2024 IEEE Symposium on Security and Privacy (SP) | 979-8-3503-3130-1/24/$31.00 ©2024 IEEE | DOI: 10.1109/SP54263.2024.00196

Time-Aware Projections: Truly Node-Private Graph Statistics under Continual Observation*

Palak Jain
Boston University
palakj@bu.edu

Adam Smith
Boston University
ads22@bu.edu

Connor Wagaman
Boston University
wagaman@bu.edu

***Abstract*—Releasing differentially private statistics about social network data is challenging: one individual's data consists of a node and all of its connections, and typical analyses are sensitive to the insertion of a single unusual node in the network. This challenge is further complicated in the *continual release* setting, where the network varies over time and one wants to release information at many time points as the network grows. Previous work addresses node-private continual release by assuming an unenforced promise on the maximum degree in a graph; indeed, the algorithms from these works exhibit blatant privacy violations when the degree bound is not met.**

In this work, we describe the first algorithms that satisfy the standard notion of node-differential privacy in the continual release setting (i.e., without an assumed promise on the input streams). These algorithms are accurate on sparse graphs, for several fundamental graph problems: counting edges, triangles, other subgraphs, and connected components; and releasing degree histograms. Our unconditionally private algorithms generally have optimal error, up to polylogarithmic factors and lower-order terms.

We provide general transformations that take a base algorithm for the continual release setting, which need only be private for streams satisfying a promised degree bound, and produce an algorithm that is unconditionally private yet mimics the base algorithm when the stream meets the degree bound (and adds only linear overhead to the time and space complexity of the base algorithm). To do so, we design new projection algorithms for graph streams, based on the batch-model techniques of [BBDS13; DLL16], which modify the stream to limit its degree. Our main technical innovation is to show that the projections are *stable*—meaning that similar input graphs have similar projections—when the input stream satisfies a privately testable safety condition. Our transformation then follows a novel online variant of the Propose-Test-Release framework [DL09], privately testing the safety condition before releasing output at each step.

1. Introduction

Graphs provide a flexible and powerful way to model and represent relational data, such as social networks, epidemiological contact-tracing data, and employer-employee relationships. Counts of substructures—edges, nodes of a given degree, triangles, connected components—are fundamental statistics that shed light on a network's structure and are the focus of extensive algorithmic study. For example, edge counts can quantify relationships in a social network; a function of triangle and 2-star counts called the "correlation coefficient" or "transitivity" of a network is of interest to epidemiologists for understanding disease spread [BS10; YJM+13]; and connected component counts have been used for determining the number of classes in a population [Goo49] and estimating fatalities in the Syrian civil war [CSS18].

When the graph contains sensitive information about individuals, one must balance the accuracy of released statistics with those individuals' privacy. Differential privacy [DMNS16] is a widely studied and deployed framework for quantifying such a trade-off. It requires that the output of an algorithm reveal little about any single individual's record (even hiding its presence or absence in the data set).

In this work, we study differentially private algorithms that continually monitor several fundamental statistics about a graph that evolves over time. We consider the *continual release* (or *continual observation*) model of differential privacy [DNPR10; CSS11] in which the input data is updated over time and statistics about it must be released continuously. (In contrast, in the *batch* model, input arrives in one shot and output is produced only once.)

There are two standard notions of differential privacy (DP) for algorithms that operate on graph data: (1) *edge DP* [NRS07], for which the algorithm must effectively obscure the information revealed by any individual edge (including its mere presence or absence), and (2) *node*

* A full version of this paper is available on arXiv [JSW24].

DOI 10.1109/SP54263.2024.00182

DP [HLMJ09; BBDS13; KNRS13; CZ13], for which the algorithm must obscure the information revealed by a node's entire set of connections (including even the node's presence or absence).

Edge privacy is typically easier to achieve and more widely studied. However, in social networks and similar settings, nodes—rather than edges—correspond to individuals and so node privacy is more directly relevant. Indeed, existing attacks infer sensitive information about a person from aggregate information about their neighborhood in the network (e.g., sexuality can be inferred from an individual's Facebook friends [JM09]), showing that privacy at the node level rather than only the edge level is important.

Background: Node-differential Privacy. To understand the challenges of designing node-private algorithms, consider the task of estimating the number of edges in a graph. For every graph G with n vertices, there is a node-neighboring graph G' with n more edges (obtained by adding a new, high-degree node connected to all existing nodes). A node private algorithm, however, must hide the difference between G and G'. Therefore, every node-private algorithm must have additive error $\Omega(n)$ on either G or G', which means large relative error when G and G' are sparse. It is thus impossible to get a useful worst-case accuracy guarantee for counting the edges in a graph or, for similar reasons, many other basic statistics.

As a result, node private algorithms are often tailored to specific families of inputs. In the batch model, instead of aiming for universal accuracy across all graph types, algorithms are designed to provide privacy for all possible graphs while providing accurate estimates for a select subset of "nice" graphs—for example, graphs that satisfy a degree bound—on which the statistic of interest is well behaved. There are now several techniques for achieving this type of guarantee in the batch model, notably projections [BBDS13; KNRS13; DLL16] and Lipschitz extensions [BBDS13; KNRS13; CZ13; RS16b; RS16a; DLL16; BCSZ18b; CD20; KRST23]. Broadly, these techniques start from an algorithm which is both private and accurate when restricted to a set of "nice" graphs, and find a new algorithm that mimics the base algorithm on the set of "nice" graphs while providing privacy for all possible graphs; such extensions exist under very general conditions [BCSZ18a; BCSZ18b].

The continual release setting complicates these approaches, causing tools for the batch setting to break down. Projections and Lipschitz extensions are harder to design: in the batch setting, the decision to remove an edge may propagate only across nodes; in the continual release setting, the change can also propagate through time, rendering existing batch-model solutions ineffective. Even the general existential result of [BCSZ18a] applies, at best, only to the offline version of continual release (in which the algorithm can inspect the entire input stream before starting to produce output). In a nutshell, ensuring low sensitivity separately at each point in time does not guarantee the type of stability that is needed to get low error with continual release (e.g., ℓ_1 stability of difference vectors [SLM+18]). For this reason, the only straightforward way to get node-private algorithms from existing batch model work is to use advanced composition [DRV10] and compose over T time steps. This potentially explains why prior works on node-private continual release of graph statistics assume restrictions on the input graphs to their private algorithms, providing privacy only in the case where the restriction is satisfied.

1.1. Our Results

Truly Node-private Algorithms. For several fundamental graph statistics, we obtain (the first) algorithms that satisfy the usual notion of node-differential privacy in the continual release setting—that is, they require no assumption on the input streams.

In contrast, previous work on node-private continual release of graph statistics [SLM+18; FHO21] develops algorithms for basic graph statistics that are accurate and private when the input graph stream has maximum degree at most a user-specified bound D but exhibit blatant privacy violations if the input graph stream violates the bound.[1] This conditional node-privacy is not standard in the literature: prior work on node-privacy in the batch model gives unconditional privacy guarantees (e.g., [BBDS13; KNRS13; CZ13; RS16b; RS16a; DLL16; BCSZ18b; CD20; KRST23]). To emphasize the conditional nature of the privacy guarantees for [SLM+18; FHO21], we say they satisfy *D-restricted node-DP*. (Similarly, algorithms whose edge-privacy depends on such an assumption satisfy *D-restricted edge-DP*.)

We consider an insertion-only model of graph streams, where an arbitrary subset of new nodes and edges arrives at each of T time steps (Definition 2.2). We do not assume any relationship between the size of the graph and T. The degree of a node u in the

1. For example, suppose edges in the graph denote transmissions of a stigmatized disease like HIV and suppose the analyst knows that all the edges associated with one individual, Bob, arrive at a given time step t (and only those edges arrive). The outputs of the [FHO21] edge-counting algorithm at times $t-1$ and t would together reveal how many disease transmissions Bob is involved in, up to error D/ε, for privacy parameter ε. When Bob's degree is much larger than D, this is a clear violation of node privacy. One can argue using this example that the algorithms of [SLM+18; FHO21] do not satisfy (ε, δ)-node differential privacy (Definition 2.8) for any finite ε with $\delta < 1$.

stream is the total number of edges adjacent to u in the stream (equivalently, in the final graph). The stream's maximum degree is the largest of its nodes' degrees; if this maximum is at most D, we say the stream is *D-bounded*.

Our main contribution is a black-box transformation that can take any D-restricted-DP "base" algorithm and transform it into an algorithm that is private on all graphs while maintaining the original algorithm's accuracy on D-bounded graphs.

We then use this transformation to produce specific node-private algorithms for estimating several fundamental graph statistics and (in all but one case) show that the error incurred by these private algorithms is near optimal. Prior work either has accuracy that is exponentially worse in T for the same privacy guarantees (e.g., [BBDS13; CZ13; KNRS13; DLL16; KRST23]), or exhibits blatant privacy violations ([SLM+18; FHO21]) and is not actually node-DP. We generally use algorithms from previous work as our base, though for connected components the restricted-DP algorithm is new. Importantly for real-world applications, our algorithms are efficient: they add only linear overhead to the time and space complexity of the base algorithm.

Table 1 summarizes the bounds we obtain on additive error—worst-case over D-bounded graph streams of length T—for releasing the counts of edges (f_{edges}), triangles ($f_{\mathsf{triangles}}$), k-stars[2] ($f_{\mathsf{k\text{-}stars}}$), and connected components (f_{CC}), as well as degree histograms ($f_{\mathsf{degree\text{-}hist}}$); it also compares with previous results. The parameters ε, δ specify the privacy guarantee (Definition 2.8).

Stable, Time-aware Projections. Central to our approach is the design of *time-aware projection* algorithms that take as input an arbitrary graph stream and produce a new graph stream, in real time, that satisfies a user-specified degree bound D. "Time-aware" here refers to the fact that the projection acts on a stream, as opposed to a single graph; we drop this term when the context is clear. "Projection" comes from the additional requirement that the output stream be identical to the input on every prefix of the input that is D-bounded. Ideally, we would simply run the restricted-DP algorithm on the projected graph stream and thus preserve the original algorithm's accuracy on D-bounded streams.

