Efficient Algorithms for Public-Private Social Networks

Flavio Chierichetti[*]
Sapienza University
Rome, Italy
flavio@chierichetti.name

Alessandro Epasto[†]
Brown University
Providence, RI
epasto@cs.brown.edu

Ravi Kumar
Google
Mountain View, CA
ravi.k53@gmail.com

Silvio Lattanzi
Google
New York, NY
silviol@google.com

Vahab Mirrokni
Google
New York, NY
mirrokni@google.com

ABSTRACT

We introduce the *public-private* model of graphs. In this model, we have a public graph and each node in the public graph has an associated private graph. The motivation for studying this model stems from social networks, where the nodes are the users, the public graph is visible to everyone, and the private graph at each node is visible only to the user at the node. From each node's viewpoint, the graph is just a union of its private graph and the public graph.

We consider the problem of efficiently computing various properties of the graphs from each node's point of view, with minimal amount of recomputation on the public graph. To illustrate the richness of our model, we explore two powerful computational paradigms for studying large graphs, namely, sketching and sampling, and focus on some key problems in social networks and show efficient algorithms in the public-private graph model. In the sketching model, we show how to efficiently approximate the neighborhood function, which in turn can be used to approximate various notions of centrality. In the sampling model, we focus on all-pair shortest path distances, node similarities, and correlation clustering.

Categories and Subject Descriptors

H.2.8 [**Database Management**]: Database Applications—*Data mining*; G.2.2 [**Discrete Mathematics**]: Graph Theory—*Graph algorithms*; K.4.1 [**Computers and Society**]: Public Policy Issues—*Privacy*

Keywords

Privacy; Social Networks; Graph Algorithms

[*]Supported in part by a Google Focused Research Award, and by the MIUR-PRIN National Project "ARS TechnoMedia" (Algorithmics for Social Technological Networks).
[†]Work partially supported by a Google Europe Ph.D. Fellowship in Algorithms, 2011 and a Google Focused Award; part of the work done while at the Sapienza U. of Rome.

KDD'15, August 10-13, 2015, Sydney, NSW, Australia.
ACM 978-1-4503-3664-2/15/08.
DOI: http://dx.doi.org/10.1145/2783258.2783354 .

1. INTRODUCTION

A social network is a perfect poster child for a massive graph. It embodies all the complexities and subtleties one might encounter in a typical large graph: degree distributions that are heavy-tailed, the abundance of local structures, the existence of global sparsity, the presence of noisy edges to the constant evolving nature, and so on. The study of social networks has blossomed into a fertile area of research in the last few years. There are several natural questions that are key and unique to social networks. Owing to the scale of the network and its associated idiosyncrasies, many of these questions cannot be answered well by traditional algorithmic tools and techniques. A large effort in the computational social sciences is devoted to the development of new algorithms and algorithmic paradigms for studying social networks — this field, though, still is in its infancy.

Privacy issues are a major factor in the algorithmic analysis of social networks. In fact, privacy controls the way information is shared among the members of the social network, and also influences the way in which the network itself can be viewed and processed by algorithms. Privacy guarantees are implemented in different ways by different social networks. In the simplest case, a user can mark some of her friends as private; this would make the edges between this user and these friends visible only to the user herself. In a different instantiation of privacy, a user can be member of a private group; in this case, all the edges among the group members (in the extreme case, a clique) are to be considered private. Thus, each user in the social network has her *own* view of the link structure of the network.

In a recent study [16], Dey et al. crawled a snapshot of 1.4 million New York City Facebook users and reported that 52.6% of them hid their friends list. To illustrate the scenario further, consider a social recommendation algorithm. The social network provider could use only the public portion of the network to run the algorithm and build recommendations; this will easily ensure privacy for all the users. But, this will not benefit nodes whose edges are overwhelmingly private; as noted in the NYC example, there can be several such nodes. Alternatively, social network providers can, naively speaking, run the algorithm once for each user, on the union of the public portion of the network and the user's private network. This also clearly respects the overall privacy requirements, however, it is grossly inefficient.

In this paper, we initiate the problem of designing efficient algorithms in the public-private network model. We seek algorithms that improve upon a naive implementation (e.g., running the vanilla algorithm on each of the different networks seen by the various users; the network seen by a user is the union of her private network and the public network). As a first step, we formalize the notion of the public-private graph model. We show our formalization is simple enough to be algorithmically useful, while is rich and realistic enough to be practically relevant.

Our contributions. In the *public-private* graph model, there is a public graph G whose nodes are the users and that is visible to all users in the network. Each node u in the public graph has a private graph G_u associated with itself; the nodes of G_u are users from the public graph, and G_u is only known to u. We stipulate that the private graph cannot be arbitrary. Each node v in the private graph G_u is at most distance two from u. While this might seem restrictive at first, it is sufficient to capture many interesting privacy settings. For example, if G_u is a star centered at u, then it captures the setting where u can mark certain of her friends as private. Likewise, if G_u is a clique containing u, then it captures the setting where u is part of a private group. Our models allows G_u to be a bit more general, supporting the following example. Consider the case of sharing a private circle in Google+. When a node u builds a private circle, it can choose that the members of the circle are visible to the rest of the circle. Consider two members v and w of this circle that are not in each others' public or private circles. Then the edge (u, v) is private to w and the edge (u, w) is private to node v. In other words, $(u, w) \in G_v$ and $(u, v) \in G_w$, but neither of v nor w is connected to the other in the public graph, or in either or their private graphs.

We call an algorithm to be efficient in this model if it can compute a function on $G \cup G_u$ in time proportional to the size of G_u, i.e., just the number of edges in the private graph. The algorithm is allowed to preprocess the public graph G to store a (succinct) synopsis of G.

In this model, we consider two powerful computational paradigms for massive graphs, namely, sketching and sampling. Sketching algorithms have been developed for some basic graph problems including connectivity and cut sizes and more social-network specific problems such as neighborhood estimation and reachability. Some of the sketching algorithms produce composable sketches, i.e., the sketches of two graphs can be combined to get a sketch of their union. Adapting such algorithms to the public-private model is immediate (and thus less interesting). Our neighborhood estimation problems, though, have a quite non-trivial composition. We obtain an efficient algorithm for estimating reachability counts in the public-private model; as it is known from earlier work, this quantity can be an effective heuristic for estimating various network centrality measures.

In the sampling setting, we illustrate our model with non-trivial algorithms for three key social network problems: estimating all-pair distances, estimating node (pairwise) similarities, and correlation clustering. For the first two problems, we obtain sampling-based algorithms that are efficient in the public-private model. For correlation clustering, we show how to efficiently update a clustering solution on the public graph using the edges in the private graph.

We illustrate the effectiveness of our model and the computational efficiency of our algorithms by performing experiments on real-world social networks.

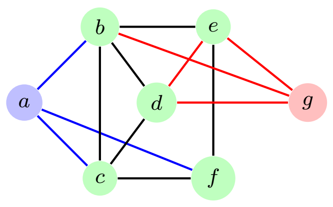

Figure 1: An example of an undirected public-private graph. Here, the blue edges out of node a are private to a and the red edges out of g and the edge (d, e) are private to g. The public graph consists of all the black edges.

2. PUBLIC-PRIVATE GRAPHS

We define the *public-private model* of graphs. In this model, we have a directed graph $G = (V, E)$, where V is the set of nodes and E is the set of edges; this graph is called the *public* graph. Let $n = |V|$ and $m = |E|$.

For each node $u \in V$, we have an associated *private* graph

$$G_u = (\{u\} \cup V_u, E_u),$$

where $V_u = \{v_1, \ldots, v_k\} \subseteq V$ and

$$E_u \subseteq (\{u\} \times V_u) \cup (V_u \times \{u\}) \cup (V_u \times V) \cup (V \times V_u).$$

Sometimes, we will consider the special case when G_u is a *star* centered at u; in this case, $E_u = \{u\} \times V_u$. For simplicity, we use $G \cup G_u$ to denote the graph with V as the set of nodes and $E \cup E_u$ as the set of edges. In the above definitions, it is without loss of generality to assume $u \in V$ and $E \cap E_u = \emptyset$. Furthermore, these notions can be extended to the weighted case in an obvious manner.

