ABSTRACT
We consider the reachability indexing problem for private-public directed graphs. In these graphs nodes come in three flavors: public—nodes visible to all users, private—nodes visible to a specific set of users, and protected—nodes visible to any user who can see at least one of the node’s parents. We are interested in computing the set of nodes visible to a specific user online. There are two obvious algorithms: precompute the result for every user, or run a reachability algorithm at query time. This paper explores the trade-off between these two strategies.

Our approach is to identify a set of additional visible seed nodes for each user. The online reachability algorithm explores the graph starting at these nodes. We first formulate the problem as asymmetric k-center with outliers, and then give an efficient and practical algorithm. We prove new theoretical guarantees for this problem and show empirically that it performs very well in practice.

1. INTRODUCTION
User-generated content makes up a large part of the modern Web, ranging from photos and comments to full-fledged documents. An essential part of serving user-generated content is deciding who has access to it. Service providers implement different strategies by which the author can control access to their content. The traditional approach either makes the content public, publishing it to the Web for all to view, or private, restricting access to a whitelisted set of authenticated users. In recent years, an alternative form of access control has emerged—using a shared secret to control access to a document. This frequently involves storing content at a hard-to-guess URL and making the content available to anyone with a link. Lying somewhere between public and private, we call such content protected.

This control mechanism is relatively common in modern social applications. For example, Facebook and Google allow users to share photos via a link containing a hashcode. Dropbox, a cloud storage provider, allows users to quickly share access to files or folders via a link. Sharing by URL balances privacy with utility. On one hand, a total stranger cannot guess a valid hashcode, and therefore is unlikely to gain access to the content. On the other, friends are free to forward the link to others, providing instant access without the need for an account, password or special group memberships.

The setting is particularly powerful when collaborating on documents in an enterprise setting: by treating the link as a secret, the author can forget about explicit access control lists (ACLs) and rely on the social contract to ensure the content is restricted to the intended recipients. This form of sharing also gives rise to a class of protected documents that are transitively accessible to a user, even if they do not directly know the secret URL for the protected document. Suppose a document \(D\) is linked to by a private document \(P\). The private document is only shared with a handful of people. Since the address of \(D\) is private, access to \(D\) is implicitly accessible only to users who can access \(P\). As the set of those who can access \(P\) grows, so does the set of those accessing \(D\), without any additional work from the owner of \(D\).

These cascading permissions have high utility but present a problem when deciding whether a particular user is allowed to see a specific document, a key step in any enterprise search system. Such a system must issue the search query against an individual user’s corpus, rather than the entire document corpus, and return results ranked by relevance. To determine whether a document belongs to a user’s corpus, one needs to trace the document’s provenance. At a high level, a document may be accessible to the user for three different reasons:

1. The document is public, and thus viewable to all users.
2. The document is private, but the user is specifically included on its ACL.
3. The document is protected, but the user has transitive access via another document that links to it.

In the third case, although the user is not explicitly listed as one who can access the document, she has access to the secret link, and could plausibly discover the document on her own.

More formally, let \(A\) be the set of documents visible to all users (public documents), \(P\) be the private documents, visible to a named subset of users, and \(D\) be the protected documents. We can create a document graph with \(V = A \cup P \cup D\) as the vertices, and a directed edge \(i \to j\) if document \(i\) links to document \(j\). If \(P_u \subseteq P\) is the set of
private documents visible to a particular user \( u \), then the set of all documents visible to \( u \) is \( R_u = \text{Tr}_{A \cup P_u \cup D}(A \cup P_u) \), where for sets \( S \subseteq U \subseteq V \), \( \text{Tr}_U(S) \) denotes the transitive closure of \( S \) in \( U \), i.e., the set of documents reachable from \( S \) in the graph induced by \( U \). Figure 1 illustrates our setting.

Formally, the reachability problem is as follows.

**Definition 1** (Reachability). Given a set of public documents \( A \), private documents \( P_u \) visible by a given user \( u \), protected documents \( D \), and directed edges \( E \) between documents, find all documents reachable from \( A \cup P_u \).

There are two ways in which our work differs from the classical work on reachability indexing. First, the ACLs in any production system are bound to be dynamic: both the permissions on individual documents, as well as the link structure between them will change over time. At query time we need to be sure to return only those documents reachable at that point in time. Fortunately, these changes are relatively rare, but still the system needs to certify its results.

Second, we can take advantage of parallelism in exploring the reachability graph. We give a principled approach to smoothly trade-off latency and storage costs. The bookends of the trade-off are obvious: if storage is not an issue we simply pre-compute the reachable set \( R_u \) for each user \( u \). In this approach, the latency at query time is small because we just need to do one table lookup. However, this approach does not certify results in the face of changes to the graph. If latency is not a worry, we can simply explore the graph at query time via breadth-first search (BFS) from the set \( A \cup P_u \): the resulting set of nodes is certified by the BFS tree itself. However, for graphs with large diameters this is infeasible in practice. Even if we parallelize the BFS computation, no layer of the BFS tree can be explored until at least one node in the previous layer has been discovered, so we will incur latency at least proportional to the depth of the tree, which can be as large as the diameter of the graph.

Instead, we suggest pre-computing a small set of seed nodes for each user so that at query time, we can perform a shallow BFS from the nodes in \( A \cup P_u \) as well as the seed nodes, in parallel. The seed nodes are chosen carefully to limit the depth of the BFS that we need to explore, and are themselves certified during the process. In this manner, we always respect the permissions and take advantage of parallelism to limit online latency. This leads us to define the seed selection for reachability problem:

**Definition 2** (Seed Selection). Given an instance of the reachability problem \((A, P_u, D, E)\) and a bound \( r \) on the number of BFS rounds, find a minimum-size set of seed nodes \( S \subseteq D \), such that:

- Every node in \( S \) is \( r \)-hop certified, meaning that it is within \( r \) hops of either \( A \cup P_u \) or another \( r \)-hop certified node in \( S \).
- Every node in \( \text{Tr}_{A \cup P_u \cup D}(A \cup P_u) \) is within \( r \) hops of \( A \cup P_u \cup S \).

