Computing Classic Closeness Centrality, at Scale

Edith Cohen
Microsoft Research
editco@microsoft.com

Daniel Delling
Microsoft Research
dadellin@microsoft.com

Thomas Pajor
Microsoft Research
tpajor@microsoft.com

Renato F. Werneck
Microsoft Research
renatow@microsoft.com

ABSTRACT

Closeness centrality, first considered by Bavelas (1948), is an importance measure of a node in a network which is based on the distances from the node to all other nodes. The classic definition, proposed by Bavelas (1950), Beauchamp (1965), and Sabidussi (1966), is (the inverse of) the average distance to all other nodes.

We propose the first highly scalable (near linear-time processing and linear space overhead) algorithm for estimating, within a small relative error, the classic closeness centralities of all nodes in the graph. Our algorithm applies to undirected graphs, as well as for centrality computed with respect to round-trip distances in directed graphs.

For directed graphs, we also propose an efficient algorithm that approximates generalizations of classic closeness centrality to outbound and inbound centralities. Although it does not provide worst-case theoretical approximation guarantees, it is designed to perform well on real networks.

We perform extensive experiments on large networks, demonstrating high scalability and accuracy.

1. INTRODUCTION

Closeness centrality is a structural measure of the importance of a node in a network, which is based on the ensemble of its distances to all other nodes. It captures the basic intuition that the closer a node is to all other nodes, the more important it is. Structural centrality in the context of social graphs was first considered in 1948 by Bavelas [4].

The classic definition measures the closeness centrality of a node as the inverse of the average distance from it and was proposed by Bavelas [5], Beauchamp [6], and Sabidussi [44]. On a graph $G = (V, E)$ with $|V| = n$ nodes, the centrality of v is formally defined by

$$B^{-1}(v) = (n-1)/\sum_{u \in V} d_{vu}, \tag{1}$$

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
COSN'14, October 1–2, 2014, Dublin, Ireland.
Copyright is held by the owner/author(s). Publication rights licensed to ACM.
ACM 978-1-4503-3198-2/14/10 ...$15.00.
http://dx.doi.org/10.1145/2660460.2660465.

where d_{vu} is the shortest-path distance between v and u in G. This textbook definition is also referred to as *Bavelas closeness centrality* or as the *Sabidussi Index* [26, 27, 50].

The classic closeness centrality of a node v can be computed exactly using a single-source shortest paths computation (such as Dijkstra's algorithm). In general, however, we are interested not only in the centrality of a particular node, but rather in the set of all centrality values. This is the case when centrality values are used to obtain a relative ranking of the nodes. Beyond that, the distribution of centralities captures important characteristics of a social network, such as its *centralization* [27, 50].

When we would like to perform many centrality queries (in particular when we are interested in centrality values for all nodes) on graphs with billions of edges, such as large social networks and Web crawl graphs, the exact algorithms do not scale. Instead, we are looking for scalable computation of approximate values, with small relative error.

The node with maximum classic closeness centrality is known as the 1-median of the network. A near-linear-time algorithm for finding an approximate 1-median was proposed by Indyk and Thorup [29, 47]. Their algorithm samples k nodes at random and performs Dijkstra's algorithm from each sampled node. They show that the node with minimum sum of distances to sampled nodes is with high probability an approximate 1-median of the network. The same sampling approach was also used to estimate the centrality values of all nodes [25] and to identify the top k centralities [40]. When the distance distribution is heavy-tailed, however, the sample average is a very poor estimator of the average distance: The few very distant nodes that dominate the average distance are likely to be all excluded from the sample C, resulting in a large expected error for almost all nodes.

Contributions

We present the first near-linear-time algorithm for estimating, with a small relative error, the classic closeness centralities of all nodes. Our algorithm provides probabilistic guarantees that hold for all instances and for all nodes.

Computationally, our algorithm selects a small uniform sample C of k nodes and performs single-source shortest paths computation from each sampled node. We provide a high-level description, illustrated in Figure 1, of how we use this information to estimate centralities of all nodes.

From the single-source computations, we know the distances from nodes in C to all other nodes and therefore the exact value of $B(u)$ for each $u \in C$, but we need to estimate

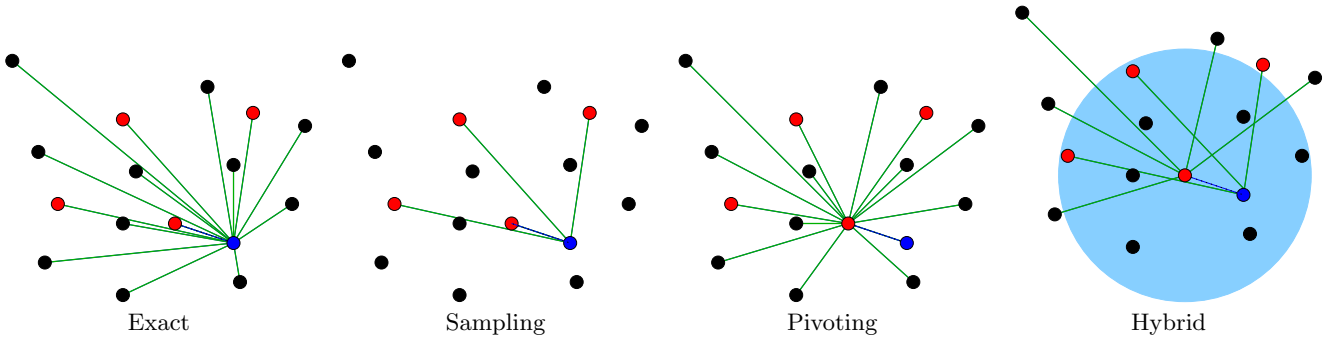

Figure 1: Exact: Average distance from blue node to all other nodes. Sampling: Average distance to sampled (red) nodes. Pivoting: Average distance from pivot (closest sampled node). Hybrid: Distances outside the threshold radius from pivot are estimated through the pivot (but distances to sampled nodes outside the threshold are exact). Shorter distances, within the threshold radius, are estimated through sampled nodes.

the centrality of other nodes. As we mentioned, a natural way to use this information is *sampling* [25, 29, 40, 47]: Estimate the centrality of a node v using the sample average $\hat{B}(v) = \sum_{u \in C} d_{vu}/k$. As we argued, however, the expected relative error can be very large when the distribution of distances from the node v to all other nodes is skewed.

A second basic approach, which we propose here, is *pivoting*, which builds on techniques from approximate shortest-paths algorithms [15, 48]. We define the *pivot* $c(v) \in C$ of a node v as the node in the sample which is closest to v. We can then estimate the centrality of v by that of its pivot, $B(c(v))$, which we computed exactly. By the triangle inequality, the value of $B(v)$ is within $\pm\, d_{vc(v)}$ of $B(c(v))$.

A large error, however, can be realized even on natural instances: The centrality of the center node in a star graph would be estimated with an error of almost 100%, using average distance of approximately 2 instead of 1. If we use the *pivoting upper bound*

$$\hat{B}(v) = B(c(v)) + d_{vc(v)}$$

as our estimator, we obtain an estimate that is about three times the value of the true average. We can show, however, that this is just about the worst case: On all instances and nodes v, the pivoting upper bound estimate is, with high probability, not much less than $B(v)$ or much more than three times the value, that is, the estimate is within a factor of 3 of the actual value. Since the argument is both simple and illuminating, we sketch it here. When the sample has size k, it is likely that the distance between v and its pivot $c(v)$ is one of the $1/k$ closest distances from v. Actually, with very high probability, $d_{vc(v)}$ is one of the $(\log n)/k$ closest distances to v. Since $B(v)$ is the average value of a set of values such that $(1 - (\log n)/k)$ of them are at least as large as $d_{vc(v)}$, we obtain that

$$B(v) \geq (1 - (\log n)/k) d_{vc(v)}. \tag{2}$$

We next apply the triangle inequality to obtain

$$B(c(v)) \leq B(v) + d_{vc(v)}. \tag{3}$$

Finally, we combine (2) and (3) to obtain that our estimate $\hat{B}(v) \equiv B(c(v)) + d_{vc(v)} \leq B(v) + 2d_{vc(v)}$ is not likely to be much larger than $3B(v)$.

Therefore, the pivoting estimator has a bounded error with high probability, regardless of the distribution of distances, a property we could not get with the sampling estimator. Neither method, sampling or pivoting, however, is satisfactory to us, since we are interested in a *small relative* error, for *all* nodes, on all instances, and with (probabilistic) guarantees.

Our key algorithmic insight is to carefully combine the sampling and pivoting approaches. When estimating centrality for a node v, we apply the pivoting estimate only to nodes u that are "far" from v, that is, nodes that have distance d_{vu} much larger than the distance to the pivot $c(v)$. The sampling approach is applied to the remaining "closer" nodes. By doing so, our hybrid approach obtains an estimate with a small relative error with high confidence, something that was not possible when using only one of the methods in isolation. Moreover, the computation needed by our hybrid algorithm is essentially the same as with the basic approaches: k single-source shortest paths computation for a small value of k. Our hybrid estimator is presented and analyzed in Section 2. The estimator is applicable to points in a general metric space and is therefore presented in this context. An efficient algorithm which computes the hybrid centrality estimate for all nodes in an undirected graphs is presented in Section 3.