Let $\mathsf{Out}(u)$ denote the out-neighborhood of u and let $\mathsf{Out}_2(u)$ denote the two-level out-neighborhood of u in $G \cup G_u$. While these definitions are for directed graphs, it is easy to extend them to undirected graphs.

In this paper we study efficient algorithms for various basic graph problems in this model. In particular, we focus on the following computational question: how best we can *preprocess* the public graph G so that we can answer queries about or compute properties of $G \cup G_u$, for an arbitrary u, as *efficiently* as possible? Ideally, the preprocessing of G should use space quasilinear in n and time poly(m); the computation on $G \cup G_u$ should use time/space near-linear in $|E_u|$ and poly$(\log n)$. Note that the central parameters of interest are the preprocessing time and space of G and the query time to compute a property of $G \cup G_u$.

We consider two flavors of algorithms in this model. In the first we focus on sketching algorithms. In the second we focus on sampling algorithms.

Warm-up: Number of connected components. To gain familiarity with the model before presenting other results, we first show how to efficiently solve a simple problem in this setting: computing the number of connected components in undirected graphs when G_u is a star. In this setting we have access to G and all the G_u in advance and we want to be able to compute efficiently the number of connected component in $G \cup G_u$ for each node u.

In the preprocessing step, we compute the connected components of the public graph. We then assign a component identifier to each node and store this information; this takes $O(m)$ time and $O(n \log n)$ space since all we need is to store the label $\ell(u)$ of the component to which each node $u \in V$ belongs. Furthermore, we also store the total number of connected components.

Now, to compute the number of connected components of $G \cup G_u$, we count the number of different connected components that G_u connects. This takes time $|E_u|$ since we only need to scan all the edges in G_u.

Proposition 1. *We can count the number of connected components in the public-private model using preprocessing time $O(m)$ and space $O(n \log n)$ and query time $|E_u|$.*

3. SKETCHING ALGORITHMS

In this section we show how to use graph sketches to efficiently compute some interesting and non-trivial functions in the public-private graph model. We will deal with directed graphs in this section though the results easily apply to the undirected case as well.

We first make an easy observation about graph sketches. Informally, a graph sketch is *composable* if the sketch of the graph $G \cup H$ can be easily obtained from the sketch of G and the sketch of H. Graph problems that are solvable by composable sketches naturally fit in the public-private model: compute the sketch of G and store it. To compute the function on $G \cup G_u$, first compute the sketch of G_u and use the composability property to compute the sketch of $G \cup G_u$ using the stored sketch for G. Such sketches (which use linear projections, and are hence composable) exist for some basic graph problems such as connectivity and cut-size estimation [1, 2, 28]. Since the answer is easy for these problems, we will not consider them further.

For other problems such as neighborhood estimation, even though sketches exist [7, 10, 11, 29] and even though they are composable, one has to be careful how to compose them efficiently in the public-private model. This is precisely what we do in this section. We show how to efficiently compute the number of nodes reachable from a node u in $G \cup G_u$ and hence efficiently estimate the number of nodes within distance $1, 2, \ldots$, and so on from u. We then use these quantities to get an estimate of some centrality measures as in [8].

3.1 Size of the reachability tree

Given a directed graph $G = (V, E)$, the reachability tree $T_G(v)$ at a node v is defined as any directed tree with v as the root that contains all the nodes reachable from v using the edges in E. Here, for each edge (x, y) in the tree, x is the parent, y is the child, and $(x, y) \in E$. This is a useful structure in general, for example, in a social network setting where the directed edges denote the "follow" relation, the size of the reachability tree represents the number of people who follow a specific user (at the root of the tree) directly or indirectly.

The main tool that we use to efficiently estimate the size of the reachability tree for a specific node is the *bottom-k sketch* [12]. Before describing our algorithm we briefly review the bottom-k sketch and show how it can be used to estimate the size of the reachability tree at a node. Suppose we are interested in estimating the size of an arbitrary subset $V' \subseteq V$. First, we assign to each node v in the graph a number $r(v)$ uniformly at random in $[0, 1]$. Let $r(V')$ denote the set $\{r(v)\}_{v \in V'}$. Assuming $|r(V')| \geq k$, let $\mathsf{Bot}_k(V') \subseteq r(V')$ be the subset of the k smallest elements in $r(V')$ and let $b_k(V') = \max\{\mathsf{Bot}_k(V')\}$, i.e., the kth smallest element in $r(V')$. Now we can estimate the size of V' just looking at $\mathsf{Bot}_k(V')$ and $b_k(V')$; it constitutes the sketch of V'. An estimate of the size of V' is given by

$$\begin{cases} |\mathsf{Bot}_k(V')| & \text{if } |\mathsf{Bot}_k(V')| < k, \\ (k-1)/b_k(V') & \text{otherwise.} \end{cases}$$

As shown in [11, 12] the estimate is accurate; indeed, for any $c > 0$, it is enough to set $k = (2 + c)\epsilon^{-2} \log n$ to have a probability of having relative error larger than ϵ bounded by n^{-c}. A nice property of this sketch is that it is composable; this is crucial in estimating the size of the reachability tree.

Using such sketches it is possible to estimate the size of the reachability tree in a graph efficiently using the following algorithm. As before, each node $v \in V$ has a random number $r(v) \in [0, 1]$. Initially each node v has a sketch $S(v, 0) = \mathsf{Bot}_k(\mathsf{Out}(v))$, i.e., the set of kth smallest numbers assigned to nodes in $\mathsf{Out}(v)$. Then at iteration i, each node v in the graph receives $S(w, i-1)$ for all the out-neighbors $w \in \mathsf{Out}(v)$, and computes $S(v, i)$ as the set of kth smallest numbers in its sketch and that of its neighbors, i.e., $S(v, i) = \mathsf{Bot}_k \left(S(v, i-1) \cup \bigcup_{w \in \mathsf{Out}(v)} S(w, i-1) \right)$. After D iterations, where D is diameter of the graph, $S(u, D)$ is equal to $\mathsf{Bot}_k(T(u))$, i.e., the sketch of $T(u)$, where $T(u)$ is the set of nodes in the reachability tree of u. Thus we can sketch the size of all the reachability tree in G in time $O(mD\epsilon^{-2} \log n)$ and using memory $O(\epsilon^{-2} n \log n)$. Now we will show how to use those sketches to estimate the reachability tree for each node u in $G \cup G_u$.

Theorem 2. *The size of the reachability tree can be approximated to within $(1+\epsilon)$-factor in the public-private model using preprocessing time $O(mD\epsilon^{-2} \log n)$, space $O(\epsilon^{-2} n \log n)$, and query time $O(|E_u|\epsilon^{-2} \log n)$.*

Proof. The main idea of our algorithm is to estimate $T(\cdot)$, i.e., re-compute $S(\cdot)$, only for nodes in $\{u\} \cup \mathsf{Out}(u)$. In order to prove that this is sufficient we first show the following lemma. Intuitively, it says that in-neighbors of u and the in-neighbors of the out-neighbors of u play no role in computing the reachability tree rooted at u.

Lemma 3. *Let $G = (V, E)$ be a directed graph and let $T(u) = T_G(u)$ be the reachability tree rooted at $u \in V$ in G. Let $G' = (V, E')$ be the graph on the same set of nodes with*

$$E' = E \setminus \{(w, v) \in E \mid v = u \text{ or } (v \in \mathsf{Out}(u) \text{ and } w \neq u)\},$$

and let $T'(u) = T_{G'}(u)$ be the reachability tree rooted at u in G'. Then $T(u) = T'(u)$.