The first property allows an \( r \)-hop parallel BFS to certify some seed nodes \( S_0 \), while also detecting “certification edges” \( u \rightarrow v \) between nodes \( u, v \in S \) for which there is an \( r \)-hop path from \( u \) to \( v \). The rest of the nodes in \( S \) can then be certified via a BFS from \( S_0 \) in the (much smaller) graph of these certification edges. We do not care about the diameter of this certification graph because it can easily be collected on a single machine (since we keep \( |S| \) small by design), so it does not require extra network communication to explore this graph once it has been constructed. The second property guarantees that the original \( r \)-hop BFS finds all reachable nodes in \( D \).

Notice that the recursive definition of \( r \)-hop certified gives us much more power to craft efficient seed sets \( S \). If we were to consider a seed node \( S_0 \) to be certified only if it is within \( r \) hops of \( A \cup P_u \), then our certified seed nodes could cover nodes only within \( 2r \) hops of \( A \cup P_u \). By allowing ourselves to certify seed nodes through the concatenation of an arbitrary number of \( r \)-hop paths, we could (in the extreme case) have a certified seed node that is as many as \( r|S| \) hops from \( A \cup P_u \), covering nodes at a distance of up to \( r(|S| + 1) \) hops from \( A \cup P_u \).

Since the set of nodes \( A \) is available to every user, one might be tempted to precompute \( \text{Tr}_{A \cup D}(A) \). However, in order to protect against changes in the graph, we need to perform BFS from \( A \cup P_u \) in order to certify reachability at query time.

The question we tackle in this paper is how to find a nearly optimal set of seed nodes, \( S \). Our main contributions are:

- We present the public-private model for access control, and define the public-private-protected indexing problem for reachability.
- We formulate the indexing problem as an instance of a classic clustering problem, namely asymmetric k-center (Section 4).
- For the version without outliers, we give an improved approximation algorithm that closes most of the gap between the worst-case approximation guarantee and the hardness lower bound. Specifically, our approximation ratio exceeds the hardness lower bound by just a small additive constant, whereas the previous gap was a multiplicative factor of 3 (Theorem 7). As a
bonus, our algorithm is vastly more memory and time-efficient.

- For the version with outliers, we present a new memory-efficient algorithm and prove that it has a small constant-factor bicriteria approximation ratio (Theorem 8).
- We empirically verify the efficacy of our approach on a number of real-world graphs (Section 6).

We must emphasize the distinction between the offline and online components of the proposed indexing system. The seed selection process itself is run offline and we report experimental results from our single-machine implementation. This algorithm outputs a seed set $S$ that could be used in a live system for computing certified reachability online in a massive distributed system, as described above.

To report system independent results, we focus on two metrics: the number of hops as a proxy for the latency in any such system, and the size $|S|$ of the seed set $S$ as a proxy for the (distributed) memory usage in this system. In addition, there are the orthogonal issues of run time and memory usage by the offline seed selection computation. Here, our main improvement over previous work is to avoid the need to take the cube of the entire graph: this seemingly small difference transforms the algorithm from impractical to practical. Of much greater interest is the quality of the seed set $S$ produced by this offline process, as this is what determines the performance of the online system.

2. RELATED WORK

There is a rich literature on reachability algorithms, spanning many decades and hundreds of papers. Here we limit our focus to the most related theoretical and practical contributions.

Theoretical Considerations. Parallel reachability has been intensively studied in the PRAM model [13, 22]. Probably the most popular algorithm in this area has been that of Ullman and Yannakakis [31]. Their algorithm is simple and elegant: the main idea is to start several BFS visits in parallel from multiple seed nodes selected uniformly at random. Interestingly, Ullman and Yannakakis show good theoretical guarantees. This algorithm forms the basis for one of our baselines; however, as shown in Section 6, it performs poorly on real-world instances.

Transitive closure spanners are another related concept. A $k$ transitive closure spanner for a graph $G$ is a directed graph $H = (V, E)$ that has the same transitive-closure as $G$ (for each seed $u \in V$), but diameter at most $k$. Several interesting upper and lower bounds have been shown for transitive closure spanners [3, 15]. This approach is well-suited for offline applications, but these spanners cannot be updated online, and a single edge change in the underlying graph may invalidate the entire spanner.

Indexing for Reachability. Several practical and efficient reachability algorithms have also been proposed in the past several years. The majority of the techniques in this area follow three main approaches: transitive closure compression [1, 20, 25, 32, 34], online search [5, 29, 30, 33], or design of reachability oracles [4, 6, 7, 11, 19, 18, 27, 35]. The transitive closure compression techniques compress the reachability tree to answer queries with a single lookup. Those techniques are CPU-efficient but incur a large memory and precomputation cost. In contrast, online search techniques tend to store very little information but incur a large query cost. Finally, reachability oracles compute a labeling that allows them to efficiently answer reachability queries on pairs of nodes (i.e., is $v$ reachable from $w$?). There are also a few attempts at combining different approaches [17, 24, 28, 38]. For example, Jin et al. [17] design an algorithm that tries to achieve the positive aspects of both transitive closure compression and online search. Practical reachability algorithms have also been proposed for dynamic [37, 39] or uncertain [16] graphs. The main difference between all of the previous techniques and our approach is in the use of parallelization. The algorithms we present can be seen as an online search technique that uses parallelization to overcome the main limitations of the online search framework.