The challenge is that the resulting process is only private if the projection algorithm is *stable*, meaning that neighboring input streams map to nearby projected streams. Specifically, the *node distance* between two streams S and S' is the minimum number of nodes that must be added to and/or removed from S to obtain S'. *Edge distance* is defined similarly (Definition 2.5). *Node-neighboring* streams are at node distance 1. The *node-to-node stability* of a projection is the largest node distance between the projections of any two node-neighboring streams; the *node-to-edge stability* is the largest edge distance among such pairs.

If we had a projection with good (that is, low) node-to-node stability, then running the restricted-DP algorithm on the projected graph stream would satisfy node privacy, and we would be done. Alas, we do not know if such a projection exists. (We show that such a transformation does exist for edge-DP—see the end of this section.) Instead, we give two simple, greedy projection algorithms that have good node-to-node and node-to-edge stability when the input graph stream satisfies a privately testable "safety" condition. The safety condition is that the stream has few large-degree vertices (Definition 3.2). Specifically, a graph (or stream) is (D, ℓ)-bounded if it has at most ℓ nodes of degree larger than D.[3]

We obtain a general transformations from D-restricted-DP algorithms to truly private ones by testing the safety condition using a novel online variant of the Propose-Test-Release framework of [DL09].

We explore two natural methods for time-aware projection, each based on a batch-model projection algorithm from the literature. Both time-aware projections greedily add edges while maintaining an upper bound on each node's degree. One of these methods bases its greedy choices on the degree of the nodes in the original graph stream ("BBDS", [BBDS13]), while the other bases its choices on the degree of the nodes in the projection that it produces ("DLL", [DLL16]). The results in Table 1 are obtained using the BBDS-based projection and our general transformation.

We give tight bounds on three measures of stability, summarized in Table 2. The table lists upper bounds; the lower bounds for the BBDS projection are identical up to small additive constants (and the edge-to-edge stability is identical), while the bounds for DLL are tight up to a constant multiplicative factor. A graph in the batch model can be represented as a length-1 graph stream, so these projections' stability properties also hold for graphs in the batch model.

The DLL projection preserves more edges than the BBDS projection when the input has some high-degree vertices (the graph returned by BBDS is a subgraph of that returned by DLL), which initially suggests that the DLL projection could be more useful. Indeed, in the batch setting, the authors of [DLL16] show that the

2. A k-star is a set of k nodes, each with an edge to a single common neighbor (which can be thought of as the k-star's center).

3. (D, ℓ)-boundedness is a computationally efficient proxy for requiring that the stream be close in node-distance to a D-bounded stream. Testing the latter condition directly is NP-hard (by reduction from vertex cover); we instead efficiently compute the distance to the nearest *not* (D, ℓ)-bounded stream—see Section 4.

	Lower bounds for $\varepsilon \geq \frac{\log T}{T}, \delta = O(\frac{1}{T})$	Reference	Additive ℓ_∞ error for $\varepsilon \leq 1$, $\delta = \Omega\left(\frac{1}{\text{poly}(T)}\right)$	Node-DP guarantee
f_{edges}	$\Omega(D \log T/\varepsilon)$	BBDS/CZ/KNRS	$\widetilde{O}(D\sqrt{T}/\varepsilon)^*$	(ε, δ)
		[FHO21]	$O(D \log^{5/2} T/\varepsilon)$	D-restricted $(\varepsilon, 0)$
		Our work	$O(D \log^{5/2} T/\varepsilon + \log^{7/2} T/\varepsilon^2)$	(ε, δ)
$f_{\text{triangles}}$	$\Omega(D^2 \log T/\varepsilon)$	BBDS/CZ/KNRS	$\widetilde{O}(D^2\sqrt{T}/\varepsilon)^*$	(ε, δ)
		[FHO21]	$O(D^2 \log^{5/2} T/\varepsilon)$	D-restricted $(\varepsilon, 0)$
		Our work	$O(D^2 \log^{5/2} T/\varepsilon + \log^{9/2} T/\varepsilon^3)$	(ε, δ)
$f_{\text{k-stars}}$	$\Omega(D^k \log T/\varepsilon)$	BBDS/CZ/KNRS	$\widetilde{O}(D^k\sqrt{T}/\varepsilon)^*$	(ε, δ)
		[FHO21]	$O(D^k \log^{5/2} T/\varepsilon)$	D-restricted $(\varepsilon, 0)$
		Our work	$O(D^k \log^{5/2} T/\varepsilon + \log^{k+5/2} T/\varepsilon^{k+1})$	(ε, δ)
$f_{\text{degree-hist}}$	$\Omega(D \log T/\varepsilon)$	[DLL16]	$\widetilde{O}(D^2\sqrt{T}/\varepsilon)^*$	(ε, δ)
		[FHO21]	$\widetilde{O}(D^2 \log^{5/2} T/\varepsilon)$	D-restricted $(\varepsilon, 0)$
		Our work	$\widetilde{O}(D^2 \log^{5/2} T/\varepsilon + \log^{9/2} T/\varepsilon^3)$	(ε, δ)
f_{CC}	$\Omega(D \log T/\varepsilon)$	[KRST23]	$\widetilde{O}(D\sqrt{T}/\varepsilon)^*$	(ε, δ)
		Our work	$O(D \log^{5/2} T/\varepsilon + \log^{7/2} T/\varepsilon^2)$	(ε, δ)

TABLE 1. ACCURACY OF OUR NODE-PRIVATE ALGORITHMS, PREVIOUSLY KNOWN *restricted* NODE-PRIVATE ALGORITHMS, AND NODE-PRIVATE BATCH MODEL ALGORITHMS ON INSERTION-ONLY, D-BOUNDED GRAPH STREAMS OF LENGTH T. "BBDS/CZ/KNRS" REFERS TO [BBDS13; CZ13; KNRS13]; THE ERROR BOUNDS WITH * WERE COMPUTED BY APPLYING ADVANCED COMPOSITION [DRV10] TO BATCH MODEL ALGORITHMS. ERROR LOWER BOUNDS FOR THESE PROBLEMS ARE FOR SUFFICIENTLY LARGE T.

	Π_D^{BBDS}	Π_D^{DLL}
edge-to-edge	3	$2\ell + 1$
node-to-edge	$D + \ell$	$D + 2\ell\sqrt{\min\{D, \ell\}}$
node-to-node	$2\ell + 1$	$2\ell + 1$

TABLE 2. STABILITY OF Π_D^{BBDS} AND Π_D^{DLL} ON (D, ℓ)-BOUNDED INPUT GRAPH STREAMS, FROM THEOREM 3.3.

projected degree distribution (and number of edges) has low sensitivity. This allows for the DLL projection to provide a better privacy-utility trade-off for these tasks in the batch model. However, this projection actually has worse stability when we measure node- or edge-distance between output graphs.[4] Therefore, more noise must be added when using the DLL projection for generic applications, as compared to the BBDS projection. In our uses, this ultimately means that the projection of [BBDS13] provides the better privacy-utility trade-off.

Truly Edge-private Algorithms. Although our focus is on node-privacy, we show along the way that the BBDS-based time-aware projection has edge-sensitivity 3, uniformly over all graphs. (This follows from a batch-model argument of [BBDS13] and a general "Flattening Lemma" (Lemma 3.6) that we establish for greedy, time-aware projections.) As a result, one can make D-restricted edge-private algorithms into truly private ones at almost no cost in accuracy. Some consequences are summarized in Theorem A.4.

Experiments. We provide experiments on synthetic graphs, which show that our transformation adds little run time overhead and results in truly node-private algorithms that improve considerably over the batch-model baseline.

1.2. Techniques

Stability Analyses. The main technical contribution lies in defining time-aware versions of the two greedy projection algorithms (Algorithm 1), and leveraging that structure to analyze the sensitivity of the entire projected graph sequence (Theorem 3.3). Our analyses differ substantially from existing batch-model analyses, both because of the sequential nature of our problem and the stronger notions of stability we consider.

Section 3.1 contains a detailed overview of the arguments; we highlight here a few simple but useful

4. This distinction is crucial in the continual-release setting. For example, even though the degree distribution of the DLL projection has node sensitivity $O(D)$ in the batch setting, the sequence of degree distributions one gets when projecting a stream has unbounded sensitivity.

ideas. The time-aware projections share two key features:

- *Shortsightedness:* the algorithm includes all nodes and makes a final decision about each edge at the time it arrives;
- *Opportunism:* if an edge connects vertices with degree at most D in the original graph stream, it will necessarily be included in the projection.

Such greedy structure is computationally convenient but also helps us analyze stability. To see why, consider two graph streams S, S' that differ in the presence of a node v^+ and its edges, and let $\Pi_D(S)$ and $\Pi_D(S')$ denote the projected streams (where Π_D could be either of our two projections). We consider for each time step t the *difference graph* Δ_t consisting of edges that have been added (at or before time t) to one projected stream but not the other.

The first feature, shortsightedness, implies that this difference graph *grows monotonically*. (Such a statement need not hold for arbitrary projections.) This allows us to show a "Flattening Lemma" (Lemma 3.6), which states that the edge- and node-distance between $\Pi_D(S)$ and $\Pi_D(S')$ depend only on the final difference graph Δ_T (or an intermediate graph Δ_t in the case that we are considering only a prefix of the streams). Thus, shortsightedness allows us to ignore the sequential structure and reduce to a batch-model version of Π_D in which arrival times affect only the order in which edges are greedily considered.

The second feature, opportunism, allows us to take advantage of (D, ℓ)-boundedness. If the larger stream has at most ℓ vertices of degree more than D, we can show that Δ_t will have a vertex cover of size at most $\ell + 1$ (Lemma 3.7).

These structural results suffice to bound the node-to-node stability of both projections by $2\ell + 1$.