Proof. Suppose that the statement is false. Since $E' \subseteq E$, this implies that $T(u) \setminus T'(u) \neq \emptyset$. Let $z \in T(u) \setminus T'(u)$. Now if z is reachable from a path in G but not in G', then this implies that every directed path connecting u to z must have an edge in $\{(w, v) \mid v = u \text{ or } (v \in \mathsf{Out}(u) \text{ and } w \neq u)\}$. Let π be one such path and let $(y, v) \in \{(w, v) \mid v = u \text{ or } (v \in \mathsf{Out}(u) \text{ and } w \neq u)\}$. Now if $v = u$, we can remove all edges after (y, v) and the edge (y, v) and we still get a directed path connecting u to z, which contradicts the hypothesis. So we can assume that $(y, v) \in \{(w, v) \mid v \in \mathsf{Out}(u) \text{ and } w \neq u\}$ be the first edge in $\{(w, v) \mid v \in \mathsf{Out}(u) \text{ and } w \neq u\}$ that we encounter in π. Now from π we can obtain a directed path from u to z that does not use any edge in $\{(w, v) \mid v \in \mathsf{Out}(u) \text{ and } w \neq u\}$ by removing from π all edges after (y, v) and the edge (y, v) and by adding the edge (u, v) that exist because $v \in \mathsf{Out}(u)$. So $z \in T'(u)$, which once again contradicts the hypothesis. This concludes the proof of Lemma 3. $\square$

We continue with the proof of Theorem 2. We will focus on having an estimate of the size reachability tree for u in $G \cup G_u$. Consider $T_G(u)$ and $T_{G'}(u)$, where $G' = (V, E')$, where $E' = E \setminus \{(w, v) \mid v = u \text{ or } (v \in \mathsf{Out}(u) \text{ and } w \neq u)\}$. From Lemma 3, we have $T_G(u) = T_{G'}(u)$.

For simplicity, we first assume we have access to all of $T_{G'}(v)$, $\forall v \in V$. (Note this is not true in practice since we only have access to $T_G(v)$, $\forall v \in V$ and we cannot precompute $T_{G'}(v)$ since they could be potentially different for each node.) We will later show how to eliminate this assumption.

With this assumption, our goal is to estimate $|T_{G\cup G_u}(u)|$. Note that instead of focusing on $T_{G\cup G_u}(u)$ we can work with $T_{G'\cup G'_u}(u)$, where $G'_u = (V_u, E'_u)$ with $E'_u = \{(w, v) \mid v = u \text{ or } (v \in \mathsf{Out}(u) \text{ and } w \neq u)\}$; indeed, using Lemma 3 $T_{G\cup G_u}(u) = T_{G'\cup G'_u}(u)$.

Note that in G' the nodes in $\{u\} \cup \mathsf{Out}(u)$ do not have any incoming edges. So adding incoming edges to $\{u\} \cup \mathsf{Out}(u)$ does not change the reachability tree of any node $w \notin \{u\} \cup \mathsf{Out}(u)$. Hence $\forall w \notin \{u\} \cup \mathsf{Out}(u)$ we have $T_{G'}(w) = T_{G'\cup G'_u}(w)$.

Thus, the only sketches that we need to recompute are the ones for the nodes in $\{u\} \cup \mathsf{Out}(u)$. We focus first on the nodes in $\mathsf{Out}(u)$; let $z \in \mathsf{Out}(u)$. Note that all nodes in $\mathsf{Out}(z)$ are at least at distance two from u in G' and so for them, we already have (a sketch of) the correct reachability tree. Using this fact we can recompute the sketch for $T_{G'\cup G'_u}(z)$ as $S(z) = \mathsf{Bot}_k\left(S(z) \cup \bigcup_{w:(z,w)\in E'_u}(S(w) \cup r(w))\right)$, where $r(w)$ is the random number associated with w. The correctness of this step follows since the reachability tree of z is equal to the union of the reachability tree of its out-neighbors plus its neighbors. So when we add a few outgoing edges to a node we just need to add the reachability tree of the new neighbors and the new neighbors to obtain the new reachability tree.

Using this observation we can compute $S(z)$ for all nodes $z \in \mathsf{Out}(u)$. So now we can use similar arguments to recompute $S(u)$. In particular we have that
$S(u) = \mathsf{Bot}_k\left(S(u) \cup \bigcup_{v:(u,v)\in E'_u}(S(v) \cup r(v))\right)$. But $T_{G'\cup G'_u}(z) = T_{G\cup G_u}(u)$ and hence we are done.

Unfortunately so far we assumed to have access to $T_{G'}(z)$ but this is not true because we only have access to $T_G(z)$. Now we argue that even if we use $T_G(z)$ instead of $T_{G'}(z)$, we will still obtain the correct sketch. Note in particular that we can restrict our attention only to nodes in $\mathsf{Out}_2(u)$. Suppose for $y \in \mathsf{Out}_2(u)$ we have $s \in T_G(y) \setminus T_{G'}(y)$. (Note that $T_{G'}(y) \subseteq T_G(y)$.)

Now if $s \in T_G(y)$, then $s \in T_{G\cup G_u}(u)$ since $y \in \mathsf{Out}_2(u)$. But $T_{G\cup G_u}(u) = T_{G'\cup G'_u}(u)$, so the fact that we consider s when computing a sketch for $T_G(y)$ will not cause a problem when computing a sketch for $T_{G'\cup G'_u}(u')$, since $s \in T_{G'\cup G'_u}(u)$. So the random number $r(s)$ associated with s would have been considered (in computing the sketch) in any case when computing $T_{G'\cup G'_u}(u')$.

Hence, we can use $T_G(z)$ instead of $T_{G'}(z)$ without changing the value of Bot_k. In this way we obtain an algorithm to compute the reachability tree of a node u, by using precomputing time $O(mD\epsilon^{-2}\log n)$, using memory $O(\epsilon^{-2} n \log n)$ and processing time $O(|E_u|\epsilon^{-2}\log n)$. $\square$

3.2 Centrality measures

Based on the results in the previous section, we now show how to compute a few interesting centrality measures, namely, Closeness centrality, Lin's centrality, and Harmonic centrality. The idea is to use the same approach used in the previous section to compute the volume of the ball of different diameters centered at each node. Then we can use those numbers to obtain approximations to the three centrality measures as in [8].

Before describing our algorithm, we recall the definition of these measures. Let $d(v, w)$ be the shortest path distance between v and w in G and let $B_u(d) = \{v \mid d(v, u) \leq d\}$. Let D be the diameter of G.

(i) *Closeness centrality* is defined as

$$C(u) = \frac{1}{\sum_{v\in V} d(v, u)} = \frac{1}{\sum_{d=1}^{D} d \cdot (|B_u(d)| - |B_u(d-1)|)}.$$

(ii) *Lin's centrality* is defined as

$$C(u) = \frac{|\{y \mid d(y, u) < \infty\}|}{\sum_{v\in V} d(v, u)} = \frac{|\{y \mid d(y, u) < \infty\}|}{\sum_{d=1}^{D} d \cdot (|B_u(d)| - |B_u(d-1)|)}.$$

(iii) *Harmonic centrality* is defined as

$$C(u) = \sum_{v\in V} \frac{1}{d(v, u)} = \sum_{d=1}^{D} \frac{1}{d \cdot (|B_u(d)| - |B_u(d-1)|)}.$$

Now note that $|\{y \mid d(y, u) < \infty\}|$ is the size of the reachability tree rooted at u, which we already saw how to estimate. So we need only to estimate $|B_u(d)| - |B_u(d-1)|$ to obtain a formal estimator for the centrality measures. Unfortunately this is hard even for simple graphs. Our approach will be to compute a good approximation of $|B_u(d)|$ for all $d > 0$ using bottom-k sketch as in [11]. We then use it to estimate $|B_u(d)| - |B_u(d-1)|$; by subtracting two estimates we eschew any theoretical guarantees but as in [8], this is an effective method to estimate these centralities in practice.

For the remainder, we focus on getting a good approximation for $|B_u(d)|$ in the public-private graph setting. This can be done using a technique similar to the one used to estimate the reachability tree. We start by showing how to compute this estimate in G, then we explain how to compute it in $G \cup G_u$.

The core idea of the algorithm is once again to use the bottom-k sketch. Initially each node u has a sketch $S(u, 0)$ and after updating the sketch as before, at the end of the dth iteration, $S_G(u, d)$ contains a sketch of $B_u(d)$ in G. Furthermore, we have the following analog of Lemma 3:

LEMMA 4. *Let $G = (V, E)$ be a directed graph and let $B_u(d)$ be the ball of diameter d rooted at u in G. Let $G' = (V, E')$ be the graph on the same set of nodes but where $E' = E \setminus \{(z, v) \mid v = u \text{ or } (v \in \mathsf{Out}(u) \text{ and } z \neq u)\}$; let $B'_u(d)$ be the ball of diameter d rooted at u in G'. Then $B_u(d) = B'_u(d)$*

The proof follows as that of Lemma 3 and by the fact that every time we change a path, we only shorten it.