Clustering for Reachability. Our work is also strongly related to the asymmetric $k$-center literature. In this context there are three important related papers. Panigrahy and Vishwanathan [26] were the first to give an approximation algorithm for this problem, obtaining an approximation ratio of $5\log^* n + O(1)$. Archer [2] improves the bound to $3\log^* k + O(1)$. Finally, Chuzhoy et al. [10] show that the problem does not admit an approximation ratio better than $\log^* n - O(1)$ unless $NP \subseteq DTIME(n^{\log \log n})$.

In addition, Gertz and Wirth [14] analyze several variants, and prove that the asymmetric $k$-center with outliers and forbidden centers problem is not approximable at all unless $P = NP$. This problem is the same as asymmetric $k$-center with outliers, except that there may also be some nodes that are disallowed as centers.

Finally, our paper is inspired by new challenges proposed in the recent work on private-public graph algorithms [8].

3. PRELIMINARIES

Given a directed graph $G = (V, E)$, for a node $v \in V$ we denote by $\Gamma^{-}(v)$ the set of in-neighbors of $v$, i.e., $\{u \in V : (u, v) \in E\}$, and by $\Gamma^{+}(v)$ the set of out-neighbors of $v$, i.e., $\{u \in V : (v, u) \in E\}$. We say that $v$ covers nodes in $\Gamma^{+}(v)$ and is covered by nodes in $\Gamma^{-}(v)$.

Furthermore, let $\Gamma^{+}_{h}(u)$ be the set of nodes that node $u$ covers within $h$ hops, and $\Gamma^{-}_{h}(u)$ be the set of nodes that cover $u$ within $h$ hops. For a set $S \subseteq V$, define $\Gamma^{+}_{h}(S) = \cup_{u \in S}\Gamma^{+}_{h}(u)$, and similarly $\Gamma^{-}_{h}(S) = \cup_{u \in S}\Gamma^{-}_{h}(u)$. We denote by $\text{Tr}_{G}(S)$ the set of documents reachable from $S$ in $G$, and will drop the subscript when it is clear from context.

We will show that the Seed Selection problem is closely related to asymmetric clustering, which we define below.

Given a set of points $V$ and a distance function $d : V \times V \to \mathbb{R}$, we call $d$ an asymmetric metric if it satisfies the following properties for all $x, y, z \in V$: (a) $d(x, y) \geq 0$, (b) $d(x, x) = 0$, and (c) $d(x, z) \leq d(x, y) + d(y, z)$. That is, $d$ satisfies all properties of a metric, except for symmetry. In this case, $(V, d)$ is called an asymmetric metric space.

A node $v \in V$ is said to cover node $u \in V$ within radius $r$ if $d(u, v) \leq r$. Let $B_{r}(u) = \{v \in V : d(u, v) \leq r\}$ denote set of all nodes covered by $u$ within radius $r$; for a set $S \subseteq V$, $B_{r}(S) = \cup_{u \in S}B_{r}(u)$, i.e., the set of all nodes covered within radius $r$ by at least one of the nodes in $S$. The coverage radius of a set $S$ with respect to a target set $T$ is $\rho(S, T) := \min\{r : B_{r}(S) \supseteq T\}$, i.e., the smallest radius $r$ such that $S$ covers all of $T$ within radius $r$. We are now in a position to define the asymmetric $k$-center problem.
DEFINITION 3 (AKC). Given a positive integer \( k \), and an asymmetric metric space \( (V,d) \), the asymmetric \( k \)-center problem is to find a set \( S \subseteq V \) of size \( |S| \leq k \) that minimizes \( \rho(S,V) \). The set of nodes in \( S \) are called centers.

Finally, we define \( G_r = (V,E_r) \), where \( E_r = \{(u,v) : u \in V, v \in V, d(u,v) \leq r \} \).

4. REACHABILITY AS ASYMMETRIC CLUSTERING

In this section we explain the connection between the Seed Selection problem defined in the introduction and the AKC problem. We then discuss the impact of outliers on the results, and define outlier versions of both problems.

Given an instance of the Seed Selection problem: nodes \( A, D, P_a \) and edges \( E \), we proceed to define an instance of AKC.

Let \( V = \text{Tr}_{A \cup D}(A \cup P_a) \), be the set of documents reachable by user \( u \), and \( E = E[V] \) be the set of edges visible to the user (i.e., all edges in \( E \) except those pointing to any private documents for which the user is not in the ACL). The directed graph \( G = (V,E) \) defines an asymmetric metric space \( (V,d) \) where \( d \) is the shortest path distance on \( G \). Since \( G \) is undirected, \( d(u,v) \) is just the number of hops in the shortest path from \( u \) to \( v \), or \( \infty \) if \( v \) is not reachable from \( u \). Let \( S \) be a solution with value \( h \) to the AKC problem on \( (V,d) \). Performing a breadth-first search (BFS) for \( h \) hops from \( S \) will discover all of the nodes in \( V \), therefore \( S \) is a good solution for our original Seed Selection problem*.

OUTLIERS. The Seed Selection problem is extremely sensitive to outliers. Consider a path graph, with documents \( x_1, \ldots, x_s \) and document \( x_i \) pointing to document \( x_{i+1} \) for \( 1 \leq i < s \). If \( x_i \) is private to the user, but all of the \( x_{2,\ldots,s} \) are protected, then even doing \( k \) parallel BFS instances will require \( s/k \) rounds. In practice such “tendrils” are common, and may disproportionately influence the value of the solution.

To avoid such large bottlenecks, we consider the outlier versions of both Seed Selection and AKC. In the former, instead of finding all of the reachable nodes, we only require that the \( r \)-step parallel BFS covers at least a \((1-\epsilon)\) fraction of them. The definition of AKC-O is similar.

