On-Device Algorithms for Public-Private Data with Absolute Privacy

Alessandro Epasto
Google
New York
aepasto@google.com

Hossein Esfandiari
Google
New York
esfandiari@google.com

Vahab Mirrokni
Google
New York
mirrokni@google.com

ABSTRACT

Motivated by the increasing need to preserve privacy in digital devices, we introduce the *on-device public-private* model of computation. Our motivation comes from social-network based recommender systems in which the users want to receive recommendations based on the information available on their devices, as well as the suggestions of their social contacts, without sharing such information or contacts with the central recommendation system. Our model allows us to solve many algorithmic problems while providing absolute (deterministic) guarantees of the privacy of on-device data and the user's contacts. In fact, we ensure that the private data and private contacts are never revealed to the central system. Our restrictive model of computation presents several interesting algorithmic challenges because any computation based on private information and contacts must be performed on local devices of limited capabilities. Despite these challenges, under realistic assumptions of inter-device communication, we show several efficient algorithms for fundamental data mining and machine learning problems, ranging from k-means clustering to heavy hitters. We complement this analysis with strong impossibility results for efficient private algorithms without allowing inter-device communication. In our experimental evaluation, we show that our private algorithms provide results almost as accurate as those of the non-private ones while speeding up the on-device computations by orders of magnitude.

CCS CONCEPTS

• **Security and privacy → Social network security and privacy**; • **Information systems → Data mining**; **Social networks**; • **Theory of computation →** *Sketching and sampling*.

ACM Reference Format:
Alessandro Epasto, Hossein Esfandiari, and Vahab Mirrokni. 2019. On-Device Algorithms for Public-Private Data with Absolute Privacy. In *Proceedings of the 2019 World Wide Web Conference (WWW '19), May 13–17, 2019, San Francisco, CA, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3308558.3313677

1 INTRODUCTION

Digital devices play a significant role in the lives of their users through the connections they provide, the information they assist in consuming, and the online purchases they enable. The wealth of information on such devices (contacts, purchases, etc.) can be potentially used to provide their owner with ever-improving services and a better user experience, but *only* if the user's privacy is respected. Providing the users with useful services while also respecting their privacy is the holy grail of data mining and machine learning and a fundamental endeavor in the research community and in the industry [1, 15, 21–23, 27, 48].

Consider the case of a standard recommender system, such as a service that suggests books to read. The system has a central recommendation provider. The user logs into the system and volunteers her ratings for the books already read and, in exchange, receives suggestions for new books to read. Now, assume that the user would like to receive suggestions based on the books read by her close friends (perhaps the contacts in her mobile phone). In a standard centralized system [12], to obtain such social recommendations, the user needs to authorize access to her list of contacts on the phone.

This has obvious limitations. The user might want to share the book reviews online and with her friends, but at the same time she might not want to share the contact list to the system. The standard centralized system architecture does not allow such a use case, because all the data necessary to solve the task of obtaining the reviews (i.e., the book ratings and the social graph) must be stored and processed in a centralized system.

In this paper, we begin the study of a new model of computation that we call the *On-Device Public-Private* model. This model allows the user to employ all the information available on her device, as well as the user's social contacts, without sharing them with the central system. In particular, we show how to solve several machine learning and data mining problems, in this model, without having to move any private data to the central system and without having to reveal the private connections in the social graph. Our method provides absolute deterministic guarantees of privacy, as no information based on the private data or the private contacts of the user is shared with the central authority.

This model is along the lines of other privacy-preserving schemes that are actively studied and that are reviewed in related workâĂŤ for instance, differential privacy [1, 22–24], federated learning [38], and the recent public-private data models [7, 15]. All these methods allow strong privacy guarantees, and may complement ours, but we observe that they do not directly model the social-graph-based information or they involve a setting where private data or some information based on it may still need to be collected by the central authority (although in an anonymized or aggregated form). We seek to study a model that allows general algorithmic problems to be solved while harnessing the social network of the users. Our

WWW '19, May 13–17, 2019, San Francisco, CA, USA

ACM ISBN 978-1-4503-6674-8/19/05.
https://doi.org/10.1145/3308558.3313677

model can be seen as an extension to the recent public-private graph model [15], along the lines of on-device learning settings in which our model, contrary to the public-private graph model, does not require any private information to be shared with the central system, and it also allows metadata on nodes of the network to address applications beyond graph analysis. Finally, we show how to achieve this without relying on insertion of noise to obfuscate the output, which might partially degrade it, while achieving, by design, an even stronger notion of privacy: private data is only used to provide outputs to the data owner.

Our restrictive model of computation presents several interesting algorithmic challenges: since no private information is shared, all computations based on private data must be performed on local devices of limited capabilities. Despite these challenges, we show efficient algorithms in our model for fundamental data mining and machine learning problems, ranging from clustering to recommendation system applications.

In a nutshell, our model works as follows: we allow the user to specify an arbitrary partitioning of the local device data as public or private. Moreover, the user is allowed to specify some contacts as private or public. Only the public data and the public contacts are shared to the central system in accordance with the choice of the user. Then, the user wants to obtain some information (e.g., the results of a recommendation system algorithm) on a dataset consisting of all the data to which the user has access (i.e., her private data, all her contacts whether private or public, as well as all the public information available in the system). All of this must be achieved without revealing the private data and the private contacts to the central authority. We present a formal summary of these results in section 3 after defining necessary preliminaries.

Under the realistic assumption of a limited amount of communication among private contacts, we show positive results for addressing general algorithmic problems with applications in areas including social-network-based recommender systems [42, 46]. We stress that our model, contrary to previous work [15], is not restricted to graph-based problems. In fact, we show results on general machine learning and data mining problems, finding heavy hitter [33] and solving clustering problems (like k-medians, k-means and k-centers) [11, 29].

To further motivate the model, we study a more restrictive case in which the devices cannot communicate directly with each other. In this *no-inter-communication* scenario, we show strong impossibility results for many problems of practical interest, including approximating the distance of users in social networks or counting the connected components of the graph. These impossibility results, in turn, motivate our model that allows minimal communication among devices. On the positive side, we show that, informally, even in this restrictive case, any problem that allows a streaming algorithm with a limited amount of space can be solved efficiently.

The rest of the paper proceeds as follows. In section 2, we introduce our models and review the prior public-private graph models. Then, in section 3, we present our main theoretical results for our model. Next, in section 4, we show empirical evaluation of algorithms in our model. In section 5, we review related work, and finally, in section 6, we present our conclusions.

2 ON-DEVICE PUBLIC-PRIVATE MODEL OF COMPUTATION

In this section, we formally define the *on-device public-private model* of computation. Before defining the model, we review the *public-private graph* model, which was introduced by Chierichetti et al. [15], to which we refer as *centralized* public-private model to distinguish it from the *on-device* version we introduce.

Centralized model. In the (centralized) public-private model of computation, we have a graph $G = (V, E)$, called the *public* graph, where V is the set of nodes representing users of a system and E is the set of public edges connecting them. Let $n = |V|$ and $m = |E|$. Then, for each node $u \in V$, we have an associated *private* graph $G_u = (\{u\} \cup V_u, E_u)$, over the set of nodes $V_u = \{v_1, \ldots, v_k\} \subseteq V$ representing the private edges of the node. Throughout the paper, we assume that the public and private graphs are undirected, unweighted graphs, but the model can be defined [15] in the obvious way for general graphs. We also focus on the most studied instance of the model (known as the star case) where the set E_u consists of a star centered in u (i.e., $E_u = \{u\} \times V_u$), which represents the private neighbors of the node u.

In the centralized public-private model of computation, we have a graph problem to solve (e.g., determining the connected components of the graph) and want to output efficiently the solution for the problem for the graph visible to any arbitrary user u (i.e., the public-private graph of node u, $G \cup G_u$) after performing some efficient preprocessing of the public graph G computed by a central machine. More precisely, let P be the preprocessing algorithm. Ideally, we want P to analyze the public graph G in a time that is polynomial in its size (i.e., $O(\text{poly}(n + m))$), and to output a space-efficient *synopsis* $P(G)$ of the public graph with space $\tilde{O}(n)$.[1] Then, the aim of the query algorithm is to take in input for node u the private graph G_u and to output the solution of the problem for $G \cup G_u$ by using the synopsis $P(G)$, in time and space that is ideally near-linear in $|E_u|$ and poly-log(n). Crucially, we observe that this model assumes that the query algorithm can access the entire synopsis by making queries to it that depend on the private data. Implicitly, this means that the private graph (or some information based on it) resides in the central machine.