The effectiveness of our hybrid estimate in practice depends on setting a threshold correctly between pivoting and sampling. Our analysis sets a threshold with which we obtain guarantees with respect to worst-case instances, i.e., for any network structure and distances distribution of a node. In our implementation, we experiment with different settings. We also propose a novel *adaptive* approach, which estimates the error for several (or effectively all relevant) choices of threshold values, on a node per node basis. The sweet spot estimate which has the smallest estimated error is then used. Our error estimator for each threshold setting and our adaptive approach are detailed in Section 4.

In applications, we are often interested in measuring centrality with respect to a particular topic or property which has a different presence at each node. Nodes can also intrinsically be heterogeneous, with different activity or importance levels. These situations are modeled by an assignment of weights $\beta(i) \geq 0$ to nodes. Accordingly, one can naturally define *weighted* classic closeness centrality of a node i as

$$B_\beta^{-1}(i) = \frac{\sum_{j \neq i} \beta(i)}{\sum_{j \neq i} \beta(i) d_{ij}}. \tag{4}$$

In Section 5, we present and analyze an extension of our algorithm designed for approximating weighted centralities. The approach is based on weighted sampling of nodes, which, for any weighting β, ensures a good approximations (small relative error) of Equation (4). The handling of weighted nodes is supported with almost no cost to scalability or accuracy when compared to unweighted instances.

In Section 6 we consider directed networks. When the graph is strongly connected, meaning that all nodes can reach all other nodes, it is often natural to consider closeness centrality with respect to *round-trip* distances. The round-trip distance between two nodes is defined as the sum $d_{uv} + d_{vu}$ of the shortest-paths distances. We show that a small modification of our hybrid algorithm, which requires both forward and reverse single-source shortest-paths computations from each sampled node, approximates round-trip centralities for all nodes with a small relative error. This follows because our hybrid estimator and its analysis apply in any metric space, and round-trip distances are a metric.

When the graph is not strongly connected, however, classic closeness centrality is not well defined: All nodes that have one or more unreachable nodes have centrality value of 0. We may also want to separately consider inbound or outbound centralities, based on outbound distances from a node or inbound distances to a node, since these can be very different on directed graphs. Proposed modification of classic centrality to directed graphs are based on a combination of the average distance within the outbound or inbound reachability sets of a node, as well as on the cardinalities of these sets [12, 35]. We therefore consider scalable estimation of these quantities, proposing a sampling-based solution which provides good estimates when the distance distribution is not too skewed.

Section 7 briefly describes other relevant related work, including other important centrality measures. The results of our experimental evaluation are provided in Section 8, demonstrating the scalability and accuracy of our algorithms on benchmark networks with up to tens of millions of nodes.

2. THE HYBRID ESTIMATOR

We present our hybrid centrality estimator, which applies for a set V of $n = |V|$ points in a metric space.

We use parameters k and ϵ, whose setting determines a tradeoff between computation and approximation quality. We sample k points uniformly at random from V to obtain a set C. We then obtain the distances d_{ij} from each point $i \in C$ to all points $j \in V$. The estimators we consider are applied to this set of nk computed distances.

Specifically, we consider estimators $\hat{S}[j]$ for $j \in V$ of the sum $S(j) = \sum_{i \in V} d_{ij}$. We then estimate the centrality of j as (the inverse of) $\hat{B}[j] \leftarrow \hat{S}[j]/(n-1)$.

For points $j \in C$, we can compute the exact value of $S(j)$, since the exact distances d_{ji} are available to all i. For $j \notin C$ we are interested in estimating $S(j)$. We define the *pivot* of j (closest node in the sample):

$$c(j) = \arg\min_{i \in C} d_{ij}$$

and the distance $\Delta(j) = d_{jc(j)}$ to the pivot.

In the introduction we discussed three basic estimators: The *sample average*

$$\hat{B}(j) = \frac{1}{k} \sum_{i \in C} d_{ij} \ , \tag{5}$$

the *pivot* estimator, $\hat{B}(j) \equiv B(c(j))$, and the *pivoting upper bound*

$$\hat{B}(j) \equiv B(c(j)) + \Delta(j) \ .$$

We argued that neither one can provide a small relative error with high probability.

The hybrid estimate $\hat{S}[j]$ for a point $j \in V \setminus C$ is obtained as follows (efficient computation is discussed in the next section). We first compute the pivot $c(j)$ and its distance $\Delta(j)$. We then partition the points $V \setminus \{j\}$ to three parts $L(j)$, $HC(j)$, and $H(j)$, where the placement of a node i is determined according to its distance $d_{ic(j)}$ from the pivot $c(j)$.

- The points $L(j)$ (L stands for "low") have distance at most $\Delta(j)/\epsilon$ from $c(j)$. The sum of distances to these points is estimated using the sum of distances to the sampled points which are in $L(j)$. Since these points are a uniform sample from $L(j)$, we compute the *effective* sampling probability $p(j) \equiv |L(j) \cap C|/|L(j)|$, and divide the sum by $p(j)$ to obtain an unbiased estimate.

- The set $HC(j)$ ("high in C") includes sampled points $i \in C$ that have distance greater than $\Delta(j)/\epsilon$ from the pivot $c(j)$. The distances from v to these points are accounted for exactly.

- The set $H(j) \subset V \setminus C$ ("high") are the points that are not sampled whose distance to the pivot $c(j)$ is greater than $\Delta(j)/\epsilon$. The sum of distances to these points is estimated by the exact sum of their distances to $c(j)$.

The estimate $\hat{S}[j]$ for $S(j)$ is thus

$$\hat{S}[j] = \sum_{i \in H(j)} d_{c(j)i} + \sum_{i \in HC(j)} d_{ji} + \frac{|L(j)|}{|L(j) \cap C|} \sum_{i \in L(j) \cap C} d_{ji}. \tag{6}$$

Since $c(j) \in L(j) \cap C$, the denominator satisfies $|L(j) \cap C| \geq 1$ and thus the estimator is well defined. It is easy to verify that the estimate $\hat{S}[j]$ for all points j can be computed from the nk distances we collected.

2.1 Quality Guarantees

We now analyse the quality of the hybrid estimator and show that the estimate $\hat{S}[j]$ has a small relative error for any point j:

THEOREM 2.1. *Using $k = 1/\epsilon^3$, the hybrid estimator* (6) *has a normalized root mean square error (NRMSE) of $O(\epsilon)$. Using $k = \frac{\log n}{\epsilon^3}$, when applying the estimator to all points in V, we get a maximum relative error of $O(\epsilon)$ with high probability.*

PROOF. We consider the error we obtain by using $\hat{S}[j]$ instead of $S[j]$ for a point $j \in V \setminus C$. Error can be accumulated on accounting for distances to $H(j)$ or to $L(j)$.

The first set, $H(j)$, includes all non-sample points that have distance greater than $\Delta(j)/\epsilon$ from $c(j)$. The accumulated error on the sum is bounded by $\pm\Delta(j)$ for each point

in $H(j)$. Since the distance from j to a point in $i \in H(j)$ is at least

$$d_{c(j)i} - \Delta(j) = \Delta(j)(1/\epsilon - 1),$$

the relative error on all of $H(j)$ is at most $\Delta(j)/[\Delta(j)(1/\epsilon - 1)] = 1/(1/\epsilon - 1) = \epsilon/(1-\epsilon)$.

We now turn to $L(j)$, where we use a sampling estimator: We estimate the sum of distances to points in $L(j)$ using the sum of distances to sample points that are in $L(j)$. The sample points constitute a random sample of $L(j)$, which includes each point in $L(j)$ with probability $p = k/n$.

We compute the variance of estimating $\sum_{i \in L(j)} d_{ji}$ using the estimate $\frac{1}{p}\sum_{i \in L(j) \cap C} d_{ji}$. Consider the ratio of the variance to the square of the sum. The ratio is maximized when the set $L(j)$ includes all points (otherwise the contribution of $H(j)$ increases the denominator but not the numerator). Therefore, since we are upper bounding the error, we can assume that the set $L(j)$ contains all points.

The points in $L(j)$ are of distance at most $\Delta(j)(1/\epsilon + 1)$ from j.

We first consider the total contribution to the centrality of the set of points A that are of distance smaller than $\Delta(j)$ from j. Since $\Delta(j)$ is the distance to the pivot, the expected number of such points is not more than n/k. Their expected total relative contribution to $B(j)$ is at most their relative fraction, which in expectation is $1/k \ll \epsilon$. Moreover, for an integer $a > 1$, the probability of there being more than an/k such points is the probability that all k sampled points selected among the $n(1-a/k)$ farthest points from j, which is at most $(1-a/k)^k \approx e^{-a}$. So the contribution of points A to centrality (and to the variance) is also well concentrated.

