ and $v^* \notin N_r(u)$. By Claim 4.2, **EccTester** accepts if and only if no such node is sampled in U. Since U is a random subset of size k, this happens with probability at most

$$\left(1 - \frac{8}{t}\right)^k = \left(1 - \frac{1}{2}\varepsilon\right)^{\lceil \frac{1}{2}\varepsilon \log(\frac{1}{\eta}) \rceil} \le \exp(-\ln(1/\eta)) = \eta.$$

Theorem 4.2 follows from Lemmas 4.1 to 4.3.

4.4 Proof of Theorem 4.1

Recall we are given an input graph $G(V, E)$ with $|V| = n$ and $\varepsilon \in (0, 1)$. We set $k = \lceil 6\varepsilon^{-1} \ln(n) \rceil$. As in Algorithm 2, we let $t = \lceil 16/\varepsilon \rceil$, and let U be a subset of k nodes chosen independently and uniformly at random from V.

Correctness. We prove the guarantee of k-**BFS$_{\mathbf{TST}}$** by arguing that it in fact performs **EccTester** for every $v^* \in V$ and $r \in [n]$ simultaneously, and for every v^*, the estimate $\hat{e}(v^*)$ equals the maximal r such that **EccTester** accepts v^*, r. The above setting of k corresponds to setting $\eta = 1/n^3$ in Thorem 4.2, which allows for a union bound over all pairs v^*, r with a total success probability of $1 - 1/n$.

Formally, fix $v^* \in V$. Note that in Algorithm 2, for every $u \in U$, $\ell(u)$ is defined as the minimal $\ell \in [n]$ such that $|N_\ell(u)| \ge t \ln t$. Let

$$\hat{e}(v^*, u) = \begin{cases} \Delta(v^*, u) & \text{if } v \in N_{\ell(u)}(u) \\ \ell(u) + 1 & \text{if } v \notin N_{\ell(u)}(u) \end{cases},$$

and note that the estimate $\hat{e}(v^*)$ returned by k-**BFS$_{\mathbf{TST}}$** for v^* satisfies

$$\hat{e}(v^*) = \max_{u \in U} \hat{e}(v^*, u). \tag{4.2}$$

Let $r \in [n]$. In the same terminology as the previous section, we say that a node $u \in U$ *rejects* r if **EccTester** rejects during the iteration that starts a BFS at u while testing the property $e(v^*) \le r$. By Claim 4.2, this happens if and only if both $|N_{r-1}(u)| < t \ln t$ and $v^* \notin N_r(u)$ hold. Otherwise we say that u *accepts* r.

Claim 4.5. *u accepts r if and only if $\hat{e}(v^*, u) \le r$.*

Proof. We prove the claim by case analysis.
Case 1: $r - 1 \ge \ell(u)$. By definition of $\ell(u)$ this implies $|N_{r-1}(u)| \ge t \ln t$. Hence by Claim 4.2, u accepts r, and we need to show that $\hat{e}(v^*, u) \le r$. Indeed,

- If $v^* \in N_{\ell(u)}(u)$ then $\hat{e}(v^*, u) = \Delta(v^*, u) \le \ell(u) < r$.
- If $v^* \notin N_{\ell(u)}(u)$ then $\hat{e}(v^*, u) = \ell(u) + 1 \le r$.

Case 2: $r - 1 < \ell(u)$, hence $|N_{r-1}(u)| < t \ln t$.

- If $v^* \in N_r(u)$ then by Claim 4.2 u accepts r, so we need to show $\hat{e}(v^*, u) \le r$. Indeed, since $r - 1 < \ell(u)$ we have $v^* \in N_r(u) \subset N_{\ell(u)}(u)$, hence $\hat{e}(v^*, u) = \Delta(v^*, u)$, which is at most r since $v^* \in N_r(u)$.
- If $v^* \notin N_r(u)$ then by Claim 4.2 u rejects r, so we need to show $\hat{e}(v^*, u) > r$. Indeed, if $v^* \in N_{\ell(u)}(u)$ then $\hat{e}(v^*, u) = \Delta(v^*, u) > r$, and if $v^* \notin N_{\ell(u)}(u)$ then $\hat{e}(v^*, u) = \ell(u) + 1 > r$.

