Almost Linear-Time Algorithms for Adaptive Betweenness Centrality using Hypergraph Sketches

Yuichi Yoshida
National Institute of Informatics, and
Preferred Infrastructure, Inc.
yyoshida@nii.ac.jp

ABSTRACT

Betweenness centrality measures the importance of a vertex by quantifying the number of times it acts as a midpoint of the shortest paths between other vertices. This measure is widely used in network analysis. In many applications, we wish to choose the k vertices with the maximum adaptive betweenness centrality, which is the betweenness centrality without considering the shortest paths that have been taken into account by already-chosen vertices.

All previous methods are designed to compute the betweenness centrality in a fixed graph. Thus, to solve the above task, we have to run these methods k times. In this paper, we present a method that directly solves the task, with an almost linear runtime no matter how large the value of k. Our method first constructs a hypergraph that encodes the betweenness centrality, and then computes the adaptive betweenness centrality by examining this graph. Our technique can be utilized to handle other centrality measures.

We theoretically prove that our method is very accurate, and experimentally confirm that it is three orders of magnitude faster than previous methods. Relying on the scalability of our method, we experimentally demonstrate that strategies based on adaptive betweenness centrality are effective in important applications studied in the network science and database communities.

Categories and Subject Descriptors

E.1 [**Data**]: Data Structures—*Graphs and networks*

Keywords

Adaptive betweenness centrality; Adaptive coverage centrality; Randomized algorithm

1. INTRODUCTION

The *centrality* of a vertex measures its relative importance within a graph. There is no unique way of measuring the importance of vertices. In fact, many criteria of a vertex have been used to define its centrality, such as its degree, distance to other vertices, and its eigenvalue [4].

In this paper, we consider centralities based on shortest paths. The most famous such centrality is the *(shortest-path) betweenness centrality*, which quantifies the number of times a vertex acts as a midpoint of the shortest paths between other vertices [12]. Betweenness centrality has been extensively used in network analysis, such as for measuring lethality in biological networks [17, 10], studying sexual networks and AIDS [20], and identifying key actors in terrorist networks [18, 9]. Betweenness centrality is also used as a primary routine for clustering and community identification in real-world networks [23, 15].

We also study the *(shortest-path) coverage centrality*, which measures the importance of a vertex by counting the number of pairs of vertices with a shortest path passing through it. Coverage centrality naturally arises in the study of indexing methods for determining shortest-path distances [2, 25, 1].

Although computing the betweenness centrality and coverage centrality is already quite costly, in many applications, we want to compute these centralities in an iterative way. Consider the problem of suppressing epidemics with immunization [16]. In this problem, a vertex is infected by a disease, and the disease spreads to other vertices under some stochastic process. We have a limited number k of vaccines, and the task is to find an effective strategy to vaccinate vertices to suppress the epidemic. For this task, it is known that greedily choosing k vertices with the maximum adaptive betweenness centrality often shows a good performance [16]. Here, the *adaptive betweenness centrality* denotes the betweenness centrality without considering the shortest paths that have been taken into account by already-chosen vertices.

In the indexing method for determining shortest-path distances proposed by [2], it is desirable to compute the vertex ordering obtained by greedily taking vertices with the maximum adaptive coverage centrality, where *adaptive coverage centrality* is defined analogously.

An obvious drawback of using the top-k vertices, or the vertex ordering with respect to adaptive centrality, is that obtaining these is computationally expensive. Indeed, the fastest exact methods for computing the top-k vertices with respect to adaptive betweenness and coverage centrality take $O(knm)$ time and $O(kn^2m)$ time, respectively, where n is the number of vertices and m is the number of edges. To ameliorate this issue, faster randomized approximation algorithms have been proposed for the betweenness centrality [8, 7, 14, 24]. However, because we must run the al-

KDD'14, August 24–27, 2014, New York, NY, USA.

http://dx.doi.org/10.1145/2623330.2623626.

gorithm k times to obtain the top-k vertices, we still need at least $\Omega(k(n+m))$ time. Hence, the effectiveness of the top-k vertices and the vertex ordering with respect to adaptive centrality has only been confirmed for small graphs, and dealing with larger graphs remains a challenging problem.

1.1 Our contributions

The main contribution of this paper is to present an algorithm that approximately computes the vertex ordering (and hence the top-k vertices) with respect to the adaptive betweenness and coverage centrality in almost linear time. This is remarkable, as finding the (single) vertex with the maximum centrality requires linear time.

Let us explain the idea of our algorithm using the coverage centrality. In this paper, we only consider undirected and unweighted graphs, but it is fairly straightforward to extend our result for directed and weighted graphs. For a graph $G = (V, E)$, the (shortest-path) coverage centrality of a vertex $v \in V$ is defined as

$$C(v) = \#\{(s,t) \in V \times V \mid v \in P_{st}\},$$

where P_{st} is the set of all vertices on shortest paths between s and t. In other words, $C(v)$ is the number of pairs such that some shortest path between them goes through v. For a vertex v and a vertex set $S \subseteq V$, the *adaptive coverage centrality (ACC) of v conditioned on (having chosen) S* is defined as

$$C(v \mid S) = \#\{(s,t) \in V \times V \mid v \in P_{st}, P_{st} \cap S = \emptyset\}.$$

Hence, we ignore pairs (s,t) if some shortest path between them is already covered by a vertex in S. Note that $C(v) = C(v \mid \emptyset)$ for any v. Our goal is to keep choosing vertices with the maximum ACC conditioned on the set of already-chosen vertices.

To this end, we construct a hypergraph H, which we call the *hypergraph sketch*, that encodes all information for estimating ACC as follows. First, we randomly sample $M = \Theta(\log n/\epsilon^2)$ pairs of vertices, where ϵ is an error parameter. For each sampled pair (s,t), we add to H a hyperedge with the vertex set P_{st}. Then, our algorithm for obtaining the vertex ordering is very simple: we keep choosing vertices with the maximum degree in the hypergraph after removing the hyperedges incident to already-chosen vertices. The runtime of this algorithm is almost linear, and we have a good accuracy if M is chosen to be sufficiently large.

Our idea is applicable for obtaining the vertex ordering based on the adaptive betweenness centrality. For a graph $G = (V, E)$, the (shortest-path) betweenness centrality of a vertex $v \in V$ is defined as

$$B(v) = \sum_{s,t \in V \setminus \{v\}} \frac{\sigma_{st}(v)}{\sigma_{st}},$$

where σ_{st} is the number of shortest paths between s and t, and $\sigma_{st}(v)$ is the number of shortest paths between s and t going through v. For a vertex $v \in V$ and a vertex set $S \subseteq V$, the *adaptive betweenness centrality (ABC) of v conditioned on (having chosen) S* is defined as

$$B(v \mid S) = \sum_{s,t \in V \setminus \{v\}} \frac{\sigma_{st}(v \mid S)}{\sigma_{st}},$$

where $\sigma_{st}(v \mid S)$ is the number of shortest paths between s and t passing through v, but not passing through any vertex in S. Our goal is to keep choosing vertices with the maximum ABC conditioned on the set of already-chosen vertices. Though we can use the hypergraph sketch again, the algorithm becomes more involved than the case of ACC, because we must impose weights on the vertices and update them each time we choose a vertex. However, the resulting runtime remains almost linear.

For both centralities, we prove that our method has a high probability of estimating the adaptive centrality of any vertex with additive error of ϵn^2. This additive error should not be critical in most applications, as numerous real-world graphs actually have vertices of centrality $\Theta(n^2)$, and it often suffices to obtain the correct ordering among these.