We now consider the contribution to variance of points that have distance between $\Delta(j)$ and $(1/\epsilon + 1)\Delta(j)$. For convenience we use $s \equiv 1/\epsilon + 1$ and $\Delta \equiv \Delta(j)$. Repeating the same argument as before, since we are computing an upper bound we can assume that this set contains all points. Given the sum of distances of these points, the "worst case" for variance is when all distances are at one of the extremes; we thus further assume that the distance of each point is either Δ or $s\Delta$. The variance contribution of a point is $(1/p - 1)$ times its distance squared. We now define $x \in [0,1]$ to be the fraction of points are of distance Δ; the remaining have distance $s\Delta$. The sum of distances is

$$n(x\Delta + (1-x)s\Delta) = n\Delta(x + (1-x)s)$$

and the variance is

$$\begin{aligned}
&((1/p) - 1)n(x\Delta^2 + (1-x)s^2\Delta^2) \\
&= ((1/p) - 1)n\Delta^2(x + (1-x)s^2) \\
&\leq \frac{n^2\Delta^2}{k}(x + (1-x)s^2).
\end{aligned}$$

We now consider the maximum over choices of n and x of the ratio of the variance to the square of the mean, which is

$$\max_{x \in [0,1]} \frac{1}{k}\frac{x + (1-x)s^2}{(x + (1-x)s)^2}.$$

This is maximized at $x = s/(s+1) = (1+\epsilon)/(1+2\epsilon)$. The maximum is $\frac{1}{k}\frac{(s+1)^2}{4s} = \frac{(1+2\epsilon)^2}{4k\epsilon(1+\epsilon)} \approx \frac{1}{4k\epsilon}$. This means that the Coefficient of Variation (CV) is about $\frac{1}{2\sqrt{k\epsilon}}$.

Balancing the sampling CV with the pivoting relative error of ϵ we obtain $k \approx \frac{1}{2\epsilon^3}$. $\square$

In our implementation, we worked with parameter settings of $\epsilon = \sqrt{k}$. This setting means that the relative error on the pivoting component is at most $\epsilon/(1-\epsilon)$. We can typically expect it to be much smaller, however. First, because distances in $H(j)$ can be much larger than $\Delta(j)/\epsilon$. Second, the estimates of different points are typically not "one sided" (the estimate is one sided when the pivot happens to be on or close to the shortest path from j to most other points), so errors can cancel out. For the sampling component, the analysis was with respect to a worst-case distance distribution, where all values lie at the extremes of the range, but in practice we can expect an error of $\approx 1/\sqrt{k} \approx \epsilon$. Moreover, when the population variance of $L(j)$ is small, we can expect a smaller relative error.

In Section 4 we propose adaptive error estimation, which for each point j, uses the sampled distances d_{ij} to obtain a tighter estimate on the actual error.

3. COMPUTING ESTIMATES

We now consider closeness centrality on undirected graphs, with a focus on efficient computation, both in terms of running time and the (run-time) storage we use, Specifically, we would like to compute estimates $\hat{S}[v]$ of $S(v) = \sum_j d_{vj}$ for all nodes $v \in V$.

All the estimators we consider, the basic sampling (5) and pivoting estimates and the hybrid estimate (6) are applied to a set of (at most) kn sampled distances. To compute these distances, we can first sample a set C of k nodes uniformly at random and then run Dijkstra's single-source shortest path algorithm from each node $u \in C$ to compute the distances d_{uv} from u to all other nodes. The computation of the estimates $\hat{S}[v]$ given these distances is linear. The issue with this approach is a run-time storage of $O(nk)$.

We first observe that both the basic sampling and the basic pivoting estimates can be computed using only $O(1)$ run-time storage per node. With sampling, we accumulate, for each node v, the sum of distances from the nodes in C. We initialize the sum to 0 for all v and then when running Dijkstra from $u \in C$, we add d_{uv} to each scanned node v. The additional run-time storage used here is the state of Dijkstra and $O(1)$ additional storage per node. With pivoting, we initialize $\Delta(v) \leftarrow \infty$ for all nodes. When running Dijkstra from u, we accumulate the sum of distances as $S(v)$. We also update $\Delta(v) \leftarrow \min\{d_{uv}, \Delta(v)\}$ when a node v is scanned. When $\Delta(v)$ is updated, we also update the pivot $c(v) \leftarrow u$. Finally, for each node v, we estimate $S(v)$ by the precomputed $S(c(v))$.

The pseudocode provided as Algorithm 1 computes the hybrid estimates (6) for all nodes using $O(1)$ additional storage per node. To do so with only $O(1)$ storage, we use an additional run of Dijkstra: For each node $v \in V$, we first compute its pivot $c(v)$ and the distance $\Delta(v) = d_{vc(v)}$. This can be done with a single run of Dijkstra's algorithm having all sampled nodes as sources.

We then run Dijkstra's algorithm from each sampled node $u \in C$. For the sampled nodes $u \in C$, the sum $S(u)$ is computed exactly; for such cases, we have $\hat{S}[u] = S(u)$. For the nodes $v \notin C$ we compute an estimate $\hat{S}[v]$.

The computation of the estimate is based on identifying the three components of the partition of $V \setminus \{v\}$ into $L(v) \cup HC(v) \cup H(v)$, which is determined according to distances

from the pivot $c(v)$. The pivot mapping computed in the additional run is used to determine this classification.

The contributions to the sum estimates $\hat{S}[v]$ are computed during the single-source shortest paths computations from C. In particular, the contribution to $\hat{S}[v]$ of sampled nodes $u \in L(v) \cup HC(v)$ are computed when we run Dijkstra from u. The contribution of $H(v)$ is computed when we run Dijkstra from the pivot $c(v)$ of v.

When running Dijkstra from a sampled node $u \in C$ and visiting v, we need to determine whether u is in $L(v)$ or $HC(v)$ in order to compute its contribution. If $u \in HC(v)$, we increase $\hat{S}[v]$ by d_{uv}. If $u \in L(v)$, we would like to increase $\hat{S}[v]$ by $d_{uv}/p[v]$. At that point, however, $p[v]$, which depends on $|L(v)|$ and $|C \cap L(v)|$, may not be available. We therefore add d_{uv} to LCSUM$[v]$, which tracks the sum of distances to nodes in $C \cap L(v)$. We also increment LCNUM$[v]$, which tracks the cardinality $|C \cap L(v)|$. When the k Dijkstra runs terminate, we can compute $p[v]$ and increase $\hat{S}[v]$ by LCSUM$[v]/p[v]$.

Deciding whether u is in $L(v)$ or $HC(v)$ can sometimes be done only after the pivot $c(v)$ was visited by the Dijkstra run from u. If $d_{uv} > \Delta(v)(1/\epsilon + 1)$ then from the triangle inequality $d_{uc(v)} > \Delta(v)/\epsilon$ and we can determine that $u \in HC(v)$. Similarly, if $d_{uv} \leq \Delta(v)(1/\epsilon - 1)$ we can determine that $u \in L(v)$. Otherwise, we can classify u only after we visit $c(v)$ and know the distance $d_{c(v)u}$. In this case, the accounting of u to $\hat{S}[v]$ is postponed: We place the pair (v, d_{uv}) in LIST$[c(v)]$. Each time a sampled node $z \in C$ is visited by u, we process the list LIST$[z]$ and for each entry (v, d_{vu}) we use $d_{uz} \equiv d_{uc(v)}$ to classify v and accordingly increase $\hat{S}[v]$ or LCSUM$[v]$. LIST$[z]$ is then deleted.

The accounting for $H(v)$ is done when running Dijkstra from the pivot $c(v)$. During Dijkstra from u, we record information on each node v for which $c(v) \equiv u$. The threshold values $\Delta(v)/\epsilon$ are recorded in increasing order in the THRESH array, as nodes are visited. The set of nodes with pivot u and a threshold value is recorded in the entry of NODES which corresponds to the threshold value. The sum of distances from u to all nodes in $V \setminus C$ with distances that are between entries in the THRESH array is computed in the corresponding entries of the BIN array. After Dijkstra's algorithm from u is completed, we process these arrays in reverse, computing for each node v such that $c(v) \equiv u$ the contribution of $H(v)$ to the estimate $\hat{S}[v]$.

This algorithm performs $k+1$ runs of Dijkstra's algorithm and uses running storage that is linear in the number of nodes (does not depend on k). This means the algorithm has very little computation overhead over the basic estimators.

4. ADAPTIVE ERROR ESTIMATION

Algorithm 1 also computes, for each node v, an estimate on the error of our estimate $\hat{S}[v]$. This estimate is *adaptive*, that is, it depends on the input. This is in contrast to the error bounds in Theorem 2.1, which are with respect to *worst-case* instances and, if used, will typically grossly overestimate the actual error and provide weak and pessimistic confidence bounds. We explain how these adaptive estimates are computed.

We also propose *adaptive error minimization* as Algorithm 2: Instead of working with a fixed value of ϵ, as in Algorithm 1, the new algorithm chooses the estimate that has the smallest estimated error.

4.1 Error Estimation

In Algorithm 1, error estimates are computed separately for each of the two components: one from the pivoting on the "distant" nodes $H(v)$, and one from the sampling, on the "closer" nodes $L(v)$.