From this point, the analyses of the two projections diverge. Each inclusion rule leads to different structure in the difference graph Δ_T. The most involved of these analyses proves a (tight) bound of $D + 2\ell\sqrt{\min\{D, \ell\}}$ on the node-to-edge stability of the DLL-based projection. At a high level, that analysis proceeds by orienting the edges of Δ_T to show that it is close in edge distance to a large DAG which is covered by at most ℓ edge-disjoint paths and then bounding the possible size of such a DAG.

The node-to-edge analysis of the BBDS-based algorithm is also subtle, but different. The key point there is that all of the edges connected to v^+ can potentially cause changes in the projected graph, even the edges which are not selected for inclusion in the projection themselves. We refer to Section 3.2 for further detail.

Testing Distance to Unsafe Streams. A second, less involved insight is that, although it is NP-hard to compute the node distance to the nearest stream that is not D-bounded, (D, ℓ)-boundedness gives us a proxy that is much easier to work with. Specifically, we observe that D-bounded streams are always distance at least $\ell + 1$ from the nearest non-$(D + \ell, \ell)$-bounded stream; furthermore, this distance can be computed in linear time (Lemma 4.3). Since the distance to non-$(D + \ell, \ell)$-boundedness at any given time step has low node-sensitivity, we can use a novel (to our knowledge) online variant of the PTR framework [DL09] based on the sparse vector technique [RR10; HR10] to monitor the distance and stop releasing outputs when the distance becomes too small. The privacy analysis of this part follows the argument of [DL09] but differs because, rather than making a binary decision to either release or not release an output, the testing process dynamically chooses to release outputs at up to T time steps (see Theorem 4.4). The resulting general transformations are summarized in Theorem 4.1.

1.3. Related Work

Our contributions draw most heavily from the literature on batch-model node-differentially private algorithms. Node privacy was first formulated by [HLMJ09]. The first nontrivial node-private algorithms emerged in three concurrent works [BBDS13; KNRS13; CZ13] that collectively identified two major families of (overlapping) approaches based on Lipschitz extensions [BBDS13; KNRS13; CZ13; RS16a; RS16b; DLL16; BCSZ18b; CD20; KRST23] on one hand, and projections [BBDS13; KNRS13; DLL16] on the other. These works provide algorithms with (tight) accuracy guarantees for D-bounded graphs for the statistics we consider here as well as families that arise in the estimation of stochastic block models and graph neural networks. They also consider other families of graphs on which their specific statistics are well behaved. Most relevant here is the batch-model projection of BBDS [BBDS13] with low edge-to-edge sensitivity, and the Lipschitz extension for degree distributions of DLL [DLL16]. This latter extension can be viewed algorithmically as a greedy projection for which the degree histogram is stable. We use their projection idea, analyzing the stability of the graph as a whole.

There is also an extensive literature on batch-model edge-private algorithms; we do not attempt to survey it here.

A second major tool we draw on is the D-restricted node- and edge-private algorithms of [SLM+18; FHO21] for continual release of graph statistics. These in turn use the widely-studied tree mechanism, whose use in the continual-release setting (for numerical data) dates back to the model's introduction [DNPR10; CSS11]. Also relevant are the edge-private streaming algorithms

of [Upa13; UUA21] for cuts and spectral clustering. (To the best of our understanding, the application of our transformations to their algorithms does not yield non-trivial utility guarantees.)

Finally, our work draws on the Propose-Test-Release framework of [DL09], combining it with the sparse vector mechanism [RR10; HR10; LSL17] to monitor the distance from the stream to the nearest non-(D, ℓ)-bounded stream.

1.4. Organization of This Manuscript

Section 2 lays out the model and basic definitions used in the remainder of the paper. Section 3 presents the time-aware projections and the results on their stability. Section 4 explains the general black-box transformation from D-restricted edge (or node) privacy to true node privacy. Section 5 develops the applications to basic graph statistics. Section 6 presents our experimental results. Because of space constraints, many proofs are deferred to the full version, which can be found at [JSW24].

2. Preliminaries

Definition 2.1 (Graph)**.** A *graph* $G = (V, E)$ consists of a set of vertices V (also known as nodes), and a set of edges E, where edge $\{v_1, v_2\} \in E$ if and only if there is an edge between nodes $v_1 \in V$ and $v_2 \in V$.

Definition 2.2 (Graph stream)**.** Given a time horizon T, a *graph stream* $S \in \mathcal{S}^T$ is a T-element vector, where each element of the vector contains some set of nodes and edges, or the symbol $\perp$ if no nodes or edges arrive in that time step. At each time $t \in [T]$, either $\perp$ arrives or some set of nodes and edges arrives. By convention, an edge's endpoints arrive no later than the edge.

We denote by S_t the set of added nodes and edges which arrive in time step t. We use $S_{[t]}$ to denote the sequence $S_1, \ldots, S_t$.

Definition 2.3 (Flattened graph)**.** Let $S \in \mathcal{S}^T$ be a graph stream of length T. The *flattened graph* of the first t terms $S_{[t]}$ of a graph stream, denoted $\mathsf{flatten}(S_{[t]})$, is the graph that can be formed by all of the nodes and edges which arrive at or before time t.

When the meaning is clear, we may refer to the graph stream through time t when stating a property of the flattened graph of the graph stream through time t. For example, when we say that $S_{[t]}$ has maximum degree at most D, we mean that $\mathsf{flatten}(S_{[t]})$ has maximum degree at most D.

We next define neighboring graphs and graph streams. In privacy-preserving data analysis, the notion of neighboring datasets is important since privacy requires that evaluating a function on similar datasets produces indistinguishable outputs. There are two natural notions of neighboring graphs and graph streams: node neighbors differ on a node (and its associated edges), while edge neighbors differ on one edge. We denote node and edge neighbors with the relations $\simeq_{node}$ and $\simeq_{edge}$ respectively.

Definition 2.4 (Neighboring graph streams)**.** Two graphs (respectively, graph streams) are *node-neighbors* if one can be obtained from the other by removing a vertex and all of its adjacent edges. (For graph streams, the adjacent edges for the removed node may have been spread over many time steps.)

Similarly, two graphs (respectively, graphs streams) are *edge neighbors* if one can be obtained from the other by either removing one edge, removing an isolated node, or removing a node of degree 1 and its adjacent edge.[5]

A generalization of node- and edge-neighboring graphs and graph streams is the notion of node and edge distance between graphs and graph streams. We note that node- and edge-neighboring datasets are at node and edge distance 1, respectively.

Definition 2.5 (Node and edge distance)**.** The *node-distance* $d_{node}(G, G')$ is defined as the length d of the shortest chain of graphs (respectively, graph streams) $G_0, G_1, \ldots, G_d$ where $G_0 = G$, $G_d = G'$, and every adjacent pair in the sequence is node neighboring.

The *edge-distance* $d_{edge}(G, G')$ is defined as the length d of the shortest chain of graphs (respectively, graph streams) $G_0, G_1, \ldots, G_d$ where $G_0 = G$, $G_d = G'$, and every adjacent pair in the sequence is edge neighboring.

Given two graphs with no isolated vertices $G = (V, E)$ and $G' = (V', E')$ (where V and V' may overlap), the edge distance between G and G' is exactly the size of the set $E \triangle E'$. (Isolated vertices that are not in both graphs add to the distance.)

Differential Privacy in the Batch Model. To define differential privacy in the batch model, we introduce the notion of (ε, δ)-indistinguishability.

Definition 2.6 ((ε, δ)-indistinguishability)**.** We say that two random variables R_1, R_2 over outcome space $\mathcal{Y}$ are (ε, δ)-*indistinguishable* (denoted $R_1 \approx_{\varepsilon,\delta} R_2$) if for all $Y \subseteq \mathcal{Y}$, we have

$$\Pr[R_1 \in Y] \leq e^{\varepsilon} \Pr[R_2 \in Y] + \delta;$$
$$\Pr[R_2 \in Y] \leq e^{\varepsilon} \Pr[R_1 \in Y] + \delta.$$

5. Another way to define edge neighbors would be to take the set of nodes as fixed and public, and only consider changes to one edge. We adopt the more general definition since it simplifies our results on node-to-edge stability.

Informally, a function is differentially private if applying the function to inputs which differ in the data of one individual results in outputs from similar distributions—more specifically, from distributions which are (ε, δ)-indistinguishable. We use the definitions of node and edge neighbors presented above to formalize the notion of what it means for graphs or graph streams to differ in the data of one individual.

Definition 2.7 (Differential privacy (DP) in the batch model [DMNS16])**.** A randomized algorithm $\mathcal{M} : \mathcal{S}^T \to \mathcal{Y}$ is (ε, δ)-*node-DP* (respectively, *edge-DP*), if for all pairs of node-neighboring (respectively, edge-neighboring) graph streams S and S', the distributions $\mathcal{M}(S)$ and $\mathcal{M}(S')$ are (ε, δ)-indistinguishable:

$$\mathcal{M}(S) \approx_{\varepsilon,\delta} \mathcal{M}(S').$$

The term *pure DP* refers to the case where $\delta = 0$, and *approximate DP* refers to the case where $\delta > 0$.

Privacy under Continual Observation. We now define privacy of graph statistics under continual observation; the general definition is borrowed from [JRSS23]. In the continual release setting, first explored by [CSS11; DNPR10], an algorithm receives a stream of inputs $S = (S_1, \ldots, S_T) \in \mathcal{S}^T$. The definition of privacy requires indistinguishability on the distribution of the entire sequence, not just one output. For simplicity, in this version, we consider only the simpler, non-adaptive concept of differential privacy. We conjecture that all our algorithms and results extend verbatim to the adaptive version [JRSS23] (since the main components of our algorithm, the tree mechanism and sparse vector technique, are known to be adaptively private).

Definition 2.8 (Privacy of a mechanism under continual observation)**.** Define $\mathcal{A}_{\mathcal{M}}$ as the batch-model algorithm that receives a dataset x as input, runs $\mathcal{M}$ on stream x, and returns the output stream y of $\mathcal{M}$. We say that $\mathcal{M}$ is (ε, δ)-*DP in the non-adaptive setting under continual observation* if $\mathcal{A}_{\mathcal{M}}$ is (ε, δ)-DP in the batch model.