For both centralities, we can define the centrality of a vertex set, and our method can be seen as an approximation of the problem of finding a set of k vertices with the maximum centrality. We prove that our method has almost the same approximation ratio as the exact greedy method, which is known to work quite well in practice. Hence, the quality of the output of our method should be very high.

We also empirically show that our method runs two or three orders of magnitude faster than previous methods.

Now that we have a method to obtain the vertex ordering based on adaptive centrality, we investigate the two applications mentioned before: suppressing epidemics with immunizations, and constructing indices for determining shortest-path distances. Our experiments demonstrate that the strategy based on vertex ordering with respect to adaptive centrality shows a better performance than other heuristic strategies in large graphs, which we have not been able to confirm due to the prohibitively large runtime of previous methods.

1.2 Related work

As we have mentioned, there are plenty of studies on computing the betweenness centrality. Currently, the fastest known algorithm for exactly computing the betweenness centralities of all the vertices is that proposed by Brandes [6], which first solves the single-source shortest-path problem (SSSP) from every vertex. An SSSP computation from a vertex s produces a directed acyclic graph (DAG) encoding all shortest paths starting at s. By backward aggregation, the contributions of these paths to the betweenness centrality can be computed in linear time. In total, Brandes' algorithm runs in $O(nm)$ time for unweighted graphs, and $O(nm + n^2 \log m)$ time for weighted graphs.

Brandes and Pich [8] investigated how the exact algorithm can be turned into an approximation algorithm by sampling a small set of vertices, from which we solve the SSSP. A random sample of starting vertices turns out to work well. To obtain better accuracy and runtime, several sampling and aggregation methods have been studied [14, 3, 24]. However, none of these methods is sufficiently fast when we wish to obtain the vertex ordering based on the adaptive centrality.

Lee et al. [19] considered the problem of updating the betweenness centrality when the graph changes. However, their method is not sufficiently fast to handle large graphs, say, millions of vertices.

Borgs et al. [5] considered the problem of *influence maximization*. In this problem, vertices are iteratively influenced by neighbors under some stochastic process, and the objective is to find a seed set of k vertices that maximizes the expected number of influenced vertices at the end of the process. To efficiently implement a greedy algorithm, they

proposed a method that constructs a hypergraph so that the degree of a vertex represents its influence. Though their method has a similar flavor to ours, we consider completely unrelated problems.

1.3 Organization

We first explain our method for ACC, and present its theoretical analysis in Section 2, because this is much simpler than for ABC. We then proceed to ABC in Section 3. In Section 4, we empirically demonstrate the accuracy and runtime of our method. Section 5 is devoted to confirming the effectiveness of the vertex ordering based on adaptive centrality using the two applications.

Notations.

For a given graph, we always use the symbols n and m to denote the number of vertices and edges, respectively.

2. ADAPTIVE COVERAGE CENTRALITY

We formalize the problem of computing the top-k vertices with respect to ACC. Let $G = (V, E)$ be a graph. For two vertices s and t, let P_{st} be the set of all vertices on any shortest path between s and t. By orienting the edges in P_{st} according to the distance from s, we can obtain a DAG rooted at s. We often identify P_{st} with this DAG. We say that a pair (s, t) is *covered* by a vertex $v \in V$ if $v \in P_{st}$. Similarly, we say that a pair (s, t) is *covered* by a vertex set $S \subseteq V$ if $S \cap P_{st} \neq \emptyset$. In other words, (s, t) is covered by v (resp., S) if some shortest path between s and t goes through v (resp., some vertex in S).

The *coverage centrality* of a vertex $v \in V$ is

$$C(v) = \#\{(s, t) \in V \times V \mid v \in P_{st}\},$$

which is the number of pairs of vertices (s, t) that is covered by v. We extend the notion to a set of vertices. For a set of vertices $S \subseteq V$, the *coverage centrality* of S is

$$C(S) = \#\{(s, t) \in V \times V \mid S \cap P_{st} \neq \emptyset\},$$

which is the number of pairs of vertices (s, t) covered by S.

Suppose that, given a parameter $k > 0$, we want to find a set of k vertices with the maximum coverage centrality. We call this problem MCC_k, which stands for the maximum coverage centrality problem with parameter k. A natural strategy for MCC_k is to keep choosing vertices that cover the maximum number of pairs that have not been covered by already-chosen vertices. This strategy motivates us to define the following notion. For a vertex v and a vertex set S, we define the *ACC of v conditioned on S* as

$$C(v \mid S) = \{(s, t) \in V \times V \mid v \in P_{st}, S \cap P_{st} = \emptyset\}.$$

Related to this definition, we say that a vertex v *covers* the pair (s, t) conditioned on S if $v \in P_{st}$ and $S \cap P_{st} = \emptyset$. The following proposition is obtained immediately from this definition.

PROPOSITION 2.1. *For any vertex v and vertex set S, we have*

$$C(S \cup \{v\}) = C(S) + C(v \mid S).$$

The greedy strategy keeps choosing vertices with the maximum ACC conditioned on the set of already-chosen vertices, or from Proposition 2.1, we can say that it keeps choosing vertices that maximize the resulting coverage centrality.

Given a set of vertices S, an obvious way to exactly compute $C(v \mid S)$ for all $v \in V$ is the following. First, for each pair (s, t), we obtain P_{st} by performing a breadth-first search (BFS). If P_{st} contains a vertex in S, then the pair is already covered. Otherwise, vertices in P_{st} cover the pair (s, t) conditioned on S. In this way, we can compute $C(v \mid S)$ for all $v \in V$ in $O(n^2 m)$ time. If we want to run the greedy algorithm to obtain a solution for MCC_k, we need $O(kn^2 m)$ time. On the other hand, our method runs in almost linear time, even when $k = n$.

2.1 Proposed method

We describe our method for computing the top-k vertices with respect to ACC. Given a graph $G = (V, E)$, we first build a hypergraph H that encodes all the information needed to compute the ACC as follows. The vertex set of H is V. Given a parameter M, we pick M pairs of vertices at random. Then, for each pair (s, t), we add P_{st} to H as a hyperedge (Algorithm 1). Let $E(H)$ denote the set of hyperedges in H, and for a hyperedge $e \in E(H)$, let $V(e)$ denote the set of vertices incident to e.

The *degree* $d_H(v)$ of a vertex $v \in V$ in H is the number of hyperedges incident to v. Similarly, we define the *degree* $d_H(S)$ of a vertex set $S \subseteq V$ in H as the number of hyperedges incident to S. That is, $d_H(S) = \#\{e \in E(H) \mid V(e) \cap S \neq \emptyset\}$. Note that $d_H(\{v\}) = d_H(v)$.

The following observation is crucial.

LEMMA 2.2. *For any vertex set $S \subseteq V$, we have*

$$\mathop{\mathbf{E}}_{H}[d_H(S)] = \frac{M}{n^2} C(S).$$

PROOF. Each time we sample a pair (s, t), the probability that P_{st} contains a vertex in S is exactly $C(S)/n^2$. Hence, the lemma holds from the additivity of expectation. □

We define $\widetilde{C}(S) = \frac{n^2}{M} d_H(S)$ for a set $S \subseteq V$. Note that $\widetilde{C}(S)$ is a random variable, and its expectation is $C(S)$. We will later show that $d_H(S)$ is highly concentrated on its expectation, and we can use $\widetilde{C}(S)$ as a good approximation to $C(S)$.