DEFINITION 4 (AKC-O). Given a positive integer \( k \), an asymmetric metric space \( (V,d) \) and an \( \epsilon > 0 \), the asymmetric \( k \)-center with outliers problem is to find a set \( S \subseteq V \) of size \(|S| \leq k \) that covers at least \((1-\epsilon)\) fraction of \( V \) with the minimum radius. That is, find a set \( S \) that minimizes \( \{r : |B_r(S)| \geq (1-\epsilon)|V|\} \).

5. ALGORITHMS AND ANALYSIS

In this section we will show improved algorithms for the AKC and AKC-O problems, which translate to improved algorithms for Seed Selection, and prove our main theoretical results. While the general proof strategy follows that of [2], the modifications we make to the algorithms lead to a memory-efficient algorithm with better approximation ratios. In what follows, let \( S_{OPT} \) be the optimum solution to the AKC problem.

5.1 Warm Up: Dominating Set

Recall the famous dominating set problem (which is a special case of set cover). Given a directed graph \( G = (V,E) \) the goal is to select the smallest subset of nodes \( V^* \subseteq V \) such that every node \( v \in V \) is either in \( V^* \), or is covered by some node \( u \) in \( V^* \). It is well known that the greedy algorithm, which repeatedly selects a node that covers the most still uncovered nodes, achieves an approximation ratio of \( H(n) \) [21]. Here, \( H() \) is the harmonic function: \( H(x) = 1 + 1/2 + \ldots + 1/[x] + (x-[x])/[x] = \ln x + O(1) \).

One can use the greedy algorithm as a building block to solve AKC. Suppose that in the graph \( G_r \), we use the greedy algorithm to find a dominating set \( S_1 \) of size at most \( kH(\sqrt[3]{k}) \), covering all \( n \) nodes of \( V \) [21]. We can now solve the dominating set problem in the same graph again, but ask to cover only \( S_1 \), this time yielding a cover \( S_2 \) of size at most \( kH(|S_1|/k) \leq kH(\sqrt[3]{k}) \). Since \( S_2 \) covers all of \( S_1 \) in 1 hop, it covers all of \( V \) in 2 hops. We iterate this process until the cover \( S_t \) is sufficiently small. After only \( i = H^{(\sqrt[3]{k})} \) iterations, we will have \(|S_i| = O(k)\), and \( S_i \) covers all of \( V \) in \( i \) hops. Here, \( H^* \) is defined analogously to \( \log^* \).

A variant of this recursive dominating set procedure forms the core of our approach.

5.2 Algorithm

We first describe our algorithm for AKC, then present its analysis, and finally show how to amend it to solve AKC-O.

Our algorithm has three main phases. The first phase is present in all of the methods for AKC [2, 26], the second phase is similar to the recursive procedure above, and we add a third phase to get a better approximation ratio.

**Phase I. Center-Capturing Nodes.** In this phase we identify a special subset of the nodes in the graph.

DEFINITION 5 (Center-Capturing node (CCN)). We say that \( u \in V \) is a center-capturing node if \( \Gamma^- (u) \subseteq \Gamma^+ (u) \).

A center-capturing node \( u \) is named as such because we know that every dominating set includes some center in \( \Gamma^- (u) \). Since the definition of CCN says that \( \Gamma^- (u) \supseteq \Gamma^+ (u) \), then node \( u \) covers this center in 1 hop, and therefore covers in 2 hops everything that this center covers in 1.

In Phase I of the algorithm, we select CCNs to cover nodes from \( S_{OPT} \). Specifically, we maintain an active set \( A \) of nodes we need to cover. If we can find a CCN \( u \in A \), then we choose \( u \) as a center and remove \( \Gamma^+ (u) \) from \( A \). We repeat the procedure until there are no more CCNs in \( A \). Let \( C \) denote the set of centers chosen in this phase.

**Phase II. Recursive Cover.** After Phase I, we are left with a subset of uncovered nodes \( A \subseteq V \) that contains no CCNs. One can show that there exist \( p \) nodes from \( S_{OPT} \) that cover \( A \), with \( p \leq k - |C| \). In Phase II of our algorithm, we use the recursive cover procedure described above to find a set \( C_1 \) of at most \( \frac{3}{2}p \) additional centers that, together with \( C \), cover all of \( V \) in \( H^*(\sqrt[3]{p}) \) hops.

Sadly, we are not yet done because our solution uses \( \frac{3}{2}p + |C| \) centers, which may exceed the \( k \) we are allowed.

**Phase III. Center Reduction.** In this phase we reduce the centers used from \( \frac{3}{2}p \) to \( p \). Although simply continuing the recursive cover phase would eventually result in only \( p \) centers, it would take too many rounds because the rate of progress slows at the end: once \( |S_i| \) drops below \( 2p \), our
bound on the quantity \( |S_i| - p \) drops by only a factor of 2 in each round, so going from \( 2p \) to \( p \) could take \( \log_2 p \) rounds.

To reduce the number of additional centers, phase III performs one more round of dominating set, but does it in the graph \( G_r^3 \) instead of in \( G_r \). Here we use the standard notation that \( G_r^k \) is the cube of \( G_r \), meaning it has the same set of nodes, and each node \( u \) has an edge in \( G_r^3 \) to each node it can reach in 3 hops in \( G_r \).

Because all active CCNs were already chosen in phase I, we can prove there exists a dominating set for \( C_i \) in \( G_r^3 \) of size at most \( \frac{2}{3} p \), so the greedy algorithm finds a cover of size at most

\[
\frac{2}{3} p H_i \left( \frac{|C_i|}{p} \right) \leq \frac{2}{3} p H_i \left( \frac{4p}{p} \right) = \frac{2}{3} p H_i(2) = p.
\]

Thus, we reduce the number of extra centers to \( p \), paying only one extra hop in \( G_r^3 \), which is at most 3 hops in \( G_r \).