We borrow the definition of accuracy from [JRSS23], which bounds the error of a mechanism with respect to a target function f. The definition takes the maximum error over both time steps and the coordinates of the output of f. (Although most of the functions we approximate return a single real value, the degree histogram $f_{\mathsf{degree\text{-}hist}}$ returns a vector at each step and requires the extra generality.)

Definition 2.9 (Accuracy of a mechanism)**.** Given a set of allowable streams $\mathcal{S} \subseteq \mathcal{X}^*$, a mechanism $\mathcal{M}$ is (α, T)-*accurate with respect to* $\mathcal{S}$ for a function $f : \mathcal{X}^* \to \mathbb{R}^k$ if, for all fixed (i.e., non-adaptively chosen) input streams $S = (S_1, \ldots, S_T) \in \mathcal{S}$, the maximum ℓ_∞ error over the outputs $a_1, \ldots, a_T$ of mechanism $\mathcal{M}$ is bounded by α with probability at least 0.99; that is,

$$\Pr_{\text{coins of } \mathcal{M}}\left[\max_{t\in[T]} \left\| f\left(S_{[t]}\right) - a_t \right\|_\infty \le \alpha\right] \ge 0.99.$$

If the indistinguishability property of differential privacy holds conditioned on the promise that both node-neighbors (respectively, edge-neighbors) lie in the set of graph streams with maximum degree at most D, we say that the algorithm offers D-*restricted* (ε, δ)-*node-DP* (respectively, D-*restricted* (ε, δ)-*edge-DP*). This is the notion of privacy explored by all prior work on node-private graph statistics under continual observation [FHO21; SLM+18].

Definition 2.10 (D-restricted DP)**.** A randomized algorithm $\mathcal{M} : \mathcal{G} \to \mathcal{Y}$ is D-*restricted* (ε, δ)-*node-DP* (respectively, *edge-DP*) if Definition 2.7 holds when restricted to the set of node-neighboring (respectively, edge-neighboring) graph streams with maximum degree at most D.

Likewise, $\mathcal{M}$ is D-*restricted* (ε, δ)-*node-DP under continual observation* (respectively, *edge-DP*) if Definition 2.8 holds when restricted to the set of node-neighboring (respectively, edge-neighboring) graph streams with maximum degree at most D.

3. Stable and Time-Aware Projections

In this section we present two *time-aware projection* algorithms Π_D^{BBDS} and Π_D^{DLL}, and prove Theorem 3.3 that presents their robust stability guarantees when run on (D, ℓ)-bounded graph streams. The two projection algorithms follow very similar strategies at a high level and are presented together in Algorithm 1. Both take as input a graph stream S of length $T \in \mathbb{N}$ and some user-specified value $D \in \mathbb{N}$ and return a projected graph stream with maximum degree at most D. If the graph stream S is already D-bounded, then both algorithms output it unchanged. At each time step, both algorithms greedily choose and output some subset of the arriving edges to include in the projection.

Algorithm 1 takes parameter c, called the *inclusion criterion*, that determines which of the two projections it executes. Let Π_D^{BBDS} denote Algorithm 1 with inclusion criterion $\mathsf{c} = \textit{original}$ and let Π_D^{DLL} denote the version with $\mathsf{c} = \textit{projected}$. (We use the author initials of [BBDS13; DLL16] to denote the algorithms inspired by their respective projections.)

The two algorithms differ from each other in terms of how they decide whether an edge should be added to the projection so far. The first algorithm Π_D^{BBDS} adds edge $e = \{u, v\}$ if the degree of both end points u and v is less than D in the *original* graph stream so far (i.e., S

Algorithm 1 Π_D for time-aware graph projection by edge addition.

Input: Graph stream $(S_1, \ldots, S_T) = S \in \mathcal{S}^T$, time horizon $T \in \mathbb{N}$, degree bound $D \in \mathbb{N}$, and inclusion criterion $\mathsf{c} \in \{original, projected\}$.
▷ $\mathsf{c} = original$ yields BBDS-based projection Π_D^{BBDS}
▷ $\mathsf{c} = projected$ yields DLL-based projection Π_D^{DLL}
Output: Graph stream $(S_1^*, \ldots, S_T^*) = S^* \in \mathcal{S}^T$

for $t = 1$ to T **do**
  Parse S_t as $(\partial V_t, \partial E_t)$
  for v in ∂V_t **do** $d(v) = 0$
  $\partial E_t^{proj} = \emptyset$
  for $e = \{u, v\}$ in ∂E_t, in consistent order, **do**
    add_edge $\leftarrow (d(u) < D) \wedge (d(v) < D)$
    if add_edge **then**
      set $\partial E_t^{proj} = \partial E_t^{proj} \cup \{e\}$
        ▷ add edge $e = \{u, v\}$ to the projection
    else ignore $e = \{u, v\}$
    if $\mathsf{c} = original$ or add_edge **then**
      $d(u)$ += 1, $d(v)$ += 1
        ▷ increment degree counters for u, v
  Output $S_t^* = (\partial V_t, \partial E_t^{proj})$

restricted to all edges considered before e). The second algorithm Π_D^{DLL} adds edge $e = \{u, v\}$ if nodes u and v both have degree less than D in the *projection* so far (i.e. $\Pi_D(S)$ restricted to all edges considered before e).

Consistent Ordering. When multiple edges arrive in a time step, the projections must decide on the order in which to consider these edges. While the exact ordering does not matter, we assume a *consistent ordering* of the edges in the input graph stream. Consistency means that any pair of edges in neighboring graph streams should be considered for addition to the projection in the same relative order. A similar ordering assumption is made by [BBDS13; DLL16].

A simple implementation of such an ordering assumes that each node u has a unique string identifier id_u—a user name, for example—and orders edges according to their endpoints (so (u, v) gets mapped to $(\mathsf{id}_u, \mathsf{id}_v)$, where $\mathsf{id}_u < \mathsf{id}_v$ and pairs are ordered lexicographically).

Since both projections process edges at the time they arrive, they end up considering edges according to a *time-aware version* of the ordering: edges end up being considered in the lexicographic order given by the triples $(t, \mathsf{id}_u, \mathsf{id}_v)$, where t is the edge's arrival time.

Ordering the edges uniformly randomly within each time step would also suffice since one can couple the random orderings on two neighboring streams so they are consistent with each other. We omit a proof of this, and assume lexicographic ordering in the rest of this manuscript.

Remark 3.1 (Running Algorithm 1 on static graphs)**.** Algorithm 1 can also take a (static, not streamed) graph as input by interpreting the graph as a length-1 graph stream, where the first element of the graph stream is equal to the graph itself.

3.1. Stability of the Time-Aware Projection Algorithms

Our analysis of the projection algorithms differs significantly from the batch-model analyses of [BBDS13; DLL16]. First, we consider the stability of the entire projected sequence, and not a single graph. Second, Blocki et al. [BBDS13] consider only the edge-to-edge stability of their projection algorithm, while Day et al. [DLL16] only analyze the stability with respect to a particular function of the projected graph (namely, its degree distribution). We analyze several stronger notions of stability for the entire sequence produced by our projections. All but one of these stability guarantees hold for streams that are (D, ℓ)-bounded.

Definition 3.2 ((D, ℓ)-bounded)**.** We say that a graph G is (D, ℓ)-*bounded* if it has at most ℓ nodes of degree greater than D. Similarly, a graph stream S of length T is (D, ℓ)-*bounded through time* $t \in [T]$ if the flattened graph $\mathsf{flatten}(S_{[t]})$ is (D, ℓ)-bounded (i.e., has at most ℓ nodes of degree greater than D).

We now present our theorem on the stability of Algorithm 1. These stabilities are summarized in Table 2.

Theorem 3.3 (Stability of projections)**.** *Let $T \in \mathbb{N}$, $D \in \mathbb{N}$, $\ell \in \mathbb{N} \cup \{0\}$, and let $\Pi_D^{BBDS}, \Pi_D^{DLL}$ be Algorithm 1 with inclusion criterion $\mathsf{c} = original$ and $\mathsf{c} = projected$, respectively.*

1) ***(Edge-to-edge stability.)*** *If $S \simeq_{edge} S'$ are edge-neighboring graph streams of length T, then for all time steps $t \in [T]$, the edge distances between the projections through time t satisfy the following:*
   a) $d_{edge}\left(\Pi_D^{BBDS}(S)_{[t]}\ ,\ \Pi_D^{BBDS}(S')_{[t]}\right) \leq 3$.
   b) *If S, S' are (D, ℓ)-bounded through time t, then* $d_{edge}\left(\Pi_D^{DLL}(S)_{[t]}\ ,\ \Pi_D^{DLL}(S')_{[t]}\right) \leq 2\ell + 1$.
2) ***(Node-to-edge stability.)*** *If $S \simeq_{node} S'$ are node-neighboring graph streams of length T, then for all time steps $t \in [T]$, the edge distances between the projections through time t satisfy the following:*
   a) *If S, S' are (D, ℓ)-bounded through time t, then* $d_{edge}\left(\Pi_D^{BBDS}(S)_{[t]}\ ,\ \Pi_D^{BBDS}(S')_{[t]}\right) \leq D + \ell$.
   b) *If S, S' are (D, ℓ)-bounded through time t, then* $d_{edge}\left(\Pi_D^{DLL}(S)_{[t]}, \Pi_D^{DLL}(S')_{[t]}\right) \leq D + 2\ell\sqrt{\min\{D, \ell\}}$.

3) ***(Node-to-node stability.)*** *If $S \simeq_{node} S'$ are node-neighboring graph streams of length T, then for all time steps $t \in [T]$, the node distances between the projections through time t satisfy the following:*
   a) *If S, S' are (D, ℓ)-bounded through time t, then $d_{node}\left(\Pi_D^{BBDS}(S)_{[t]}\ ,\ \Pi_D^{BBDS}(S')_{[t]}\right) \leq 2\ell + 1$.*
   b) *If S, S' are (D, ℓ)-bounded through time t, then $d_{node}\left(\Pi_D^{DLL}(S)_{[t]}\ ,\ \Pi_D^{DLL}(S')_{[t]}\right) \leq 2\ell + 1$.*

Furthermore, the bounds above are all tight in the worst case, either exactly (bound 1(a)), up to an additive constant of 2 (bounds 2(a) and 3(a)), or up to multiplicative constants.