□

Since **EccTester** accepts upon testing $e(v^*) \le r$ if and only if every $u \in U$ accepts r, we have the following corollary by Equation (4.2):

Corollary 4.1. *For a fixed subset $U \subset V$ of BFS sources, **EccTester** accepts upon testing $e(v^*) \le r$ if and only if $\hat{e}(v^*) \le r$.*

The corollary implies that for a fixed subset $U \subset V$, $\hat{e}(v^*)$ is the minimal $r \in [n]$ for which **EccTester** accepts. Theorem 4.2 guarantees that **EccTester** accepts with probability 1 all r such that $e(v^*) \le r$. Furthermore, by a union bound over all values $r \in [n]$, with probability at least $1 - n\eta$ it rejects all r for which the input graph does not satisfy $e(v^*) \preceq_\varepsilon r$. Consequently, with probability at least $1 - n\eta$, $\hat{e}(v^*)$ satisfies $\hat{e}(v^*) \le e(v^*) \preceq_\varepsilon \hat{e}(v^*)$. Taking another union bound over all $v^* \in V$ guarantees this for all nodes simultaneously with probability $1 - n^2\eta$. By setting $\eta = 1/n^3$, which corresponds to the setting $k = \lceil 6\varepsilon^{-1} \ln(n) \rceil$ in Theorem 4.1, the guarantee holds for all nodes with probability at least $1 - 1/n$, as was to be shown.

Work and depth. To bound the total work, consider $u \in U$. The algorithm starts a BFS at u and halts it when $t \ln t$ nodes have been reached, which takes at most $O((t \ln t)^2)$ work. Then it proceeds to complete the level $\ell(u)$ in which the BFS was halted. This final level can have as many as $\Omega(n)$ nodes, but the total additional work required to complete it is governed by the number of edges between the nodes in it and the nodes in all previous levels. There are at most $t \ln t$ nodes in the previous levels, and hence at most $nt \ln t$ such edges. Therefore the total work of the BFS started at u is $O(nt \ln t + (t \ln t)^2)$. Summing over all $u \in U$, we have $O(k(nt \ln t + (t \ln t)^2)) = O(\varepsilon^{-2} \log(1/\varepsilon) \cdot n \log n)$ work for the main loop.

The depth of k-**BFS**$_{\mathbf{TST}}$ is governed by the depth of any BFS it performs. For every $u \in U$ the BFS is capped at depth $\ell(u)$. Since we assume the graph is connected, every BFS level reaches at least one previously unreached node, and therefore we reach $t \ln t$ nodes after at most $t \ln t$ levels. Hence $\ell(u) \le t \ln t$, and the depth of any BFS is at most $t \ln t = O(\varepsilon^{-1} \log(1/\varepsilon))$.

Graph name	*Nodes*	*Edges*	*Diam.*	*Rad.*	*Avg. $e(v)$*
Oregon-1$_{010526}$	11,174	23,409	10	5	7.15
p2p-Gnutella30	36,646	88,303	11	7	8.69
loc-Brightkite	56,739	212,945	18	9	11.75

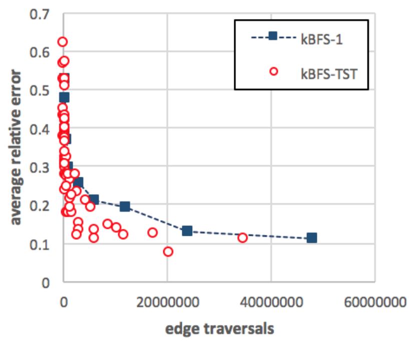
Table 1: Graphs used in experiments

5 Experiments

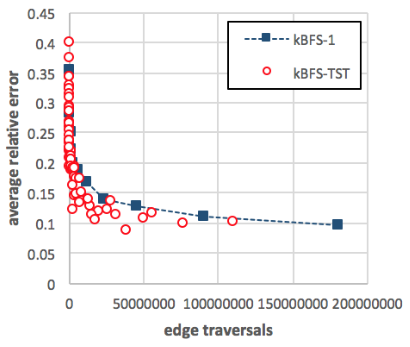
In this section we present experimental results in order to test our proposed algorithms and validate their analysis. As input graphs, we use three real-world graphs from the Stanford Network Analysis Project (available at `http://snap.stanford.edu/data/`). Their properties are summarized in Table 1. In each graph we use only the largest connected component (which contains almost all nodes), and treat all edges as undirected.

k-BFS$_\mathbf{1}$ and k-BFS$_\mathbf{TST}$. For practical convenience, our implementation of k-**BFS**$_{\mathbf{TST}}$ is parameterized by the BFS cutoff (which is $\tau = t \ln t = \Theta(\varepsilon^{-1} \log(1/\varepsilon))$ in Algorithm 2) instead of ε. We run k-**BFS**$_\mathbf{1}$ and k-**BFS**$_{\mathbf{TST}}$ with $k = 2^i$ for $i = 0, 1, ..., 10$ and with different cutoffs for k-**BFS**$_{\mathbf{TST}}$. The results are depicted in Figure 1. The x-axis counts the total number of edge traversals performed by each algorithm, as a proxy for their total work. The y-axis measures their average relative error, defined as $\frac{1}{n} \sum_{v \in V} \frac{|e(v) - \hat{e}(v)|}{e(v)}$. The results show that introducing an aggressive BFS cutoff to k-**BFS**$_\mathbf{1}$ makes it significantly more efficient at the cost of only a small decrease in accuracy, yielding an overall better performance tradeoff, as per Section 4. In particular, it seems preferable to spend a fixed budget of edge traversals on a larger number of truncated BFS invocations than on a smaller number of complete ones.