Now using Lemma 4 and using a similar reasoning as before, we can compute a sketch of $B_u(d)$ in $G \cup G_u$. In fact, for each $v \in \mathsf{Out}(u)$ we can compute an intermediate bottom-k sketch as $\mathsf{Bot}_k\left(S_G(v, d-1) \cup \bigcup_{z:(z,v)\in E'}(S_G(z, d-2) \cup r(z))\right)$,

Finally we can compute
$S_{G\cup G_u}(u,d) = \mathsf{Bot}_k\left(S_G(u)(d) \cup \bigcup_{v:(v,u)\in E'}(S_G(v,d)\cup r(v))\right)$.
The correctness of the estimator can be proved using the same technique showed before; we omit the details in this version. Note that in this case to compute an estimate for the ball of diameter d we need to keep a good estimate for balls of diameter $d-1$ and $d-2$.

THEOREM 5. *For any d, we can estimate $|B_u(d)|$ to within $(1+\epsilon)$-factor in the public-private model using preprocessing time $O(mD\epsilon^{-2}\log n)$, space $O(\epsilon^{-2}Dn\log n)$, and query time $O(|E_u|\epsilon^{-2}\log n)$.*

4. SAMPLING ALGORITHMS

In this section we show how a few well-known sampling algorithms can be efficiently realized in the public-private graph model. The examples that we exhibit will be of increasing interest and hardness.

4.1 All-pair shortest paths

The distance between two nodes in a social network is a useful feature for many applications. For example, it can be a feature to predict which celebrity a particular user will follow. In this section we study how to approximate efficiently the distance between any two nodes in the graph in the public-private model. Note that this problem is particularly interesting in our model because the distances between two nodes can change dramatically even if we add a single edge. In this section we assume that the graph is undirected.

THEOREM 6. *We can approximate the distance between two nodes to within $O(\log n)$ in the public-private model using preprocessing time $O(m\log^2 n)$, space $O(n\log^2 n)$, and query time $O(|E_u|\log^2 n)$.*

PROOF. Consider the all-pair shortest path approximation of Das Sarma et al. [32]. The basic idea behind the algorithm is to estimate the distances between two nodes v and w in a graph $G=(V,E)$ by precomputing the distances to a random subset of nodes $S\subseteq V$ and then to estimate the distance $d(v,w)$ between v and w by looking at a subset of the shortest paths that go through S.

More formally, the algorithm computes an $O(\log^2 n)$-sized sample by computing for $\log n$ times, a sample of size $2\log n + 2$ in the following way. It first generates $\lfloor \log n\rfloor + 1$ random sets of nodes of sizes $1,2,\ldots,2^r$ called $S_0,S_1,\ldots,S_r$, where $r=\lfloor\log n\rfloor$. Then it computes for each node v and for all i, the closest node $v_i\in S_i$ to v and the corresponding distance. Finally for each node v, the algorithm stores as a sample the pairs $\langle v_i, d(v,v_i)\rangle$ for all i. Note that the previous computation can be executed in time $O(m\log^2 n)$ by doing a breadth-first search (BFS) from each set S_i.

Now we can estimate the distance between two nodes v and w simply by looking at their respective samples. Let W_v be the sample for each node v. Then, the approximate distance $\hat{d}(v,w) = \min_{k:(k,*)\in W_v[i],W_w[i]}(d(v,k)+d(k,w))$. Essentially, the above estimator sums the distance from a common element in the sample. Note that if the graph is connected, then this distance is always well defined, otherwise if there is no common node in the samples, the distance is set to ∞. Using such an estimator, it is possible to approximate the distances between any two nodes in undirected graphs[1] by multiplicative factor of $O(\log n)$ w.h.p. Thus, it is possible to compute in time $O(m\log^2 n)$ samples of total size $O(n\log^2 n)$ that allow to approximate the distances between any two pairs of nodes within a multiplicative factor $O(\log n)$ in time $O(\log^2 n)$ in G.

Now we discuss how to use those samples to compute an approximation of the shortest path in $G\cup G_u$. In the following let $d_G(\cdot,\cdot)$ denote the shortest path between two nodes in G and $d_{G\cup G_u}(\cdot,\cdot)$ denote the shortest path between two nodes in $G\cup G_u$. The key observation is to note that the shortest path between two nodes u and v in $G\cup G_u$ is:

$$d_{G\cup G_u}(u,v) = \min\begin{cases} d_G(u,v); \\ 1+d_G(w,v), \\ \quad \forall w\in N(u) \text{ and } (u,w)\in E_u; \\ 2+d_G(z,v), \\ \quad \forall z\in N_2(u)\setminus N(u) \text{ and } (z,*)\in E_u. \end{cases} \tag{1}$$

Using the samples described earlier, we can obtain an $O(\log n)$ multiplicative approximation for all the distances in $d_G(*,v)$. Given that in expression (1) we consider at most $O(|E_u|)$ such distances and that we can estimate each of them in $O(\log^2 n)$ time, the proof is complete. □

4.2 Pairwise node similarities

The shortest path is a classic way to estimate the closeness between nodes in a social network but unfortunately it takes into account only the length of a path between two nodes and not the number of paths between them. For this reason several different distances have been introduced to capture the affinity between two nodes in a social network, including the work of Katz [21] and personalized PageRank [18].

Personalized PageRank. One of the popular algorithms for node similarity is personalized PageRank (PPR). Here we propose an heuristic to efficiently estimate the PPR of a node u in the public-private model. (For more background on PPR, see [18].)

For the public graph G we precompute for each node $v\in G$, the vector $\mathrm{PPR}_v(G)$ in G. In this phase we use the algorithm of Andersen et al. [4] with the given parameter ϵ to approximate the vectors efficiently.

For each private graph G_u we obtain the PPR vector of u based on the decomposition result of Jeh and Widom [20]:

$$\mathrm{PPR}_u(G_u) = (1-\alpha)\deg_{G_u}(u)^{-1}\sum_{(u,v)\in E_u}\mathrm{PPR}_v(G_u) + \alpha\mathbf{1_u}.$$

In our heuristic we substitute the results for the precompute graph G instead of the private graph.

$$\mathrm{PPR}_u(G_u) \approx (1-\alpha)\deg_{G_u}(u)^{-1}\sum_{(u,v)\in E_u}\mathrm{PPR}_v(G) + \alpha\mathbf{1_u}.$$

Note that even if this computation may look rough at first sight, we can show experimentally that this simple heuristic is very effective in practice (Section 5.3). Unfortunately we cannot show any theoretical guarantees on the approximation of PPR and therefore we study another similarity metric for which we can show formal guarantees.

[1]Unfortunately it is not possible to get a similar theoretical guarantees for directed graphs. Nevertheless the authors in [32] show that similar samples give good results in practice also for some structured directed graphs as the Web graph.

Social affinity. We now focus on the similarity measure introduced in Panigrahy et al. [30]; we chose this measure for its elegance and for its conducive properties. This metric captures both the number of paths between the two nodes and the length of the paths between two nodes. Formally, the similarity $A^{\theta}(v, w)$ between two nodes v and w is defined as the maximum fraction of edges that can be deleted randomly from the graph without disconnecting v and w with probability at least θ. The authors also show how to compute this sampling-based similarity measure efficiently and show empirically that this measure nicely captures the semantic similarity between users in Twitter. For the remainder of this section we assume that the graph is undirected.