The proofs that the bounds are tight appear in the full version. Here, we focus on proving the upper bounds.

Proof Sketch for Stability of Algorithm 1 (Theorem 3.3). To analyze stability, we consider a pair of graphs streams S, S' that are either edge (part (1)) or node neighbors (parts (2) and (3)). Assume without loss of generality that S' is the larger of the two streams. When S' is larger by virtue of including an additional node, we use v^+ to denote the additional node in S'.

Our proofs of stability rely heavily on two observations. First, the greedy nature of Algorithm 1 ensures that once an edge is added to the projection of a graph stream, that edge will not be removed from the projection at any future time step. Moreover, an edge may only be added to the projection at the time it arrives—it will not be added to the projection at any later time. This greedy behavior simplifies the analysis dramatically. It allows us to reason only about the distances between the flattened projected graphs at time t, rather than about the entire projected sequence: an edge that appears at some time $t' < t$ in one projected sequence but not the other will still differ between the flattened graphs at time t. This fact is captured in the "Flattening Lemma" (Lemma 3.6), which we use throughout the remainder of the argument. (A different projection algorithm, for example one that recomputes a projection from scratch at each time step, would require us to more explicitly analyze the entire projected sequence.)

The other important observation for our analysis is the following. Consider two neighboring graphs, where one graph contains (at most) one additional node v^+. In the larger graph, if an edge is between two nodes of degree at most D and is not incident to the added node v^+, then it will be in both projections. In other words, only edges that do not satisfy this condition may differ between projections. This idea is captured in Lemma 3.7.

(Stability of Π_D^{BBDS}.) To prove edge-to-edge stability, we apply Lemma 3.6 and largely borrow the analysis of [BBDS13, Proof of Claim 13]. To prove node-to-node stability, we only need to consider the projections through times $t \in [T]$ for which the graph streams S and S' are both (D, ℓ)-bounded. We then use Lemma 3.6, in addition to the fact that only an edge with an endpoint node of degree greater than D in one of the original graphs may differ between projections of neighboring streams (Lemma 3.7) to see that all nodes and edges that differ between graph streams belong to a vertex cover of size at most $\ell+1$. Therefore, we can obtain $S_{[t]}$ from $S'_{[t]}$ by removing v^+ and changing the remaining ℓ nodes in the vertex cover.

The node-to-edge stability also applies only through times $t \in [T]$ for which the graph streams S and S' are both (D, ℓ)-bounded. Its proof requires more careful analysis of exactly how many edges incident to nodes of degree greater than D may change, in addition to using Lemmas 3.6 and 3.7. We first show that at most D edges incident to the added node v^+ may appear in the projection of S'; these edges cannot appear in S or its projection.

We next consider whether any of the other edges differ between projections. First, consider an added edge e^+ incident to v^+ and some "high-degree" node u (i.e., with degree greater than D in S'). The presence of e^+ may mean exactly one edge incident to u that was included in the projection of S will now be dropped, since u may already have degree D when that edge is considered for addition. Now, suppose instead that e^+ is incident to v^+ and some "low-degree" node (i.e., with degree greater than D in S'). None of the edges incident to u will be dropped from the projection due to the inclusion of e^+, because although the degree of u is now larger, it is still safely at or below the threshold of D. Of the remaining edges, then, only edges incident to high-degree nodes may change. Since there are at most ℓ nodes with degree greater than D, there are at most ℓ additional edges that differ between projections. Therefore, by combining this with the above observation that at most D edges incident to v^+ appear in the projection of S', we see that at most $D + \ell$ edges differ between projections through time t for node-neighboring graph streams.

(Stability of Π_D^{DLL}.) The analysis of this projection, especially its node-to-edge stability, is generally more complex. One exception is the proof of node-to-node stability, which follows from the same argument used to prove node-to-node stability for Π_D^{BBDS}. All bounds on the stability of this projection only apply through times $t \in [T]$ for which the graph streams S and S' are both (D, ℓ)-bounded (note that the edge-to-edge stability for Π_D^{BBDS} does not rely on this assumption).

The node-to-edge stability analysis of this algorithm is more involved, though it also relies on Lemma 3.6. We consider a pair of arbitrary node neighbors and leverage the greedy nature of our algorithm to iteratively construct

a *difference graph* that tracks which edges differ between the projections of each graph. We show that the edge distance between the projections of neighboring graphs is exactly the number of edges in the difference graph. If the difference graph on any two node neighbors were to form a DAG on $\ell+1$ nodes with at most k paths of length at most ℓ, then we would be able to bound its edge count by $2\ell\sqrt{k}$ and prove, by setting $k = \min\{D, \ell\}$, that the projections are at edge distance at most $2\ell\sqrt{\min\{D, \ell\}}$ (in the full version, we prove this upper bound on the number of edges in such a DAG).

In reality, we show that the difference graph is edge distance at most D from a DAG with this special structure. This introduces an additive D term in the edge distance between the projections (which is to be expected since the projections of two node neighbors could differ in D edges that are incident to the differing node). The "pruning" argument, which shows how to remove edges from the difference graph to obtain the special DAG, and the construction of the resulting DAG form the most involved part of our analysis.

The edge-to-edge stability of Π_D^{DLL} also uses Lemma 3.6. If both graphs have maximum degree at most D, Algorithm 1 acts as the identity, and the projections differ in at most one edge. For graphs with at most ℓ nodes of degree greater than D, we observe that all of the edges in the corresponding difference graph form two paths of length at most ℓ, where all edges in each path are incident either to nodes with degree greater than D in S' or to the node with the added edge. Since there are at most ℓ of these high-degree nodes, the path has length at most $2\ell + 1$.

Useful Lemmas for Proving Stability. As described in the proof sketch, Lemmas 3.6 and 3.7 are used for proving many of the stability statements in Theorem 3.3. Before presenting the proofs of stability, we present those lemmas, along with definitions for terms that are used in the lemmas and their proofs.

The first definition is motivated by the observation that, if we run the algorithm on edge- or node-neighboring graph streams, edges are considered for inclusion in the output stream S^* in the same relative order for both graph streams (i.e., edge e_1 is considered before edge e_2 in stream S if and only if e_1 is considered before e_2 in stream S').

Definition 3.4 (Projection stage of an edge)**.** Let $T \in \mathbb{N}$ be a time horizon, $D \in \mathbb{N}$ be a degree bound, S be a graph stream of length T, and let $\Pi_D \in \{\Pi_D^{\text{BBDS}}, \Pi_D^{\text{DLL}}\}$ denote one of the variants of Algorithm 1. An edge e in S *is processed at projection stage* i of algorithm $\Pi_D(S)$ (denoted $\mathsf{ProjStage}(e) = i$) if e is the i^{th} edge to be considered by $\Pi_D(S)$.

In the proofs that follow, we are often interested in the value of a counter $d(\cdot)$ (see Line 3 of Algorithm 1) in relation to a projection stage; we say that a counter $d(u)$ has value j at projection stage i if, when the j^{th} edge is considered for inclusion in the projection, $d(u) = j$ on Line 6.

Multiple edges may arrive in a time step, so the *projection stage* of an edge is related to but distinct from the *arrival time* of the edge.

The second definition comes from the following observation. Consider a node u and counter $d(u)$ in Algorithm 1. If an edge e incident to u is processed at a projection stage where Algorithm 1 has $d(u) \geq D$, then e will not be included in the output stream. Since no more edges incident to u will be included, node u can be thought of as being *saturated*. Definition 3.5 allows us to talk about the order in which this saturation occurs.

Definition 3.5 (Saturation stage of a node)**.** Let $T \in \mathbb{N}$ be a time horizon, $D \in \mathbb{N}$ be a degree bound, S be a graph stream of length T, and let $\Pi_D \in \{\Pi_D^{\text{BBDS}}, \Pi_D^{\text{DLL}}\}$ denote one of the variants of Algorithm 1. A node u in S has *saturation stage* b (denoted $\mathsf{SatStage}_S(u) = b$) if b is the first projection stage such that $d(u) \geq D$ by the end of Line 12 in $\Pi_D(S)$. We define $\mathsf{SatStage}_S(u) = \infty$ if there is no b for which the described condition holds.

Lemma 3.6 (Flattening Lemma)**.** *Let* $\Pi_D \in \{\Pi_D^{BBDS}, \Pi_D^{DLL}\}$ *denote one of the variants of Algorithm 1. For every edge- or node-neighboring pair of graph streams* S *and* S' *of length* T*, the edge- and node-distance between the* projected streams *through time* t *is the same as the edge- and node-distance between the* flattened graphs *through time* $t \in [T]$*:*

$$d_{edge}\big(\Pi_D(S)_{[t]}, \Pi_D(S')_{[t]}\big) = d_{edge}\big(\mathsf{flatten}\big(\Pi_D(S)_{[t]}\big), \mathsf{flatten}\big(\Pi_D(S')_{[t]}\big)\big) \quad (1)$$

and

$$d_{node}\big(\Pi_D(S)_{[t]}, \Pi_D(S')_{[t]}\big) = d_{node}\big(\mathsf{flatten}\big(\Pi_D(S)_{[t]}\big), \mathsf{flatten}\big(\Pi_D(S')_{[t]}\big)\big) \quad (2)$$

Lemma 3.7 (Edges between low-degree nodes remain in both projections)**.** *Let* $\Pi_D \in \{\Pi_D^{BBDS}, \Pi_D^{DLL}\}$ *denote one of the variants of Algorithm 1. Consider a pair of edge- or node-neighboring graph streams* S, S' *of length* T*, where* S' *contains (at most) one additional node as compared to* S*. For all edges* $e = \{u, v\}$ *that arrive in* S *at (or before) time step* $t \in [T]$*, if* u *and* v *have degree at most* D *in* $S'_{[t]}$*, then* e *is in both* $\Pi_D(S)$ *and* $\Pi_D(S')$*.*

Lemma 3.7 has an important consequence: if U is the set of nodes that have degree more than D in $S'_{[t]}$ (where S' is the larger of two neighboring graph

streams), then $U \cup \{v_+\}$ forms a vertex cover for the edges that differ between the projections $\Pi_D(S)_{[t]}$ and $\Pi_D(S')_{[t]}$. We use this in the proof of Theorem 3.3 in a few ways. Most directly, if we have an upper bound of $\ell + 1$ on the size of U, then we immediately obtain an upper bound of $2\ell + 1$ on the node distance between the projections $\Pi_D(S)_{[t]}$ and $\Pi_D(S')_{[t]}$, which gives us the proof of node-to-node stability. It is also the first step in the proofs of other stability statements.