We can estimate the coverage centralities of vertices in G using the hypergraph H, and, in particular, find the vertex with the maximum coverage centrality. We now proceed to find the vertex v with the maximum $C(v \mid S)$, given a set S of already-chosen vertices. Note that $C(v \mid S) = C(S \cup \{v\}) - C(S)$ from Proposition 2.1. Hence, we can approximate $C(v \mid S)$ by

$$\begin{aligned}\widetilde{C}(v \mid S) &:= \widetilde{C}(S \cup \{v\}) - \widetilde{C}(S) \\ &= \frac{n^2}{M}(d_H(S \cup \{v\}) - d_H(S)) = \frac{n^2}{M} d_{H-S}(v),\end{aligned}$$

where $H - S$ is the hypergraph obtained from H by removing S and all hyperedges incident to S. With this observation, to estimate $C(v \mid S)$, we can simply use the degree of v in the hypergraph $H - S$. Algorithm 2 describes how to compute the top-k vertices with respect to ACC. We choose $M = O(\log n/\epsilon^2)$ to guarantee the quality of its output.

2.2 Accuracy

In this section, we first show that, for any set of vertices S, $\widetilde{C}(S)$ has a high probability of being a good approximation

Algorithm 1 Build-Coverage-Hypergraph(G, M)

Input: A graph $G = (V, E)$ and an integer $M \geq 1$.

Initialize $H = (V, \emptyset)$.
for $i = 1$ to M **do**
  Pick a pair of vertices (s, t) at random.
  Add a hyperedge with the vertex set P_{st} to H.
return H.

Algorithm 2 Top-k-ACC(G, k)

Input: A graph $G = (V, E)$ and an integer $k \geq 1$.

$M \leftarrow O(\log n/\epsilon^2)$.
$H \leftarrow$ Build-Coverage-Hypergraph(G, M).
for $i = 1$ to k **do**
  $v_i \leftarrow \arg\max_v \{d_H(v)\}$.
  Remove v_i and all hyperedges incident to v_i.
return the set $\{v_1, \ldots, v_k\}$.

to $C(S)$. Then, we examine the quality of the output of Top-k-ACC. We say that an algorithm is an (α, β)-*approximation algorithm* to a maximization problem if, for any instance of the problem, it outputs a solution whose value is at least $\alpha \cdot \mathrm{opt} - \beta$, where opt is the optimal value of the instance. When $\beta = 0$, this is simply called an α-*approximation algorithm*. We will show that Top-k-ACC is a $(1 - 1/e, \epsilon n^2)$-approximation algorithm to MCC$_k$.

To show that $\widetilde{C}(S)$ is a good approximation to $C(S)$, we recall Hoeffding's inequality:

LEMMA 2.3 (HOEFFDING'S INEQUALITY). *Let $X_1, \ldots, X_n$ be independent random variables in $[0, 1]$ and $\overline{X} = \frac{1}{n}\sum_{i=1}^{n} X_i$. Then,*

$$\Pr[|\overline{X} - \mathbf{E}[\overline{X}]| \geq t] \leq 2\exp(-2t^2 n).$$

It is not hard to show that, with high probability over the construction of H, we can estimate the coverage centrality to within a small error:

LEMMA 2.4. *For any set of vertices $S \subseteq V$, we have*

$$\Pr_H\left[|\widetilde{C}(S) - C(S)| \geq \frac{\epsilon n^2}{2}\right] < \frac{1}{n^3}$$

over the construction of the hypergraph H.

PROOF. From Lemma 2.2, we have $\mathbf{E}[\widetilde{C}(S)] = C(S)$. Because $\widetilde{C}(S)$ is the sum of M random variables whose values are in $[0, 1]$, Hoeffding's inequality implies that

$$\begin{aligned}
&\Pr\left[\left|\widetilde{C}(S) - C(S)\right| \geq \frac{\epsilon n^2}{2}\right] \\
&= \Pr\left[\left|\frac{n^2}{M} d_H(S) - \frac{n^2}{M}\,\mathbf{E}[d_H(S)]\right| \geq \frac{\epsilon n^2}{2}\right] \\
&= \Pr\left[\left|\frac{1}{M} d_H(S) - \mathbf{E}[\frac{1}{M} d_H(S)]\right| \geq \frac{\epsilon}{2}\right] \leq 2\exp\left(-\frac{\epsilon^2 M}{2}\right)
\end{aligned}$$

By choosing $M = 2\log(2n^3)/\epsilon^2 = O(\log n/\epsilon^2)$, we have the desired result. □

Now we show that our method is a $(1-1/e, \epsilon n^2)$-approximation algorithm to MCC$_k$. We note that the problem can be seen as a *monotone submodular function maximization problem*. A function $f : 2^V \to \mathbb{R}$ is called *monotone* if

$$f(S) \leq f(T) \text{ for any } S \subseteq T \subseteq V,$$

and is *submodular* if

$$f(S \cup \{e\}) - f(S) \geq f(T \cup \{e\}) - f(T)$$
$$\text{for any } S \subseteq T \subseteq V \text{ and } e \in V \setminus T.$$

In the monotone submodular function maximization problem (MSFM for short), we are given a non-negative monotone submodular function $f : 2^V \to \mathbb{R}$ and a parameter k, and the objective is to find a set $S \subseteq V$ of k elements that maximizes $f(S)$. A standard heuristic for this problem is a greedy algorithm, that is, we keep choosing elements v that produce the maximum marginal gain $f(S \cup \{v\}) - f(S)$, where S is the set of already-chosen vertices. The following fact is well known.

LEMMA 2.5 ([21, 22]). *The greedy algorithm is a $(1 - 1/e)$-approximation algorithm to MSFM, and obtaining a better approximation ratio requires an exponential number of queries in $|V|$.*

We can exploit this result to solve MCC$_k$. We need the following properties of coverage centrality.

LEMMA 2.6. *The function $C : 2^V \to \mathbb{Z}$ is non-negative, monotone, and submodular.*

PROOF. As non-negativity and monotonicity are clear, we consider the submodularity. Let $S \subseteq T \subseteq V$ and $v \in V$. Then, $C(S \cup \{v\}) - C(S) = C(v \mid S)$ and $C(T \cup \{v\}) - C(T) = C(v \mid T)$. Hence, it suffices to show $C(v \mid S) \geq C(v \mid T)$, but this should be true because the number of pairs newly covered by v is larger when having chosen a smaller set of vertices. □

COROLLARY 2.7. *The greedy algorithm is a $(1 - 1/e)$-approximation algorithm to MCC$_k$.*

Of course, we do not want to run the greedy algorithm on the function C, as evaluating C would take a very long time. Instead, we show that Algorithm 2 has almost the same quality, that is, it is a $(1 - 1/e, \epsilon n^2)$-approximation algorithm. We first show the following.

LEMMA 2.8. *The function $\widetilde{C} : 2^V \to \mathbb{Z}$ is non-negative, monotone, and submodular for any realization of H.*

PROOF. As non-negativity and monotonicity are clear, we consider the submodularity. Let $S \subseteq T \subseteq V$ and $v \in V$. Then, $\widetilde{C}(S \cup \{v\}) - \widetilde{C}(S) = \widetilde{C}(v \mid S)$ and $\widetilde{C}(T \cup \{v\}) - \widetilde{C}(T) = \widetilde{C}(v \mid T)$. Hence, it suffices to show $\widetilde{C}(v \mid S) \geq \widetilde{C}(v \mid T)$, but this should be true because the number of hyperedges newly incident to v is larger when having chosen a smaller set of vertices. □

THEOREM 2.9. *Let $\widetilde{S}$ be the output of Algorithm 2, and S^* be the optimal solution to MCC$_k$. Then, with a probability of at least $1 - 1/n$,*

$$C(\widetilde{S}) \geq \left(1 - \frac{1}{e}\right) C(S^*) - \epsilon n^2.$$

PROOF. Let $\widetilde{S}^* = \arg\max_{S \subseteq V : |S| = k} \widetilde{C}(S)$. By Lemma 2.2, there is a probability of at least $1 - \frac{1}{n^3}$ that $\widetilde{C}(S^*) \geq C(S^*) - \epsilon n^2/2$. In particular, $\widetilde{C}(\widetilde{S}^*) \geq C(S^*) - \epsilon n^2/2$.