The pivoting error is estimated by considering distant sampled nodes, that is, nodes in $HC(v)$. These nodes are treated as a representative sample of $H(v)$. For these nodes, we take the average of the squared difference between the distance of the node from v and its distance from the pivot $c(v)$:

$$\widehat{SQ}(H(v)) = \frac{1}{|HC(v)|} \sum_{u \in HC(v)} \left(d_{uv} - d_{c(v)u}\right)^2. \tag{7}$$

Note that for nodes in $HC(v)$, both these distances are available from the single-source shortest-paths computations we performed. Finally, to obtain an estimate on the contribution of the pivoting component to the squared error of $\hat{S}[v]$, we multiply by the magnitude $|H(v)|$ of the set $H(v)$, which we know exactly. In cases when there are not enough or no samples (when $HC(v)$ is empty), we instead compute the average squared difference over a "suffix" of the farthest nodes in C.

The sampling error applies to the remaining "closer" nodes $L(v)$ and depends on the distribution of distances in $L(v)$, that is, on the population variance of $L(v)$, and on the sample size from this group, which is $L(v) \cap C$. We first estimate the population variance of the set of distances from v to the set of nodes $L(v)$. This is estimated using the sample variance of the uniform sample $L(v) \cap C$, as

$$\begin{aligned}\hat{\sigma^2}(L(v)) &= \frac{1}{|C \cap L(v)|} \sum_{u \in C \cap L(v)} \left(d_{uv} - \frac{\sum_{u \in C \cap L(v)} d_{uv}}{|C \cap L(v)|}\right)^2 \\ &= \frac{\sum_{u \in C \cap L(v)} d_{uv}^2}{|C \cap L(v)|} - \left(\frac{\sum_{u \in C \cap L(v)} d_{uv}}{|C \cap L(v)|}\right)^2. \end{aligned} \tag{8}$$

We then divide the estimated population variance by the number of samples $|L(v) \cap C|$ (variable LCNUM in the pseudocode) to estimate the variance of the average of $|L(v) \cap C|$ samples from the population. To estimate the variance contribution of the sampling component to the sum estimate $\hat{S}[v]$, we multiply by $|L(v)|$ (variable LNUM in the pseudocode). The combined square error of $\hat{S}[v]$ is estimated by summing these two components:

$$|H(v)|\widehat{SQ}(H(v)) + \frac{|L(v)|}{|L(v) \cap C|}\hat{\sigma^2}(L(v)).$$

4.2 Adaptive Error Minimization

In order to get the most mileage from the k single source shortest paths computations we performed, we would like to adaptively select the best "threshold" between pivoting and sampling, rather than work with a fixed value.

For a node $v \in V$ and a threshold value T let

$$\begin{aligned} H(v,T) &= \{u \in V \setminus C \mid d_{c(v)u} > T\} \\ HC(v,T) &= \{u \in C \mid d_{c(v)u} > T\} \\ L(v,T) &= \{u \in V \mid d_{c(v)u} \leq T\}. \end{aligned}$$

The set $H(v,T)$ contains all non-sampled nodes with distance from $c(v)$ greater than T, the set $HC(v,T)$ contains all sampled nodes with distance from $c(v)$ greater than T,

Algorithm 1 Centrality estimation for all nodes: undirected

```
Input: Network G, integer k > 0, ε > 0
select uniformly at random k nodes C = {c_1, ..., c_k} ⊂ V
for v ∈ V do                                                   ▷ Computation equivalent to a single Dijkstra
    c[v] ← arg min_{i=1,...,k} d_{c_i v}                       ▷ Pivot of v
    Δ[v] ← d_{v,c_{c[v]}}                                      ▷ distance of v to its pivot
    Ŝ[v] ← 0 ; LCsum[v] ← 0; LCnum[v] ← 0 ; LCsumSq[u] ← 0 ; HCsum[u] ← 0 ; HCsumSqErr[u] ← 0 ;
for i = 1, ..., k do
    t ← 0; curt ← 0; Thresh[0] ← 0                             ▷ Initialize thresholds array and counters
    Run Dijkstra from the sampled node c_i
    for each new node u visited by Dijkstra do
        d ← d_{c_i u}                                          ▷ distance from c_i to u
        Ŝ[c_i] ← Ŝ[c_i] + d
        if u ∈ C then                                          ▷ equivalently, c_{c[u]} = u
            j ← c[u]                                           ▷ a sampled node is its own pivot, we get its index
            last[j] ← i; dist[j] ← d                           ▷ c_j was visited from c_i and has distance dist[j]
            for z ∈ List[j] do
                if d > Δ[z.node]/ε then HCsum[z.node] ←+ z.d   ▷ c_i ∈ HC(z.node)
                    HCsumSqErr[z.node] ←+ (z.d − d)^2
                else LCsum[z.node] ←+ z.d; LCnum[z.node] ←+ 1; LCsumSq[z.node] ←+ z.d^2   ▷ c_i ∈ L(z.node)
            Delete List[j]
        else                                                   ▷ u ∉ C
            if (d ≤ Δ[u](1/ε − 1)) or (last[c[u]] = i) and (dist[c[u]] ≤ Δ[u]/ε) then   ▷ c_i ∈ L(u)
                LCsum[u] ←+ d; LCnum[u] ←+ 1
                LCsumSq[u] ←+ d^2
            else                ▷ We can not determine if c_i ∈ L(u) or we know c_i ∈ HC(u) but c[u] was not yet visited
                z.node ← u; z.d ← d
                List[c[u]] ← List[c[u]] ∪ {z}
            if c[u] = i then                                   ▷ c_i is the pivot of u
                if Thresh[t] = d/ε then                        ▷ same threshold as previous
                    nodes[t] ← nodes[t] ∪ {u}
                else t ← t + 1; Thresh[t] ← d/ε; nodes[t] ← {u}; bin[t] ← 0; count[t] ← 0
            while curt < t and d > Thresh[curt + 1] do curt ←+ 1
            if d > Thresh[curt] then bin[curt] ←+ d; count[curt] ←+ 1
                                                               ▷ Compute tail sums for nodes for which c_i is pivot
    tailsum ← 0; tailnum ← 0
    while t > 0 do
        tailsum ←+ bin[t]
        tailnum ←+ count[t]
        for u ∈ nodes[t] do
            Hsum[u] ← tailsum
            Hnum[u] ← tailnum                                  ▷ Hnum[u] = |H(u)|; Hsum[u] = Σ_{v∈H(u)} d_{c(u)v}
        t ← t − 1
for u ∈ V \ C do
    Lnum ← n − 1 − Hnum[u] − k + LCnum[u]; HCnum ← k − LCnum
    p ← LCnum[u] / Lnum                                        ▷ Fraction of sampled nodes that are in L(u)
    Ŝ[u] ← Hsum[u] + HCsum[u] + LCsum[u]/p
    SqErrEst[u] ← 1/LCnum[u] ( LCsumSq[u]/LCnum[u] − ( LCsum[u]/LCnum[u] )^2 ) Lnum[u] + HCsumSqErr[u]/HCnum Hnum[u]
return For all u: (u, Ŝ[u], SqErrEst[u])
```

and the set $L(v,T)$ contains all nodes with distance from $c(v)$ at most T.

We can then define an estimator with respect to a threshold T, as in Equation (6):

$$\hat{S}(v,T) = \sum_{u \in H(v,T)} d_{c(v)u} + \sum_{u \in HC(v,T)} d_{vu} + \frac{|L(v,T)|}{|L(v,T) \cap C|} \sum_{u \in L(v,T) \cap C} d_{vu}. \tag{9}$$

In Algorithm 1 we used the threshold value $T_v = \Delta(v)/\epsilon$ for a node v. Here we choose T_v adaptively so as to balance the estimated error of the first and third summands.

One way to achieve this is to apply Algorithm 1 simultaneously with several choices of ϵ. Then, for each node, we take the value with the smallest estimated error. We propose here Algorithm 2, which maintains $O(k)$ state per node but looks for the threshold sweet spot while covering the full range between pure pivoting and pure sampling.

Algorithm 2 computes estimates and corresponding error estimates as in Algorithm 1. The estimates, however, are computed for k values of the threshold T_v which correspond to the distances from $c(v)$ to each of the other sampled nodes. From these k estimates, the algorithm selects the one which minimizes the estimated error.

The reason for considering only these k threshold values (for each pivot) is that they represent all the possible assignments of sampled nodes to $L(v)$ or $HC(v)$.

Finally, we note that the run-time storage we use depends linearly in the sets of threshold values and therefore it can be advantageous, when run-time storage is constrained, to reduce the size further. One way to do this is, for example, to only use values of T_v which correspond to discretized distances.

5. WEIGHTED CENTRALITY

We now consider weighted classic closeness centrality with respect to node weights $\beta : V \geq 0$, as defined in Equation (4). We limit our attention to estimating the denominator

$$S_\beta(i) = \sum_{j \neq i} \beta(i) d_{ij},$$

since the numerator $\sum_{j \neq i} \beta(i)$ can be efficiently computed exactly for all nodes by computing the sum $\sum_i \beta(i)$ once and, for each node j, subtracting the weight of the node j itself from the total. We show how to modify Algorithm 1 to compute estimates for $S_\beta(i)$ for all nodes. We will also argue that the proof of Theorem 2.1 goes through with minor modifications, that is, we obtain a small relative error with high probability.