3.2. Proof of Node-to-Edge Stability for Π_D^{BBDS}

Here we prove item (2a) of Theorem 3.3, which we repeat below for convenience. The proof uses Lemmas 3.6 and 3.7, though it requires a more careful analysis of the flattened graphs than the proofs of items (1a) and (3) of Theorem 3.3.

Theorem 3.8 (Item (2a) of Theorem 3.3)**.** *Let $T \in \mathbb{N}$, $D \in \mathbb{N}$, $\ell \in \mathbb{N} \cup \{0\}$, and let Π_D^{BBDS} be Algorithm 1 with inclusion criterion $\mathsf{c} = original$. If $S \simeq_{node} S'$ are node-neighboring graph streams of length T, then for all time steps $t \in [T]$ such that S and S' are (D, ℓ)-bounded through time t, the edge distances between the projections through time t satisfy*

$$d_{edge}\left(\Pi_D^{BBDS}(S)_{[t]}\ ,\ \Pi_D^{BBDS}(S')_{[t]}\right) \leq D + \ell.$$

Proof of Theorem 3.8. Without loss of generality, let S' be the larger graph stream—that is, it contains an additional node v^+ and associated edges, which we represent with the set E^+. To simplify notation, let $F_{[t]}, F'_{[t]}$ denote $\mathsf{flatten}\left(\Pi_D^{\text{BBDS}}(S)_{[t]}\right)$ and $\mathsf{flatten}\left(\Pi_D^{\text{BBDS}}(S')_{[t]}\right)$ respectively. Note that we only care to bound the node distance between projections for times $t \in [T]$ where $S_{[t]}, S'_{[t]}$ are (D, ℓ)-bounded. By Lemma 3.6, we only need to bound $d_{edge}(F_{[t]}, F'_{[t]})$.

By Lemma 3.7, only edges incident (1) to v^+ or (2) to nodes with degree greater than D in $S'_{[t]}$ will differ between flattened graphs $F_{[t]}$ and $F'_{[t]}$. We now count the number of edges in these categories that differ between $F_{[t]}$ and $F'_{[t]}$.

We first bound the number of edges in (1). There are at most D edges incident to v^+ in $F'_{[t]}$ because $F'_{[t]}$ has maximum degree at most D, and none of these edges show up in $F_{[t]}$ since v^+ is not in $F_{[t]}$. Therefore, there are at most D edges in category (1).

We next bound the number of edges in (2). Each flattened graph $F_{[t]}$ and $F'_{[t]}$ contains at most $D \cdot \ell$ edges incident to nodes with degree greater than D in $S'_{[t]}$. However, we show that many of these edges will be the same in both flattened graphs. Let U denote the set of nodes, excluding v^+, that both have degree greater than D in $S'_{[t]}$ and are incident to edges in E^+. Consider an edge $e = \{u, v\}$, where neither u nor v is in U. The added edges in E^+ will not affect the values of $d(u)$ or $d(v)$, so either e will be in both $F_{[t]}$ and $F'_{[t]}$, or it will be in neither flattened graph.

Now consider edges e' incident to nodes in U. All edges in E^+ are of the form $e^+ = \{v^+, w\}$. If e^+ is processed at a projection stage where $d(w) \geq D$ in Π_D^{BBDS} on S', then e^+ will not cause any edges e' to differ between $F_{[t]}$ and $F'_{[t]}$. However, if e^+ is processed prior to this projection stage, then there may be one edge e_w incident to w that appears in $F_{[t]}$ but does not appear in $F'_{[t]}$, due to having $d(w) = D$ at the projection stage when e_w is processed instead of $d(w) = D - 1 < D$ (as is the case when running Π_D^{BBDS} on S). Note that e^+ will not affect the inclusion of other edges in the output stream: any edge incident to w that is processed at a projection stage after e_w will appear in neither flattened graph, and any edge that is processed at a projection stage prior to e_w will have $d(w) < D$ when considered for inclusion in both output streams.

We see that each edge $e = \{v^+, w\}$ in E^+ causes at most one edge incident to w to differ between projections (in particular, to appear in $F_{[t]}$ but not appear in $F'_{[t]}$). Therefore, if we bound $|E^+|$, we can bound the number of edges in category (2). Since all edges in E^+ are incident to v^+ and a node in U, there are at most $|U|$ edges in E^+. By the fact that the streams $S_{[t]}$ and $S'_{[t]}$ are (D, ℓ)-bounded, there are at most ℓ nodes in U. Therefore, the number of edges in category (2) is at most ℓ.

Categories (1) and (2) contain a total of at most $D + \ell$ edges, so at most $D + \ell$ edges differ between $F_{[t]}$ and $F'_{[t]}$, which is what we wanted to show. □

4. From D-restricted Privacy to Node Privacy

In this section, we present our general transformation from restricted edge- and node-DP algorithms to node-DP algorithms (Algorithm 3). As input, Algorithm 3 takes several user-specified parameters $(\varepsilon_{\mathsf{Test}}, \beta_{\mathsf{Test}}, \beta, D, T)$, a length-$T$ graph stream S of arbitrary degree, and black-box access to a base algorithm. At every time step, Algorithm 3 returns either the result of running one more step of the base algorithm on the projection of the graph stream, or a special symbol $\perp$ denoting failure.

The following theorem encapsulates its privacy and accuracy properties, and its efficiency. Roughly, the overall algorithm is private for all graph streams as long as the base algorithm satisfies either edge or node variants of D-restricted DP. It is accurate on graph streams through all time steps that satisfy the assumed

degree bound, and adds only linear overhead to the runtime of the base algorithm.

Theorem 4.1 (Privacy for all graph streams and restricted accuracy)**.** *Consider running Algorithm 3 with parameters* $\varepsilon_{\mathsf{Test}}, \beta_{\mathsf{Test}}, \beta, D, T$*, and let* $\ell = \left\lceil 8 \ln\left(\frac{T}{\beta\beta_{\mathsf{Test}}}\right)/\varepsilon_{\mathsf{Test}}\right\rceil$ *and* $D' = D+\ell$ *as in lines 2 and 3.*

1) ***(Node privacy from restricted edge privacy.)*** *Suppose* $\mathsf{RestrictedPrivAlg}_{D'}$ *satisfies* D'*-restricted* (ε', δ')***-edge****-DP under continual observation for graph streams of length* T*. Then Algorithm 3 satisfies (unrestricted)* (ε, δ)***-node****-DP under continual observation for graph streams of length* T*, with*

$$\begin{aligned}\varepsilon &= \varepsilon_{\mathsf{Test}} + \varepsilon' \cdot (D' + \ell) \quad \textit{and}\\ \delta &= (1 + e^{\varepsilon_{\mathsf{Test}}})e^{\varepsilon}\beta_{\mathsf{Test}} + \delta' \cdot e^{\varepsilon' \cdot (D'+\ell)} \cdot (D' + \ell).\end{aligned}$$

In particular, for $\varepsilon \leq 1$ *and* $T \geq 2$*, it suffices to set* $\varepsilon_{\mathsf{Test}} = \varepsilon/2$ *and* $\beta_{\mathsf{Test}} = \delta/30$*, and*

$$\varepsilon' = \Theta\left(\frac{\varepsilon}{B}\right) \quad \textit{and} \quad \delta' = \Theta\left(\frac{\delta}{B}\right), \quad \textit{where} \quad B = D + \frac{\log(T/(\beta\delta))}{\varepsilon}.$$

2) ***(Node privacy from restricted node privacy.)*** *Suppose* $\mathsf{RestrictedPrivAlg}_{D'}$ *satisfies* D'*-restricted* (ε', δ')***-node****-DP under continual observation for graph streams of length* T*. Then Algorithm 3 satisfies (unrestricted)* (ε, δ)***-node****-DP under continual observation for graph streams of length* T*, with*

$$\begin{aligned}\varepsilon &= \varepsilon_{\mathsf{Test}} + \varepsilon' \cdot (2\ell + 1) \quad \textit{and}\\ \delta &= (1 + e^{\varepsilon_{\mathsf{Test}}})e^{\varepsilon}\beta_{\mathsf{Test}} + \delta' \cdot e^{\varepsilon' \cdot (2\ell+1)} \cdot (2\ell + 1).\end{aligned}$$

In particular, for $\varepsilon \leq 1$ *and* $T \geq 2$*, it suffices to set* $\varepsilon_{\mathsf{Test}} = \varepsilon/2$ *and* $\beta_{\mathsf{Test}} = \delta/30$*, and*

$$\varepsilon' = \Theta\left(\frac{\varepsilon}{B}\right) \quad \textit{and} \quad \delta' = \Theta\left(\frac{\delta}{B}\right), \quad \textit{where} \quad B = \frac{\log(T/(\beta\delta))}{\varepsilon}.$$

3) ***(Accuracy.)*** *If the input graph stream* S *is* $(D, 0)$*-bounded through time step* t'*, then the output from* $\mathsf{RestrictedPrivAlg}_{D'}$ *is released at all time steps* $t \leq t'$ *with probability at least* $1 - \beta$*.*

4) ***(Time and space complexity.)*** *Algorithm 3 adds linear overhead to the time and space complexity of* $\mathsf{RestrictedPrivAlg}_{D'}$*. More formally, let* $R_{[t]}$ *and* $S_{[t]}$ *be the runtime and space complexity of* $\mathsf{RestrictedPrivAlg}_{D'}$ *through* t *time steps, and let* $n_{[t]}$ *and* $m_{[t]}$ *be the number of nodes and edges in the graph stream through time* t*. The total time complexity of Algorithm 3 through time* $t \in [T]$ *is* $R_{[t]} + O(n_{[t]} + m_{[t]} + t)$*, and the total space complexity through time* t *is* $S_{[t]} + O(n_{[t]})$*.*

The basic idea of the algorithm follows the Propose-Test-Release (PTR) framework of [DL09]: we use the sparse vector technique [DNR+09; RR10; HR10] to continually check that the conditions of Theorem 3.3 are met. As long as they are, we can safely run the base algorithm with parameters scaled according to the edge (or node) sensitivity of the projection so that, by group privacy, its outputs are (ε, δ)-indistinguishable on all pairs of node-neighboring inputs, satisfying (ε, δ)-node-DP.