Algorithm 2 runs the greedy algorithm on the function $\widetilde{C}$, and outputs the set $\widetilde{S}$. Because $\widetilde{C}$ is a non-negative

submodular function by Lemma 2.8, we have $\widetilde{C}(\widetilde{S}) \geq (1 - 1/e)\widetilde{C}(\widetilde{S}^*)$ by Lemma 2.5.

Let $\widetilde{S}_i$ be the set of vertices chosen up to and including the i-th iteration (with $\widetilde{S}_0 = \emptyset$). In particular, $\widetilde{S}_k = \widetilde{S}$. On the i-th iteration, we consider each set of the form $\widetilde{S}_{i-1} \cup \{v\}$, where v is a vertex. There are at most n of these sets, and hence the union bound implies that C and $\widetilde{C}$ differ by at most $\epsilon n^2/2$ on each of these sets, with probability at least $1 - 1/n^2$. In particular, $|\widetilde{C}(\widetilde{S}_i) - C(\widetilde{S}_i)| < \epsilon n^2/2$. Taking the union bound over all iterations, we have that $|\widetilde{C}(\widetilde{S}_k) - C(\widetilde{S}_k)| < \epsilon n^2/2$ with probability at least $1 - 1/n$. Therefore, we have

$$C(\widetilde{S}_k) \geq \widetilde{C}(\widetilde{S}_k) - \frac{\epsilon n^2}{2} \geq \left(1 - \frac{1}{e}\right)\widetilde{C}(\widetilde{S}^*) - \frac{\epsilon n^2}{2}$$
$$\geq \left(1 - \frac{1}{e}\right) C(S^*) - \epsilon n^2$$

conditioned on an event of probability $1 - 1/n$.

□

Hence, there is a high probability that Algorithm 2 outputs a $(1-1/e, \epsilon n^2)$-approximation. In Section 4, we will see that the output is close to that of the exact greedy algorithm.

2.3 Runtime

We finally consider the runtime of Algorithm 2.

THEOREM 2.10. *Algorithm 2 can be implemented to run in $O((n+m)\log n/\epsilon^2)$ time.*

PROOF. Build-Coverage-Hypergraph clearly runs in $O((n+m)\log n/\epsilon^2)$ time. For Top-k-ACC, we will maintain a list of vertices sorted by their degree in H; this will enable us to iteratively choose the vertex with the maximum degree in constant time. We need $O(n \log n)$ time for the initial sort. We must bound the time needed to remove an edge from H and correspondingly update the sorted list. The sorted list is implemented as a doubly linked list of groups of vertices, where each group itself is implemented as a doubly linked list containing all vertices of a given degree (with only non-empty groups present). Each edge of H will maintain a list of pointers to its vertices. When an edge is removed, the degree of each vertex in the edge decreases by 1. We modify the list by shifting any decremented vertex to the preceding group (creating new groups and removing empty groups as necessary). The time taken to remove an edge from H and update the sorted list is therefore proportional to the size of the edge. As each edge in H can be removed at most once over all iterations of Top-k-ACC, the total runtime is at most the sum of degrees in H, which is at most $O((n+m)M) = O((n+m)\log n/\epsilon^2)$. □

3. ADAPTIVE BETWEENNESS CENTRALITY

In this section, we describe our method for ABC.

We formalize the problem of computing the top-k vertices with respect to ABC. Let $G = (V, E)$ be a graph. For a vertex $v \in V$, we define the *betweenness centrality* of v as

$$B(v) = \sum_{s,t \in V \setminus \{v\}} \frac{\sigma_{st}(v)}{\sigma_{st}},$$

where σ_{st} is the number of shortest paths between s and t, and $\sigma_{st}(v)$ is the number of shortest paths between s and t passing through v. For a vertex set $S \subseteq V$, we similarly define the *betweenness centrality* of S as

$$B(S) = \sum_{s,t \in V} \frac{\sigma_{st}(S)}{\sigma_{st}},$$

where $\sigma_{st}(S)$ is the number of shortest paths between s and t passing through some vertex in $S \setminus \{s,t\}$. The betweenness centrality of a set is also called the *group betweenness centrality* in [11, 13].

Suppose that, given a parameter $k > 0$, we want to find a set of k vertices with the maximum betweenness centrality. We call this problem MBC$_k$, which stands for the maximum betweenness centrality problem with parameter k. Similar to the coverage centrality, a natural strategy for MBC$_k$ is to keep choosing vertices v that maximize the resulting betweenness centrality. It is convenient to introduce some definitions. For vertices $v \in V$, $s,t \in V \setminus \{v\}$, and a vertex set $S \subseteq V$, let $\sigma_{st}(v \mid S)$ denote the number of shortest paths passing through v but avoiding all vertices in $S \setminus \{s,t\}$. In particular, $\sigma_{st}(v \mid S)$ is zero when $v \in S$. For a vertex v and a vertex set S, the *ABC of v conditioned on S* is

$$B(v \mid S) = \sum_{s,t \in V \setminus \{v\}} \frac{\sigma_{st}(v \mid S)}{\sigma_{st}}.$$

PROPOSITION 3.1. *For any vertex v and vertex set S, we have*

$$B(S \cup \{v\}) = B(S) + B(v \mid S).$$

PROOF.

$$B(S \cup \{v\}) = \sum_{s,t \in V \setminus \{v\}} \frac{\sigma_{st}(S \cup \{v\})}{\sigma_{st}}$$
$$= \sum_{s,t \in V \setminus \{v\}} \frac{\sigma_{st}(S) + \sigma_{st}(v \mid S)}{\sigma_{st}} = B(S) + B(v \cup S). \quad \square$$

By Proposition 3.1, we can say that the greedy strategy keeps choosing vertices v that maximize the ABC conditioned on the set of already-chosen vertices.

For a vertex set S, we can exactly compute $B(v \mid S)$ for all v simultaneously using the following variant of Brandes' algorithm [6]. For $s \in V \setminus \{v\}$, let $\delta_s(v \mid S) = \sum_{t \neq v} \frac{\sigma_{st}(v|S)}{\sigma_{st}}$. Because $B(v \mid S) = \sum_{s \in V \setminus \{v\}} \delta_s(v \mid S)$, it suffices to compute $\delta_s(v \mid S)$ for every $s \in V \setminus \{v\}$. Let P_s be the DAG representing the shortest paths from s, which can be constructed by performing a BFS from s. Let $\mathrm{succ}_s(v)$ be the set of successors of v in the DAG P_s. In particular, $\mathrm{dist}(s,w) = \mathrm{dist}(s,v) + 1$ for all $w \in \mathrm{succ}_s(v)$, where $\mathrm{dist}(u,v)$ is the distance between u and v. We process vertices in the DAG P_s in reverse topological order, that is, by non-increasing distance from s. We have the following recursion:

$$\delta_s(v \mid S) = \sum_{w \in \mathrm{succ}_s(v)} \left(\frac{\sigma_{sv}(v \mid S)}{\sigma_{sw}} + \frac{\sigma_{sv}(v \mid S)}{\sigma_{sw}(w \mid S)} \delta_s(w \mid S) \right).$$

The first term in the summand deals with the contribution to $\delta_s(v \mid S)$ of the pair (s,w). The second term in the summand deals with the contribution to $\delta_s(v \mid S)$ of pairs (s,t), where t ranges over all descendants of w in the DAG P_s. We need the scaling factor $\frac{\sigma_{sv}(v|S)}{\sigma_{sw}(w|S)}$, because only a

Algorithm 3 Build-Betweenness-Hypergraph(G, M)

Input: A graph $G = (V, E)$ and an integer $M \geq 1$.