If the node weights are in $\{0,1\}$, the modification is straightforward. We obtain our sample C only from nodes i with weight $\beta(i) = 1$ and account only for these nodes in our estimate of S.

We now provide details on the modification needed to handle general weights β. The first component is the node sampling. We apply a weighted sampling algorithm; in particular, we use VarOpt stream sampling [13, 17], which is a weighted version of reservoir sampling [30, 49]. We obtain a sample of exactly k nodes so that the inclusion probability of each node is proportional to its weight. More precisely, VarOpt computes a threshold value τ (which depends on k and on the distribution of β values). A node v is sampled with probability $\min\{1, \beta(v)/\tau\}$. These sampling probabilities are PPS (Probability Proportional to Size), but with VarOpt we obtain a sample of size exactly k (whereas independent PPS only guarantees an expected size of k). For each sampled node we define its *adjusted weight* $\hat{\beta}(v) = \max\{\tau, \beta(v)\}$, where τ is the VarOpt threshold.

The weighted algorithm is very similar to Algorithm 1, but requires the modification stated as Algorithm 3. The contributions to $\hat{S}[u]$ of nodes v that are in $H[u]$ (accounted for in the tail sums computed in the bin array) or in $HC[u]$ are multiplied by $\beta(v)$. For nodes in $L(v)$, we compute the inverse probability estimate with respect to the inclusion probability $\min\{1, \beta(v)/\tau\}$. We divide the contribution, which is $\beta(v)d_{uv}$, by the inclusion probability, obtaining $\hat{\beta}(v)d_{uv}$.

Our error estimates can also be easily modified to work with weighted centralities. Instead of the cardinality of each set, we use the total β weight of the set; instead of a sum of distances, we use the β-weighted sum.

The analysis of the approximation quality of $\hat{S}$ in Algorithm 3 carries over to the weighted algorithm. In fact, the skewness of β can only improve estimation quality: intuitively, the sample would contain in expectation more than k/n fraction of the total β weight, since heavier items are more likely to be sampled.

6. DIRECTED GRAPHS

6.1 Round-trip Centralities

For a strongly connected directed graph, it is natural to consider the round-trip distances $\overleftrightarrow{d}_{ij} \equiv d_{ij} + d_{ji}$, and *round-trip centrality* values computed with respect to these round-trip distances.

Since round-trip distances are a metric, the hybrid estimator (6) applies, as does Theorem 2.1, which provides the strong guarantees on approximation quality. Moreover, a simple modification of the algorithms we presented for undirected graphs applies to estimation of round-trip centralities in strongly connected directed graphs. We choose a uniform random sample of k nodes, as we did in the undirected case. Then, for each sampled node $u \in C$, we perform two single-source shortest paths computations, to compute the forward and a backward distances to all other nodes. Then for each node $v \in V \setminus C$, we compute the sum $\overleftrightarrow{d}_{uv} = d_{uv} + d_{vu}$ of these distances. We sort the nodes v by increasing $\overleftrightarrow{d}_{uv}$. We then use the sorted order and round-trip distances the same way we used the Dijkstra order in the undirected version of the algorithm.

6.2 Inbound and Outbound Centralities

As mentioned in the introduction, for general (not necessarily strongly connected) directed graphs, we may also be interested in separating *outbound* or *inbound centralities*. In particular, we are interested in the average distance from a particular node v to all nodes it can reach (outbound centrality) or from nodes that can reach v (inbound centrality), as well as in the cardinalities of these sets.

The size of the outbound reachability set of v is

$$\overrightarrow{R}[v] = |\{u \in V \setminus \{v\} \mid v \rightsquigarrow u\}|,$$

Algorithm 2 Classic closeness centralities with adaptive error minimization

```
select a set C = {c_1, ..., c_k} ⊂ V of sampled nodes, uniformly at random; for j = 1, ..., k, use c[c_j] ← j.
for v ∈ V do Δ[v] ← ∞
for i = 1, ..., k do
    Δ[c_i] ← 0                                   ▷ pivot of c_i is itself, distance to pivot is 0
    cvisited ← 1; vvisited ← 0                   ▷ number of nodes in C and V \ C, respectively, visited so far
    distsumvisited ← 0                           ▷ sum of distances to nodes in V \ C visited so far
    δ[i, i] ← 0                                  ▷ δ[i, j] is the distance between sampled nodes c_i and c_j
    π[i, 1] ← i                                  ▷ π[i, ∗] is the permutation of sampled nodes by increasing distance from c_i
    Run Dijkstra's algorithm from c_i
    for v ∈ V in order of first visit by Dijkstra do
        d ← d_{c_i v}
        if v ∈ C then
            j ← c[v]                             ▷ index of sampled node v
            cvisited ← cvisited + 1; π[i, cvisited] ← j; δ[i, j] ← d
            TailNum[i, cvisited] ← vvisited
            TailSum[i, cvisited] ← distsumvisited
        else                                     ▷ v ∉ C
            if d < Δ[v] then
                Δ[v] ← d, c[v] ← i
            D[v, i] ← d                          ▷ (n − k) × k matrix of distances of v ∈ V \ C to sampled nodes 1, ..., k
            vvisited ←+ 1; distsumvisited ←+ d
    After Dijkstra ends:
    for j = 1, ..., k do
        TailNum[j, cvisited] ← vvisited − TailNum[j, cvisited]
        TailSum[j, cvisited] ← distsumvisited − TailSum[j, cvisited]
    Ŝ[c_i] ← distsumvisited + Σ_{j=1}^{k} δ[i, j]   ▷ Exact S[c_i] of sampled node c_i
    EstErr[c_i] ← 0;                             ▷ estimated errors (no errors) for Ŝ[c_i].
for v ∈ V \ C do                                 ▷ Compute Ŝ, EstErr for all remaining nodes
    LCsum ← 0; HCsum ← Σ_{i=1}^{k} D[v, i]; HCsumSqErr ← Σ_{i=1}^{k} (D[v, i] − δ[c(v), i])^2
    Ŝ[v] ← Ŝ[c[v]]; EstErr[v] ← HCsumSqErr · (n − 1 − k)/k
    MinErr ← EstErr[v]
    for i = 1, ..., k do                         ▷ scan sampled nodes π[c(v), i] by increasing distances from c(v)
        LCsumSq ←+ D[v, π[c(v), i]]^2
        Hnum ← TailNum[c(v), π[c(v), i]]
        Lnum ← n − 1 − Hnum − k + i              ▷ |L(v)| for current threshold
        LCnum ← i; p ← LCnum/Lnum
        LCsum ←+ D[v, π[c(v), i]]                ▷ sum of distances to sampled nodes within threshold
        HCsum ←− D[v, π[c(v), i]]                ▷ sum of distances to sampled nodes outside threshold
        Hsum ← TailSum[c(v), π[c(v), i]]
        HCsumSqErr ←− (D[v, π[c(v), i]] − δ[c(v), π[c(v), i]])^2
        Est ← LCsum/p + Hsum + HCsum             ▷ estimated S[v]
        EstErr ← 1/LCnum (LCsumSq/LCnum − (LCsum/LCnum)^2) Lnum + HCsumSqErr/HCnum Hnum   ▷ est. error for threshold δ[c(v), π[c(v), i]]
        if EstErr < MinErr then
            MinErr ← EstErr                      ▷ Look for the estimation sweet spot
            Ŝ[v] ← Est; SqErrEst[v] ← EstErr
return Ŝ, SqErrEst
```

Algorithm 3 Modifications of Alg. 1 for weighted centrality

$\hat{S}[c_i] \overset{+}{\leftarrow} \beta(u) d_{c_i u}$ ▷ when computing $\hat{S}$ for $c_i \in C$
$\hat{S}[u] \overset{+}{\leftarrow} \beta(c_i) d_{c_i u}$ ▷ when $c_i \in HC(u)$
$\hat{S}[u] \overset{+}{\leftarrow} \hat{\beta}(c_i) d_{c_i u}$ ▷ when $c_i \in L(u)$
if $\beta(c_i) < \tau$ **then** VAREST$[u] \overset{+}{\leftarrow} d^2_{c_i u}(\tau - \beta(c_i))\tau$ ▷ when $c_i \in L(u)$; when $\beta(c_i) > \tau$ then c_i is included with probability 1 and its contribution to variance is 0.
BIN$[curt] \overset{+}{\leftarrow} \beta(u) d_{c_i u}$; COUNT$[curt] \overset{+}{\leftarrow} \beta(u)$ ▷ when computing tail sums/counts for c_i

where $v \rightsquigarrow u$ indicates that u is reachable from v. Similarly, the size of the inbound reachability set of v is

$$\overleftarrow{R}[v] = |\{u \in V \setminus \{v\} \mid u \rightsquigarrow v\}| .$$

Accordingly, we define the total distance to the outbound reachability set of v as

$$\overrightarrow{S}[v] = \sum_{u|v \rightsquigarrow u} d_{vu},$$

and the total distance to the inbound reachability set of v as

$$\overleftarrow{S}[v] = \sum_{u|u \rightsquigarrow v} d_{uv}.$$

The outbound and inbound centralities are accordingly defined as the (inverse of the) ratios $\overrightarrow{S}[v]/\overrightarrow{R}[v]$ and $\overleftarrow{S}[v]/\overleftarrow{R}[v]$.