THEOREM 7. *The social affinity between two nodes can be estimated in the public-private model using preprocessing time $O(m \log^2 m)$, space $O(n \log^2 n)$, and query time $O(|E_u| + \log^2 m)$.*

PROOF. Let $\epsilon > 0$ be a constant. The main idea in [30] is to construct $\log m$ distinct telescoping sequences of $\log m$ subgraphs[2] $G_{1,1}, \ldots, G_{\log m, (\log m)/\epsilon}$ of G where for all i the probability that an edge is not deleted in $G_{i,j}$ is $(1-\epsilon)^j$, for $1 \le j \le (\log m)/\epsilon$. We then compute the connected components in all the $(\log^2 m)/\epsilon$ subgraphs. Using these, we estimate $A^{\theta}(v, w)$ as:
$\frac{\epsilon}{\log^2 m} \sum_{i=1}^{(\log^2 m)/\epsilon} \left[\theta < \frac{1}{\log m} \sum_{j=1}^{\log m} [v, w \text{ connected in } G_{i,j}]\right]$,
where we used the Iverson bracket notation: $[\xi]$ equals 1 if the predicate ξ is true and is 0 otherwise. They show that this estimate is within an additive $\pm\epsilon$ factor from the correct $A^{\theta}(v, w)$, with high probability. Note that this estimation can be computed in time $O(m \log^2 m)$ for constant ϵ, and all the samples can be stored in space $O(n \log^2 m)$. In fact we need to store only the component ids for each node.

We now show how to efficiently compute this measure on the $G \cup G_u$ graph. The basic intuition is to use an efficient algorithm to compute the new connected components. Unfortunately using a union-find algorithm on the entire graph would be impractical because it would take too long to update the connected components ids of all the nodes in the graphs. For this reason we need to keep the component ids of all nodes in the graphs in a slightly different way to allow for quick updates.

More precisely, each node in the public graph does the following: instead of storing $O(\log^2 m)$ components ids for the graphs $G_{1,1}, \ldots, G_{\log m, (\log m)/\epsilon}$, it keeps $O(\log^2 m)$ pointers to its components ids. When the edges in G_u are added to the public graph, we first establish via sampling to which graph $G_{1,1}, \ldots, G_{\log m, (\log m)/\epsilon}$ they are added. Then instead of considering their effect on the entire graph, we analyze their effect on the compressed graph where nodes with the same component ids are collapsed together.

Note that for a single subgraph this can be done in time linear in $|E_u|$. For example, consider the subgraph $G_{i,j}$. Let $E_u(i, j)$ be the edges in E_u that exist in the graph $G_{i,j}$ after the random deletions. To compute the new connected components in $G_{i,j}$, using the pointers to the connected components id for all the nodes incident in $E_u(i, j)$, it is possible to see which components are connected by edges in $E_u(i, j)$. Then by starting a few BFS visits in the collapsed version

[2]It is easy to construct such a sequence by constructing graph G_i from G_{i-1} by keeping the edges in G_{i-1} with probability $(1-\epsilon)$.

(nodes in the same connected components are contracted together) of $G_{i,j}$ from connected components incident to $E_u(i, j)$, it is possible to update the component ids (we keep an arbitrary id for all the component in the same BFS tree) in time linear in $|E_u|$. Using this idea, it is possible to update the component ids for all the nodes in the graph in time $O(|E_u|)$ and hence the proof is complete. □

4.3 Correlation clustering

In this section we consider the problem of clustering in the public-private graph model. Correlation clustering is a fundamental problem in social network analysis that has received a lot of attention over the past decade.

We start with some definitions. A *partition* of V is a collection $\mathcal{C}$ of subsets of V such that $\forall \{C_i, C_j\} \in \binom{\mathcal{C}}{2}$ it holds $C_i \cap C_j = \varnothing$ and $\bigcup_{C_i \in \mathcal{C}} = V$. We will use the term *clusters* to denote the sets $C_i \in \mathcal{C}$. For this section we assume that there are two types of edges in the public graph: positive edges, denoted E^+ and negative edges denoted E^-. Recall the correlation clustering problem [5].

DEFINITION 8. *The triple (V, E^+, E^-) is a* correlation clustering instance *if let $E^+ \cap E^- = \varnothing$ and $E^+ \cup E^- = \binom{V}{2}$. A solution of the correlation clustering problem is a partition $\mathcal{C}$ of V. The cost of $\mathcal{C}$ is equal to:*

$$\left|\left\{e \in E^- \;\middle|\; e \text{ is contained in some } C_i \in \mathcal{C}\right\}\right| +$$
$$\left|\left\{e \in E^+ \;\middle|\; e \text{ is not contained in any } C_i \in \mathcal{C}\right\}\right|.$$

The goal of the correlation clustering problem is to find a partition $\mathcal{C}$ of V that minimizes the cost of the clustering. A partition is an α-approximation if its cost is no larger than α times the cost of the optimal partition.

Similar to the edges in the public graph, the edges in G_u are also of two types: E_u^+ and E_u^-. In the following we will show that given an (approximate) correlation clustering solution to the public graph G, we can quickly find an approximate correlation clustering to $G \cup G_u$. The solution to G would be through a sampling algorithm. For the rest of the subsection we assume that G_u is a star; extending our results to more general G_u as in the other sections is an open problem.

Given a correlation clustering instance $\mathcal{I} = (V, E^+, E^-)$, and some $u \notin V$, let the instance $\mathcal{I}_u = (V \cup \{u\}, E^+ \cup E_u^+, E^- \cup E_u^-)$ be obtained from $\mathcal{I}$ by adding u and the edges incident on u to $\mathcal{I}$ (so that $E_u^+ \cap E_u^- = \varnothing$ and $E_u^+ \cup E_u^- = \{\{u, v\} \mid v \in V\}$). Note that we assumed $u \notin V$; this is for simplicity of exposition. After the proof we will show how to handle this.

Given a partition $\mathcal{C}$ of V and a $v \in V$, we consider partitions of $V \cup \{u\}$ that can be obtained by adding u to one of the clusters of $\mathcal{C}$, or by placing u in a new singleton cluster. These $|\mathcal{C}| + 1$ partitions are called the *u-neighbors* of $\mathcal{C}$. We now bound the quality of a solution to $G \cup G_u$.

LEMMA 9. *Let $\mathcal{C}$ be an α-approximation of $\mathcal{I}$. Consider an instance $\mathcal{I}_u$. Then, at least one of the u-neighbors of $\mathcal{C}$ is an $O(\alpha)$-approximation of the instance $\mathcal{I}_u$.*

PROOF. Let $\mathcal{C} = \{C_1, \ldots, C_t\}$ for some integer t. Let k_i^+ (resp., k_i^-) be the number of positive (resp., negative) neighbors of u in the cluster C_i. Then $\sum_{i=1}^{t}(k_i^+ + k_i^-) = |V|$.

Let c be the cost of $\mathcal{C}$ on the instance $\mathcal{I}$, let $c^\star$ be the minimum cost on the instance $\mathcal{I}$. Also, let $c_u^\star$ be the minimum cost on the instance $\mathcal{I}_u$. Observe that $c_u^\star \ge c^\star$. Let us also use $c_u(C_i)$ to denote the cost of the edges incident on u, if we assign u to the cluster C_i, and $c_u(\varnothing)$ be the cost of those edges if we let u be in a singleton cluster.

Let us define the cost of a cluster C to be equal to: $c(C) = |E^- \cap C| + \frac{1}{2} \cdot |E^+ \cap \{\{v, v'\} \mid v \in C \text{ and } v' \notin C\}|$. Observe that the cost of a clustering $\mathcal{C}$ can be decomposed in the sum of the cost of its clusters: $c(\mathcal{C}) = \sum_{C \in \mathcal{C}} c(C)$.

Suppose that $\mathcal{C}$ is an α-approximate solution of the instance $\mathcal{I}$. Suppose we add u to $\mathcal{I}$ to obtain $\mathcal{I}_u$. Then, let us produce a solution $\mathcal{C}_u$ by adding u to one of the clusters of $\mathcal{C}$, or to a singleton cluster, to minimize the total cost of $\mathcal{C}_u$. If u is placed in a new singleton cluster, then $c(\mathcal{C}_u) - c(\mathcal{C}) = \sum_{i=1}^t k_i^+$. If, instead u is added to the cluster C_i, then we have $c(\mathcal{C}_u) - c(\mathcal{C}) = k_i^- + \sum_{i=1, i \neq j}^t k_i^+$.

Now, suppose that $\mathcal{C}_u$ places u into a new cluster by itself. By the minimality of the cost of $\mathcal{C}_u$, we have that, for each i, $k_i^+ \leq k_i^-$.