On-device model. Motivated by the problem of social-network-based recommendations, in this paper we refine the previous model to enforce that each private graph G_u is stored exclusively on the device owned by the agent u to protect her privacy. We assume that private edges of E_u represent reciprocal private connections of the user u (i.e., $v \in E_u$ implies that $u \in E_v$ and that u and v know each other and they want their connection to be private to them and not shared to the central authority or any other user). Hence, contrary to the centralized model, the query algorithm must be executed on a device of limited computation power and (crucially) cannot make queries to the central machine that depend on the private edges, as this would reveal the private connections to the shared machine.

To model this, we assume, as in the previous model, that a central machine with high computational power can preprocess the public graph G in polynomial time. Then, contrary to the previous model, we assume that the query algorithm for $G \cup G_u$ is run on the

[1] we use $\tilde{O}(\cdot)$ to neglect polylogarithmic factors.

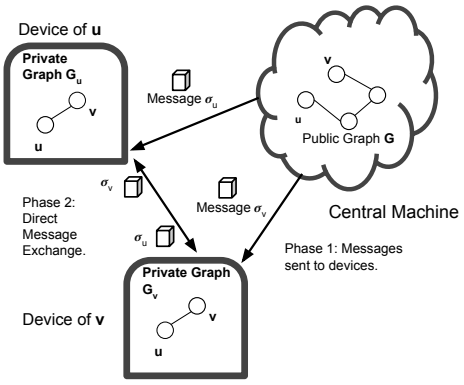


Figure 1: Description of the on-device public private model of computation

private device of u that is not capable of storing the whole public graph. Indeed, as the complete graph might be large, we restrict the computation power (both space and running time) on each device to be sublinear in the size of the public graph $o(n + m)$, preferably poly-logarithmic in n, and polynomial in the size of the private graph, ideally $\tilde{O}(E_u)$. As this rules out the possibility for the device to store a synopsis of G of size $\Theta(n)$, we allow the device to communicate with the central machine and to the other private neighbors' devices with small-size messages.

The on-device communication protocol. More formally, in the on-device model, we assume that the central machine, after preprocessing the public graph G, sends a single message σ_u to the device corresponding to the node u of (ideally) poly-log(n) bits (we refer to Figure 1 for a pictorial description of the model). No other communication is allowed with the central machine from the device thereafter. Indeed, the shared machine has no information about the private graph G_u, and σ_u just depends on the public graph G and node u, thus preserving completely the privacy of the private contacts in G_u. Next, the device of u is allowed to directly query the device of each private neighbor $v \in E_u$ to receive the *same* message σ_v sent to v by the central machine. We observe that this communication does not involve the central authority and so does not leak any information to it. We also observe that the messages exchanged between a pair of private contacts do not contain private information (as they are obtained from the public machine and shared verbatim with the private contact), so no leakage is possible. Finally, after this exchange, the query algorithm on the device uses the messages received *only* to solve the problem.

For instance, in our book recommendation system example in the introduction, v might be a contact of u in u's favoring secure messaging app.[2] User u may obtain σ_v using the same messaging app to contact v, without thus sharing the app contact list to the book recommendation system provider. Then, u will use the messages received to compute the recommendations on her device.

[2]In our model we abstract away the details of the communication protocols involved. We only assume that there is a way for the devices to obtain small messages from their private contacts without involving the central authority providing the recommendation system.

Guarantees. Notice that, in this model, we protect *by design* all private information from being known by the central authority, as it is never reported to it. Notice, also, that nodes do not share private data with their neighbors (the messages exchanged contain only public data processed by the central machine). Thus, no leakage is possible from neighbors, by design, under the assumptions that the devices are able to establish a trusted connection with their private contacts, and the private contacts will not leak the existence of their connection to others. Notice that even if a private neighbor were to collude with the central machine, the only information that could be obtained by the machine would be the existence of the connection with that neighbor.

Alternative models without communication or a centralized system. Our model requires a single round of small-sized communication among private contacts. This assumption is indeed necessary to solve many important problems, as, in section 3.8, we show strong impossibility results in a model with no-inter-communication between devices. Finally, we observe that the protocol could be simulated without a central authority, if computational constraints were not present, but that would require to process all public data on the user's devices, which is not feasible for large-scale systems.

Metadata. So far, we have described a model which, as in [15], only involves graph information. Recently, another model has been defined that allows public-private (problem-specific) data associated with users as the public-private framework of data summarization [39]. In our on-device framework, as well, we allow both graph data and non-graph metadata on nodes. We assume that each node has a problem-specific public metadata S_u^P that can be shared publicly to everyone and a problem-specific private metadata S_u^V that is only available to u. This allows us to address non-graph-related problems such as clustering and heavy hitters, as we show in our paper. Similar to the graph information, the public metadata can be stored in the central machine, while the private metadata must be processed locally. We observe that more sophisticated access controls (who-can-access-what) can be defined, but we leave this to a future work. To handle the metadata, we seek algorithms that use polynomial time and space in the size of all public data for preprocessing and linear time in the private data of a user for the query algorithm.

3 ALGORITHMIC RESULTS FOR THE ON-DEVICE MODEL

In this section, we report our main algorithmic results. First, we study formally the relationships between the centralized and the on-device model in Section 3.1. Then, we show novel algorithmic results for the heavy hitters problem and k-clustering problem. Finally, we show impossibility results when the devices are not allowed to communicate between private contacts.

3.1 Relationships between the models

Before introducing our main novel algorithmic results, as a warm-up, we present a few basic lemmas to clarify the relationships between the on-device and the centralized model for graph algorithms. In the on-device model, we define Σ as the maximum size

of a message exchanged in the protocol, while in the centralized algorithm, we define $|P(G)|$ as the size of the preprocessing synopsis obtained.

First, we prove that the on-device model is at least as hard as the centralized model, because any on-device algorithm can be implemented in the centralized setting.

Lemma 1. *Any on-device public-private graph algorithm with polynomial time preprocessing, $\Sigma \in \tilde{O}(1)$ message size and $\tilde{O}(|E_u|)$ query time, can be implemented in the centralized public-private graph model with polynomial time preprocessing, $|P(G)| \in \tilde{O}(n)$ preprocessing space and $\tilde{O}(|E_u|)$ query time.*

Proof. It is easy to observe that using the central machine, one can simulate the on-device protocol by storing in the synopsis all the messages sent to the nodes and reading them from memory during query time. The total space for the preprocessing is hence $O(n\Sigma)$, while the preprocessing time is clearly polynomial. At query time, the central machine reads the $\tilde{O}(|E_u|)$ messages "sent" to u and simulates the on-device query algorithm in $\tilde{O}(|E_u|)$ time. □

Another simple lemma shows that the contrary is not always true: the on-device mode is *strictly* harder.

Lemma 2. *There exist problems that admit a $|P(G)| \in O(n)$ preprocessing space in the centralized model and for which there is no $o(n)$ message size algorithm in the on-device public-private model.*

The proof is omitted for lack of space, but it is based on standard results in communication theory. All omitted proofs are deferred to the extended version of the paper.

Despite the on-device model being generally harder, we now show that for a significant class of centralized public-private graph algorithms, there is an immediate reduction to the on-device model.

Lemma 3. *Suppose for a given problem there exists a centralized public-private algorithm with synopsis $P(G)$ decomposable in n data-structures, $P(G)_u$ for $u \in V$ such that $|P(G)_u| \in \tilde{O}(1)$ and such that the query algorithm for node u solves the problem in $\tilde{O}(|E_u|)$ time by having access only to $P(G)_u$ and $P(G)_v$ for $v \in E_u$. Then, there exists a $\Sigma \in \tilde{O}(1)$ message size and $\tilde{O}(|E_u|)$ query time algorithm in the on-device model.*

Proof. We observe that the reduction is immediate because the central machine can send message $\sigma_u := P(G)_u$ with space $\tilde{O}(1)$ to each node u, and then the devices can implement the query algorithm using only the messages received from the central machine and their private contacts. □

The previous reductions yield directly algorithms in the on-device model for many problems already studied in the centralized model such as counting the number of connected components, estimating the size of the reachability set of a user, and several centrality estimations (we refer the reader to [15] for details) when the centralized public-private algorithm defined therein has the property required by Lemma 3. Conversely, we observe that this property does not hold for many important algorithms defined in centralized settings, including approximating shortest path distances between nodes u and v and estimating several notions of node similarities between them, because such algorithms require arbitrary access to the centralized synopsis. In particular, the central machine needs to be informed about the node v queried from u. Solving these problems while protecting the privacy of the query in our restrictive model is an interesting open problem.