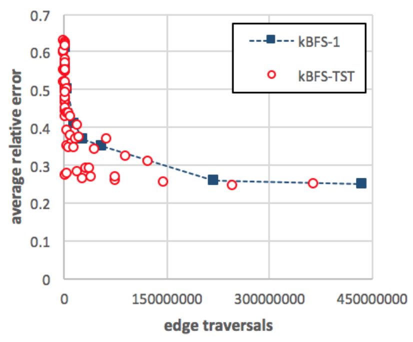
k-BFS$_\mathbf{2}$ and k-BFS$_\mathbf{SC}$. Here, for a more informed comparison, we introduce two more algorithmic variants. Note that while k-**BFS**$_\mathbf{2}$ uses k sources in the second phase, k-**BFS**$_{\mathbf{SC}}$ uses only ϕ sources, where ϕ is the cover size computed by the greedy Set Cover algorithm for the sample from the first phase. It is often the case that ϕ is much smaller than k. To equate the total work, we introduce the variant k-**BFS**$_{\mathbf{SC-full}}$, which is



(a) Oregon-1_{010526}

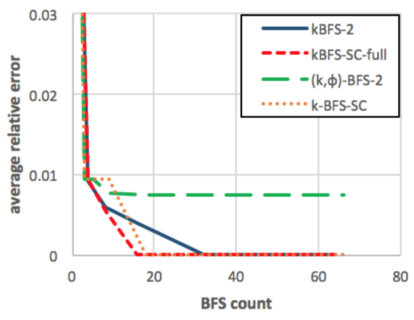


(b) p2p-Gnutella30

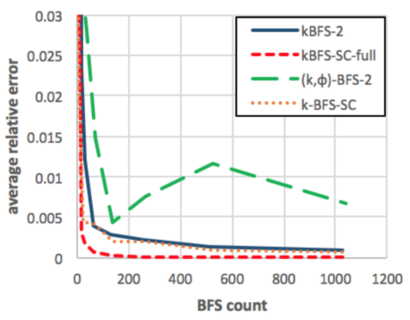


(c) loc-Brightkite

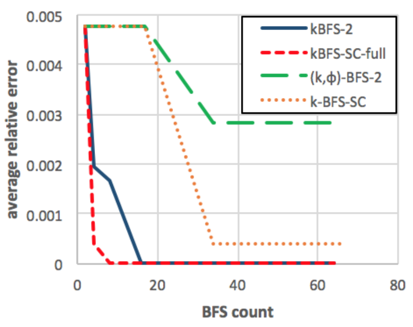
Figure 1: Evaluation of k-$\mathbf{BFS_1}$ and k-$\mathbf{BFS_{TST}}$



(a) Oregon-1_{010526}



(b) p2p-Gnutella30



(c) loc-Brightkite

Figure 2: Evaluation of k-$\mathbf{BFS_2}$, k-$\mathbf{BFS_{SC-full}}$, (k,ϕ)-$\mathbf{BFS_2}$, k-$\mathbf{BFS_{SC}}$

Graph	Eccentric cover size	k	k-$\mathbf{BFS_2}$		k-$\mathbf{BFS_{SC-full}}$		1st phase cover size (ϕ)	(k,ϕ)-$\mathbf{BFS_2}$		k-$\mathbf{BFS_{SC}}$	
			CR	*ARE*	*CR*	*ARE*		*CR*	*ARE*	*CR*	*ARE*
Oregon-1_{010526}	≤ 64	16	0.969	0.004	1	0	2	0.945	0.007	0.999	$9 \cdot 10^{-5}$
		64	1	0	1	0	2	0.945	0.007	0.999	$9 \cdot 10^{-5}$
p2p-Gnutella30	≤ 1024	16	0.802	0.022	0.988	0.001	3	0.731	0.032	0.961	0.004
		64	0.960	0.004	0.998	$2 \cdot 10^{-4}$	5	0.869	0.015	0.964	0.004
loc-Brightkite	≤ 16	16	1	0	1	0	2	0.958	0.004	0.958	0.004
		64	1	0	1	0	4	0.968	0.003	0.999	$4 \cdot 10^{-4}$

Table 2: Evaluation of k-$\mathbf{BFS_2}$, k-$\mathbf{BFS_{SC-full}}$, (k,ϕ)-$\mathbf{BFS_2}$, k-$\mathbf{BFS_{SC}}$

similar to k-$\mathbf{BFS_{SC}}$ except that in the second phase it uses the greedy strategy to choose a possibly redundant set cover of size exactly k. Thus, both k-$\mathbf{BFS_2}$ and k-$\mathbf{BFS_{SC-full}}$ use a total of $2k$ BFS sources, k in each phase, differing in how the second phase sources are chosen. k-$\mathbf{BFS_{SC}}$ uses a total of $k + \phi$ sources, k in the first phase and ϕ in the second. To complete the picture, we also include the variant (k, ϕ)-$\mathbf{BFS_2}$, which uses k sources in the first phase and ϕ sources in the second, chosen by the same rule as k-$\mathbf{BFS_2}$, i.e., by a top-ϕ selection step. For this variant, ϕ is an external parameter which we set according to the results of k-$\mathbf{BFS_{SC}}$, for the sake of comparison between them.