Unfortunately, the hybrid estimator, and even the special case of the pivoting estimator, do not work well with direction. This is because directed distances are not a metric (they are not symmetric). Intuitively, distances from the pivot (closest sampled node) can be much larger than distances from the node for which we estimate centrality.

Sampling can be used with direction, but, when naively applied, will not provide relative error guarantees even when the distance distribution is not skewed. The reason is that it is not enough to use all distances from a small sample of nodes. For sampling to work, we need to obtain a sample of a certain size from the reachability set of each node. Some nodes, however, may reach few or no nodes from this sample. Therefore the sample provides very little information (or none at all) for estimating the centrality of these nodes.

We extend the basic sampling approach to directed graphs using an algorithm of Cohen [14] that efficiently computes for each node a uniform sample of size k from its reachability set (for outbound centrality) or from nodes that can reach it (for inbound centrality). We modify the algorithm so that respective distances are computed as well. (We apply Dijkstra's algorithm instead of generic graph searches.) This algorithm also computes nk distinct distances, but does so adaptively, so that they are not all from the same set of sources.

The same algorithm also provides approximate cardinalities of these sets [14]. This means that, when the distance distribution is not too skewed, we can obtain good estimates of the average distance to reachable nodes (or from nodes our node is reachable from).

Algorithm 4 contains pseudocode for estimating outbound average distance ($\overrightarrow{B} = \overrightarrow{S}/\overrightarrow{R}$) and reachability ($\overrightarrow{R}$) for all nodes. By applying the same algorithm on G instead of the reverse graph G^T, we can obtain estimates for the inbound quantities.

Algorithm 4 Estimate for all $v \in V$ average distance to reachable nodes $\hat{B}$ and cardinality $\hat{R}$: directed graphs

```
t ← 0,
for v ∈ V do MARK[v] ← False; COUNT[v] ← 0; T[v] ← 0; DISTSUM[v] ← 0
for nodes u ∈ V in random order do
    t ← t + 1; MARK[u] ← True
    Perform pruned Dijkstra from u on G^T
    for each scanned node v of distance d_vu do
        if COUNT[v] = k then Prune Dijkstra at v
        else
            if u ≠ v then
                DISTSUM[v] +← d_vu
                COUNT[v] +← 1
                if COUNT[v] = k then
                    T[v] ← t
                    if MARK[v] then T[v] ← t − 1
for v ∈ V do
    if COUNT[v] = 0 then B̂[v] ← 0
    else B̂[v] ← DISTSUM[v]/COUNT[v]
    if COUNT[v] < k then R̂[v] ← COUNT[v]
    else R̂[v] ← 1 + (k−1)(n−2)/(T[v]−1)
```

The algorithm computes for each node a uniform random sample of size k from its reachability set. It does so by running Dijkstra's algorithm from each node u in random order, adding u to the sample of all nodes it reaches. Since these searches are pruned at nodes whose samples already have k nodes, no node is scanned more than k times during the entire computation. The total cost is thus comparable to k full (unpruned) Dijkstra computations. This algorithm does not offer worst-case guarantees. However, on realistic instances, where centrality is in the order of the median distance, it performs well.

The algorithm applies a bottom-k variant [19] of the reachability estimation algorithm of Cohen [14] and also computes distances. The cardinality estimator is unbiased with coefficient of variation (CV) at most $1/\sqrt{k-2}$ [14]. The quality of the average distance estimates depends on the distribution of distances and we evaluate it experimentally.

We also consider non-uniform node weights and the respective weighted definitions, $\overrightarrow{S}_\beta[v] = \sum_{u|v \rightsquigarrow u} \beta(u) d_{vu}$ and $\overrightarrow{R}_\beta[v] = \sum_{u|v \rightsquigarrow u} \beta(u)$. A pseudocode for a weighted version is provided as Algorithm 5. The algorithm assigns nodes with ranks that depend on their weight, effectively having each node count for a bottom-k sample of its reachability set, as proposed by Cohen and Kaplan [14, 19]. The pseudocode uses priority sampling [24, 39]. The algorithm then processes nodes according to increasing rank order. The weighted reachability estimate is applied to the rank of the kth sample (this is a bottom-k estimator).

7. RELATED WORK

Closeness centrality is only one of several common definitions of importance rankings. These include degree centrality, intended to capture activity level, betweenness central-

Table 1: Evaluating algorithms on *undirected* instances. For each instance, we report its number of nodes and edges, and for several algorithms the running time and average relative error.

				Exact	**Sampling**		**Pivoting**		**Hyb.-0.1**		**Hyb.-ad**	
type	instance	$\lvert V\rvert$ [$\cdot 10^3$]	$\lvert E\rvert$ [$\cdot 10^3$]	time ≈ [h:m]	err. [%]	time [sec]	err. [%]	time [sec]	err. [%]	time [sec]	err. [%]	time [sec]
road	fla-t	1 070	1 344	59:30	5.4	24.4	3.2	21.6	2.5	28.3	2.8	73.2
	usa-t	23 947	28 854	44 222:06	2.9	849.4	3.7	736.4	2.0	2 344.3	2.6	9 937.9
grid	grid20	1 049	2 095	70:34	4.3	26.5	3.5	26.8	2.9	29.2	3.3	69.7
triang	buddha	544	1 631	19:07	3.6	14.5	3.3	13.6	2.4	15.9	3.2	30.7
	buddha-w	544	1 631	21:25	3.5	16.4	2.6	15.5	2.2	18.5	2.9	38.1
	del20-w	1 049	3 146	72:06	2.7	27.4	3.6	26.7	2.6	32.6	2.7	71.0
	del20	1 049	3 146	67:54	4.1	25.6	5.3	25.2	3.7	27.0	3.6	54.7
game	FrozenSea	753	2 882	38:25	3.0	22.1	4.1	20.2	2.1	24.0	3.4	49.3
sensor	rgg20	1 049	6 894	137:36	1.6	54.2	3.8	49.3	2.1	63.7	2.2	123.3
	rgg20-w	1 049	6 894	160:29	1.6	61.2	3.8	57.1	2.1	73.3	2.3	142.3
comp	Skitter	1 695	11 094	248:27	0.7	59.7	14.3	55.2	0.7	61.6	3.6	109.5
	MetroSec	2 250	21 643	269:51	0.6	52.1	2.3	47.5	0.6	53.2	0.3	93.2
social	rws20	1 049	3 146	113:40	0.9	45.6	3.0	41.3	0.9	49.4	0.9	98.6
	rba20	1 049	6 291	132:35	0.8	56.8	9.7	48.4	0.8	60.2	1.0	117.4
	Hollywood	1 069	56 307	226:42	1.0	86.5	14.6	81.8	1.0	85.7	1.9	117.6
	Orkut	3 072	117 185	2 973:09	1.7	377.4	7.2	367.6	1.7	376.4	2.1	553.0

Algorithm 5 Estimate for all $v \in V$ weighted sum of distances to reachable nodes $\hat{S}$ and weighted sum of reachable nodes $\hat{R}$: directed graphs

```
for v ∈ V do COUNT[v] ← 0; BCOUNT[v] ← 0; DISTSUM[v] ← 0
V+ ← {v ∈ V | β[v] > 0}
for u ∈ V+ do r[v] ← RAND()/β[v]   ▷ RAND() ∼ U[0,1] is uniform at random from [0,1]
for u ∈ V+ in increasing r order do
    Perform pruned Dijkstra from u on G^T
    for each scanned node v of distance d_vu do
        if COUNT[v] = k then Prune Dijkstra at v
        else
            if u ≠ v then
                COUNT[v] ←+ 1
                if COUNT[v] < k then
                    DISTSUM[v] ←+ β[u]d_vu
                    BCOUNT[v] ←+ β[u]
                if COUNT[v] = k then
                    T[v] = r[u]
for v ∈ V do
    if COUNT[v] = 0 then R̂[v] ← 0; Ŝ[v] ← 0
    else if COUNT[v] < k then R̂[v] ← BCOUNT[v]; Ŝ[v] ← DISTSUM[v]
    else Ŝ[v] ← DISTSUM/T[v]; R̂[v] ← (k−1)/T[v]
```

ity, which captures power, and eigenvalue centralities, which capture reputation [27, 50].

We only consider the classic definition of closeness centrality. A well-studied alternative is *distance-decay* closeness centrality, where the contribution of each node to the centrality of another is discounted (is non-increasing) with distance [11, 12, 16, 18, 21, 41]. The subtle difference between distance-decay and classic closeness centrality is that the latter emphasizes the penalties for far nodes, whereas the distance-decay measures instead emphasize the reward from closer nodes. Distance-decay centrality is well defined on disconnected or directed graphs. In terms of scalable computation, efficient algorithms with a small relative error guarantee were known for two decades and engineered to handle graphs with billions of edges [2, 8, 9, 14, 16, 18, 20, 42]. These algorithms, however, provide no guarantees for estimating classic closeness centrality. The intuitive reason is that they are based on sampling that is biased towards closer nodes, whereas correctly estimating classic closeness centrality requires accounting for distant nodes, which can be missed by such a sample.