3.2 Problems addressed

We now introduce our main algorithmic contributions in which we study a class of problems related to recommendation systems in the on-device public-private model. In a social-network recommendation system, there is a set of arbitrary items $\mathcal{S}$ and a graph indicating the social network. We assume the social graph includes a public part G and a private graph G_u for each node u as described above. For each vertex u, we have a set $S_u^P \subseteq \mathcal{S}$ of public items and a set $S_u^V \subseteq \mathcal{S}$ of private items, respectively, that u likes.

We model the problem of implementing a social recommendation system that, given the information in the neighborhood reachable within distance d of a vertex u in the graph visible to u, $G \cup G_u$, returns a subset of items I, with $|I| = k$ to recommend to u. More formally, for a graph $\mathcal{G}$, we let $N_u^d(\mathcal{G})$ be the set of vertices reachable from u within a distance at most d on graph $\mathcal{G}$. We let

$$S_u^d(\mathcal{G}) = S_u^V \cup \bigcup_{v \in N_u^d(\mathcal{G})} S_v^P$$

be the set of items visible to u in $\mathcal{G}$ (i.e., the private items of u plus all public items of nodes reachable from u at distance less than d in the graph $\mathcal{G}$). We will drop $\mathcal{G}$ when we refer to the public graph G. In particular, in our on-device model, we want to approximate the output of certain functions specifying relevant items over the set $S_u^d(G \cup G_u)$ to recommend. We show results for the following problems.

Uniform recommendation function. A uniform recommendation function is a randomized function that returns a subset of items uniformly at random without replacement from $S_u^d(G \cup G_u)$.

Linear recommendation function. Fix a node u, let S_v be $S_v := S_v^P$ for $v \neq u$ and $S_u := S_u^V \cup S_u^P$ for u. Notice that $S_u^d(G \cup G_u) = \bigcup_{v \in N_u^d(G \cup G_u)} S_v$. For a fixed u, item $i \in \mathcal{S}$, and distance d, we define $C_u^i(G \cup G_u)$ as the number of distinct nodes $v \in N_u^d(G \cup G_u)$ for each i that appears in the set S_v defined above. A linear recommendation function is a randomized function that returns a subset of k items I, where the items in I are chosen from $S_u^d(G \cup G_u)$ without replacement with probability proportional to $C_u^i(G \cup G_u)$.

Heavy hitter. Let M be a multiset of $\mathcal{S}$, i.e., a set of items with repetitions. For parameters ϵ and κ, heavy hitter is a function that returns a subset of items I from a multiset M with two properties. 1) I contains all items that appear in at least $\frac{|M|}{\kappa}$ times in M where $|M|$ is the total number of items counting repetitions; and 2) each item $i \in I$ appears at least $\frac{|M|}{\kappa} - \epsilon$ times in M.

For a node u and distance d, we let M be the multiset that contains the union (with repetitions) of S_v for distinct $v \in N_u^d(G \cup G_u)$ (i.e., all items visible to u counted with the number of users liked by them). We seek to find all heavy hitters as defined above.

k-Clustering. In this case, each item in $\mathcal{S}$ represents a point in an arbitrary metric space, and the goal is to pick a set of k points

Algorithm 1: General Algorithm.

Input: $G, \forall_{u \in V}(S_u^P, S_u^V, G_u), d, k$
Output: Output k items for each node u

1: **Pre-processing on central machine** using $G, \forall_{u \in V} S_u^P$.
2: **for** all $u \in V$, for $d' \leq d$ **do**
3: Let $\bar{S}_u^{d'}(G) := \bigcup_{v \in N_u^d(G)} S_v^P$.
4: Compute $I_u^{d'} := sketch(\bar{S}_u^{d'}(G))$.
5: Send to u, $\sigma_u := (I_u^{d-1}, I_u^d)$.
6: **Query algorithm on u's device** using G_u, S_u^V.
7: Receive σ_u from central machine.
8: Send σ_u to v and receive σ_v from v for all $v \in E_u$.
9: Compute $I_u^V := sketch(S_u^V)$.
10: Compose sketches I_u^V, I_v^{d-1} for $v \in E_u$, and I_u^d to obtain sketch I_u.
11: Extract k items from I_u.

$r_1, \ldots, r_k$ that minimizes the cost function.

$$\Big(\sum_{s \in S_u^d(G \cup G_u)} \min_{j=1}^{k} \text{dist}(r_j, s)^\rho \Big)^{1/\rho},$$

for a particular fixed ρ. $\rho = 1$ corresponds to k-median, $\rho = 2$ to k-means and $\rho \to \infty$ to k-center.

In the next sections, we show algorithms with provable theoretical guarantees for all these problems in our model.

3.3 General algorithmic approach

The general approach of our algorithms (reported in Algorithm 1) for the previously mentioned problems is to define a composable sketch function *sketch* over sets of items such that 1) from $sketch(A)$, one can extract k items, approximately solving the problem; 2) $sketch(A)$ has small space $\tilde{O}(k)$; and 3) the sketches computed over two (possibly overlapping) sets of items A and B can be computed easily by composing $sketch(A)$ and $sketch(B)$. Assume that such a function exists; we will use it to compute a sketch $I_u^{d'}$ for the items in the neighborhood at distance at most d' from a node u on the public graph G over the public items S_u^P for all $u \in V$. These sketches will be computed over the public data in the centralized machine in the preprocessing phase for all integers $1 \leq d' \leq d$ using the public graph G and the public items S_u^P for all $u \in V$. We will send to each individual node u, as the message σ_u, the sketches for the greatest two distances (i.e., $\sigma_u := (I_u^{d-1}, I_u^d)$). Then, the nodes will share the message received from the central machine with their private neighbors and collect their messages. In the query phase, node u will compose: the sketch I_u^d and the sketches $I_v^{d-1} \forall v \in |E_u|$ received, as well as the sketch computed over the private local items S_u^V, to obtain a sketch I_u and to extract from it the recommendations.

Given this general framework, for each specific problem, we only need to show how the preprocessing phase obtains the sketches $I_u^{d'}$ efficiently, how they can be composed, and how to extract the solution from the sketch.

3.4 Uniform recommendation function

We define the sketch for the uniform case. We start with a simple sketching algorithm that does not work. Let us assume the public server defines as sketch I_u^d of uniform and independent sample

Algorithm 2: Algorithm for uniform recommendation function.

Input: $G, \forall_{u \in V}(S_u^P, S_u^V, G_u), d, k$, a hash function $h(.)$
Output: Output k uniform sample items for each node u

1: **Obtaining $I^{d'}$'s during preprocessing**
2: For $d' = 0$
3: **for** all $u \in V$ **do**
4: Sort items in $i \in S_u^P$ based on hash $h(i)$ and let I_u^0 be the first k items by $h(\cdot)$.
5: **for** $d' \in \{1 \cdots d\}$ for $u \in V$ **do**
6: Let $\hat{I}_u^{d'}$ be the union of $I_u^{d'-1} \cup \bigcup_{v \in N(u)} I_v^{d'-1}$ where $N(u)$ is refers to graph G.
7: Sort items in $\hat{I}_u^{d'}$ based on $h(i)$ and let $I_u^{d'}$ be the first k items.
8: **Query algorithm on u's device** using same $h(\cdot)$ function.
9: Let I_u^V be the top k items by $h(\cdot)$ in S_u^V.
10: Compute $\hat{I}_u$ as union of I_u^V, I_v^{d-1} for $v \in E_u$, and I_u^d.
11: Extract k items from $\hat{I}_u$ by $h(\cdot)$.

of items from S_u^d. Suppose u has two neighbors, v' and v''. Let item $i \in S_{v'}^d$, $i \in S_{v''}^d$, and $i' \in S_{v'}^d$, but $i' \notin S_{v''}^d$. In this case, the probability that i exists in either one of the samples of v' or v'' is more than the probability for element i', which is not desired because it does not allow a simple combination of the two samples.