Initialize $H = (V, \emptyset)$.
for $i = 1$ to M **do**
  Pick a pair of vertices (s, t) at random.
  Make a weighted hyperedge $e = \{(v, \frac{\sigma_{st}(v)}{\sigma_{st}}) \mid v \in P_{st} \setminus \{s, t\}\}$, and add it to H.
return H.

Algorithm 4 Top-k-ABC(G, k)

Input: A graph $G = (V, E)$ and an integer $k \geq 1$.

$M \leftarrow O(\log n/\epsilon^2)$.
$H \leftarrow$ Build-Betweenness-Hypergraph(G, M).
for $i = 1$ to k **do**
  $v_i \leftarrow \arg\max_v \{w_H(v)\}$.
  for Each hyperedge e incident to v_i **do**
    Replace it with a new weighted hyperedge $\{(v, \frac{\sigma_{s(e)t(e)}(v|v_1,\ldots,v_i)}{\sigma_{s(e)t(e)}} \mid v \in P_{s(e)t(e)} \setminus \{s, t\}\}$.
return The set $\{v_1, \ldots, v_k\}$.

$\frac{\sigma_{sv}(v|S)}{\sigma_{sw}(w|S)}$-fraction of shortest paths from s to w pass through the edge (v, w). We do not give a detailed proof of this here, as the only difference from Brandes' original algorithm is that we exclude shortest paths passing through S, and the proof remains almost the same.

Let us consider the runtime of the algorithm explained above. First, we must compute P_s for every vertex s, which takes $O(nm)$ time. For each DAG P_s, we need $O(m)$ time to calculate the recursion. Hence, in total, we require $O(nm)$ time. Moreover, if we want to run the greedy algorithm to obtain a solution for MBC_k, we need $O(knm)$ time. Though this runtime is much better than the case of MCC_k, it is still quite large. On the other hand, our method runs in almost linear time, even when $k = n$.

3.1 Proposed method

In this section, we explain our method for ABC. The idea is similar to the case of ACC, but is technically more involved. Algorithm 3 describes how to construct a hypergraph sketch H for the betweenness centrality. We pick a set of M pairs of vertices (s, t), and for each pair (s, t), we add a hyperedge with a weight on each vertex. Specifically, we add to H a set of pairs $\{(v, \frac{\sigma_{st}(v)}{\sigma_{st}}) \mid v \in P_{st} \setminus \{s, t\}\}$. In what follows, we simply call the set a *hyperedge*. For a hyperedge e of H, let $V(e)$ be the set of vertices in e. For a vertex v, let $w_e(v)$ be the weight of v imposed by e. If $v \notin V(e)$, we set $w_e(v) = 0$. Let $s(e)$ and $t(e)$ denote the pair (s, t) used to make the hyperedge e.

For a vertex v, we define the *weight* of v in H as $w_H(v) = \sum_{e \in E(H)} \frac{\sigma_{s(e)t(e)}(v)}{\sigma_{s(e)t(e)}}$. Similarly, for a vertex set $S \subseteq V$, we define the *weight* of S in H as $w_H(S) = \sum_{e \in E(H)} \frac{\sigma_{s(e)t(e)}(S)}{\sigma_{s(e)t(e)}}$. Note that $w_H(\{v\}) = w_H(v)$.

We often use the following observation.

Lemma 3.2. *For any vertex set $S \subseteq V$,*

$$\mathop{\mathbf{E}}_{H}[w_H(S)] = \frac{M}{n^2} B(S).$$

Proof. If we have sampled a pair (s, t), then the contribution to $w_H(S)$ is $\frac{\sigma_{st}(S)}{\sigma_{st}}$. Hence, the expected contribution to $w_H(S)$ over a pair chosen at random is exactly $B(S)/n^2$. The lemma follows from the linearity of expectation. □

We define $\widetilde{B}(v) = \frac{n^2}{M} w_H(v)$ and $\widetilde{B}(S) = \frac{n^2}{M} w_H(S)$. Similar to the case of coverage centrality, we can show that $w_H(S)$ is highly concentrated on its expectation. Hence, we can use $\widetilde{B}(S)$ as a good approximation to $B(S)$.

We note that $w_H(v) = \sum_{e \in E(H)} \frac{\sigma_{s(e)t(e)}(v)}{\sigma_{s(e)t(e)}} = \sum_{e \in E(H)} w_e(v)$. Hence, we can obtain the vertex with the maximum betweenness centrality by choosing that with the maximum sum of weights imposed by hyperedges in H. We now show how to modify the hypergraph H so that we can compute ABC. Suppose that we have chosen a vertex set S. Because $B(v \mid S) = B(S \cup \{v\}) - B(S)$, we want to approximate it by

$$\begin{aligned}\widetilde{B}(v \mid S) &:= \widetilde{B}(S \cup \{v\}) - \widetilde{B}(S) \\ &= \sum_{e \in E(H)} \frac{\sigma_{s(e)t(e)}(S \cup \{v\})}{\sigma_{s(e)t(e)}} - \sum_{e \in E(H)} \frac{\sigma_{s(e)t(e)}(S)}{\sigma_{s(e)t(e)}} \\ &= \sum_{e \in E(H)} \frac{\sigma_{s(e)t(e)}(v \mid S)}{\sigma_{s(e)t(e)}}.\end{aligned}$$

The last expression suggests how we should modify the hypergraph H. That is, after choosing a vertex set S, for each hyperedge e and a vertex $v \in V(e)$, we need to change the weight $w_e(v)$ to $\frac{\sigma_{s(e)t(e)}(v|S)}{\sigma_{s(e)t(e)}}$. We can use the variant of Brandes' algorithm mentioned before to compute $\frac{\sigma_{s(e)t(e)}(v|S)}{\sigma_{st}}$. However, as we only consider shortest paths between $s(e)$ and $t(e)$, we have $\sigma_{st}(v \mid S) = \sigma_{sv}(v \mid S) \cdot \sigma_{tv}(v \mid S)$ if $v \in P_{st}$ and $v \notin S$, and we have $\sigma_{st}(v \mid S) = 0$ otherwise. Hence, we can compute $\frac{\sigma_{st}(v|S)}{\sigma_{st}}$ for all $v \in V(e)$ in linear time with respect to the number of edges in the DAG $P_{s(e)t(e)}$.

Algorithm 4 summarizes how to compute the top-k vertices with respect to ABC. We choose $M = O(\log n/\epsilon^2)$ to guarantee the quality of its output.