8. EXPERIMENTS

We implemented our algorithms in C++ using Visual Studio 2013 with full optimization. We conducted all tests on a machine with two Intel Xeon E5-2690 CPUs and 384 GiB of DDR3-1066 RAM, running Windows 2008R2 Server. Each CPU has 8 cores (2.90 GHz, 8×64 kiB L1, 8×256 kiB, and 20 MiB L3 cache), but all runs are sequential. We use 32-bit integers to represent arc lengths.

We test a variety of instances, including *social networks* (Epinions [43], WikiTalk [31, 32], Flickr [38], Hollywood [7, 10], Twitter [22], LiveJournal [34], and Orkut [52]), *computer networks* (Gnutella [37], Skitter [33], Slashdot [34], MetroSec [36]), and *web graphs* (NotreDame [1], Indo [7, 10], Indochina [7, 10]). All these instances are unweighted, and some are directed. We consider two additional synthetic instances: rws20 is generated according to a preferential attachment model [51] and rba20 is a small-world graph [3].

We also test *road networks* [23]. Instances fla-t (Florida) and usa-t (USA) are undirected and use TIGER data; eur-t and eur-d are directed and represent Western Europe. For these instances, the suffix indicates whether edge costs represent travel times (-t) or distances (-d). Instance grid20 is a 1024×1024 unweighted grid.

The buddha instance is a computer graphics mesh representing a three-dimensional object [45]. Instance del20 is a Delaunay triangulation of 2^{20} random points on the unit square [28]. Nodes also represent random points in the unit square for rgg20, but now two nodes are connected by an edge if the corresponding Euclidean distance is below

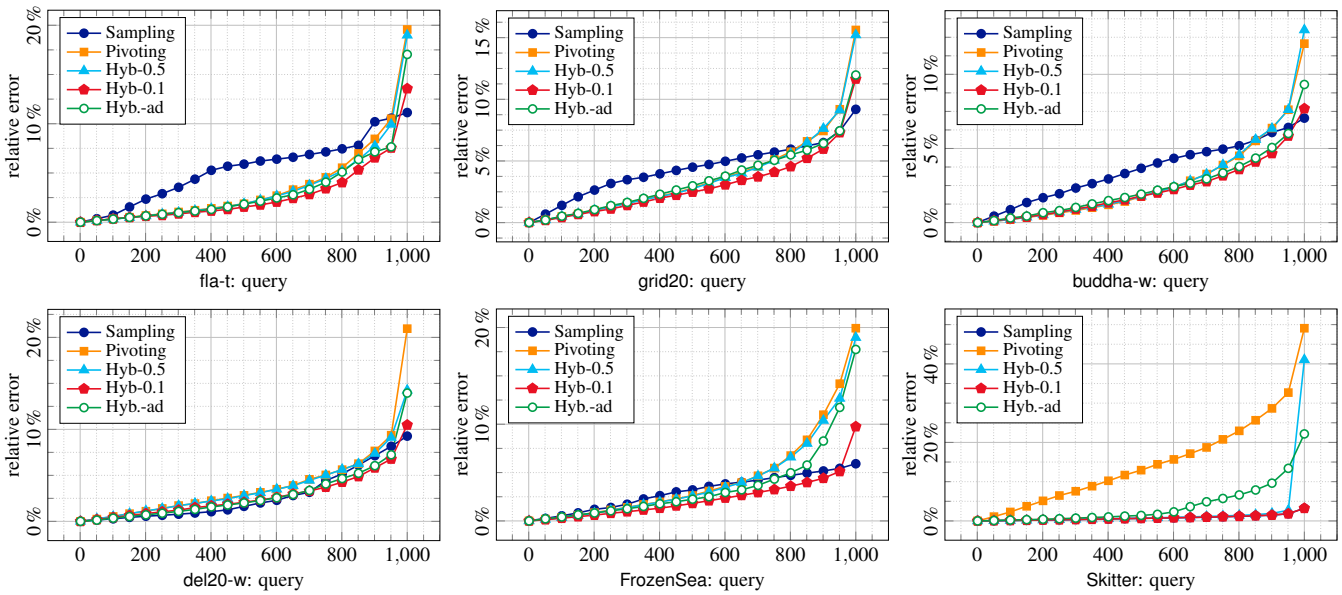

Figure 2: Cumulative quality distribution (over 1000 queries) for varying ϵ.

a given threshold (chosen to ensure the graphs are almost connected [28]). Such *random geometric graphs* often model sensor networks. These three instances are unweighted; their counterparts with a -w suffix have edge lengths corresponding to Euclidean distances. Instance FrozenSea is a grid with obstacles from Starcraft (a computer game) available from `movingai.com` [46]. Edge lengths are set to 408 for axis-aligned moves and 577 for diagonal moves ($577/408 \approx \sqrt{2}$).

8.1 Undirected Closeness Centrality

Table 1 summarizes the main results for undirected instances. We set $k = 100$ for this experiment. We evaluate sampling, pivoting, and our novel hybrid algorithm with respect to running time and solution quality. We consider two versions of our algorithm, both based on Algorithm 1: the first uses $\epsilon = \sqrt{1/k} = 0.1$; the *adaptive* version picks, for each node, the ϵ value from $\{0.001, 0.025, 0.05, 0.1, 0.2, 0.5, 0.99\}$ that minimizes the estimated error.

For each instance, Table 1 shows the number of nodes and edges it contains (in thousands), followed by the estimated time needed to compute exact centralities for all nodes. Then, for each approximate algorithm, we show its average relative error (over 1000 random nodes queried) and the total time for computing centrality estimates for all nodes (including preprocessing).

We observe that the exact algorithm is prohibitively time-consuming for large graphs, justifying our settling for approximations. Among those, all methods do reasonably well, with average relative error always below 15%. The sampling algorithm is in general more robust than pivoting, with average relative error below 6%. For some high-diameter graphs (such as road networks and meshes), however, pivoting finds better results. Our hybrid algorithm successfully achieves a good tradeoff between these two approaches. Its quality usually matches the best among pivoting and sampling, and often outperforms them.

The adaptive version of our algorithm goes one step further and actually uses different values of ϵ to obtain even finer tradeoffs. This can occasionally be helpful (as in MetroSec), but in general using fixed ϵ is better in terms of running time and quality. Although Algorithm 2 uses additional space to make even finer choices, it leads to very similar results (not shown in the table). We conclude that fixing $\epsilon = \sqrt{1/k}$ is a good strategy: It is more robust than either sampling or pivoting, with very little overhead. On the biggest graph we tested (Orkut), with 117 million edges, we obtained centrality estimates with approximation guarantees for all nodes in about six minutes.

Figure 2 examines the quality of the algorithms in Table 1 in more detail. For comparison, we also show results for the hybrid algorithm with $\epsilon = 0.5$. Once again, we compute the relative error for 1000 queries, plotted in order of increasing error. In other words, for each value $1 \leq i \leq 1000$, we report the i-th smallest relative error observed for each algorithm. We consider six representative instances. For fla-t, grid20, and buddha-w, sampling yields better results than pivoting; for del20-w, FrozenSea, and Skitter, sampling behaves better. On all cases, our default hybrid algorithm (with $\epsilon = 0.1$) is generally better than either method. We note that, unsurprisingly, pivoting tends to have more outliers than pure sampling (i.e., the worst queries for pivoting are worse than the worst for sampling). Although some of this effect is transferred to the hybrid algorithm, it is much less pronounced. This is not true with higher ϵ, which causes the hybrid algorithm to rely more heavily on pivoting.

8.2 Directed Centrality

We now consider centrality on arbitrary directed graphs. Table 2 gives the results obtained by Algorithm 4. Once again, we use $k = 100$ and evaluate the algorithm with 1000 random queries. The "Exact" column shows the estimated time for computing all n outbound centralities using Dijkstra computations. We then show the average relative error (over the 1000 random queries) and the total running time to compute all n centralities using Algorithm 4. Although this algorithm has no theoretical guarantees, its average rel-

Table 2: Evaluating algorithms on *directed* instances. As in Table 1, we report the number of nodes and directed edges and for several algorithms the running time and average relative error.

				Exact	Sampling	
type	instance	$\lvert V\rvert$ $[\cdot 10^3]$	$\lvert E\rvert$ $[\cdot 10^3]$	time $\approx$ [h:m]	err. [%]	time [sec]
road	eur-t	18 010	42 189	28 399:47	3.2	655.9
	eur-d	18 010	42 189	22 306:20	3.2	517.0
web	NotreDame	326	1 470	0:54	2.4	1.5
	Indo	1 383	16 540	58:46	4.1	21.1
	Indochina	7 415	191 607	2 884:19	4.7	174.7
comp	Gnutella	63	148	0:02	2.8	0.6
social	Epinions	76	509	0:07	5.4	1.1
	Slashdot	82	870	0:18	2.2	2.2
	Flickr	1 861	22 614	227:01	4.3	65.1
	WikiTalk	2 394	5 021	22:01	0.5	5.4
	Twitter	457	14 856	28:16	1.2	26.1
	LiveJournal	4 848	68 475	2 757:01	1.9	276.8

ative error is consistently below 6% in practice. Moreover, it is quite practical, taking less than three minutes even on a graph with almost 200 million edges.