To resolve this issue, we assume that there exists a k-min-wise independent uniform hash function $h : \mathcal{S} \to [0, 1]$ from the items to $[0, 1]$. We assume that the function has shared random seeds that are available to all devices and the central machine (i.e., we can compute it consistently in every device). The algorithm is reported in Algorithm 2, in which we report only the details missing from the general Algorithm 1. To sketch a set of items S, we hash the items in S and keep the k items with minimum values. Two sketches for sets A, B can be easily composed because one can sort the items in $A \cup B$ and return the top k by h. Algorithm 2 shows how to use this property to iteratively construct $I_u^{d'}$ for all nodes based on $I_u^{d'}$ and how to initialize I_u^0, which is the sketch of S_u^P. Then, $I_u^{d'}$ is obtained by combining the sketches of $I_v^{d'-1}$ for the public neighbors of u and the previous sketch $I_u^{d'-1}$ of u. Finally, the last two sketches for d and $d-1$ are sent to each device. To construct uniform recommendations, for any v in the private neighborhood of u, we query v to obtain I_v^{d-1}, construct $\cup_{v' \in E_u \cup G_v} I_{v'}^{d-1} \cup I_u^d$, and compose it with the sketch of the private items in u. Finally, we recommend the k items with minimum hash value from the combined sketch.

The following lemma can be shown.

Theorem 4. *Algorithm 2 obtains k uniform recommendations without replacement for each node in the on-device model with total preprocessing time $\tilde{O}\left(\sum_u |S_u^P| + kd(m+n)\right)$ using $\tilde{O}(k)$ space messages and $\tilde{O}\left(|S_u^V| + k|E_u|\right)$ query time.*

We observe that one could use l_0 sketches [30] on each vertex to provide a uniform recommendation function. However, l_0 samplers provide independent samples (i.e., with replacements). This can be resolved by using more sketches on each vertex and removing duplicated items, but l_0 samplers are designed to handle deletions of items as well, which is not necessary here.

3.5 Linear recommendation function

In this subsection, we provide a simple technique to recommend k items without replacement from $S_u^d(G \cup G_u)$ with probability proportional to $C_u^i(G \cup G_u)$.

First, we give a simple example to show that unlike the uniform sampling case, several copies of independent samplers are not useful to solve the problem efficiently. Suppose we have only items i and i', and $C_u^i(G \cup G_u) = n$ and $C_u^{i'}(G \cup G_u) = 1$. Note that any sampler that samples 2 items without replacement reports both i and i'. However, sampling one item gives i with probability $\frac{n}{n+1}$, and i' with probability $\frac{1}{n+1}$. Hence, in expectations, we need to make $O(n)$ linear samples with replacement to observe i' once, which is not desirable.

Here again, we use k-min-wise hash functions with shared random bits in a similar fashion to Algorithm 2. The key difference with the algorithm is in the sketch method, which instead of hashing items $\mathcal{S}$ to $[0, 1]$, it hashes pairs of items (i, u) to $[0, 1]$, where i is an item and u is a vertex containing i in its items. We first focus on the preprocessing. For each vertex u, we define $T_u^{d'} = \{(v, i) | v \in N_v^{d'}(G)$ and $i \in S_v^P\}$. We define the sketch $I_u^{d'}$ to be the set of k pairs in $T_u^{d'}$ with minimum hash values and with distinct items (i.e., when an item appears several times, we only keep the pair with lowest hash value for that item). It is easy to see that such sketches can be computed iteratively as in the uniform case. Notice also that at query time for node u, by having access to I_u^d, I_v^{d-1} for $v \in E_u$ and S_u^V, one can construct in the same way a sketch for $S_u^d(G \cup G_u) = \cup_{v \in N_u^d(G \cup G_u)} S_v$, where S_v is defined as $S_v := S_v^P$ for $v \neq u$ and $S_u = S_u^P \cup S_u^V$.

To construct the linear recommendations, we recommend the k distinct items with minimum hash values in the combined sketch of $S_u^d(G \cup G_u)$. Notice that the recommended set is equivalent to the set of the k distinct items with minimum hash values in $T_u^d(G \cup G_u) = \{(v, i) | v \in N_u^d(G \cup G_u)$ and $i \in S_v\}$. Hence, the probability that a pair corresponding to an item i has the minimum hash value is proportional to its frequency C_u^i as desired. After removing the pairs corresponding to the first item, the same argument holds for the second item, and so on. This allows the following theorem to be shown.

Theorem 5. *The previous algorithm obtains k linear recommendations without replacement for each node in the on-device model with total preprocessing time $\tilde{O}\left(\sum_u |S_u^P| + kd(m+n)\right)$ using $\tilde{O}(k)$ space messages and $\tilde{O}\left(|S_u^V| + k|E_u|\right)$ query time.*

3.6 Heavy Hitters

In this subsection, we provide a technique to recommend heavy hitters. Fix a probability $0 < \delta < 1$. Let $s = |\mathcal{S}|$ be the total number of items. We use a similar technique to the linear case, except that we use $H = 12 \log \frac{2s}{\delta} \epsilon^{-2}$ fully independent hash functions. As in the linear recommendations case, each hash function maps a pair (u, i) of user and item to $[0, 1]$. The sketch I_u^d keeps H pairs, one per function, such that for each function, the pair stored is the one with minimum hash value over the items inserted in the sketch. It can be observed that the sketch is composable. The construction of the sketches in the pre-processing and query time is the same as in the linear case. It can be noted that the final sketch I_u^d constructed at query time is a list of $\frac{12 \log \frac{2s}{\delta}}{\epsilon^2}$ items sampled independently, this time with replacement, from $N_u^d(G \cup G_u)$ with probability proportional to $C_u^d(G \cup G_u)$. We recommend an item if and only if it appears at least $\left(\frac{1}{\kappa} - \frac{\epsilon}{2}\right) \frac{12 \log \frac{2s}{\delta}}{\epsilon^2}$ times in the list I_u^d.

Given the result in Theorem 5, the proof can be adapted to show the following bound on the running time and message space.

Lemma 6. *The previous algorithm obtains runs in the on-device model with total pre-processing time $\tilde{O}\left(\sum_u |S_u^P| H + d(m+n)H\right)$ using $\tilde{O}(H)$ space messages and $\tilde{O}\left(|S_u^V| H + |E_u| H\right)$ query time.*

With a little more work, we can also show the following lemma.

Lemma 7. *The algorithm defined above returns the heavy hitter items in the neighborhood of u, with probability $1 - \delta$.*

3.7 k-clustering

Our algorithm in this section is based on the notion of composable core-sets [11]. Bateni et al. [11] show that given a linear space α-approximation algorithm for k-clustering problem for a fixed ρ norm (e.g., $\rho = 1$ for k-median, $\rho = 2$ for k-means, $\rho = \infty$ for k-center), the following procedure provides an $O(\alpha)$-approximate solution of the problem. Arbitrarily decompose the points into *disjoint* sets $U_1, \cdots, U_t$. Inside each set U_i select an approximate solution of size $\tilde{\Theta}(k)$. We call these points representatives. Then, for each point in U_i that is not a representative we move it to the location of the closest representative point in U_i. Now one can represent the new dataset by distinct point locations and their multiplicities, which are composable core-sets. Then, combine the composable core-sets and solve the problem on the new dataset. This is known to provide a $O(\alpha)$-approximation. Notice that the challenge to applying this method to our case is that the points in the sets S_u^P, S_u^V for $u \in V$ are not necessarily disjoint. We show, nevertheless, how to use this technique in our case.