Let us split the total cost $c_u(\mathcal{C}_u)$ of the clustering $\mathcal{C}_u$ as follows. For each cluster $C_i \in \mathcal{C}$, let us define $c_u(C_i)$ as the total number of negative edges inside C_i, plus half the number of the positive edges split between C_i and another cluster in $\mathcal{C}$, plus k_i^-, i.e., $c_u(C_i) = c(C_i) + k_i^-$. Observe that $c_u(\mathcal{C}_u) = c(\mathcal{C}) + \sum_{i=1}^t k_i^-$.

Before continuing our proof, we now recall that a *bad* triangle is a set of three elements of an instance that are connected by exactly one negative edge (and two positive edges). A fractional packing of triangles is an assignment of positive weights to the triangles of the instance such that, for each edge $\{v', v''\}$, the sum of the weights of the triangles that contain that edge does not exceed 1 ($\forall \{v', v''\} : \sum_{v''' \notin \{v', v''\}} w_{v', v'', v'''} \leq 1$). The total weight of a fractional packing is equal to the sum of the weights of its *bad* triangles. It is known [3] that the total weight of a fractional packing is a lower bound on the cost of the optimal solution of a correlation clustering instance.

Let us consider the generic cluster $C_i \in \mathcal{C}$. If it induces at least γk_i^+ negative edges, then $\gamma k_i^+ \leq c(C_i) \leq c_u(C_i)$. Otherwise, we have that C_i induces fewer than γk_i^+ negative edges. Now, since C_i induces fewer than γk_i^+ negative edges, and since $k_i^+ \leq k_i^-$, we have that there must exist a positive matching, of cardinality $(1-\gamma)k_i^+$, composed of edges $\{v, v'\} \in \binom{C_i}{2}$ such that $\{v, u\}$ is positive and $\{v', u\}$ is negative. For each $\{v, v'\}$ in the matching, we give weight 1 to the triangle $\{v, v', u\}$. This (integral) packing then proves that the total cost of the edges between the nodes in $C_i \cup \{u\}$, regardless of how the nodes are partitioned, has to be at least $(1-\gamma)k_i^+$. If we choose $\gamma = \frac{1}{2}$, we have that — regardless of whether the number of negative edges in C_i is lower or upper bounded by γk_i^+ — the total cost of $C_i \cup \{u\}$ has to be at least $\frac{1}{2} \sum_{i=1}^t k_i^+$. Since the cost is at most $c_u(\mathcal{C}_u) \leq c(\mathcal{C}) + \sum_{i=1}^t k_i^+$ we get that if u ends up in its own cluster we obtain an $O(\alpha)$ approximation.

Suppose instead that the optimal $\mathcal{C}_u$ positions u in some cluster $C_i \in \mathcal{C}$. Then $k_i^- \leq k_i^+$ and, moreover, $k_i^+ - k_i^- \geq k_j^+ - k_j^-$, for each $j = 1, \ldots, t$. Then, again, either $c(C_i)$ was larger than γk_j^-, or we can create a matching between a fraction of $(1-\gamma)$ of the k_i^- nodes of C_i that are connected to u through negative edges and the k_i^- nodes of C_i that are connected to u through positive edges. This gives a packing of total weight $\geq (1-\gamma) \cdot k_i^-$, so (by choosing $\gamma = \frac{1}{2}$) if $k_i^- \geq \sum_{j=1, j \neq i}^t k_i^+$, we have proved an $O(\alpha)$-approximation.

Finally, we consider the case $k_i^- \leq \sum_{j=1, j \neq i}^t k_i^+$ (and u is placed in C_i by the solution). Now, either the cost of $c(\mathcal{C})$ was larger than $\frac{1}{2} \sum_{j=1, j \neq i}^t k_i^+$, or there are at most $\frac{1}{2} \sum_{j=1, j \neq i}^t k_i^+$ positive edges in the cuts induces by $\mathcal{C}$.

We now define a new fractional packing of bad triangles. We will give positive weight to a triangle iff it contains u, intersects two different $C_j, C_{j'} \in \mathcal{C}$, and the two triangle's nodes in C_j and $C_{j'}$ are connected by a negative edge. Each triangle with positive weight will have the same weight $\frac{1}{\sum_{j=1}^t k_j^+}$. By the upper bound on the number of positive edges in the cut, the number of such triangles is at least $\sum_{\{j,j'\} \in \binom{[t]}{2}} \left(k_j^+ \cdot k_{j'}^+\right) - \frac{1}{2} \sum_{j=1, j \neq i}^t k_i^+ \geq \Omega\left(\sum_{\{j,j'\} \in \binom{[t]}{2}} \left(k_j^+ \cdot k_{j'}^+\right)\right)$.

No edge will have weight more than 1, and the total weight of the triangles will equal:

$$\Omega\left(\sum_{\{j,j'\} \in \binom{[t]}{2}} \frac{k_j^+ \cdot k_{j'}^+}{\sum_{\ell=1}^t k_\ell^+}\right) = \Omega\left(\sum_{j=1}^t k_j^+\right).$$

Since the cost of the solution is at most $c(\mathcal{C}) + O\left(\sum_{j=1, j \neq i}^t k_i^+\right)$, the approximation ratio is at most $O(\alpha)$. $\square$

The above result tell us that a simple algorithm can start from a clustering and place a (new) node greedily without losing much in the approximation ratio. In the public-private graph model, note that the node u is not new. To handle this, given an instance $\mathcal{I} = (V, E^+, E^-)$, and some $u \in V$, let the instance $\mathcal{I}_{-u} = (V \setminus \{u\}, E^+_{-u}, E^-_{-u})$ be obtained from $\mathcal{I}$ by removing u and the edges incident on u from $\mathcal{I}$.

Suppose that we precompute an approximate solution to the public graph G. Then, for an arbitrary node u we can apply Lemma 9 to G_u, with the new edges of u, to obtain an $O(\alpha)$-approximation algorithm. If we use the sampling-based algorithm of Ailon et al. [3], we get $\alpha = 3$.

THEOREM 10. *We can approximate the correlation clustering within a factor $O(1)$ in the public-private model (when G_u is a star) using preprocessing time $O(m)$, space $O(m)$, and query time $O(|E_u| \log n)$.*

5. EXPERIMENTS

In this section we demonstrate the efficiency of our algorithms in the public-private graph model. For purposes of showcasing the public-private graph model, we will choose the following algorithms: reachability tree size estimation by sketching, shortest path by sampling, and correlation clustering by sampling.

For experimentation purposes, we use some of the social networks publicly available as part of the Stanford's SNAP dataset (`http://snap.stanford.edu/data`). In Table 1 we give some basic statistics of the graphs that we used in the experiments.

To obtain examples in the public-private model, we do the following: given a graph (V, E), we pick a node $u \in V$ uniformly at random and designate the graph induced on the $V \setminus \{u\}$ to be the public graph G. To define the private graph G_u at u, we do one of the two modifications: in the star-case, we let the private edges E_u be the subset $\{(v, w) \in E\}$ where either $v = u$ or $w = u$, i.e., the nodes connected to or

Graph	# nodes	# edges
SLASHDOT [26]	58,228	428,156
EPINIONS [26]	77,357	516,575
WIKI-VOTE [25,26]	7,115	103,689
YOUTUBE [34]	1,134,890	2,987,624
EMAIL-EUALL [27]	265,214	420,045
GNUTELLA [31]	62,586	147, 892
DBLP [34]	317,080	1,049,866
LIVEJOURNAL [34]	3,997,962	34,681,189
ORKUT [34]	3,072,441	117,185,083

Table 1: Social networks used in the experiments.

from u. In the clique-case, we let the private edges of E_u to be all the edges in $\{u\} \cup \mathsf{Out}(u)$. As we mentioned earlier, these two cases represent two different types of privacy.

Since our main emphasis is to show the gains in running time, we will present the ratio A/B of the following two running times: A is the running time of our update algorithm that works on the private graph G_u and a sketch or sample of the public graph G and B is the running time of the usual algorithm on $G \cup G_u$, i.e., a naive implementation. This ratio will reflect the average running time savings factor we can obtain by using our algorithm instead of running the usual algorithm on $G \cup G_u$. Since our algorithms are randomized, each running time is averaged over 10 independent trials and we report the ratio A/B averaged over 100 independent private graphs. Unless otherwise specified, the quality of the solutions produced by these two separate methods, namely, the update-based algorithm and the usual algorithm, are near-identical (modulo randomness).