3.2 Accuracy

We can prove the accuracy of Algorithm 4 in a similar manner to the case of the coverage centrality:

Theorem 3.3. *Let $\widetilde{S}$ be the output of Algorithm 4 and S^* be the optimal solution to MBC_k. Then, with a probability of at least $1 - 1/n$,*

$$B(\widetilde{S}) \geq \left(1 - \frac{1}{e}\right) B(S^*) - \epsilon n^2.$$

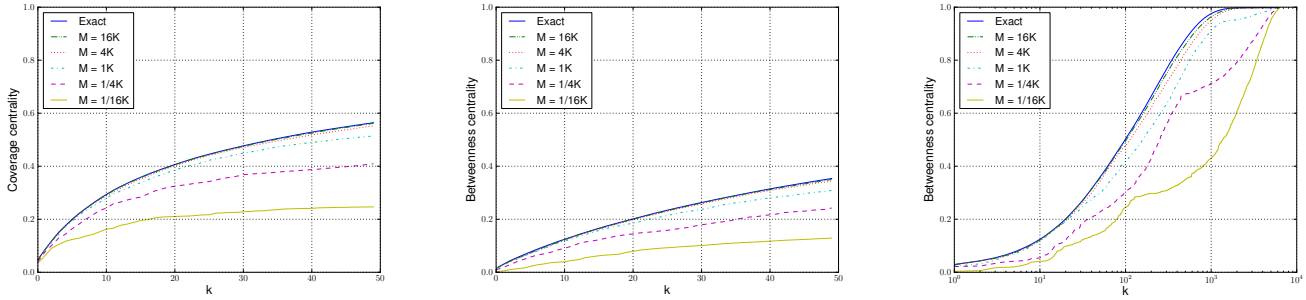
Hence, there is a high probability that Algorithm 4 outputs a $(1 - 1/e, \epsilon n^2)$-approximation to MBC_k.

3.3 Runtime

As opposed to the coverage centrality, we do not have a linear runtime in the worst case, because we must keep updating the vertex weights. However, we can bound the runtime with the following parameter

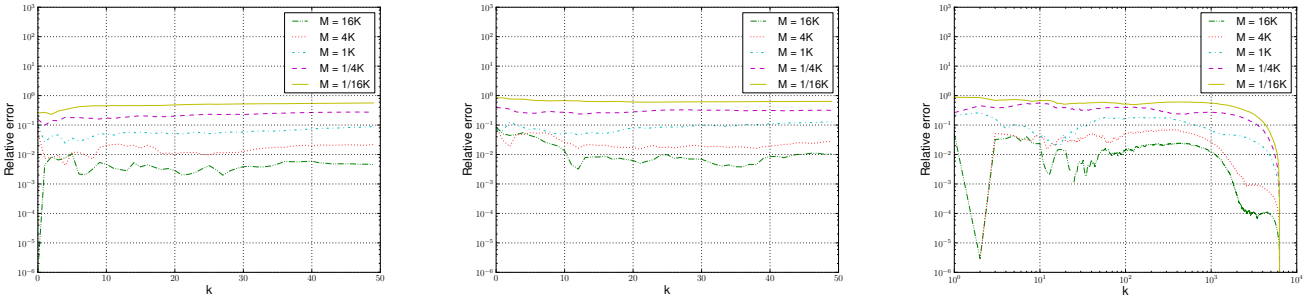
$$h = \mathop{\mathbf{E}}_{s,t \in V}[|P_{st}| \cdot |E(P_{st})|].$$

Theorem 3.4. *Algorithm 4 can be implemented so that its expected runtime is $O((n + m + h \log n) \log n/\epsilon^2)$.*



(a) Coverage centrality ($k \leq 50$) (b) Betweenness centrality ($k \leq 50$) (c) Betweenness centrality ($k \leq n$)

Figure 1: Accuracy of our method against the exact method



(a) Coverage centrality ($k \leq 50$) (b) Betweenness centrality ($k \leq 50$) (c) Betweenness centrality ($k \leq n$)

Figure 2: Relative error of our method against the exact method

PROOF. Build-Betweenness-Hypergraph clearly runs in $O((n+m)M)$ time.

For Top-k-ABC, we first examine the total runtime caused by updating the vertex weights. Let e be a hyperedge in H, with corresponding DAG P_{st}. When choosing a vertex $v \in V(e)$, we need $O(|E(P_{st})|)$ time to update the weights of vertices in $V(e)$. In addition, we need to update the weights of vertices in e at most $|P_{st}|$ times throughout the algorithm. As we sample M pairs of vertices at random when constructing H, the expected total runtime is $O(hM)$.

Because we must find vertices with the maximum weight at most hM times, using a standard priority queue (note that we are dealing with real values), the total time required to find vertices with the maximum weight is at most $O(hM \log n)$.

Hence, the expected total runtime is $O((n+m)M + hM + hM \log n) = O((n+m+h\log n)\log n/\epsilon^2)$. □

COROLLARY 3.5. *With a probability of at least* 99/100, *Algorithm 4 outputs a* $(1-1/e, \epsilon n^2)$*-approximation to* MBC_k *in* $O((n+m+h\log n)\log n/\epsilon^2)$ *time.*

PROOF. By Theorem 3.5 and Markov's inequality, Algorithm 4 stops in $O((n+m+h)\log n/\epsilon^2)$ time with a probability of at least 199/200. By Theorem 3.3, we have a probability of at least 199/200 of obtaining the desired approximation. By the union bound, we have the desired result. □

In Section 4, we empirically show that h is much smaller than $n+m$ in real-world graphs, and hence our method runs in almost linear time.

Table 1: Datasets

Dataset	n	m	h
ego-Facebook	4,039	88,234	2.27×10^2
ca-GrQc	5,242	14,490	1.97×10^2
p2p-Gnutella08	6,301	20,777	5.39×10^2
email-Enron	36,692	183,831	2.63×10^3
soc-Epinions1	75,878	405,740	6.01×10^3
ego-Twitter	81,306	1,342,303	9.85×10^3
web-Google	875,713	4,322,051	3.58×10^3
roadNet-PA	1,088,092	1,541,898	2.57×10^5
roadNet-TX	1,393,383	1,921,660	4.20×10^5
as-Skitter	1,696,415	11,095,298	4.18×10^5

4. EXPERIMENTS

We conducted experiments on a Linux server with an Intel Xeon E5-2690 (2.90 GHz) processor and 256 GB of main memory. The experiments required, at most, 4 GB of memory. All algorithms were implemented in C++.

We considered various types of network, including social, computer, and road networks. All datasets utilized in this paper are available from the Stanford Network Analysis Project (http://snap.stanford.edu/index.html). All datasets were treated as undirected and unweighted graphs. Table 1 shows

Table 2: Runtime of algorithms for MCC_k. DNF means that the experiment did not finish in 12 h.

	$k = 50$					$k = n$				
Dataset	Exact	Sampling method		This work		Exact	Sampling method		This work	
		$M = 1$K	16K	$M = 1$K	16K		$M = 1$K	16K	$M = 1$K	16K
ego-Facebook	0.54 h	15.2 s	245 s	0.31 s	5.06 s	DNF	8.68 h	DNF	0.33 s	4.95 s
ca-GrQc	1.57 h	7.4 s	111 s	0.20 s	2.21 s	DNF	3.93 h	DNF	0.27 s	2.36 s
p2p-Gnutella08	2.91 h	14.1 s	221 s	0.30 s	4.46 s	DNF	8.17 h	DNF	0.41 s	4.69 s

Table 3: Runtime of algorithms for MBC_k. DNF means that the experiment did not finish in 12 h.