Similar to the technique in previous sections, for each node u we keep a composable core-set of the points in $\bigcup_{v \in N_u^{d'}(G)} S_v^P$ for $d' = d$ and $d' = d - 1$, for each node. We do this, in this section, by first running a BFS search from each node u, gathering the points in S_v^P for $v \in N_u^{d'}(G)$, and finding the corresponding core-sets $I_u^{d'}$ as in [11]. Note that this preprocessing step can be implemented in $\tilde{O}(k|V|)$ space and polynomial time. Then, u receives I_v^{d-1} from each vertex v in its private neighborhood. Node u merges the userâĂŹs core-set I_u^d and the core-sets from private neighbors I_v^{d-1} and solves the problem. Note that core-sets I_v^{d-1} correspond to not necessarily disjoint sets. However, each point appears at most $|E_u| + 1$ times in these core-sets, where $|E_u|$ is the number of private neighbors of u. Therefore, we can show that our approximation factor is increased by a factor of at most $|E_u| + 1$. This implies the following theorem, where, as in [11], we assume there is a polynomial time, linear space, α-approximation algorithm *Alg* for k-clustering in the norm ρ.

Theorem 8. *The previous algorithm obtains a $O(\alpha |E_u|)$-approximate k-clustering in the norm ρ, for each node u, in the on-device model*

with polynomial time preprocessing, using $\tilde{O}(k)$ space messages, and using query time polynomial in $|S_u^V|$, k and $|E_u|$, as desired.

Specifically, the total pre-processing time is

$$\tilde{O}\left(\sum_u f\left(\left|\bigcup_{v\in N_u^{d'}(G)} S_v^P\right|\right) + m + \sum_u \sum_{v\in N_u^d(G)} \left|S_u^P\right|\right)$$

and query time is $\tilde{O}\left(f\left(\left|S_u^V\right| + k|E_u|\right)\right)$, where $f(.)$ indicates the running time of the offline algorithm *Alg*.

The increase $|E_u|$ in the approximation factor in the previous algorithm is due to the possibility of items appearing in multiple core-sets. In the next subsection we provide a sketching technique that allows us to reduce the weight of the duplicated points from the core-sets. By applying this sketching, our algorithm is (approximately) using the true weight of the points in the core-set. While this heuristic does not improve the approximation factor in the worst case, this theoretically sound heuristic improves the results in practice.

3.7.1 Removing duplicated items. As we mentioned earlier in this subsection, composable core-sets are designed to work with disjoint sets (i.e., a decomposition of the input set). However, in the case of our application, if a vertex v is reachable from two private neighbors of u (via distance less than d), the items (i.e., points) of v appear in both of these neighbors. Hence, we double count the weight of these items on the representative items. If we could explicitly keep the set of the items assigned to the representative items (which we refer to as $A_1, \ldots, A_t$ for simplicity of notation), we could directly indicate the duplicated items and remove their weights from the representative items. However, this might not be practical due to the large size of these sets. In this subsection we provide a technique to sketch these sets such that we can remove the weight of the duplicated items fairly accurately (i.e., with an ϵ error) using a smaller memory.

Next, we provide our tool to deal with sets that are not disjoint for composable core-sets. This tool might be of independent interest.

Let A be a collection of items and let $A_1, \ldots, A_t \subseteq A$ be a collection of subsets of A. Without loss of generality, we assume $|A_1| \geq |A_2| \geq \cdots \geq |A_t|$. Let Δ be an upper bound on the number sets that contain a, for any arbitrary $a \in A$. Let $h(.)$ be a universally shared hash function that hashes $A \rightarrow [0,1]$ uniformly at random. We let $\hat{A}_i$ be the subset of A_i with hash value at most $\lambda_i = \frac{6\log(\delta t)\Delta^2}{\epsilon^2}\frac{1}{|A_i|}$. Note that we have $E[|\hat{A}_i|] = |A_i|\frac{6\log(\delta t)\Delta^2}{\epsilon^2}\frac{1}{|A_i|} = \frac{6\log(\delta t)\Delta^2}{\epsilon^2} \in \tilde{O}(\Delta^2)$.

For $i \in \{1, \ldots, t\}$ we define $B_i = \left\{a \middle| a \in A_t \& \nexists_{j>i} a \in A_j\right\}$. This intuitively says that to construct B_is from A_is, we remove duplicated items from the larger sets and keep them in the smaller ones. Similarly, for $i \in \{1, \ldots, t\}$ we define $\hat{B}_i = \left\{a \middle| a \in \hat{A}_t \& \nexists_{j>i} a \in \hat{A}_j\right\}$.

Lemma 9. *With probability at least $1 - \delta^{-1}$, simultaneously, for all $i \in \{1, \ldots, t\}$ we have*

$$\left||B_i| - \frac{1}{\lambda_i}|\hat{B}_i|\right| \leq \epsilon\frac{|A_i|}{\Delta}.$$

The proof is omitted for lack of space.

3.8 Hardness results for no-inter-communication models

To motivate our model, here we consider a more restrictive case where we do not allow the nodes to exchange the messages σ's with their private contacts. Hence, after the message σ_u is received, the query algorithm has to solve the problem based exclusively on the local private graph G_u and the message σ_u. We call this model *no-inter-communication* model.

We show that it is impossible to solve several basic graph problems in the *no-inter-communication* public-private model, which motivates our more general model and distinguishes the computation power of these two models. We summarize our negative results in the following theorem.

Theorem 10. *It is information-theoretically impossible to solve the following problems in the* no-inter-communication *public-private model using $o(n)$ size messages:*

- *Approximate the distance of u and v.*
- *Decide if u is connected to v.*
- *Approximate the number of connected components.*

The full proof of all the results is available in the full version. For lack of space, here we describe the intuition behind them. Our hardness results are based on two-party communication complexity techniques. Note that we provide information-theoretic hardness, and hence we do not require computational complexity assumptions like $P \neq NP$.

We use the hardness of the indexing problem [32] and set disjointness [31, 44]. In the indexing problem, we have two parties: namely, Alice and Bob. Alice holds a subset $A \subseteq \{1, 2, \ldots, n\}$ and Bob holds an element a. The goal is for Bob to detect whether or not $a \in A$, using a one-way communication link from Alice to Bob. It is known that $\Omega(n)$ bits of communication are required to solve this problem.

In the set disjointness problem as well, we have two parties, Alice and Bob, each holding a subset of $\{1, 2, \ldots, n\}$. Say Alice holds A and Bob holds B. The goal here is to detect whether or not $A \cap B = \emptyset$, using a two-way communication link. This problem requires $\Omega(n)$ bits of communication as well.

To prove the above theorem, let us define the reachability problem as follow. We have a network G with a vertex r indicated as the root. A vertex v is reachable if there is a path from v to the root. Indeed, the goal is to detect whether each vertex is reachable. The next theorem shows the hardness of solving the reachability problem in the *no-inter-communication* model.

Lemma 11. *It is not possible to solve the reachability problem in the* no-inter-communication *model using $o(n)$ size messages to the device.*

Next we show the hardness of counting the number of connected components.

Lemma 12. *Let $\epsilon \in (0, 1]$ be an arbitrary small constant. It is not possible to approximate the number of connected components with an approximation factor ϵ in the* no-inter-communication *model using $o(n)$ communication with each device.*

Let us define the delay approximation problem as follows. We have an undirected unweighted network G with a vertex r indicated

as the root. The delay of a vertex v is equal to the length of the shortest path between v and the root. The delay approximation problem is to approximate the length of the shortest path between each vertex and the root. The next theorem shows the hardness of delay approximation in the *no-inter-communication* model.

Lemma 13. *It is not possible to solve the delay approximation problem with an approximation factor $\omega(\frac{1}{n})$ in the* no-inter-communication *model using $o(n)$ communication with each local device.*

Positive results. Despite the strong impossibility results presented above, we show a reduction from the popular class of streaming algorithms (with $\tilde{O}(1)$ space). This result translates any streaming algorithm to a *no-inter-communication* on-device public-private algorithm and directly provides a few basic algorithms for the *no-inter-communication* model.

Theorem 14. *Consider a problem that admits an (insertion-only) streaming algorithm with $\tilde{O}(1)$ space, T_0 update time per element inserted, and T_1 query time. The problem can be solved in the* no-inter-communication *public-private setting using $\tilde{O}(1)$ message space and $\tilde{O}(|E_u|T_0 + T_1)$ query time.*

One application of this theorem is to estimate the size of the maximum matching in planar graphs (and bounded arboricity graphs). This can be done with $\tilde{O}(1)$ space, $\tilde{O}(1)$ update time, and $\tilde{O}(1)$ query time [37]. In fact, none of the other results presented in this paper can be derived from this theorem.