The results are shown in Figure 2. The x-axis counts the number of BFS invocations as a proxy for the total work. The y-axis measures their average relative error. All algorithms were run with $k = 2^i$ for $i = 0, 1, ..., 10$, though for better visual clarity, the x-axes are truncated when all plots have stabilized. Some additional numbers are given in Table 2, which in addition to the average relative error (ARE) contains the correctness ratio (CR), which is the fraction of nodes whose eccentricity was computed exactly, and an upper bound on the eccentric cover size of each graph.

The results show that k-$\mathbf{BFS_{SC-full}}$ dominates the other algorithms and converges faster to near-zero error. The greedy Set Cover selection rule for the BFS sources in the second phase is significantly preferrable to top-k selection, improving the accuracy by up to an order of magnitude.

6 Additional Related Work

Since the publication of [30], there has been some theoretical progress on constant-factor approximation algorithms for all-eccentricities. In terms of upper bounds, [2] gave a $(2+\delta)$-approximation algorithm that runs in nearly linear time $\tilde{O}(m/\delta)$. The approximation factor is almost twice better than SimpleApx. In terms of lower bounds, it was shown that a 2-approximation is optimal for nearly-linear time algorithms, and that the (5/3)-approximation algorithm of [9] (which is included in [30]) cannot be improved in neither running time nor approximation factor, unless SETH is false [1, 2].

With regard to empirical literature, it is worth pointing out that the application of the distributed *extrema propagation* technique [3, 4] to estimating all-eccentricities [8, 15] is equivalent to k-$\mathbf{BFS_1}$ with truncated BFS (albeit with a different truncation rule than k-$\mathbf{BFS_{TST}}$). Let us briefly describe this, starting with the $k = 1$ case. Each node v initially samples a random value ζ_v, say uniformly in $[0, 1]$. In every timestep, v transmits ζ_v to all its neighbors, and then updates its own ζ_v to the minimum among its current value and the values received from its neighbors. It also keeps track of the timestep τ_v in which the last update to ζ_v was made. This serves as the eccentricity estimate, i.e., $\hat{e}(v) = \tau_v$.

Let v^* be the node whose initial ζ_{v^*} is minimal. By symmetry v^* is uniformly random. After $e(v^*)$ steps the value ζ_{v^*} propagates to all nodes, and their τ_v equals their distance to v^*. Hence this is equivalent to k-$\mathbf{BFS_1}$ with $k = 1$ and v^* as the random source.

Actual extrema propagation does not use a single ζ_v, but an array $z_v = (\zeta_v^{(1)}, \ldots, \zeta_v^{(k)})$ which it is updated in every timestep by entry-wise minimum. This yields k-$\mathbf{BFS_1}$ with the k random sources $v_i^* = \mathrm{argmin}_v \zeta_v^{(i)}$ for $i = 1, \ldots, k$, where the argmin is over the initial random values $\{\zeta_v^{(i)}\}_{v \in V}$. Finally, in practice every node decides to halt its updates and output its current eccentricity estimate once its vector z_v has not been updated with new values for several consecutive steps, which yields k-$\mathbf{BFS_1}$ with truncated BFS.

Acknowledgements. The author thanks Ronitt Rubinfeld and Ali Vakilian for helpful feedback.

References

[1] A. Abboud, V. Williams, and J. Wang. Approximation and fixed parameter subquadratic algorithms for radius and diameter in sparse graphs. In *SODA*, 2016.

[2] A. Backurs, L. Roditty, G. Segal, V. Williams, and N. Wein. Towards tight approximation bounds for graph diameter and eccentricities. In *STOC*, 2018.

[3] C. Baquero, P. S. Almeida, and R. Menezes. Fast estimation of aggregates in unstructured networks. In *ICAS*, 2009.

[4] C. Baquero, P. S. Almeida, R. Menezes, and P. Jesus. Extrema propagation: Fast distributed estimation of sums and network sizes. *IEEE Transactions on Parallel and Distributed Systems*, 23(4):668–675, 2012.

[5] G. E. Blelloch, R. Peng, and K. Tangwongsan. Linear-work greedy parallel approximate set cover and variants. In *SPAA*, 2011.