For clarity we now give the pseudocode of our algorithm in Algorithm 1.

Algorithm 1: Asymmetric k-center, given a guess at the optimal covering radius \( r \).

**Input:** Asymmetric metric space \((V,d), k \in \mathbb{N}\), a guess \( r \) at the optimal covering radius.

**Construct:** \( G_r = (V, E_r) \), where \( E_r = \{(u,v) : u \in V, v \in V, d(u,v) \leq r\} \). All in- and out-neighborhoods \( \Gamma^+(\cdot) \) and \( \Gamma^-(\cdot) \) are with respect to \( G_r \).

- \( A \leftarrow V, C \leftarrow \emptyset \)
- **while** \( A \) contains a center-capturing node \( u \) do
  - \( C \leftarrow C \cup \{u\}, A = A \setminus \Gamma^+_r(u) \)
  - \( p = k - |C|, S_0 = A, i = 0 \)
- **while** \( |S_i| > \frac{2}{3} p \) do
  - \( S_i \leftarrow S_i \setminus \Gamma^+_r(C) \)
  - **use** the greedy algorithm to find a dominating set \( S_{i+1} \) for the nodes \( S_i \) in graph \( G_r \).
  - \( i \leftarrow i + 1 \)
  - \( C_i \leftarrow S_i \) [For analysis: \( i \leftarrow \hat{i} \)]
  - **construct** \( G_r^3 \), restricted to nodes \( C_i \cup \Gamma^+_r(C_i) \).
  - **use** the greedy algorithm to find a dominating set \( S_2 \) for the nodes \( C_i \setminus \Gamma^+_r(C) \) in graph \( G_r^3 \).
  - **set** the set of centers \( C \cup S_2 \).

In the remaining subsections, we formally analyze each phase of the algorithm and prove its theoretical guarantees.

5.3 Using CCNs to cover the optimal centers

Recall that \( S_{OPT} \) denotes the optimal set of centers. In the first phase of our algorithm we focus on covering centers in \( S_{OPT} \). We claim that every time we select a new CCN \( u \) and add it to \( C \), at least one new center from \( S_{OPT} \) appears in \( \Gamma^+(C) \). Since \( S_{OPT} \) is a dominating set for \( V \), there exists some \( v \) in \( S_{OPT} \) (possibly \( u \) itself) that covers \( u \), which implies \( v \in \Gamma^+(u) \). Since \( u \notin \Gamma^+_r(C) \), we know \( v \notin \Gamma^+_r(C) \).

Therefore, at the end of phase I, there exists a subset of \( S_{OPT} \) of size at most \( p = k - |C| \) that dominates \( A \).

5.4 Recursive dominating set

As noted before, the greedy algorithm yields a dominating set \( S_1 \) satisfying \( |S_1| \leq p H_i(S_0/p) \leq p H_i(n/p) \). Inductively, \( |S_i| \leq p H_i^{(i)}(n/p) \), where the notation \( H_i^{(i)} \) means to apply the \( H(\cdot) \) function iteratively, \( i \) times. For \( 1 < y < x \), let \( H_i^y(x) \) denote the smallest iterate \( j \) such that \( H_i^{(j)}(x) \leq y \). By iteration \( i = H_i^{\hat{i}}(n/p) \), we would have \( |S_i| \leq \frac{1}{3} p \), and phase II would have already ended. Therefore, \( \hat{i} \leq H_i^{\hat{i}}(n/p) \), where \( \hat{i} \) is defined in the pseudocode from Algorithm 1.

We can prove by induction that at the end of the \( i \)th iteration in phase II, each node in \( A \) is covered either in \( i \) hops by one of the centers in \( S_i \), or in at most \( i + 1 \) hops by one of the centers in \( C \). Therefore, at the end of phase II, we have the following guarantees: \( |C_1| \leq \frac{2}{3} p, i \leq H_i^{\hat{i}}(n/p), \) and \( V \subseteq \bigcup_{i=1}^{p} (C_i) \).

5.5 Reducing to \( k \) centers

To conclude our analysis we need to reduce the numbers of centers outside \( C \) from \( \frac{2}{3} p \) to \( p \). Fortunately, we can do it by running the greedy algorithm to find a near-optimal dominating set \( C_2 \) for the nodes \( C_i \setminus \Gamma^+_r(C) \) in graph \( G_r^2 \). In fact, we can use the following useful lemma from [2].

Lemma 6. [Restated from [2]] Suppose \( A = V \setminus \Gamma^+_r(C) \), \( A \) contains no CCNs, and \( S \) covers \( A \). Then there exists a set \( T \subseteq S, |T| \leq \frac{1}{3} |S| \) that covers \( A \setminus \Gamma^+_r(C) \) in 3 hops.

At the end of phase I, \( S_{OPT} \setminus \Gamma^+_r(C) \) contains at most \( p \) centers and covers \( A \). By construction, \( A \) contains no CCNs. Therefore, Lemma 6 implies there exists some set of at most \( \frac{2}{3} p \) centers from \( S_{OPT} \) that covers \( C \setminus \Gamma^+_r(C) \) within 3 hops in \( G_r \). Applying the greedy guarantee, \( |C_2| \leq \frac{2}{3} p H_i(\frac{4p}{p^2}) = \frac{2}{3} p H_i(2) = p \).

At the end of phase II, recall that some of the nodes \( V_i \subseteq V \) are covered by \( C_1 \) in \( \hat{i} \) hops, and the rest are covered by \( C \) in \( i + 1 \) hops. Some of \( C_1 \) is covered by \( C_2 \) in 1 hop in \( G_r^2 \) (3 hops in \( G_r \)), and the rest is covered by \( C \) in 4 hops (in \( G_r \)). Therefore, our final solution \( C \cup C_2 \) covers \( V_i \) in at most \( 4 + i \) hops and \( V \setminus V_i \) in at most \( i + 1 \) hops. Therefore, we arrive at the following theorem.