4 EXPERIMENTS

In this section, we empirically evaluate the efficiency of our algorithms in the on-device public-private model. For experimentation purposes, to ensure replicability of the results, and for privacy reasons, we exclusively use public-available data for all our experiments. All our datasets are obtained from the Stanford's SNAP dataset (http://snap.stanford.edu/data) or from the DBLP public data repository of computer science publication records (https://dblp.uni-trier.de/). Of course, these datasets do not contain private data, so as in previous works [15, 39], we simulate the public-private data split in our dataset. We first describe the datasets used and then show how we simulated the public-private data split.

Uniform sampling, linear sampling, and heavy hitter. To evaluate our algorithms for uniform sampling, linear sampling, and heavy hitters, we use three datasets, referred to as `dblp`, `foods` [35], and `movies` [35]. From these three datasets we extract a collection of user-user relationships (representing the social network of the users) and a collection of user-item pairs (representing the items associated to each user). We first present statistics on these datasets.

- `dblp` contains 1.1×10^6 nodes 8.6×10^6 links and 8.1×10^3 distinct items (for 3.5×10^6 total user-items relationships)
- `foods` contains 2.1×10^5 nodes 1.8×10^7 links and 6.8×10^4 distinct items (for 4.7×10^5 total user-items relationships).
- `movies` contains 9.4×10^5 nodes 1.7×10^8 links and 2.4×10^5 distinct items (for 6.7×10^6 total user-items relationships).

For `dblp` the nodes are authors, and the items associated with a node correspond to the venues in which the author published. The social graph instead represents co-authorship relationships among nodes. For `foods` and `movies`, the nodes are users in a recommender system (see [35] for more details), while the user-items relationships represent the objects each user reviewed. The edges in the social network represent co-reviewing relationships.

k-Clustering. To evaluate our algorithms for the k-clustering problem instead we use a dataset where items are associated with a point in a metric space. We use the `gowalla` dataset [16] which contains 2.0×10^5 nodes 9.5×10^5 links and a total of 6.4×10^6 distinct items. The items represent check-ins corresponding to 2d location data. We use $\rho = \infty$ for the norm, corresponding to the well studied k-center problem. The edges in this dataset are the social connections among users.

Simulating the public-private graphs. To obtain examples of public-private graphs as in [15, 39] we mark independently and uniformly a fraction p of the edges in each dataset as private and remove them from the public graph G while adding each private edge u, v to the private graphs G_u and G_v. We use $p = 1/2$ in the experiments reported, but this coefficient does not qualitatively affect the results obtained. As the most salient aspect of the model is the sketching of the public data in the preprocessing phase we mark all items as public in our experiments, again this does not qualitatively affect the results significantly.

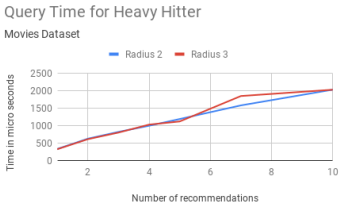
4.1 Results for uniform sampling, linear sampling, and heavy hitter

Here we experiment with our algorithms for uniform sampling, linear sampling, and heavy hitter. In this setting, we use $d = 2$ and $d = 3$ for the radius of the neighborhood around a node for which we extract the suggestions. We chose this range as it is large enough to be algorithmically challenging but not so large that it results in little variability between node suggestions (it is well known that most nodes are at distance 4 to 5 from each other [8, 34]). We experiment with the recommendation number k in the range $[1, 10]$. For heavy hitter we set $\kappa = k$, $\epsilon = \frac{1}{2k}$, and probability of success to 90% (i.e, we upper bound the number of recommendations by k). For stability of the results, when we report the running time, we show the mean of 100 independent executions.

Recall that our aim is to provide to node u suggestions from $S_u^d(G \cup G_u) = S_u^V \cup \bigcup_{v \in N_u^d(G \cup G_u)} S_v^P$ according to the definitions of the specific problem (uniform sampling, linear sampling, and heavy hitter). Notice that our on-device algorithms for uniform sampling and linear sampling are guaranteed to have the same output of a centralized algorithm that analyzes all the data (private and public) in $S_u^d(G \cup G_u)$ and then computes the suggestions. Moreover, our on-device algorithm for heavy hitters has the same output (and hence accuracy) of the centralized sketch-based algorithm running on all the data in $S_u^d(G \cup G_u)$. As such, in our experiments for these problems, we focus on the efficiency of the algorithms. Our main aim is to show that executing the query algorithm for a single node is highly efficient and can be performed on an inexpensive device. In particular, we seek to show that our preprocessing allows the query time to be much faster than without preprocessing the data. To show this, we make the conservative comparison of the running time of our query algorithm, executed for one node u, with an algorithm that simply loads in memory all data in $S_u^d(G \cup G_u)$ (which is much faster than an algorithm computing the suggestions).

This will show the gains that come from the preprocessing. Also, we show the preprocessing algorithm running time to evaluate its scalability in a centralized system. For ease of comparison of the preprocessing and query algorithm running times, we ran all our algorithms (preprocessing and query) as a standard sequential single-core job on the same machine (we used an Intel Xeon CPU E5-1650 v4 with 6 cores at 3.60 GHz, running Debian). Each job uses a single core, but we observe that the preprocessing algorithms lead to an obvious distributed implementation. We observe for the query algorithms very little memory use (less than 1 MB for the data storage), while the preprocessing used less than 10 GB of RAM.

(a) `movies` preprocessing time



(b) `movies` query time

Figure 2: Running time of the preprocessing and query algorithm for heavy hitter.

Preprocessing. We report the running time of our preprocessing algorithms in Figures 2a for the heavy hitter case in `movies`. We omit the figures for the uniform and linear sampling case for lack of space as well as the figures for `dblp`, `foods`. The results are qualitatively similar in all datasets and problems more details are deferred to the full version of the paper.

From the charts, we can make the following observations. Note that as expected, the number of recommendations has only an almost-linear effect on the preprocessing time. The radius d has also a moderate effect on the preprocessing time.

Query time. We now focus on the query time, which is reported in Figure 2b for heavy hitter in `movies` (results for linear and uniform sampling and for other datasets are omitted for lack of space, but are qualitatively very similar). Here we can observe that the diameter, as expected, does not affect the query time. As expected, increasing the number of recommendations has a linear effect on the query time. We observe running times that range from 0.5 to 2.00 milliseconds per query in heavy hitter for the movies dataset (see Figure 2b), showing great efficiency.

Recall that to show a conservative estimate of the speedup of the running time of our query algorithm due to the preprocessing, we compare its mean running time (over 100 random nodes u) with the mean running time of an algorithm that simply loads in memory all data in $S_u^d(G \cup G_u)$. Our experiments for linear sampling, uniform sampling, and heavy hitter shows that the query time for `dblp` obtains a speedup of a factor 10 to 100 for $d = 2$ and a factor 100 to 1000 for $d = 3$ by the preprocessing. Similarly, for `foods` we see a factor 20 to 200 for $d = 2$ and a factor 400 to 4000 for $d = 3$ and for `movies` a factor 200 to 2000 for $d = 2$ and a factor 15000 to 150000 for $d = 3$.

4.2 Results for clustering

In this section, we show similar experimental results for our algorithm for k-center (i.e., k-clustering with $\rho = \infty$) in the `gowalla` dataset. We set the number of points in the core-set to be k and we set the de-duplication method to use $E[|\hat{A}_i|] = 10|A_i|$. As clustering algorithm, to obtain the core-sets we use the standard k-center greedy algorithm. The running times are reported in Figure 3a and Figure 3b for preprocessing and query algorithm, respectively, for $d = 2$ and $k \in [1, 10]$ clusters. The preprocessing time is again linearly related to k. The average running time per query is less than 9 milliseconds. Here we observe that the running time has a more than linear relationship with k but still is extremely fast for all k. Again, as a benchmark, we compare the running times of the query algorithm with a centralized algorithm: gathering the items on a 2-neighborhood of the query node, we run the same greedy k-center algorithm and observe an increase in speed from a factor of 38 to a factor of 700 times for $d = 2$. Again, this confirms that the preprocessing improves performances significantly.