	$k = 50$					$k = n$				
Dataset	Exact	Sampling method		This work		Exact	Sampling method		This work	
		$M = 1$K	16K	$M = 1$K	16K		$M = 1$K	16K	$M = 1$K	16K
ego-Facebook	215 s	19.3 s	309 s	1.22 s	14.29 s	3.60 h	8.48 h	DNF	1.80 s	29.77 s
ca-GrQc	165 s	12.1 s	190 s	0.48 s	6.04 s	3.74 h	3.96 h	DNF	0.62 s	9.90 s
p2p-Gnutella08	392 s	28.5 s	449 s	0.70 s	11.61 s	9.58 h	7.88 h	DNF	1.33 s	22.48 s

the basic statistics of the datasets used in our experiments. To estimate the parameter $h = \mathbf{E}_{s,t \in V}[|P_{st}| \cdot |E(P_{st})|]$, we sampled 100,000 pairs of vertices (s, t) at random, and took the average of $|P_{st}| \cdot |E(P_{st})|$.

For the exact greedy algorithm for MCC_k (resp., MBC_k), we use that described in Section 2 (resp., Section 3), whose runtime is $O(kn^2m)$ (resp., $O(knm)$).

In this section, the symbol K denotes $2^{10} = 1024$.

4.1 Accuracy

To confirm the accuracy of our method, we conducted the following experiment for both MCC_k and MBC_k using the p2p-Gnutella08 dataset: For each problem, we executed the exact method to obtain the set of k vertices. We then executed our method to obtain the set of k vertices, and recomputed its exact centrality.

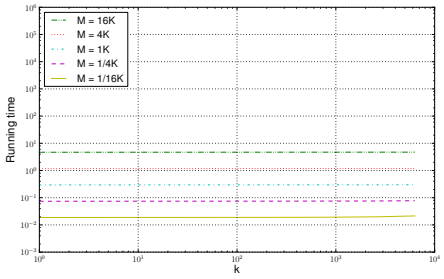
Figure 1 shows the accuracy of our method with various choices of k and M. For MCC_k, we could only use $k \leq 50$, as the exact method for MCC_k is very slow. Centralities are divided by n^2 so that the range becomes $[0, 1]$ and we can easily see how the addtive error of ϵn^2 affects the quality. As can be observed from the figure, our method is very accurate when $M = 4$K and $M = 16$K. When $k = n$, the accuracy of our method deteriorates, as it outputs more vertices, especially when M is small. This is because our method only approximates the centrality to within ϵn^2, and the ordering among vertices with an adaptive centrality of less than ϵn^2 could be random.

Figure 2 shows the relative error of our method against the exact method using the same experimental setting as in Figure 1. As we can confirm from the figure, the relative error is less than 1% in most cases when $M = 16$K.

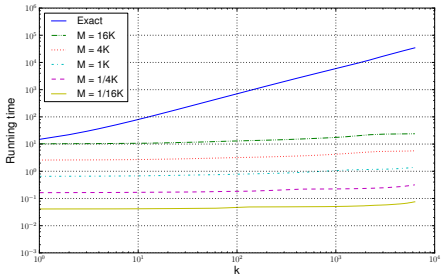
Note that the set (and the ordering) of vertices obtained by the exact method and our method may be very different. From these experiments, however, we can confirm that our method is accurate in the sense of solving MCC_k and MBC_k.

4.2 Runtime

Tables 2 and 3 summarize the runtime of our method and previous methods for MCC_k and MBC_k, respectively. Besides the exact greedy algorithm, we also implemented a sampling method that considers a fixed number, say M, of pairs of vertices, instead of all pairs, when computing the centrality. This can be viewed as a greedy algorithm that uses the method of Brandes and Pich [8] when computing the centralities.



(a) Coverage centrality



(b) Betweenness centrality

Figure 3: Runtime as a function of k

When $k = 50$, our method runs two or three orders of magnitude faster than the exact method, and at least one order of magnitude faster than the sampling method.

The difference is even larger when $k = n$. In many cases, previous methods did not finish the computation within 12 h. As our method finishes in less than 30 s for all cases, our method runs at least three orders of magnitude faster than previous methods. The main reason for this is that the runtime of previous methods is proportional to k, whereas our method always runs in nearly linear time, no matter how large the value of k. We can confirm this observation from Figure 3, which illustrates how the runtime increases as k increases for the p2p-Gnutella08 dataset. There is no

Table 4: Runtime for large graphs

Dataset	MCC_n		MBC_n	
	$M = 1$K	16K	$M = 1$K	16K
soc-Epinions1	4.75 s	75.7 s	10.4 s	156 s
email-Enron	1.47 s	24.0 s	3.28 s	53.0 s
ego-Twitter	18.3 s	268 s	31.3 s	526 s
web-Google	148 s	2333 s	316 s	6131 s
as-Skitter	201 s	3541 s	472 s	7501 s

line corresponding to the exact method for MCC_k, as the program did not finish in 12 h.

Table 4 summarizes the runtime of our method for larger graphs, which demonstrates its scalability.

5. APPLICATIONS

In this section, we consider two applications of adaptive centralities that we have not previously been able to investigate.

5.1 Suppressing epidemics with immunization

Suppose a person is infected with a disease, and the disease spreads out through a social network. We have a limited number of k vaccines, and can immunize at most k people. The objective is to determine an effective immunization strategy that prevents an epidemic. This problem is well-studied in the network science community. It has been reported that a strategy based on ABC, that is, immunizing the top-k vertices with respect to ABC, is the most effective natural strategy [16]. However, as it is quite costly to compute ABC, its effectiveness has only been confirmed for small graphs.

We demonstrate the effectiveness of ABC for larger graphs using our method. To measure the effectiveness of immunization, we use the size of the largest connected component in the graph that results from removing vaccinated vertices, which is standard in the network science community.

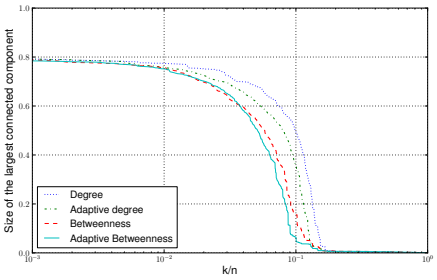
Figure 4 illustrates the performance of the strategy based on ABC. We compare this with strategies based on betweenness, degree, and adaptive degree, which means the degree of the resulting graph after removing selected vertices. As we can see, the strategy based on ABC is most effective in these datasets. This improvement could have a huge impact in the context of suppressing epidemics.

5.2 Indexing methods for answering distances

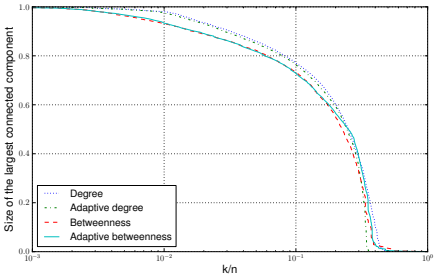
Computing the distance between two vertices is a fundamental graph operation. To quickly compute the distance, it is natural to construct an index from the input graph, and use this when answering queries. Among many others, Akiba et al. [2] proposed an efficient method based on *pruned landmark labeling*. Given a vertex ordering $v_1, \ldots, v_n$, this approach performs a BFS from v_i, and records the distances from v_i for each $i = 1, \ldots, n$ in this order. A non-trivial claim of this method is that, when the BFS from v_i reaches a vertex w along the shortest path passing through some of $v_1, \ldots, v_{i-1}$, there is no need to expand vertex w in the BFS. Hence, the runtime of the i-th BFS is proportional to

$$\#\{w \mid P_{v_i w} \cap \{v_1, \ldots, v_{i-1}\} = \emptyset\}.$$

In other words, after performing (pruned) BFSs from vertices $v_1, \ldots, v_i$, we do not have to consider pairs (s, t) such



(a) ca-GrQc



(b) ego-Twitter

Figure 4: Size of the largest connected component

that $P_{st} \cap \{v_1, \ldots, v_i\} \neq \emptyset$, and s and t appear after $v_1, \ldots, v_i$ in the ordering. With this observation, it is natural to utilize the vertex ordering based on ACC.