Theorem 7. When run with the true optimal covering radius \( r \), Algorithm 1 is a \( \left(H_i^{\hat{i}}(n/p) + 4\right)\)-approximation algorithm for ACK.

By comparison, the AKC algorithms in [2] achieve an approximation ratio of \( 3H_i^{\hat{i}}(n/p) + O(1) \), and the hardness lower bound of [10] is \( H_i^{\hat{i}}(n/p) + O(1) \).

5.6 Guessing \( r \) and theoretical guarantees

So far we have assumed that we know the correct value for \( r = r^* \), even though \( r^* \) is NP-hard to compute. Fortunately, we do not actually need to know \( r^* \). Since \( r^* \) is the distance between some pair of nodes in \( V \), there are only \( O(n^2) \) possible values, so we can try all possibilities and return the best solution. Better yet, we can binary search among these \( O(n^2) \) values for the smallest value of \( r \) for which our algorithm returns a set of at most \( k \) centers. Whenever we guess \( r \geq r^* \), the algorithm will run as described in the analysis above. Whenever we guess \( r < r^* \), the algorithm may fail, either because phase II does not terminate within the expected number of iterations, or because phase III returns too many centers. But if the algorithm luckily succeeds, the covering radius of the solution returned will be even better than the guarantee.

Running time. In all of our experiments, the seed set computations finished within minutes, despite not being optimized for speed. Below, we bound the construction time
our algorithm is more efficient from both the memory and
computational point of view. As we discuss in the next subsection, we can
limit our attention to the region near $G_T$.

Since we must try at most $O(\log n)$ values in the binary
search for $r$, the overall running time is at most

$$O((H_{\epsilon/3}(\gamma/p)\log n) + |E(G^3_\epsilon)|) \log n).$$

This is an improvement over the running time in [2], since
for each guess at $r$, we perform only one dominant set
computation on $G^3_\epsilon$ and the rest on $G_T$ whereas [2] performs all
of them on $G^3_\epsilon$.

**Practical Dominating Set.** We describe how to run the
greedy dominating set algorithm on a directed graph $G = (V, E)$ in $O(|E|)$ time. Then we explain how to run it
on $G^3_\epsilon$ without blowing up the memory footprint.

In the greedy algorithm, we maintain a priority queue of
nodes to select as centers. Denote the priority of node $v$
by $Pr(v)$. Initially, $Pr(v) = |\Gamma^+(v)|$. In each iteration, we
select a node $u$ with maximum priority. We then iterate
over all nodes in $\Gamma^+(u)$ and mark them as “covered.” For
each $v$ that is newly covered, iterate over all $w \in \Gamma^+(v)$
and decrement $Pr(w)$, since $w$ now covers one fewer active
node. Each node is marked as newly covered exactly once, so
there are exactly $|E|$ decrement operations. Instead of using
a standard heap-based priority queue, which would lead to
a time of $O(|E| \log |V|)$, we implement it as a sequence of
arrays indexed by $i = 1, \ldots, \max_{v \in V} |\Gamma^+(v)|$. Array $i$
holds all of the node ids whose priority is currently $i$. We maintain
the priority queue in a lazy fashion, as follows. When a node
$u$ is popped off the top of the priority queue at level $i$, we
check that $Pr(u)$ is really still $i$. If so, we use it. If not, we
reinsert it into the queue at the end of array number $Pr(u)$. This
yields a running time of $O(|E|)$.

To run this greedy algorithm on $G^3_\epsilon$, we need to compute
the sets $\Gamma^{-}_3(v)$ only for the nodes $v \in C_1$ since those are
the ones we need to cover. From our theoretical guarantees
there are at most $\frac{3}{4}p = O(k)$ such nodes. So the relevant
portion of $G^3_\epsilon$ contains only $O(kn)$ edges. For this reason
our algorithm is more efficient from both the memory and
time perspective when compared with previous algorithms\footnote{In fact, our algorithm need not cube the entire graph
explicitly. As we discuss in the next subsection, we can
limit our attention to the region near $C_1$.}

### 5.7 AKC with outliers

To address the AKC-O problem, we make a small change to
phase II of Algorithm 1. In the first iteration of that
phase, we terminate the greedy algorithm as soon as we
cover $(1 - \epsilon)\gamma |V|$ nodes. By the standard analysis, $S_1$ will
contain at most $pH(\sqrt{\gamma})$ centers. In iterations $i \geq 2$ of phase
II, we must cover $\gamma$ of the centers in $S_i$, so the outliers
benefit us only once. The rest of the analysis goes through
exactly the same way, leading to the following theorem.

\footnote{Previous algorithms [2, 26] are impractical as they compute
all of $G^3_\epsilon$ explicitly.}

---

**Table 1:** Statistics on Graphs used in the experimental evaluation.

| Graph      | $|V|$    | $|E|$     |
|------------|---------|-----------|
| AS-Skitter | 1,696,415 | 11,095,298 |
| Gowalla    | 196,591  | 950,327   |
| YouTube    | 1,134,890 | 2,987,624  |
| DocSample  | 2,406,082 | 4,939,033  |

**Theorem 8.** With the modification above, Algorithm 1 is
a $4 + H_{\epsilon/3}(\sqrt{\gamma})$-approximation algorithm for AKC-O.

### 6. EXPERIMENTS

In previous sections we formulated the public-private graph
indexing problem, and gave algorithms with provable
approximation guarantees. In this section we explore the
performance of our approach on real world datasets, and discuss
its advantages over simpler heuristic baselines.

#### 6.1 Experimental Setup

We begin by describing the data, the metrics, and the
baselines we compare against.