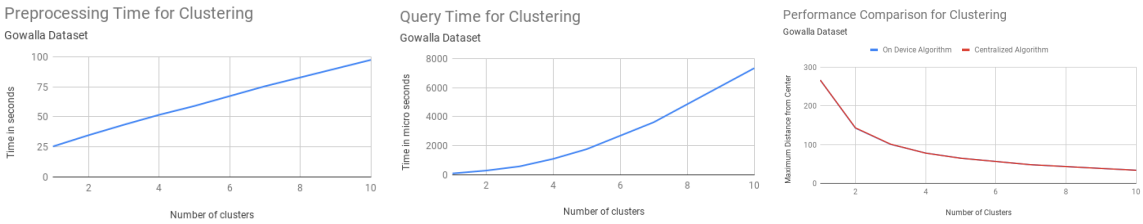
Then, in Figure 3c, we evaluated the cost of the clustering solution obtained by our algorithm (using the point de-duplication technique) and the cost obtained by a centralized clustering algorithm that runs over the correct dataset without duplicates. We recall that the issue with the composability of the sketches is that the points in the sketches for the private neighbors are not disjoint (notice that this is true even if each user has distinct points, because the sketches include points at distance 2 from the neighbor). For this reason, we introduced a sketching technique to estimate the correct multiplicity of the points in the core-sets. As it is possible to see in the figure, the two lines are almost coincident. In fact, we observe less than 1% variation in the cost of the two solutions, confirming the accuracy of the method beyond the worst case guarantees in this dataset.

5 RELATED WORK

Related work to our paper spans a variety of areas, so we will focus only on the most directly related ones.

Our paper is motivated by the importance of preserving privacy in social-network-based systems. This area has attracted significant attention [21, 26, 27, 48]. Several authors describe attacks and remedies to potential attacks on social-networks' privacy [9, 47, 48]. Users are increasingly interested in privacy controls of social-network data as Dey et al. [21] observe.

Private-public data model. This interest has motivated many privacy-preserving frameworks in the past. The one most related to our work is the public-private graph framework [15] described in section 2. Chierichetti et al. [15] show how the framework allows



(a) `gowalla` preprocessing time **(b) `gowalla` query time** **(c) `gowalla` clustering solution cost**

Figure 3: Running time of the preprocessing and query algorithm for clustering and cost of the clustering solution.

multiple graph problems to be solved, including connected components, all-pair distance estimation [19], node similarity scores as social affinity [41], correlation clustering [10], shortest paths [20], and Personalized PageRank computation [25]. Notice that Chierichetti et al. [15] do not explicitly address the issue of keeping the user's private data on the device of limited capacity because it models a centralized setting.

The original framework was defined for graph problems only, however, other non-graph problems have been studied such as the data summarization framework [39], where users possess both private and public data and one wants to summarize the data for personalized recommender systems. The authors show succinct summaries for optimizing submodular user utility functions. Similarly, Archer et al. [7] study indexing problems in public-private graphs, where one wants to retrieve nodes visible to a user. Neither Mirzasoleiman et al. [39] nor Archer et al. [7] define explicitly a notion of on-device computation.

Other privacy-preserving learning models. Related to ours is the rich area of research on differential privacy [1, 22, 23]. Differential privacy was originally proposed as a model for preventing leakage of individuals' information from the output of a computation over private databases. In a nutshell, a differentially private algorithm is guaranteed to have an output that is mostly unaffected by the data of any given user, in a strong probabilistic sense. This strong constraint limits the information over the private data that an attacker can learn. This method has found wide adoption in many areas of computer science, including deep learning [1], for its theoretical elegance and its strong guarantees.

The original applications of differential privacy were in centralized settings, but recently, several studies have used it to protect communication of on-device data to a central authority. An example is Qin et al. [43], who address recommender system problems using differential privacy in a setting not involving social-graph-based information (unlike our paper). More generally, the local differential privacy [24] technique consisting of perturbing local information before sending it to the central authority has received wide attention because it allows the computation of aggregate statistics over user data.

Another related topic is the setting of federated learning [38]. In a federated learning scenario, the users of a system want to optimize a shared machine learning model (e.g., training a single model for all users, using their own private on-device data in a distributed setting without sharing it to the central machine). In this setting, several works [14, 38] show strong theoretical guarantees of privacy based on differential privacy and sophisticated cryptographic protocols [14]. We observe that the setting is different from ours because it does not directly model social-graph-based information, and it focuses on the problem of using every user's data to improve machine learning models for all users.

Contrary to previous work, in our on-device framework, we do not rely directly on differential privacy techniques, as we guarantee, by design, an even stronger notion of privacy because private data is only used to provide output to the data owner. We also observe that our model does not require the insertion of noise in the output, the estimation of output sensitivity [23], or the use of specific cryptographic protocols. However, we believe that the wealth of theoretical results on differential privacy, federated learning, and other areas conceivably can be used to address a wide class of problems in variants of our model, an effort that we leave as an open problem.

Algorithmic problems addressed in our work. Our paper addresses several algorithmic problems in heavy hitters [33] and clustering [4, 29], both of which have a vast literature. Related to our algorithmic techniques is the area of streaming algorithms and streaming data analysis [30, 36, 45]. In particular, we use extensively techniques from sketching [5, 6] that have found applications in graph problems [13, 17, 18, 36, 40]. Particularly related are the notions of composable sketches, which are also related to the concept of composable core-sets [3, 28], mergeable summaries [2], and mapping core-sets for clustering [11].

Finally, to obtain our impossibility results, we use a few important results such as [31, 32, 44].

6 CONCLUSIONS

We introduce a new model for addressing several computational problems while providing strong privacy guarantees for the data and contacts present on a user's device. We believe that our paper further shows how techniques from streaming and sketching can be used for privacy-preserving computations in on-device computations. As a future work, we would like to further explore the connection of the area of streaming and privacy to address more problems in the on-device public-private model. Also, an interesting area of study is the use of differential-privacy methods in this model.

Acknowledgments. We thank Flavio Chierichetti, Ravi Kumar and Silvio Lattanzi for their work on the public-private graph model that inspired this work.

REFERENCES

[1] Martin Abadi, Andy Chu, Ian Goodfellow, H Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. 2016. Deep learning with differential privacy. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 308–318.

[2] Pankaj K Agarwal, Graham Cormode, Zengfeng Huang, Jeff M Phillips, Zhewei Wei, and Ke Yi. 2013. Mergeable summaries. *ACM Transactions on Database Systems (TODS)* 38, 4 (2013), 26.

[3] Pankaj K Agarwal, Sariel Har-Peled, and Kasturi R Varadarajan. 2004. Approximating extent measures of points. *Journal of the ACM (JACM)* 51, 4 (2004), 606–635.

[4] Sara Ahmadian, Ashkan Norouzi-Fard, Ola Svensson, and Justin Ward. 2017. Better guarantees for k-means and euclidean k-median by primal-dual algorithms. In *Foundations of Computer Science (FOCS), 2017 IEEE 58th Annual Symposium on*. Ieee, 61–72.

[5] Kook Jin Ahn, Sudipto Guha, and Andrew McGregor. 2012. Analyzing graph structure via linear measurements. In *Proceedings of the twenty-third annual ACM-SIAM symposium on Discrete Algorithms*. SIAM, 459–467.

[6] Kook Jin Ahn, Sudipto Guha, and Andrew McGregor. 2012. Graph sketches: sparsification, spanners, and subgraphs. In *Proceedings of the 31st ACM SIGMOD-SIGACT-SIGAI symposium on Principles of Database Systems*. ACM, 5–14.

[7] Aaron Archer, Silvio Lattanzi, Peter Likarish, and Sergei Vassilvitskii. 2017. Indexing Public-Private Graphs. In *Proceedings of the 26th International Conference on World Wide Web*. International World Wide Web Conferences Steering Committee, 1461–1470.

[8] Lars Backstrom, Paolo Boldi, Marco Rosa, Johan Ugander, and Sebastiano Vigna. 2012. Four degrees of separation. In *Proceedings of the 4th Annual ACM Web Science Conference*. ACM, 33–42.