5.1 Size of reachability tree

In this section we demonstrate our algorithm for computing the size of the reachability tree. We construct the public and private graphs as described before, for both star-case and the clique-case. We implement the naive sketching algorithm on $G \cup G_u$. We then compare its running time against our update algorithm, which uses the sketch for G, generates E'_u from E_u using the definition in Lemma 3, and updates the sketch of u and nodes in $\mathsf{Out}(u)$ according to the algorithm in Section 3.1.

Table 2 shows the average running time gains over the naive algorithm for both the star-case and clique-case of private graphs. Unsurprisingly, the efficiency gains made by our algorithm are several orders of magnitude, uniformly across different datasets. Figure 2 shows the effect of ϵ

Graph	A/B (star-case)	A/B (clique-case)
WIKI-VOTE	2.02e-05	1.82e-05
GNUTELLA	2.56e-06	2.57e-06
EMAIL-EUALL	1.89e-06	3.96e-06
SLASHDOT	1.63e-06	1.12e-06
EPINIONS	1.79e-06	1.48e-06

Table 2: Efficiency gains for reachability in the star-case and the clique-case; $\epsilon = 0.3$.

(which controls the accuracy of the estimate) on the efficiency gain. Recall that a smaller value of ϵ means a higher accuracy. Interestingly, the gains increase as ϵ decreases: note that a smaller ϵ leads to a larger k (see the algorithm in Section 3.1) and hence might result in more book-keeping for maintaining the bottom-k data structure $\mathsf{Bot}_k(\cdot)$.

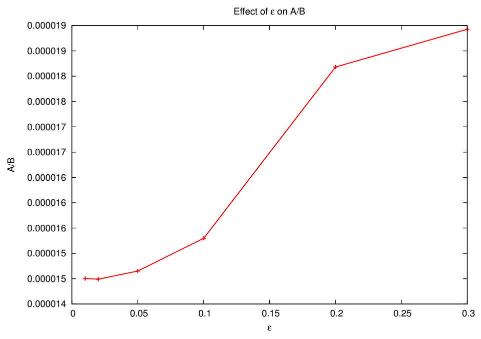

Figure 2: Role of ϵ in efficiency gains for WIKI-VOTE.

5.2 Shortest path approximations

We next focus on analyzing the performance of our algorithm to compute the shortest path from a node u to another arbitrary node in the private graph $G \cup G_u$. We run our experiments on YOUTUBE data [34] and we only analyze the star-case since the clique-case affects the shortest path in the same way as in the star-case.

More specifically, we compare the running time and the accuracy of our algorithm with the naive algorithm that executes a single-source shortest path algorithm (i.e., Dijkstra's algorithm) to compute the distance between two nodes. After showing that our algorithm vastly outperforms the naive solution, we turn our attention on the trade-off between the size of the sketches that we use and the accuracy of the solution. In our experiments, we observe $A/B \approx 1/827$, where the comparison is with the classic Dijkstra algorithm. We also observe that our algorithm obtain a good approximation of the shortest path (in fact our estimation is almost never more than a factor 1.5 away from the real value.)

5.3 Personalized PageRank

In this section we evaluate our heuristic for PPR. For each graph in our dataset with ground truth communities (Orkut, YouTube, DBLP, LiveJournal) we sampled a set of 20 nodes whose neighbors belong to at least two different communities. Let u be such a node and let C_u be the set of the communities to which the neighbors of u belong. For each node u, we declare a random half of the communities in C_u as private and mark as private all the edges to and within these communities; the remaining edges are marked public.

For each private subgraph G_u we run the following experiment. We determine the ground truth PPR ranking of u using the algorithm of Andersen et al. [4], with a given ϵ. We also compute the ranking obtained by our heuristic using the result of preprocessing the public graph, which is again obtained using the same algorithm. We evaluate the accuracy of the rankings and the performance of our heuristic. In all the experiments, we set $\alpha = 0.15$ and vary ϵ.

Table 3 (Column A/B) and Figure 3 shows the ratio of the running time of our heuristic and the algorithm of Andersen et al. on $G \cup G_u$. In all datasets, it is clear that the heuristic is faster by several orders of magnitude. It is interesting to notice that as ϵ gets smaller, the performance of the heuristic degrades. This is because if ϵ is very small, then the rankings become very long (n in the limit) and hence the heuristic takes significantly more time to evaluate them.

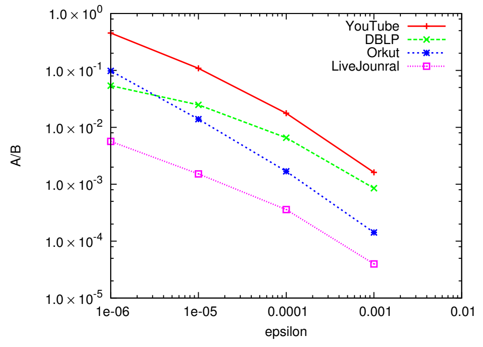

Figure 3: Role of ϵ in efficiency gains for our social networks.

Graph	A/B	RMSE	Cosine	τ@50
DBLP	6.5e-3	7.8e-4	99.8%	88.5%
LiveJournal	3.5e-4	5.1e-3	99.1%	69.3%
Orkut	1.6e-3	1.0e-3	99.9%	54.6%
YouTube	1.7e-2	6.8e-3	99.8%	80.9%

Table 3: Efficiency and accuracy of our heuristic for the PPR, $\epsilon = 0.001$. Column A/B, RMSE, Cosine, and τ@50 represent the ratio between ground truth time and our heuristics, the Root Mean Square Error, the Cosine Similarity and the Kendall-τ index for the first 50 positions of the ranking.

Table 3 shows the accuracy measure for the rankings computed. It is clear that the heuristic produces rankings whose RMSE with the ground truth is remarkably close to 0 and the cosine similarity is close to 1; this suggests that the distributions are approximated well. Also the Kendall-τ correlation of the first 50 positions of the rankings (which is more useful from a practical viewpoint) is quite high.

5.4 Correlation clustering

In this section we demonstrate our algorithm for correlation clustering in the public-private graph model. As discussed before, we construct the public graph G and the private graph G_u for many u's. Since our algorithm for correlation clustering works only for the star-case, we do not consider the clique-case here. We compute a correlation clustering of the public graph G using the sampling Pivot algorithm of Ailon et al. [3]. We use the output of this algorithm, the edges in G_u, and guarantee of Lemma 9 in order to compute the correlation clustering on $G \cup G_u$.

Since correlation clustering is on signed networks, we use two signed networks from SNAP. The network Epinions consists of who-trusts-whom network of `epinions.com`. The network Slashdot consists of the social network in `slashdot.com` from February 2009. We treat every non-edge of the two networks as a negative edge. Table 4 shows the efficiency gains for two sample graphs. We see that our update algorithm obtains highly significant savings over the naive implementation.

Graph	A/B
Slashdot	1.6e-04
Epinions	8.9e-05

Table 4: Efficiency gains for correlation clustering (in the star-case).

6. RELATED WORK

The related work falls into several categories including dynamic graph algorithms, graph sketching, and graph sampling algorithms.

On the face of it, the public-private model seems related to the dynamic (aka incremental) graph algorithms. In this model, at each time step, a new edge or a node is inserted or deleted, and the goal is to maintain an appropriate data structure of the changing graph so that one can efficiently compute certain functions of the graph at any point in time. There are several natural functions for which efficient (and in many cases, optimal) dynamic algorithms are known, e.g., connectivity [23], minimum spanning trees [19], transitive closure [24], all-pair shortest paths [15], etc. See the survey by Demetrescu et al. [14] for an overview of the area. There are basic differences between dynamic graphs and public-private graphs. First of all, while many dynamic graph algorithms deal with both insertions and deletions and they are optimized for single insert/delete operations, public-private graph computation mainly deals with batch additions. Also the central parameters of interest for dynamic algorithms are the data structure update time and the (incremental) function computation time; the space of the data structure plays a lesser role. In addition, existing algorithms for dynamic graphs are tailored for computing the function exactly, for obtaining asymptotic bounds, and for classical graph algorithms; they have not been applied broadly to the social network settings. Finally, recent attempts [6, 33] to present more general results for social networks are mainly tailored toward community detection, and their techniques do not apply to our problem setting.