[6] G. E. Blelloch, H. V. Simhadri, and K. Tangwongsan. Parallel and i/o efficient set covering algorithms. In *SPAA*, 2012.

[7] M. Cairo, R. Grossi, and R. Rizzi. New bounds for approximating extremal distances in undirected graphs. In *SODA*, 2016.

[8] J. C. Cardoso, C. Baquero, and P. S. Almeida. Probabilistic estimation of network size and diameter. In *LADC*, 2009.

[9] S. Chechik, D. H. Larkin, L. Roditty, G. Schoenebeck, R. E. Tarjan, and V. Williams. Better approximation algorithms for the graph diameter. In *SODA*, 2014.

[10] E. D. Demaine, P. Indyk, S. Mahabadi, and A. Vakilian. On streaming and communication complexity of the set cover problem. In *DISC*, 2014.

[11] I. Dinur and D. Steurer. Analytical approach to parallel repetition. In *STOC*, 2014.
[12] U. Feige. A threshold of ln n for approximating set cover. *JACM*, 45(4):634–652, 1998.
[13] P. Flajolet, É. Fusy, O. Gandouet, and F. Meunier. Hyperloglog: the analysis of a near-optimal cardinality estimation algorithm. In *DMTCS*, 2007.
[14] P. Flajolet and G. N. Martin. Probabilistic counting algorithms for data base applications. *JCSS*, 31(2):182–209, 1985.
[15] F. Garin, D. Varagnolo, and K. Johansson. Distributed estimation of diameter, radius and eccentricities in anonymous networks. In *NecSys*, 2012.
[16] A. George, J. Liu, and E. Ng. Computer solution of sparse linear systems. 1994.
[17] O. Goldreich. *Introduction to Property Testing*. Cambridge University Press, 2017.
[18] R. Impagliazzo and R. Paturi. On the complexity of k-sat. *JCSS*, 62(2):367–375, 2001.
[19] K. Iwabuchi, G. Sanders, K. Henderson, and R. Pearce. Computing exact vertex eccentricity on massive-scale distributed graphs. In *CLUSTER*, 2018.
[20] U. Kang, C. E. Tsourakakis, A. P. Appel, C. Faloutsos, and J. Leskovec. Hadi: Mining radii of large graphs. *ACM TKDD*, 5(2):8, 2011.
[21] T. Kaufman, M. Krivelevich, and D. Ron. Tight bounds for testing bipartiteness in general graphs. *SICOMP*, 33(6):1441–1483, 2004.
[22] M. Kaufmann, M. Then, A. Kemper, and T. Neumann. Parallel array-based single-and multi-source breadth first searches on large dense graphs.
[23] H. Liu, H. H. Huang, and Y. Hu. ibfs: Concurrent breadth-first search on gpus. In *SIGMOD*, 2016.
[24] D. Magoni and J.-J. Pansiot. Analysis and comparison of internet topology generators. In *International Conference on Research in Networking*, 2002.
[25] M. Mneimneh and K. Sakallah. Computing vertex eccentricity in exponentially large graphs: Qbf formulation and solution. In *SAT*, 2003.
[26] M. Parnas and D. Ron. Testing the diameter of graphs. *Random Structures & Algorithms*, 20(2):165–183, 2002.
[27] G. A. Pavlopoulos, M. Secrier, C. N. Moschopoulos, T. G. Soldatos, S. Kossida, J. Aerts, R. Schneider, and P. G. Bagos. Using graph theory to analyze biological networks. *BioData mining*, 4(1):10, 2011.
[28] L. Roditty and V. Williams. Fast approximation algorithms for the diameter and radius of sparse graphs. In *STOC*, 2013.
[29] A. Sarıyüce, E. Saule, K. Kaya, and Ü. Çatalyürek. Regularizing graph centrality computations. *Parallel and Distributed Computing*, 76:106–119, 2015.
[30] J. Shun. An evaluation of parallel eccentricity estimation algorithms on undirected real-world graphs. In *SIGKDD*, 2015.
[31] F. W. Takes and W. A. Kosters. Determining the diameter of small world networks. In *CIKM*, 2011.
[32] F. W. Takes and W. A. Kosters. Computing the eccentricity distribution of large graphs. *Algorithms*, 6(1):100–118, 2013.
[33] M. Then, M. Kaufmann, F. Chirigati, T.-A. Hoang-Vu, K. Pham, A. Kemper, T. Neumann, and H. T. Vo. The more the merrier: Efficient multi-source graph traversal. *VLDB*, 8(4):449–460, 2014.
[34] R. Williams. Some estimated likelihoods for computational complexity. *Springer LNCS 10,000*, 2018.
[35] V. Williams. On some fine-grained questions in algorithms and complexity. 2018.