[9] Lars Backstrom, Cynthia Dwork, and Jon Kleinberg. 2007. Wherefore art thou r3579x?: anonymized social networks, hidden patterns, and structural steganography. In *Proceedings of the 16th international conference on World Wide Web*. ACM, 181–190.

[10] Nikhil Bansal, Avrim Blum, and Shuchi Chawla. 2004. Correlation clustering. *Machine learning* 56, 1-3 (2004), 89–113.

[11] MohammadHossein Bateni, Aditya Bhaskara, Silvio Lattanzi, and Vahab Mirrokni. 2014. Distributed balanced clustering via mapping coresets. In *Advances in Neural Information Processing Systems*. 2591–2599.

[12] Jesús Bobadilla, Fernando Ortega, Antonio Hernando, and Abraham Gutiérrez. 2013. Recommender systems survey. *Knowledge-based systems* 46 (2013), 109–132.

[13] Paolo Boldi, Marco Rosa, and Sebastiano Vigna. 2011. HyperANF: Approximating the neighbourhood function of very large graphs on a budget. In *Proceedings of the 20th international conference on World wide web*. ACM, 625–634.

[14] Keith Bonawitz, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H Brendan McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, and Karn Seth. 2017. Practical secure aggregation for privacy-preserving machine learning. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 1175–1191.

[15] Flavio Chierichetti, Alessandro Epasto, Ravi Kumar, Silvio Lattanzi, and Vahab Mirrokni. 2015. Efficient algorithms for public-private social networks. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 139–148.

[16] Eunjoon Cho, Seth A Myers, and Jure Leskovec. 2011. Friendship and mobility: user movement in location-based social networks. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 1082–1090.

[17] Edith Cohen. 1997. Size-estimation framework with applications to transitive closure and reachability. *J. Comput. System Sci.* 55, 3 (1997), 441–453.

[18] Edith Cohen. 2015. All-distances sketches, revisited: HIP estimators for massive graphs analysis. *IEEE Transactions on Knowledge and Data Engineering* 27, 9 (2015), 2320–2334.

[19] Atish Das Sarma, Sreenivas Gollapudi, Marc Najork, and Rina Panigrahy. 2010. A sketch-based distance oracle for web-scale graphs. In *Proceedings of the third ACM international conference on Web search and data mining*. ACM, 401–410.

[20] Camil Demetrescu and Giuseppe F Italiano. 2004. A new approach to dynamic all pairs shortest paths. *Journal of the ACM (JACM)* 51, 6 (2004), 968–992.

[21] Ratan Dey, Zubin Jelveh, and Keith Ross. 2012. Facebook users have become much more private: A large-scale study. In *Pervasive Computing and Communications Workshops (PERCOM Workshops), 2012 IEEE International Conference on*. IEEE, 346–352.

[22] Irit Dinur and Kobbi Nissim. 2003. Revealing information while preserving privacy. In *Proceedings of the twenty-second ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*. ACM, 202–210.

[23] Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. 2006. Calibrating noise to sensitivity in private data analysis. In *Theory of cryptography conference*. Springer, 265–284.

[24] Úlfar Erlingsson, Vasyl Pihur, and Aleksandra Korolova. 2014. Rappor: Randomized aggregatable privacy-preserving ordinal response. In *Proceedings of the 2014 ACM SIGSAC conference on computer and communications security*. ACM, 1054–1067.

[25] Dániel Fogaras, Balázs Rácz, Károly Csalogány, and Tamás Sarlós. 2005. Towards scaling fully personalized pagerank: Algorithms, lower bounds, and experiments. *Internet Mathematics* 2, 3 (2005), 333–358.

[26] David Garcia. 2017. Leaking privacy and shadow profiles in online social networks. *Science advances* 3, 8 (2017), e1701172.

[27] Ralph Gross and Alessandro Acquisti. 2005. Information revelation and privacy in online social networks. In *Proceedings of the 2005 ACM workshop on Privacy in the electronic society*. ACM, 71–80.

[28] Piotr Indyk, Sepideh Mahabadi, Mohammad Mahdian, and Vahab S Mirrokni. 2014. Composable core-sets for diversity and coverage maximization. In *Proceedings of the 33rd ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*. ACM, 100–108.

[29] Anil K Jain. 2010. Data clustering: 50 years beyond K-means. *Pattern recognition letters* 31, 8 (2010), 651–666.

[30] Hossein Jowhari, Mert Sağlam, and Gábor Tardos. 2011. Tight bounds for lp samplers, finding duplicates in streams, and related problems. In *Proceedings of the thirtieth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*. ACM, 49–58.

[31] Bala Kalyanasundaram and Georg Schintger. 1992. The probabilistic communication complexity of set intersection. *SIAM Journal on Discrete Mathematics* 5, 4 (1992), 545–557.

[32] Ilan Kremer, Noam Nisan, and Dana Ron. 1999. On randomized one-round communication complexity. *Computational Complexity* 8, 1 (1999), 21–49.

[33] Kasper Green Larsen, Jelani Nelson, Huy L Nguyên, and Mikkel Thorup. 2016. Heavy hitters via cluster-preserving clustering. *arXiv preprint arXiv:1604.01357* (2016).

[34] Jure Leskovec, Jon Kleinberg, and Christos Faloutsos. 2005. Graphs over time: densification laws, shrinking diameters and possible explanations. In *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*. ACM, 177–187.

[35] Julian John McAuley and Jure Leskovec. 2013. From amateurs to connoisseurs: modeling the evolution of user expertise through online reviews. In *Proceedings of the 22nd international conference on World Wide Web*. ACM, 897–908.

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

[37] Andrew McGregor and Sofya Vorotnikova. 2018. A simple, space-efficient, streaming algorithm for matchings in low arboricity graphs. In *OASIcs-OpenAccess Series in Informatics*, Vol. 61. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.

[38] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. 2017. Communication-Efficient Learning of Deep Networks from Decentralized Data. In *Artificial Intelligence and Statistics*. 1273–1282.

[39] Baharan Mirzasoleiman, Morteza Zadimoghaddam, and Amin Karbasi. 2016. Fast distributed submodular cover: Public-private data summarization. In *Advances in Neural Information Processing Systems*. 3594–3602.

[40] Christopher R Palmer, Phillip B Gibbons, and Christos Faloutsos. 2002. ANF: A fast and scalable tool for data mining in massive graphs. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 81–90.

[41] Rina Panigrahy, Marc Najork, and Yinglian Xie. 2012. How user behavior is related to social affinity. In *Proceedings of the fifth ACM international conference on Web search and data mining*. ACM, 713–722.

[42] Moon-Hee Park, Jin-Hyuk Hong, and Sung-Bae Cho. 2007. Location-based recommendation system using Bayesian user's preference model in mobile devices. In *International Conference on Ubiquitous Intelligence and Computing*. Springer, 1130–1139.

[43] Zhan Qin, Yin Yang, Ting Yu, Issa Khalil, Xiaokui Xiao, and Kui Ren. 2016. Heavy hitter estimation over set-valued data with local differential privacy. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 192–203.

[44] Alexander A. Razborov. 1992. On the distributional complexity of disjointness. *Theoretical Computer Science* 106, 2 (1992), 385–390.

[45] Jonathan A Silva, Elaine R Faria, Rodrigo C Barros, Eduardo R Hruschka, Andre CPLF De Carvalho, and João Gama. 2013. Data stream clustering: A survey. *ACM Computing Surveys (CSUR)* 46, 1 (2013), 13.

[46] Frank Edward Walter, Stefano Battiston, and Frank Schweitzer. 2008. A model of a trust-based recommendation system on a social network. *Autonomous Agents and Multi-Agent Systems* 16, 1 (2008), 57–74.

[47] Lingjing Yu, Sri Mounica Motipalli, Dongwon Lee, Peng Liu, Heng Xu, Qingyun Liu, Jianlong Tan, and Bo Luo. 2018. My Friend Leaks My Privacy: Modeling and Analyzing Privacy in Social Networks. In *Proceedings of the 23nd ACM on Symposium on Access Control Models and Technologies*. ACM, 93–104.

[48] Elena Zheleva and Lise Getoor. 2009. To join or not to join: the illusion of privacy in social networks with mixed public and private user profiles. In *Proceedings of the 18th international conference on World wide web*. ACM, 531–540.