##### 6.1.1 Data

We evaluate our algorithms on a number of real world
graphs. We give the salient statistics about each of these in
Table 1. The AS-Skitter, Gowalla, and YouTube graphs
are taken from the SNAP library [9, 23, 36]; we complement
these with DocSample, a graph derived from a subset of an
anonymized corporate document corpus.

To construct the DocSample graph we consider a subset of
Google’s internal corporate document corpus. We create a
node for each document in the subset, and create a directed
edge $u \rightarrow v$ between two documents if document $u$ links to
document $v$. We do not consider the content of the
documents, and we label the documents sequentially, thereby
further obfuscating the document id. This real DocSample
data set is the most interesting graph to us, as it is precisely
the one that motivated this work.

To imitate the motivating scenario of public, protected,
and private documents for each graph we select a set of roots
R, uniformly at random from among all of the nodes, and
compute the transitive closure $Tr(R)$: that is, the set of all
nodes reachable from $R$.

##### 6.1.2 Metrics

Given a root set $R$ and the nodes $Tr(R)$ we wish to cover,
our goal is to find the fewest number of seeds $k$, that cover
as many nodes $(1 - \epsilon)|Tr(R)|$ as possible, in as few hops $r$
as possible. Therefore, the output of the algorithm can be
parametrized by a triple $(k, r, \epsilon)$. Fixing any two of these
dimensions leads to a well-defined optimization problem.
For example, fixing the number of seeds $k$ and the maximum
number of hops $r$, the goal is to cover as many of the nodes
in the transitive closure $Tr(R)$ as possible. For consistency,
we always augment the solution $S$ of the algorithm with the
initial roots $R$ and report the overall results $(k, r, \epsilon)$. For
any given augmented solution $S \cup R$, we report $k = |S|$, and
$\epsilon$ is a function of $r$: the more hops we allow, the more
nodes we cover, corresponding to smaller $\epsilon$. Including $R$ in
the solution guarantees that we can always achieve $\epsilon = 0$ for
sufficiently high $r$. 

---

- 1466
6.1.3 Baselines

We consider two baselines for the problem. The first is to select \( k \) seeds uniformly at random from the graph. This approach is motivated by the parallel algorithm for this problem due to Ullman and Yannakakis [31]. As we will see below, this approach is more of a sanity check, and tends to perform very poorly in practice.

The second approach, which we denote by \texttt{Degree} does a natural greedy approximation: we first sort all of the nodes by out-degree, and then select the top \( k \) nodes in this sequence. As shown below, this approach tends to perform well only when the problem is "easy," that is, either the allowed number of outliers is large, the radius is large, or the number of seeds is large.

6.1.4 Practical considerations

We describe two considerations for making our asymmetric \( k \)-center algorithm practical. First, we specify how to construct the asymmetric \( \epsilon \)-center instance starting from our reachability instance. Then we discuss the impact of center-capturing nodes (CCNs) in the outlier setting.

Constructing an instance. In the formal algorithm for AKC, we guess the optimal number of hops \( r \) in an outer loop, and pass that as a parameter to Algorithm 1, which then performs computations on \( G_r \). This is the same as \( G^{\epsilon} \) in our case, where the metric is given by shortest paths in graph \( G \). For real graphs, even modest values of \( r \) make explicit construction of \( G^{\epsilon} \) impractical as the graph quickly becomes very dense. In our experiments, we run the algorithm on the original graph \( G \), thereby losing some of our theoretical guarantees, but gaining a practical algorithm.

Specifically, we fix \( r \) and \( \epsilon \) and employ our algorithm to find the smallest set of seeds possible, such that they (along with \( R \)) cover at least \( (1 - \epsilon) \) of the nodes in the graph. The seeds we find will be augmented by the root nodes \( R \) so we initialize \( A \) to be only nodes in \( V \) that are not already covered by \( R \) within \( r \) hops.

We then follow Algorithm 1, identifying CCNs, and running \( r - 3 \) rounds of recursive cover to produce \( C_1 \). The last step is to compute a dominating set for \( C_1 \) in \( G^{\epsilon} \). Critically, this does not require constructing all of \( G^{\epsilon} \), but rather only the \( 3 \)-hop in-neighbors of \( C_1 \), a much smaller set of nodes.

Center-Capturing Nodes. In the first stage of our AKC-O algorithm, we selected CCNs in the graph. As shown in Section 5, this step of the algorithm is fundamental in our theoretical analysis. Nevertheless, removing CCNs is not always a good idea in practice. The reason for this discrepancy between practice and theory is that our theoretical analysis compares the algorithm’s output to the optimum asymmetric \( k \)-center solution \emph{without} outliers. In practice, we are interested in analyzing the problem in the presence of outliers.

As an example where outliers make a difference, consider a large graph with an additional isolated node \( v \). Note that \( v \) is a CCN, since its only \( r \)-hop in- or out-neighbor is itself, so our algorithm will select it. Every optimal solution for AKC has to include \( v \), since no other node covers \( v \). However, in AKC-O, we should clearly let \( v \) be one of our outliers so we can spend our center elsewhere.

In our experiments, we stayed true to the algorithm and included the CCN phase, but we observed cases where skipping the CCN phase would have improved the solution.

Figure 2: Percentage of nodes unreachable from the seeds and roots for a specific radius. The number of baseline seeds selected matched the number of seeds output by our algorithm run with \( \epsilon = 1\% \).

6.2 Overall Coverage

To investigate the performance of our algorithm against the baseline, we look at the overall coverage as a function of \( k \) and \( r \). For each graph, we fix a single, randomly-chosen seed set \( R \). In order to get an apples-to-apples comparison, we fix \( r \) and \( \epsilon \), run our algorithm and obtain some number of seeds \( k \) (which depends on \( r \) and \( \epsilon \)). For the \texttt{Degree} baseline, we then generate \( k \) seeds. For both solutions, we augment the set of seeds with the roots \( R \), and measure the number of nodes that are not covered within \( r \) hops.