4.1. Testing for Bad Graphs

To use the sparse vector technique, we need a stream of queries with (node) sensitivity 1. Below, we define a function DistToGraph with node-sensitivity 1 that returns the minimum, over all graphs, of the node distance between the input graph and a graph with at least ℓ nodes of degree greater than D. In other words, it tells how close the input graph is to a graph that is *not* $(D, \ell - 1)$-bounded. This function can be used to make such a stream of node-sensitivity 1 queries for continually checking whether the conditions of Theorem 3.3 are satisfied.

Definition 4.2 ($\mathsf{DistToGraph}_{D,\ell}$)**.** Let $\mathsf{DistToGraph}_{D,\ell} : \mathcal{G} \to \mathbb{N} \cup \{0\}$ return the minimum node distance between the input graph and a graph with at least ℓ nodes of degree greater than D.

We now present some properties of DistToGraph, which we will use in the proof of privacy for the black-box framework described in Algorithm 3.

Lemma 4.3 (Properties of DistToGraph)**.** $\mathsf{DistToGraph}_{D,\ell}$ *has the following properties:*

1) $\mathsf{DistToGraph}_{D,\ell}$ *has node-sensitivity 1.*
2) *For an input graph* G *with* $|V|$ *nodes and* $|E|$ *edges,* $\mathsf{DistToGraph}_{D,\ell}$ *can be computed in time* $O(|V| + |E|)$*.*
   Furthermore, given a graph stream S*, one can determine the sequence of distances for all prefixes of the stream, online. With each node or edge arrival, the distance can be updated in constant time.*
3) *Let* $G \in \mathcal{G}$ *be a* (D, ℓ)*-bounded graph (i.e., with at most* ℓ *nodes of degree greater than* D*). Then* $\mathsf{DistToGraph}_{D+k,\ell+k}(G) \geq k$ *for all* $k \in \mathbb{N} \cup \{0\}$*.*

We omit a full proof; however, the following algorithm computes DistToGraph and can be implemented in linear time. Given a graph $G \in \mathcal{G}$, first check if G is (D, ℓ)-bounded. While that condition is not satisfied, add a node to G with edges incident to all existing nodes (and check (D, ℓ)-boundedness again). $\mathsf{DistToGraph}_{D,\ell}(G)$

equals the number of nodes that have been added to G when the check fails.

4.2. PTR in the Continual Release Setting

The high-level idea of our general transformation is to use a novel, online variant of the Propose-Test-Release framework (PTR) of [DL09]. To our knowledge, PTR has not been used previously for designing algorithms in the continual release setting. In this section, we show that PTR can, in fact, be applied in the continual release setting when the algorithm checking the safety condition is itself private under continual observation, as is the sparse vector algorithm.

Theorem 4.4 (Privacy of PTR under Continual Observation)**.** *Let* $\mathsf{Base} : \mathcal{X}^T \to \mathcal{Y}^T$ *be a streaming algorithm and* $\mathcal{Z} \subseteq \mathcal{X}^T$ *a set of streams such that, for all neighbors* $S, S' \in \mathcal{Z}$, $\mathsf{Base}(S) \approx_{\varepsilon_{\mathsf{Base}}, \delta_{\mathsf{Base}}} \mathsf{Base}(S')$.

Let $\mathsf{Test} : \mathcal{X}^T \to \{\perp, \top\}$ *be* $(\varepsilon_{\mathsf{Test}}, \delta_{\mathsf{Test}})$*-DP under continual observation such that for every stream* S*, if there is a time* t *such that* $S_{[t]}$ *does not equal the* t*-element prefix of any item in* $\mathcal{Z}$*, then* $\mathsf{Test}(S)$ *outputs* $\perp$ *w.p. at least* $1 - \beta_{\mathsf{Test}}$ *at or before time* t*.*

If Algorithm 2 is initialized with Test *and* Base*, it will satisfy* (ε, δ)*-DP under continual observation on all input streams* $x \in \mathcal{X}^T$*, for*

$$\varepsilon = \varepsilon_{\mathsf{Base}} + \varepsilon_{\mathsf{Test}} \quad \textit{and}$$

$$\delta = \delta_{\mathsf{Base}} + (1 + e^{\varepsilon_{\mathsf{Base}} + \varepsilon_{\mathsf{Test}}})\delta_{\mathsf{Test}} + (1 + e^{\varepsilon_{\mathsf{Test}}})e^{\varepsilon_{\mathsf{Base}} + \varepsilon_{\mathsf{Test}}}\beta_{\mathsf{Test}}.$$

Algorithm 2 Algorithm PTR for propose-test-release under continual observation.

Input: Stream $x \in \mathcal{X}^T$, streaming algorithms $\mathsf{Test} : \mathcal{X}^T \to \{\perp, \top\}^T$ and $\mathsf{Base} : \mathcal{X}^T \to \mathcal{Y}^T$.
Output: Stream in $(\{\perp, \top\} \times \{\mathcal{Y}, \perp\})^T$.

passed $\leftarrow$ *True*
$b_0 \leftarrow$ initial state of Base
$s_0 \leftarrow$ initial state of Test
for all $t \in [T]$ **do**
  $(\mathsf{verdict}, s_t) \leftarrow \mathsf{Test}(x_t; s_{t-1})$
    ▷ Send x_t to Test, and set verdict to be its output
  Output verdict
  if verdict $= \perp$ **then** passed $\leftarrow$ *False*
  if passed $=$ *True* **then**
    $(y_t, b_t) \leftarrow \mathsf{Base}(x_t; b_{t-1})$
      ▷ Send x_t to Base and output the result
    Output y_t
  else output $\perp$

While it is perhaps unsurprising that PTR applies to the continual release setting, the proof of Theorem 4.4 does not follow immediately from the standard PTR analysis: the standard analysis is binary (either release the output of the base algorithm, or don't), while the version we need releases a dynamically chosen prefix of the base algorithm's output.

4.3. Accuracy and Privacy of Algorithm 3

In Algorithm 3, we present our method for obtaining node-DP algorithms from restricted edge-DP and restricted node-DP algorithms.

Our algorithm works as follows. Where we set $D' = D + \ell$ and $\ell \approx \frac{\log(T/\delta)}{\varepsilon}$, we initialize PTR (Algorithm 2) with (1) a Base algorithm that offers indistinguishability on neighbors from the set of (D', ℓ)-bounded graphs and (2) a Test algorithm that uses sparse vector to ensure that $\mathsf{DistToGraph}_{D',\ell}$ is non-negative (i.e., that the graph stream is (D', ℓ)-bounded), where ℓ is an additive slack term to account for the error of the sparse vector technique. Specifically, Base is the composition of $\mathsf{RestrictedPrivAlg}_{D'}$ with $\Pi^{\mathrm{BBDS}}_{D'}$, where $\mathsf{RestrictedPrivAlg}_{D'}$ satisfies D'-restricted edge- or node-DP.

If Test succeeds, the projection will be stable with high probability, so we can safely release the result of running Base on the projected graph; if Test fails, the symbol $\perp$ is released. Algorithm 3 post-processes the outputs from PTR, so it inherits the privacy properties of PTR. The privacy properties of Algorithm 3, stated in Theorem 4.1, follow from Theorem 4.4 and are proven in the full version.

5. Optimal Algorithms for f_{edges}, $f_{\mathsf{triangles}}$, f_{CC}, $f_{\mathsf{k\text{-}stars}}$

We now use our transformation in Algorithm 3 to convert restricted edge- and node-DP algorithms for several fundamental problems into node-DP algorithms that achieve the same asymptotic error as the analogous restricted node-DP algorithms given by [SLM+18; FHO21], up to lower-order terms. Table 1 shows the additive error for privately counting edges (f_{edges}), triangles ($f_{\mathsf{triangles}}$), k-stars ($f_{\mathsf{k\text{-}stars}}$), and connected components (f_{CC}), and privately releasing degree histograms[6] ($f_{\mathsf{degree\text{-}hist}}$) of the input graph stream. Moreover, for f_{edges}, $f_{\mathsf{triangles}}$, $f_{\mathsf{k\text{-}stars}}$, and f_{CC}, the accuracy of our algorithms is asymptotically optimal, up to lower order terms and polylogarithmic factors.

For f_{edges}, $f_{\mathsf{triangles}}$, $f_{\mathsf{k\text{-}stars}}$, and $f_{\mathsf{degree\text{-}hist}}$, the errors for our transformation follow from substituting the ε' term from Theorem 4.1 into the error bounds for the

6. The degree histogram $f_{\mathsf{degree\text{-}hist}}(G)$ for a graph G with maximum degree at most D is the $(D+1)$-element vector $(a_0, \ldots, a_D) \in \mathbb{R}^{D+1}$, where a_i is the number of nodes with degree i in G.

Algorithm 3 BBRestrictedToNodePriv for transforming restricted-DP algorithms to node-DP algorithms.