Algorithm 4 : NodeObliviousEccTester

```
Input: $G(V,E)$; $v^* \in V$; integer $r > 0$; $\varepsilon, \eta \in (0,1)$
Output: Accept or Reject

$t \leftarrow \lceil 16/\varepsilon \rceil$
$k \leftarrow \lceil 2\ln(1/\eta)/\varepsilon \rceil$
$U \leftarrow k$ nodes chosen i.i.d. uniformly at random

foreach $u \in U$:
    Use neighbor queries to start a BFS from $u$, halted
    either when $N_r(u)$ has been fully scanned, or when
    $t \ln t$ nodes are reached, whichever happens first.
    if $|N_r(u)| < t \ln t$ and $v^* \notin N_r(u)$:
        return Reject

return Accept
```

A Property Testing of Eccentricities

In this section we elaborate on aspects of testing node eccentricities in the classical property testing setting.

A.1 Node-Oblivious Eccentricity Tester

We start with a simplified version of **EccTester**, called **NodeObliviousEccTester**, given in Algorithm 4. It provides a somewhat weaker guarantee than **EccTester**, with the same query complexity. On the other hand, it uses only neighbor queries, and more importantly, its queries are independent of the node v^* whose eccentricity is being tested. This will facilitate some of the result later in this section.

Theorem A.1. *Let $G(V,E)$ be an input graph on n nodes, let $v^* \in V$, let $r > 0$ be an integer, and let $\varepsilon, \eta \in (0,1)$. **NodeObliviousEccTester** makes $O(\varepsilon^{-3}\log^2(1/\varepsilon)\log(1/\eta))$ queries to the graph, and satisfies the following: If $e(v^*) \le r$ then it accepts with probability 1, and if G is ε-far from satisfying $e(v^*) \le r+1$ then it rejects with probability at least $1-\eta$.*

The proof is very similar to that of Theorem 4.2, and we refrain from repeating the details.

We remark that if we use Theorem A.1 instead of Theorem 4.2 for k-**BFS**$_{\mathbf{TST}}$, we obtain a variant with the weaker guarantee $\hat{e}(v) \le e(v) \preceq_\varepsilon \hat{e}(v) + 1$ for every

$v \in V$, but better dependence on ε in the work bound, which is $O(\varepsilon^{-1} n \log n + \varepsilon^{-3} \log^2(1/\varepsilon) \log n)$. This holds since each of the k BFS invocations traverses $O((t \ln t)^2)$ edges and thus takes $O(n + (t \ln t)^2)$ work, and updating the eccentricity estimate takes $O(kn)$ work. The total work is $O(kn + k(t \ln t)^2)$, and as in k-**BFS$_{\mathbf{TST}}$** we have $t = O(\varepsilon^{-1} \log(1/\varepsilon))$ and $k = O(\varepsilon^{-1} \log n)$.

A.2 Diameter Testing

In [26], Parnas and Ron provide a graph diameter tester that for a given $d > 0$, accepts with probability 1 if $\mathcal{D}(G) \leq d$, and rejects with probability 2/3 if G is ε-far from satisfying $\mathcal{D}(G) \leq 2d + 2$. This result also follows from Theorem A.1 (with a similar but somewhat different testing algorithm) since the triangle inequality implies that $e(v^*) \leq \mathcal{D}(G) \leq 2e(v^*)$ for every v^*, so we can invoke **NodeObliviousEccTester** with $r = d$ and an arbitrary node v^*, and return its output. Applying the same reasoning to **EccTester** and Theorem 4.2, we can remove the +2-term from the reject condition, obtaining the following somewhat improved result.

Theorem A.2. *Let $G(V,E)$ be an input graph on n nodes, let $d > 0$ be an integer, and let $\varepsilon \in (0,1)$. There is a testing algorithm that makes $O(\varepsilon^{-3} \log^2(1/\varepsilon))$ queries to the graph, and satisfies the following: If $\mathcal{D}(G) \leq d$ then it accepts with probability 1, and if G is ε-far from satisfying $\mathcal{D}(G) \leq 2d$ then it rejects with probability at least 2/3.*

All the above results pertain to testing with one-sided error, meaning that the tester accepts with probability 1 if the graph satisfies $\mathcal{D}(G) \leq d$. In [26] it is also shown how to trade two-sided error for better bounds, and obtain a tester that accepts with probability 2/3 if $\mathcal{D}(G) \leq d$ and rejects with probability 2/3 if G is ε-far from satisfying $\mathcal{D}(G) \leq d + 4$, under certain limitations on the parameter regime. The same transformation applied to Theorem A.2 improves the latter bound to $\mathcal{D}(G) \leq d + 2$. We omit further details.

A.3 Radius Testing

The radius of the graph is the minimal eccentricity, and will be denoted by $\mathcal{R}(G)$. In this section we give a testing algorithm for the graph radius. It is stated as **RadiusTester** in Algorithm 5.