Figures 3, and 4 plot the results of both \texttt{Degree} and our algorithm with \( \epsilon = 1\%, 3\%, 5\% \), respectively. Note that our algorithm, by construction, is guaranteed to leave at most an \( \epsilon \) fraction of the nodes uncovered, but may perform even better. Also note that the number of seeds used decreases...
Figure 4: Percentage of nodes unreachable from the seeds and roots for a specific radius. The number of baseline seeds selected matched the number of seeds output by our algorithm run with $\epsilon = 5\%$.

...with $r$, which is why $\epsilon$ can either decrease or increase. For the $\epsilon = 1\%$ and $\epsilon = 3\%$ cases, our algorithm significantly outperforms the greedy heuristic at the same $(k, r)$ pair; the latter is often significantly higher than the desired $\epsilon$. Overall, as $\epsilon$ increases, the problem gets easier, and we see the advantage of our algorithm slowly ebb.

We do not plot the performance of the random heuristic as it performs significantly worse across the board. It gives an $\epsilon$ in the double digits for the same $(k, r)$ values, and needs a radius of 15 or higher to be comparable in terms of coverage.

6.3 Certified Coverage

As mentioned in the introduction, the real graphs motivating our approach are dynamic in nature, and nodes that may be reachable and accessible during the offline pre-computation may no longer be reachable at query time. This problem creates new challenges for any indexing system, but it also highlights a strength of the one we propose. At query time, we must certify that a seed selected by the algorithm can still be reached from one of the nodes to which the user has explicit access.

At query time we restrict ourselves to explore only $r$ hops away from each of the seeds and explicit nodes. Therefore, we can certify a seed if and only if there exists a path of at most $r$ hops to the seed, either from an explicit node or from another seed we have already certified. We then consider as covered only those nodes that are within $r$ hops from a certified seed.

Our asymmetric $k$-center algorithm and the two heuristics do not consider this constraint directly, but we show that our approach remains effective even with this additional constraint. To demonstrate this, we present the results of the certified coverage on the same, unmodified graphs using the same sets of chosen seeds.

We show the results on all of the graphs, for $\epsilon \in \{1\%, 3\%, 5\%\}$ in Figures 5, 6, and 7. As expected, to achieve the same error $\epsilon$, the certified radius is higher than the original. Nevertheless, our algorithm still achieves good performance, and is significantly better than the greedy baseline at most $\epsilon$

Figure 5: Percentage of nodes unreachable from the seeds and roots for a specific radius. The number of baseline seeds selected matched the number of seeds output by our algorithm run with $\epsilon = 1\%$.

Figure 6: Percentage of nodes unreachable from the seeds and roots for a specific radius. The number of baseline seeds selected matched the number of seeds output by our algorithm run with $\epsilon = 3\%$.

Figure 7: Percentage of nodes unreachable from the seeds and roots for a specific radius. The number of baseline seeds selected matched the number of seeds output by our algorithm run with $\epsilon = 5\%$.

6.4 Unreasonable effectiveness of greedy

Although one can construct instances where the greedy dominating set algorithm (without outliers) selects $\text{OPT} \cdot \log |V|$ seeds, conventional wisdom holds that it typically performs much better. For our test graphs, we ran some experiments to show that this is indeed the case. Specifically, we constructed an integer program (IP) model of the dominating set problem, and solved it using the SCIP software package [12]. In Table 6.4, the column labeled $\text{OPT}_{IP}$ is the optimal number of fractional centers in the linear programming relaxation, $\text{OPT}_{IP}$ is the optimal number of actual centers in the integer program, greedy is the number of centers chosen by the greedy algorithm, and gap is the
Figure 7: Percentage of nodes unreachable from the seeds and roots for a specific radius. The number of baseline seeds selected matched the number of seeds output by our algorithm run with $\epsilon = 5\%$.

<table>
<thead>
<tr>
<th>Graph</th>
<th>OPT_{IP}</th>
<th>OPT_{IP}$^*$</th>
<th>greedy</th>
<th>gap %</th>
</tr>
</thead>
<tbody>
<tr>
<td>AS-Skitter</td>
<td>191,027</td>
<td>191,969</td>
<td>196,447</td>
<td>2.33</td>
</tr>
<tr>
<td>Gowalla</td>
<td>41,601</td>
<td>41,613</td>
<td>42,502</td>
<td>2.14</td>
</tr>
<tr>
<td>YouTube</td>
<td>219,166</td>
<td>219,189</td>
<td>220,253</td>
<td>0.49</td>
</tr>
<tr>
<td>DocSample</td>
<td>1,124,571</td>
<td>1,124,571</td>
<td>1,129,564</td>
<td>0.44</td>
</tr>
</tbody>
</table>

Table 2: Comparing optimal fractional, integer, and greedy solutions for dominating set.

percentage gap between OPT_{IP} and greedy. The main take-home point is that greedy is very close to optimal in all four instances, with the largest gap being 2.33%.

7. CONCLUSION

In this work we formulated the indexing problem in modern sharing scenarios, where content can be public, private, or protected. We showed that the indexing problem is an instance of the classic AKC problem, and gave improved approximation algorithms for both AKC and AKC-O. We compared our approach against greedy baselines across a number of different corpora and parameter settings and showed that it performs well in practice.

As has been previously noted by Chierichetti et al. [8], performing computations in scenarios that mix public and private content is a rich avenue for future research. Here we show one such direction. Further improving the indexing schemes to take real-time updates into account remains an interesting open problem. In addition to indexing, there are many open questions in the areas of ranking, learning, and personalization in this domain.

8. REFERENCES