Our model is also related to the problem of sketching graphs to answer queries about them. In the graph sketching model, a succinct representation of a graph, called a sketch, is constructed from the graph. This sketch can be used to compute some functions on the whole graph. The sketch depends on the actual function to be computed and efficient graph sketches are known for a variety of problems including connectivity [1], cut size and distance between nodes [2]. However, graph sketching is still in its infancy and sketches have been developed only for some very basic graph problems; discouragingly, some of the more interesting problems are faced with strong lower bounds. On a topic more relevant to social networks, the so-called All Distance Sketches were developed to approximate the neighborhood function of graphs; see the original work of Cohen [10, 11], ANF [29], and hyperANF [7]. Some of the graph sketches have the property that they are composable: the sketch of the union of two graphs can be computed from their respective individual sketches. This is ideal for our setting, where we can compute a sketch of the public graph and sketches for private graphs, and consequently, can compute the sketch of the union of the public graph with any private graph. Even though this looks directly applicable, in some cases it is not clear if such a composition can be efficiently implemented in the public-private graph model. In fact, one of our contributions is to show that the neighborhood approximation sketch can be implemented efficiently in our model.

Sampling algorithms have been used for several graph and social network problems. Typically, sampling algorithms pick nodes (or edges) in the graph according to some distribution, e.g., uniform on the nodes, or proportional to the degrees (e.g., by a uniform random walk), etc. Sampling algorithms have been used to estimate basic graph properties including the size of the network [22], average degree [13], clustering coefficient [17]. In the public-private model, these problems are less interesting since they can be solved trivially and with ease. More sophisticated sampling algorithms exist for other social network problems including all-pair distance estimation [32], similarity estimation [30], correlation clustering [3], densest subgraphs [9], etc. It is not a priori clear if it is possible at all to adapt these algorithms to the public-private model. We show how to extend the all-pair distance estimation, similarity estimation, and correlation clustering to public-private graphs; showing a similar result for the densest subgraph problem is an open problem.

7. CONCLUSIONS

In this paper we introduced the public-private model of computation for online social networks. This model is motivated by a new classes of features introduced by online social networks, where users can define public and private friends, or public and private lists or circles. Our model abstracts the privacy aspects in the link structure of the social network and establishes a formal framework to study efficient social network algorithms that respect the privacy of the links; such algorithms operate, for a given user, on this user's private graph and on the common public graph. As a demonstration of the simplicity and the potential of our model, we study two classes of popular social network algorithms using our framework: sketching and sampling. We show efficient algorithms for many important social network problems and illustrate the computational benefits of the framework by experimental analysis.

We believe that the public-private model is an abstraction that can be used to develop efficient social network algorithms. Our work leaves a number of open interesting research directions such as: obtaining efficient algorithms for the densest subgraph/community detection problems, influence maximization, computing other pairwise similarity scores, and most importantly, recommendation systems.

8. REFERENCES

[1] K. J. Ahn, S. Guha, and A. McGregor. Analyzing graph structure via linear measurements. In *SODA*, pages 459–467, 2012.

[2] K. J. Ahn, S. Guha, and A. McGregor. Graph sketches: sparsification, spanners, and subgraphs. In *PODS*, pages 5–14, 2012.

[3] N. Ailon, M. Charikar, and A. Newman. Aggregating inconsistent information: Ranking and clustering. *JACM*, 55(5), 2008.

[4] R. Andersen, F. Chung, and K. Lang. Local partitioning for directed graphs using pagerank. *Internet Mathematics*, 5(1-2):3–22, 2008.

[5] N. Bansal, A. Blum, and S. Chawla. Correlation clustering. *Machine Learning*, 56(1-3):89–113, 2004.

[6] T. Y. Berger-Wolf and J. Saia. A framework for analysis of dynamic social networks. In *KDD*, pages 523–528, 2006.

[7] P. Boldi, M. Rosa, and S. Vigna. HyperANF: approximating the neighbourhood function of very large graphs on a budget. In *WWW*, pages 625–634, 2011.

[8] P. Boldi and S. Vigna. In-core computation of geometric centralities with hyperball: A hundred billion nodes and beyond. In *ICDM Workshops*, pages 621–628, 2013.

[9] M. Charikar. Greedy approximation algorithms for finding dense components in a graph. In *APPROX*, pages 84–95, 2000.

[10] E. Cohen. Size-estimation framework with applications to transitive closure and reachability. *JCSS*, 55:441–453, 1997.

[11] E. Cohen. All-distances sketches, revisited: HIP estimators for massive graphs analysis. In *PODS*, pages 88–99, 2014.

[12] E. Cohen and H. Kaplan. Summarizing data using bottom-k sketches. In *PODC*, pages 225–234, 2007.

[13] A. Dasgupta, R. Kumar, and T. Sarlós. On estimating the average degree. In *WWW*, pages 795–806, 2014.

[14] C. Demetrescu, D. Eppstein, Z. Galil, and G. F. Italiano. Dynamic graph algorithms. In M. J. Atallah and M. Blanton, editors, *Algorithms and Theory of Computation Handbook*, pages 9–9. Chapman & Hall/CRC, 2010.

[15] C. Demetrescu and G. F. Italiano. A new approach to dynamic all pairs shortest paths. *JACM*, 51(6):968–992, 2004.

[16] R. Dey, Z. Jelveh, and K. W. Ross. Facebook users have become much more private: A large-scale study. In *PerCom Workshops*, 2012.

[17] S. J. Hardiman and L. Katzir. Estimating clustering coefficients and size of social networks via random walk. In *WWW*, pages 539–550, 2013.

[18] T. H. Haveliwala. Topic-sensitive pagerank. In *WWW*, pages 517–526, 2002.

[19] J. Holm, K. de Lichtenberg, and M. Thorup. Poly-logarithmic deterministic fully-dynamic algorithms for connectivity, minimum spanning tree, 2-edge, and biconnectivity. *JACM*, 48(4):723–760, 2001.

[20] G. Jeh and J. Widom. Scaling personalized web search. In *WWW*, pages 271–279. ACM, 2003.

[21] L. Katz. A new status index derived from sociometric analysis. *Psychometrika*, 18(1):39–43, 1953.

[22] L. Katzir, E. Liberty, and O. Somekh. Estimating sizes of social networks via biased sampling. In *WWW*, pages 597–606, 2011.

[23] V. King. Fully dynamic connectivity. In *Encyclopedia of Algorithms*. Springer, 2008.

[24] V. King. Fully dynamic transitive closure. In *Encyclopedia of Algorithms*. Springer, 2008.

[25] J. Leskovec, D. P. Huttenlocher, and J. M. Kleinberg. Predicting positive and negative links in online social networks. In *WWW*, pages 641–650, 2010.

[26] J. Leskovec, D. P. Huttenlocher, and J. M. Kleinberg. Signed networks in social media. In *CHI*, pages 1361–1370, 2010.

[27] J. Leskovec, J. M. Kleinberg, and C. Faloutsos. Graph evolution: Densification and shrinking diameters. *TKDD*, 1(1), 2007.

[28] A. McGregor. Graph stream algorithms: a survey. *SIGMOD Record*, 43(1):9–20, 2014.

[29] C. R. Palmer, P. B. Gibbons, and C. Faloutsos. ANF: a fast and scalable tool for data mining in massive graphs. In *KDD*, pages 81–90, 2002.

[30] R. Panigrahy, M. Najork, and Y. Xie. How user behavior is related to social affinity. In *WSDM*, pages 713–722, 2012.

[31] M. Ripeanu, A. Iamnitchi, and I. T. Foster. Mapping the Gnutella network. *IEEE Internet Computing*, 6(1):50–57, 2002.

[32] A. D. Sarma, S. Gollapudi, M. Najork, and R. Panigrahy. A sketch-based distance oracle for web-scale graphs. In *WSDM*, pages 401–410, 2010.

[33] C. Tantipathananandh, T. Y. Berger-Wolf, and D. Kempe. A framework for community identification in dynamic social networks. In *KDD*, pages 717–726, 2007.

[34] J. Yang and J. Leskovec. Defining and evaluating network communities based on ground-truth. In *ICDM*, pages 745–754, 2012.