Theorem A.3. *Let $G(V,E)$ be an input graph on n nodes, let $r > 0$ be an integer, and let $\varepsilon \in (0,1)$. There is a testing algorithm that makes $O(\varepsilon^{-3} \log^3(1/\varepsilon) \log\log(1/\varepsilon))$ queries to the graph, and satisfies the following: If $\mathcal{R}(G) \leq r$ it accepts with probability 1, and if G is ε-far from satisfying $\mathcal{R}(G) \leq r+1$ it rejects with probability at least 2/3.*

Algorithm 5 : RadiusTester

```
Input: Graph $G(V,E)$, integer $r > 0$, $\varepsilon \in (0,1)$
Output: Accept or Reject

$t \leftarrow \lceil 16/\varepsilon \rceil$
$k \leftarrow \lceil 2 \ln(6t \ln t)/\varepsilon \rceil$
$U \leftarrow k$ nodes chosen i.i.d. uniformly at random

foreach $u \in U$:
    Use neighbor queries to start a BFS from $u$, halted
    either when $N_r(u)$ has been fully scanned, or when
    $t \ln t$ nodes are reached, whichever happens first.
if the intersection
```

$$\bigcap_{u \in U : |N_r(u)| \leq t \ln t} N_r(u) \tag{A.1}$$

```
is non-empty, return Accept. Else, return Reject.
Note that if $|N_r(u)| > t \ln t$ for all $u \in U$, we consider
the intersection as non-empty and return Accept.
```

It is instructive to first observe that Theorem A.3 follows almost immediately from Theorem A.1, if one is willing to blow up the number of random BFS sources (and thus the query complexity) by a factor of $O(\log n)$. To this end, note that the property $\mathcal{R}(G) \leq r$ is the union of properties $e(v^*) \leq r$ for all $v^* \in V$. **RadiusTester** is identical to **NodeObliviousEccTester** except that the former checks whether the intersection in Equation (A.1) is non-empty, whereas the latter checks whether it contains v^*. Hence **RadiusTester** can be seen as invoking **NodeObliviousEccTester** simultaneously for every $v^* \in V$, with the same random sources, with the former accepting if and only if at least one invocation of the latter accepts. Here we have crucially used the fact that the queries of **NodeObliviousEccTester** are oblivious to v^*. The guarantee of Theorem A.3 follows from Theorem A.1 by setting $\eta = 1/(2n)$ in the latter, allowing for a union bound over all $v^* \in V$.

With little additional work, we can reduce the number of nodes in the union bound to $O(\varepsilon^{-1} \log(1/\varepsilon))$, obtaining Theorem A.3. As in Algorithm 5, let $t = \lceil 16/\varepsilon \rceil$ and $k = \lceil 2 \ln(6t \ln t)/\varepsilon \rceil$, and let U be a subset of $2k$ nodes chosen independently and uniformly at random from V.

Claim A.1. *If at most $8n/t$ nodes $v \in V$ have $N_r(v) \leq t \ln t$, then G is ε-close to satisfying $\mathcal{R}(G) \leq r + 1$.*

Proof. [sketch] By a probabilistic argument similar to the one in Claims 4.3 and 4.4, there exists a subset $Z \subset V$ of size $|Z| \leq \varepsilon n$ such that $Z \cap N_r(v)$ is non-empty for all $v \in V$. By picking an arbitrary $v^* \in V$

and adding edges between v^* and every node in Z, we obtain a graph in which v^* has eccentricity at most $r+1$, and hence this graph has radius at most $r+1$. □

Proof of Theorem A.3. Suppose $\mathcal{R}(G) \le r$. Then there is a node v^* with $e(v^*) \le r$, hence $v^* \in N_r(v)$ for every $v \in V$, and hence **RadiusTester** returns Accept wth probability 1.

Suppose G is ε-far from satisfying $\mathcal{R}(G) \le r+1$. Let us mentally partition U arbitrarily into two sets U_1, U_2 of size k each. Say, let U_1 be the first k sampled sources, and U_2 be the remaining k. By the contrapositive of Claim A.1, there are more than $8n/t$ nodes $v \in V$ that satisfy $N_r(v) \le t \ln t$. The probability that U_1 misses all of them is at most

$$\text{(A.2)} \quad \left(1 - \tfrac{8}{t}\right)^k = \left(1 - \tfrac{1}{2}\varepsilon\right)^k \le \exp(-\ln(6t \ln t)) < \frac{1}{6}.$$

Suppose this does not occur. Let $u^* \in U_1$ be an arbitrary node for which $|N_r(u^*)| \le t \ln t$. We now aim to upper-bound the probability that **RadiusTester** accepts, which only increases if we restrict the intersection in Equation (A.1) to $u \in \{u^*\} \cup U_2$, disregarding the rest of U_1. This is equivalent to invoking **NodeObliviousEccTester** simultaneously for every $v \in N_r(u^*)$, with the random sources in U_2. (We remark this is not true for nodes $v \notin N_r(u^*)$, since **RadiusTester** necessarily rejects them while **NodeObliviousEccTester** accepts them if $v \in \bigcap_{u \in U_2} N_r(u)$.)