In [2], the authors used a vertex ordering based on degree. This strategy works well when the input graph is a web graph or a social network, as the shortest paths of many pairs pass through high-degree vertices. However, the degree-based strategy does not work well for other kinds of graphs, such as road networks. For such graphs, coverage centrality might be more plausible.

Table 5 summarizes the performance of each strategy for road networks. In all datasets, we set the number of bit-parallel BFSs to be 64 (see [2] for details). The label size of a vertex means the number of distances recorded to that vertex. Note that the index size is (roughly) $(64 + \mathrm{LN})n$, where LN is the average label size. We also chose $M = 4$K in our method.

The best criteria with respect to the indexing time and the label size are strategies based on adaptive coverage and betweenness centrality, respectively. Adaptive centralities are better than their corresponding centralities in terms of the label size and the indexing time, and are comparable in terms of the ordering time. Thus, a strategy based on adaptive centrality may be preferable to one based on the corresponding centrality, which justifies the importance of adaptive centrality.

The label size of the degree-based strategy is larger than that of other strategies. Though its ordering time is much faster than other strategies, the combined ordering and indexing time is slightly longer than that of the ACC strategy.

To conclude, the strategy based on ACC is the best of those studied here, as it has the lowest total indexing time, and the label size is half that of the strategy developed by Akiba et al. [2].

Table 5: Comparison of performance of the pruned landmark labeling method using various strategies. OT, IT, and LN denote the time required to obtain the vertex ordering, the indexing time excluding the vertex ordering, and the average label size of a vertex, respectively.

Dataset	Degree			CC			ACC			BC			ABC		
	OT	IT	LN	OT	IT	LN	OT	IT	LN	OT	IT	LN	OT	IT	LN
roadNet-PA	**0.116** s	331 s	358	251 s	99.9 s	227	252 s	**71.3** s	176	578 s	316 s	411	1865 s	119 s	**166**
roadNet-TX	**0.148** s	446 s	372	303 s	180 s	256	306 s	**137 s**	208	695 s	374 s	404	2832 s	159 s	**192**

6. CONCLUSIONS

We have proposed an almost linear-time approximate method for obtaining the vertex orderings with respect to the *adaptive* coverage and betweenness centralities. Our method is remarkable, because simply obtaining the vertex with the maximum coverage or betweenness centrality requires linear time. The output quality of our method against the exact method is high in the sense of centralities, and our method is three orders of magnitude faster than previous methods.

Our method opens the door to use the vertex orderings with respect to the adaptive coverage and betweenness centralities for large graphs, say, millions of vertices. As illustrating examples, we have empirically shown the effectiveness of strategies based on these orderings in applications that arise from the network science and database communities.

Acknowledgment

The author thanks T. Akiba, C. Seshadhri, and T. Takaguchi for helpful comments.

Y. Y. is supported by JSPS Grant-in-Aid for Young Scientists (B) (No. 26730009), MEXT Grant-in-Aid for Scientific Research on Innovative Areas (No. 24106003), and JST, ERATO, Kawarabayashi Large Graph Project.

7. REFERENCES

[1] T. Akiba, Y. Iwata, K. Kawarabayashi, and Y. Kawata. Fast shortest-path distance queries on road networks by pruned highway labeling. In *ALENEX*, 2014. to appear.

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

[3] D. A. Bader, S. Kintali, K. Madduri, and M. Mihail. Approximating betweenness centrality. In *WAW*, pages 124–137. Springer-Verlag, 2007.

[4] S. P. Borgatti and M. G. Everett. A graph-theoretic perspective on centrality. *Soc. Networks*, 28(4):466–484, 2006.

[5] C. Borgs, M. Brautbar, J. Chayes, and B. Lucier. Maximizing social influence in nearly optimal time. In *SODA*, 2014.

[6] U. Brandes. A faster algorithm for betweenness centrality. *J. Math. Sociol.*, 25(2):163–177, 2001.

[7] U. Brandes. On variants of shortest-path betweenness centrality and their generic computation. *Soc. Networks*, 30(2):136–145, 2008.

[8] U. Brandes and C. Pich. Centrality estimation in large networks. *Int. J. Bifurcat. Chaos*, 17(07):2303–2318, 2007.

[9] T. Coffman, S. Greenblatt, and S. Marcus. Graph-based technologies for intelligence analysis. *Commun. ACM*, 47(3):45–47, 2004.

[10] A. del Sol, H. Fujihashi, and P. O'Meara. Topology of small-world networks of protein-protein complex structures. *Bioinformatics*, 21(8):1311–1315, 2005.

[11] M. G. Everett and S. P. Borgatti. Extending centrality. *Models and methods in social network analysis*, 35(1):57–76, 2005.

[12] L. C. Freeman. A set of measures of centrality based on betweenness. *Sociometry*, 40(1):35–41, 1977.

[13] T. Fushimi, K. Saito, T. Ikeda, and N. Mutoh. Proposing set betweenness centrality measures focusing on nodes' collaboratative behaviors and its application. *IEICE T. Inf. Syst.*, J96-D(5):1158–1165.

[14] R. Geisberger, P. Sanders, and D. Schultes. Better approximation of betweenness centrality. In *ALENEX*, pages 90–100, 2008.

[15] M. Girvan and M. E. J. Newman. Community structure in social and biological networks. *P. Natl. Acad. Sci.*, 99(12):7821–7826, 2002.

[16] P. Holme, B. Kim, C. N. Yoon, and S. K. Han. Attack vulnerability of complex networks. *Phys. Rev. E*, 65(5):056109, 2002.

[17] H. Jeong, S. P. Mason, A. L. Barabasi, and Z. N. Oltvai. Lethality and centrality in protein networks. *Nature*, 411(6833):41–42, 2001.

[18] V. E. Krebs. Mapping networks of terrorist cells. *Connections*, pages 43–52, 2002.

[19] M.-J. Lee, J. Lee, J. Y. Park, R. H. Choi, and C.-W. Chung. Qube: a quick algorithm for updating betweenness centrality. In *WWW*, page 351, 2012.

[20] F. Liljeros, C. R. Edling, L. A. N. Amaral, H. E. Stanley, and Y. Åberg. The web of human sexual contacts. *Nature*, 411(6840):907–908, 2001.

[21] G. L. Nemhauser and L. A. Wolsey. Best algorithms for approximating the maximum of a submodular set function. *Math. Oper. Res.*, 3(3):177–188, 1978.

[22] G. L. Nemhauser, L. A. Wolsey, and M. L. Fisher. An analysis of approximations for maximizing submodular set functions—I. *Math. Program.*, 14(1):265–294, 1978.

[23] M. Newman and M. Girvan. Finding and evaluating community structure in networks. *Phys. Rev. E*, 69(2):026113, 2004.

[24] M. Riondato and E. M. Kornaropoulos. Fast approximation of betweenness centrality through sampling. In *WSDM*, 2014. to appear.

[25] Y. Yano, T. Akiba, Y. Iwata, and Y. Yoshida. Fast and scalable reachability queries on graphs by pruned labeling with landmarks and paths. In *CIKM*, pages 1601–1606, 2013.