9. CONCLUSION

We presented a comprehensive solution to the problem of approximating, within a small relative error, the classic closeness centrality of all nodes in a network. We proposed the first near-linear-time algorithm with theoretical guarantees and provide a scalable implementation. Our experimental analysis demonstrates the effectiveness of our solution.

Our basic design and analysis apply in any metric space: Given the set of distances from a small random sample of the nodes to all other nodes, we can estimate, for each node, its average distance to all other nodes, with a small relative error. We therefore expect our estimators to have further applications.

10. REFERENCES

[1] R. Albert, H. Jeong, and A.-L. Barabási. Internet: Diameter of the World-Wide Web. *Nature*, 401:130–131, September 1999.

[2] L. Backstrom, P. Boldi, M. Rosa, J. Ugander, and S. Vigna. Four degrees of separation. In *WebSci*, pp. 33–42, 2012.

[3] A.-L. Barabási and R. Albert. Emergence of scaling in random networks. *Science*, 286(5439):509–512, 1999.

[4] A. Bavelas. A mathematical model for small group structures. *Human Organization*, 7:16–30, 1948.

[5] A. Bavelas. Communication patterns in task oriented groups. *Journal of the Acoustical Society of America*, 22:271–282, 1950.

[6] M. A. Beauchamp. An improved index of centrality. *Behavioral Science*, 10:161–163, 1965.

[7] P. Boldi, M. Rosa, M. Santini, and S. Vigna. Layered label propagation: A multiresolution coordinate-free ordering for compressing social networks. In *Proceedings of the 20th international conference on World Wide Web*, pp. 587–596. 2011.

[8] P. Boldi, M. Rosa, and S. Vigna. HyperANF: Approximating the neighbourhood function of very large graphs on a budget. In *WWW*, 2011.

[9] P. Boldi, M. Rosa, and S. Vigna. Robustness of social networks: Comparative results based on distance distributions. In *SocInfo*, pp. 8–21, 2011.

[10] P. Boldi and S. Vigna. The WebGraph framework I: Compression techniques. In *Proceedings of the 13th international conference on World Wide Web*, pp. 595–602. 2004.

[11] P. Boldi and S. Vigna. In-core computation of geometric centralities with hyperball: A hundred billion nodes and beyond. In *ICDM workshops*, 2013. `http://arxiv.org/abs/1308.2144`.

[12] P. Boldi and S. Vigna. Axioms for centrality. *Internet Mathematics*, 2014.

[13] M. T. Chao. A general purpose unequal probability sampling plan. *Biometrika*, 69(3):653–656, 1982.

[14] E. Cohen. Size-estimation framework with applications to transitive closure and reachability. *J. Comput. System Sci.*, 55:441–453, 1997.

[15] E. Cohen. Undirected shortest-paths in polylog time and near-linear work. *J. Assoc. Comput. Mach.*, 47:132–166, 2000. Extended version of a STOC 1994 paper.

[16] E. Cohen. All-distances sketches, revisited: HIP estimators for massive graphs analysis. In *PODS*. ACM, 2014.

[17] E. Cohen, N. Duffield, C. Lund, M. Thorup, and H. Kaplan. Efficient stream sampling for variance-optimal estimation of subset sums. *SIAM J. Comput.*, 40(5), 2011.

[18] E. Cohen and H. Kaplan. Spatially-decaying aggregation over a network: Model and algorithms. *J. Comput. System Sci.*, 73:265–288, 2007. Full version of a SIGMOD 2004 paper.

[19] E. Cohen and H. Kaplan. Summarizing data using bottom-k sketches. In *ACM PODC*, 2007.

[20] P. Crescenzi, R. Grossi, L. Lanzi, and A. Marino. A comparison of three algorithms for approximating the distance distribution in real-world graphs. In *TAPAS*, 2011.

[21] C. Dangalchev. Residual closeness in networks. *Phisica A*, 365, 2006.

[22] M. De Domenico, A. Lima, P. Mougel, and M. Musolesi. The anatomy of a scientific rumor. *Scientific Reports*, 3:2980, 2013.

[23] C. Demetrescu, A. V. Goldberg, and D. S. Johnson, editors. *The Shortest Path Problem: Ninth DIMACS Implementation Challenge*, DIMACS Book 74. American Mathematical Society, 2009.

[24] N. Duffield, M. Thorup, and C. Lund. Priority sampling for estimating arbitrary subset sums. *J. Assoc. Comput. Mach.*, 54(6), 2007.

[25] D. Eppstein and J. Wang. Fast approximation of centrality. In *SODA*, pp. 228–229, 2001.

[26] L. C. Freeman. A set of measures of centrality based on betweeness. *Sociometry*, 40:35–41, 1977.

[27] L. C. Freeman. Centrality in social networks: Conceptual clarification. *Social Networks*, 1, 1979.

[28] M. Holtgrewe, P. Sanders, and C. Schulz. Engineering a scalable high quality graph partitioner. In *24th International Parallel and Distributed Processing Symposium (IPDPS'10)*, pp. 1–12. IEEE Computer Society, 2010.

[29] P. Indyk. Sublinear time algorithms for metric space problems. In *STOC*. ACM, 1999.

[30] D. E. Knuth. *The Art of Computer Programming, Vol 2, Seminumerical Algorithms*. Addison-Wesley, 1st edition, 1968.

[31] J. Leskovec, D. Huttenlocher, and J. Kleinberg. Predicting positive and negative links in online social networks. In *Proceedings of the 19th international conference on World wide web*, pp. 641–650. ACM, 2010.

[32] J. Leskovec, D. Huttenlocher, and J. Kleinberg. Signed networks in social media. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pp. 1361–1370. ACM, 2010.

[33] J. Leskovec, J. Kleinberg, and C. Faloutsos. Graphs over time: Densification laws, shrinking diameters and possible explanations. In *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, pp. 177–187. ACM, 2005.

[34] J. Leskovec, K. J. Lang, A. Dasgupta, and M. W. Mahoney. Community structure in large networks: Natural cluster sizes and the absence of large well-defined clusters. *Internet Mathematics*, 6(1):29–123, 2009.

[35] N. Lin. *Foundations of Social Search*. McGraw-Hill Book Co., New York, 1976.

[36] C. Magnien, M. Latapy, and M. Habib. Fast computation of empirically tight bounds for the diameter of massive graphs. *Journal of Experimental Algorithmics (JEA)*, 13:10:1–10:9, 2009.

[37] R. Matei, A. Iamnitchi, and I. Foster. Mapping the Gnutella network: Properties of large-scale peer-to-peer systems and implications for system design. *IEEE Internet Computing Journal*, 2002.

[38] A. Mislove, M. Marcon, K. P. Gummadi, P. Druschel, and B. Bhattacharjee. Measurement and analysis of online social networks. In *Proceedings of the 7th ACM SIGCOMM conference on Internet measurement*, pp. 29–42. 2007.

[39] E. Ohlsson. Sequential poisson sampling. *J. Official Statistics*, 14(2):149–162, 1998.

[40] K. Okamoto, W. Chen, and X. Li. Ranking of closeness centrality for large-scale social networks. In *Proc. 2nd Annual International Workshop on Frontiers in Algorithmics*, FAW. Springer-Verlag, 2008.

[41] T. Opsahl, F. Agneessens, and J. Skvoretz. Node centrality in weighted networks: Generalizing degree and shortest paths. *Social Networks*, 32, 2010. `http://toreopsahl.com/2010/03/20/`.

[42] C. R. Palmer, P. B. Gibbons, and C. Faloutsos. ANF: A fast and scalable tool for data mining in massive graphs. In *KDD*, 2002.

[43] M. Richardson, R. Agrawal, and P. Domingos. Trust management for the semantic web. In *The Semantic Web – ISWC 2003*, pp. 351–368. Springer, 2003.

[44] G. Sabidussi. The centrality index of a graph. *Psychometrika*, 31(4):581–603, 1966.

[45] P. V. Sander, D. Nehab, E. Chlamtac, and H. Hoppe. Efficient traversal of mesh edges using adjacency primitives. *ACM Transactions on Graphics (TOG)*, 27(5):144, 2008.

[46] N. R. Sturtevant. Benchmarks for grid-based pathfinding. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(2):144–148, 2012.

[47] M. Thorup. Quick k-median, k-center, and facility location for sparse graphs. In *ICALP*. Springer-Verlag, 2001.

[48] J. D. Ullman and M. Yannakakis. High-probability parallel transitive closure algorithms. *SIAM J. Comput.*, 20:100–125, 1991.

[49] J. Vitter. Random sampling with a reservoir. *ACM Trans. Math. Softw.*, 11(1):37–57, 1985.

[50] S. Wasserman and K. Faust, editors. *Social Network Analysis: Methods and Applications*. Cambridge University Press, 1994.

[51] D. J. Watts and S. H. Strogatz. Collective dynamics of 'small-world' networks. *Nature*, 393(6684):440–442, 1998.

[52] J. Yang and J. Leskovec. Defining and evaluating network communities based on ground-truth. In *Proceedings of the ACM SIGKDD Workshop on Mining Data Semantics*, MDS '12, pp. 3:1–3:8, New York, NY, USA, 2012. ACM.