Our current assumption that G is ε-far from satisfying $\mathcal{R}(G) \le r+1$ implies that G is ε-far from satisfying $e(v) \le r+1$ for every $v \in N_r(u^*)$. The choice of k in **RadiusTester** corresponds to setting $\eta = 1/(6t \ln t)$ in Theorem A.1, and hence **NodeObliviousEccTester** accepts each invocation with probability at most $1/(6t \ln t)$. By a union bound over all $v \in N_r(u^*)$, the probability that any of the invocations accepts is at most $1/6$. By a final union bound over the latter event and Equation (A.2), **RadiusTester** accepts with probability at most $1/3$, as needed.

A.4 A Model for Sketching Graphs for Testing

Let us highlight one more consequence of Theorem A.1, pertaining to sketching large graphs for testing. Sketching is an algorithmic paradigm whose goal is to summarize a large object (say, a graph) into a memory-efficient representation, called a *sketch*, such that certain parameters of the object (say, node distances or cut values) can be approximately recovered from the sketch. In this section, our goal is to combine sketching with property testing, and define a model for sketching graphs in sublinear time, so that property testing queries can be answered from the sketch.

Formally, sketching is defined in terms of a one-way communication problem. In our setting, Alice has query access to a graph $G(V,E)$ (in the model defined in Section 4.1), which she uses to produce a small-size sketch, and sends it to Bob. Bob needs to use the sketch to test a property $\mathcal{P} \in \Pi$ of G from a set of properties Π, where Π is known to both parties but $\mathcal{P}$ is known only to Bob. Π can be thought of as a set of properties parameterized by nodes, edges, cuts, and so on.

Instantiating the model for eccentricities, we let $\Pi = \{\mathcal{P}_{r,v} : v \in V, r \in [n]\}$, where $\mathcal{P}_{v,r}$ is the property that $e(v) \le r$. The fact that the queries of **NodeObliviousEccTester** are oblivious to v^*, and in a sense also to r, allows us to separate the querying part from the testing part, and obtain the following result in this model.

THEOREM A.4. *Let $G(V,E)$ be an input graph on n nodes, and let $\varepsilon \in (0,1)$. There is a randomized sketching algorithm (Alice) and a deterministic estimation algorithm (Bob), with the following guarantees:*

- *Alice makes $O(\varepsilon^{-3} \log^2(1/\varepsilon) \cdot \log n)$ queries to G.*
- *Alice computes a sketch of size $O(\varepsilon^{-2} \log(1/\varepsilon) \cdot \log^2 n)$ bits.*
- *With high probability over Alice's random coins, Bob can compute from the sketch eccentricity estimates $\{\hat{e}(v)\}_{v \in V}$ that satisfy $\hat{e}(v) \le e(v) \preceq_\varepsilon \hat{e}(v) + 1$ for every $v \in V$.*

Proof. Let $\eta = 1/n^3$. As in Algorithm 4, let $t = \lceil 16/\varepsilon \rceil$ and $k = \lceil 2 \ln(1/\eta)/\varepsilon \rceil$. Alice samples a subset $U \subset V$ of k uniformly random nodes, starts a BFS at each, and halts each BFS after $t \ln t$ nodes have been reached. (Unlike **NodeObliviousEccTester**, she does not halt when $N_r(u)$ has been fully scanned for some r.) The query complexity is $O(\varepsilon^{-3} \log^2(1/\varepsilon) \log(1/\eta))$ as in Theorem A.1. The sketch consists of the list of $t \ln t$ nodes reached from each $u \in U$ along with their distances from u. Since every node label and every distance are represented by $\log n$ bits, the sketch size is $k \cdot t \ln t \cdot 2 \log n = O(\varepsilon^{-2} \log(1/\varepsilon) \cdot \log^2 n)$ bits.

The sketch consists of all the information required for Bob to emulate **NodeObliviousEccTester** on every $v^* \in V$ and $r \in [n]$ with the random sources U, i.e., to check whether $v^* \in N_r(u)$ for every $u \in U$ for which $|N_r(u)| < t \ln t$. By Theorem A.1, with a union bound over all $v^* \in V$ and $r \in [n]$, all n^2 invocations succeed simultaneously with probability at least $1 - 1/n$. This means that each pair v^*, r is accepted if $e(v^*) \le r$ and rejected if G is ε-far from satisfying $e(v^*) \le r+1$. By setting $\hat{e}(v^*)$ to be the minimal $r \in [n]$ for which the pair v^*, r is accepted, the theorem follows. □