Input: Privacy params $\varepsilon_{\mathsf{Test}} > 0$, $\beta_{\mathsf{Test}} \in (0,1]$; accuracy param $\beta \in (0,1]$; degree bound $D \in \mathbb{N}$; time horizon $T \in \mathbb{N}$; graph stream $S \in \mathcal{S}^T$; alg RestrictedPrivAlg

Output: A stream, where each term is $\perp$ or an estimate from RestrictedPrivAlg.

$\tau = -8\ln(1/\beta_{\mathsf{Test}})/\varepsilon_{\mathsf{Test}}$

$\ell = \lceil 8\ln(T/(\beta\beta_{\mathsf{Test}}))/\varepsilon_{\mathsf{Test}} \rceil$

$D' = D + \ell$

$\mathsf{Base} = \mathsf{RestrictedPrivAlg}_{D'} \circ \Pi^{\mathrm{BBDS}}_{D'}$

▷ $\Pi^{\mathrm{BBDS}}_{D'}$ is Algorithm 1 with $\mathsf{c} = \textit{original}$

for all $t \in [T]$ **do**

  $q_t(\cdot) = -1 \cdot \mathsf{DistToGraph}_{D',\ell}(\mathsf{flatten}(\ \cdot\ _{[t]}))$

$$\mathsf{Test} = \begin{cases} \mathsf{SVT} \text{ with privacy param } \varepsilon_{\mathsf{Test}}, \\ \text{thresh } \tau, \text{ queries } q_1, \ldots, q_T \end{cases}$$

$s_0 \leftarrow$ initial state for PTR

Initialize PTR with algorithms Test and Base

for all $t \in [T]$ **do**

  $(\mathsf{Test\text{-}val}, \mathsf{Base\text{-}val}, s_t) \leftarrow \mathsf{PTR}(S_t; s_{t-1})$

  Output Base-val

▷ Base-val will be $\perp$ once the test fails

restricted edge-DP algorithms of [FHO21] (algorithms with slightly worse lower-order terms follow from using restricted node-DP algorithms). The bound for f_{CC} follows from a new edge-DP algorithm based on the binary tree mechanism. This algorithm and the accompanying proof appear in the full version.

By item (1a) of Theorem 3.3, we can also immediately obtain $(\varepsilon, 0)$-edge-DP algorithms from restricted edge-private algorithms by using the projection Π^{BBDS}_{D} and running the corresponding restricted edge-private algorithm with $\varepsilon' = \varepsilon/3$ on the projection. This technique gives the first algorithms for $f_{\mathsf{triangles}}$, $f_{\mathsf{k\text{-}stars}}$, and f_{CC} that are edge-private under continual observation for all graphs and, for graph streams with maximum degree at most D, have asymptotically optimal accuracy (up to polylogarithmic factors).

The lower bounds in the table follow by reductions from a version of the $\Omega(\log T/\varepsilon)$ lower bound for binary counting from [DNPR10], modified by us for the approximate-DP setting. Proofs of these lower bounds appear in the full version.

6. Experiments

We implemented Algorithm 3 for the task of estimating the number of edges. We ran our algorithm on several synthetic graph streams (which we describe below) and compared the accuracy of our algorithm for counting edges to the accuracy of a direct application of batch model algorithms for counting edges with advanced composition. We used the implementation of the (standard) binary tree mechanism from [AP23].

The Python code for the algorithm and synthetic graph generation is on GitHub: https://github.com/cwagaman/time-aware-proj.

Parameters for node privacy. All of our experiments use $\varepsilon = 1$ and $\delta = 10^{-10}$ (about 2^{-33}), and are (ε, δ)-node-DP.

Input streams. We look at (1) *random graphs* with $n = 10^6$ nodes and $m = 2 \cdot 10^8$ edges (with the edges drawn uniformly without replacement from the set of possible edges), and (2) *two-block graphs* with $n = 10^6$ nodes and $m = 2 \cdot 10^8$ edges (with the edges drawn uniformly without replacement from the set of possible edges), except for 5,000 randomly selected nodes that have degree 10,000 (with these edges drawn uniformly at random from the set of edges incident to the randomly chosen nodes). In both cases, the average degree in the graph is $\frac{2m}{n} = 400$, but in the two-block cases there are nodes with 25 times the average degree.

In both cases, we consider a stream with $T = 10^6$ time steps. A uniformly random subset of $\frac{m}{T} = 200$ edges arrives at each time step.

Degree cutoffs. Recall that the algorithms we consider require an analyst-specific degree cutoff D. We conduct three experiments overall. For the random graphs (with maximum degree about 400), we conduct experiments with $D = 400$ and $D = 1{,}000$—these reflect settings where the maximum degree estimate is tight or conservatively large. For the two-block stream with maximum degree 10,000, we use $D = 15{,}000$, which corresponds to a slight overestimate of the maximum degree.

One could get around the need to specify D by running several parallel copies of the algorithm with different degree cutoffs (say, using powers of 2 up to some reasonable limit, and choosing the output corresponding to the smallest value of D for which the PTR test has not yet rejected). We did not consider this approach in our experiments.

Results. We present results via two different types of plots (Fig. 1). The first type shows the actual edge count, the value reported by our algorithm, and the value reported by algorithms that follow from prior work. The second type shows the relative error of our algorithm and that of prior work. The denominator in the relative error is the current edge count, which increases linearly over time. We show these plots for all 10^6 time steps, along with a "zoomed-in" version of these plots for the first 50,000 time steps. These latter plots allow one to see the variability of the error over time.

Running time. As stated in Theorem 4.1, our algorithm requires $O(n+m)$ additional time (total) and $O(n)$ additional space on a stream with n nodes and m edges. On a laptop, our (unoptimized) implementation for edge counting runs in about $6 \cdot 10^{-5}$ seconds per edge on the random graph described above with $n = 10^6$ nodes and $m \approx 2 \cdot 10^8$ edges. For comparison, merely reading the graph and tracking the degree of each node (with the same machine and programming environment) takes $1.5 \cdot 10^{-5}$ seconds per edge. Compared to this elementary processing, our algorithm takes more time by a factor of 4; optimized implementations would presumably see a similarly small run time increase.

Discussion. Our experimental results, displayed in Figure 1, show natural regimes where our new algorithm offers much better accuracy than the batch-model baseline. Furthermore, in all three experiments, the relative error of our algorithm drops below 1 early in the stream (10,000 time steps for the random graphs, and 50,000 for the two-block model). The error of our algorithm largely reflects the asymptotic error bounds, showing that the advantage of our algorithm holds even for modest settings of parameters. Additionally, implementing our algorithm was straightforward, and it is lightweight and runs quickly, even without optimizations.

Our algorithm's error is driven by the specified degree cutoff D (assuming it produces any output at all, which requires roughly that D be not much smaller than the actual maximum degree). The experiments reflect this: the relative error is lower in the random graph stream, where the maximum and average degree are basically the same, than in the two-block graph. And the relative error is lower when the degree cutoff is close to the maximum degree ($D = 400$ as opposed to $D = 1{,}000$, for the random graph stream). Choosing the value of D automatically (as discussed under "Degree cutoffs", above) would help address overestimation of the maximum degree. However, variance among the degrees is a more fundamental obstacle, since the sensitivity of the edge count to the removal of any single vertex is equal to the maximum degree, but relative error compares to the total number of edges.

These experiments also reveal some limitations of our algorithm. Our algorithm's advantage over the baseline comes the fact that its error scales with $\operatorname{polylog} T$, instead of the $\sqrt{T}$ scaling that appears when composing the batch-model algorithms. As a result, T must be large to obtain a significant accuracy advantage. Our algorithm's advantage over the baseline also improves for larger values of D, since then the fixed, additive slack term from the PTR framework becomes less significant. Designing an algorithm with low additive error in practice for all ranges of D (including, say, with D a small constant) remains an open question.

Acknowledgements

The authors would like to thank the anonymous Oakland reviewers as well as colleagues—Ephraim Linder, Sofya Raskhodnikova, Satchit Sivakumar, Theresa Steiner, and Marika Swanberg—for many helpful comments. The authors were supported in part by NSF awards CCF-1763786 and CNS-2120667, Faculty Awards from Google and Apple, and Cooperative Agreement CB16ADR0160001 with the Census Bureau. The views expressed in this paper are those of the authors and not those of the U.S. Census Bureau or any other sponsor.

References

[AP23] J. D. Andersson and R. Pagh, "A smooth binary mechanism for efficient private continual observation," in *NeurIPS*, 2023.

[BBDS13] J. Blocki, A. Blum, A. Datta, and O. Sheffet, "Differentially private data analysis of social networks via restricted sensitivity," in *ITCS*, 2013. DOI: 10.1145/2422436.2422449.

[BCSZ18a] C. Borgs, J. T. Chayes, A. D. Smith, and I. Zadik, "Private algorithms can always be extended," *CoRR*, 2018.

[BCSZ18b] ——, "Revealing network structure, confidentially: Improved rates for node-private graphon estimation," in *FOCS*, 2018. DOI: 10.1109/FOCS.2018.00057.

[BS10] J. Badham and R. Stocker, "The impact of network clustering and assortativity on epidemic behaviour," *Theor. Popul. Biol.*, 2010.

[CD20] R. Cummings and D. Durfee, "Individual sensitivity preprocessing for data privacy," in *SODA*, 2020. DOI: 10.1137/1.9781611975994.32.

[CSS11] T.-H. H. Chan, E. Shi, and D. Song, "Private and continual release of statistics," *ACM Trans. Inf. Syst. Secur.*, 2011. DOI: 10.1145/2043621.2043626.

[CSS18] B. Chen, A. Shrivastava, and R. C. Steorts, "Unique entity estimation with application to the syrian conflict," *The Annals of Applied Statistics*, 2018. DOI: 10.1214/18-AOAS1163.

[CZ13] S. Chen and S. Zhou, "Recursive mechanism: Towards node differential privacy and unrestricted joins," in *SIGMOD*, 2013. DOI: 10.1145/2463676.2465304.

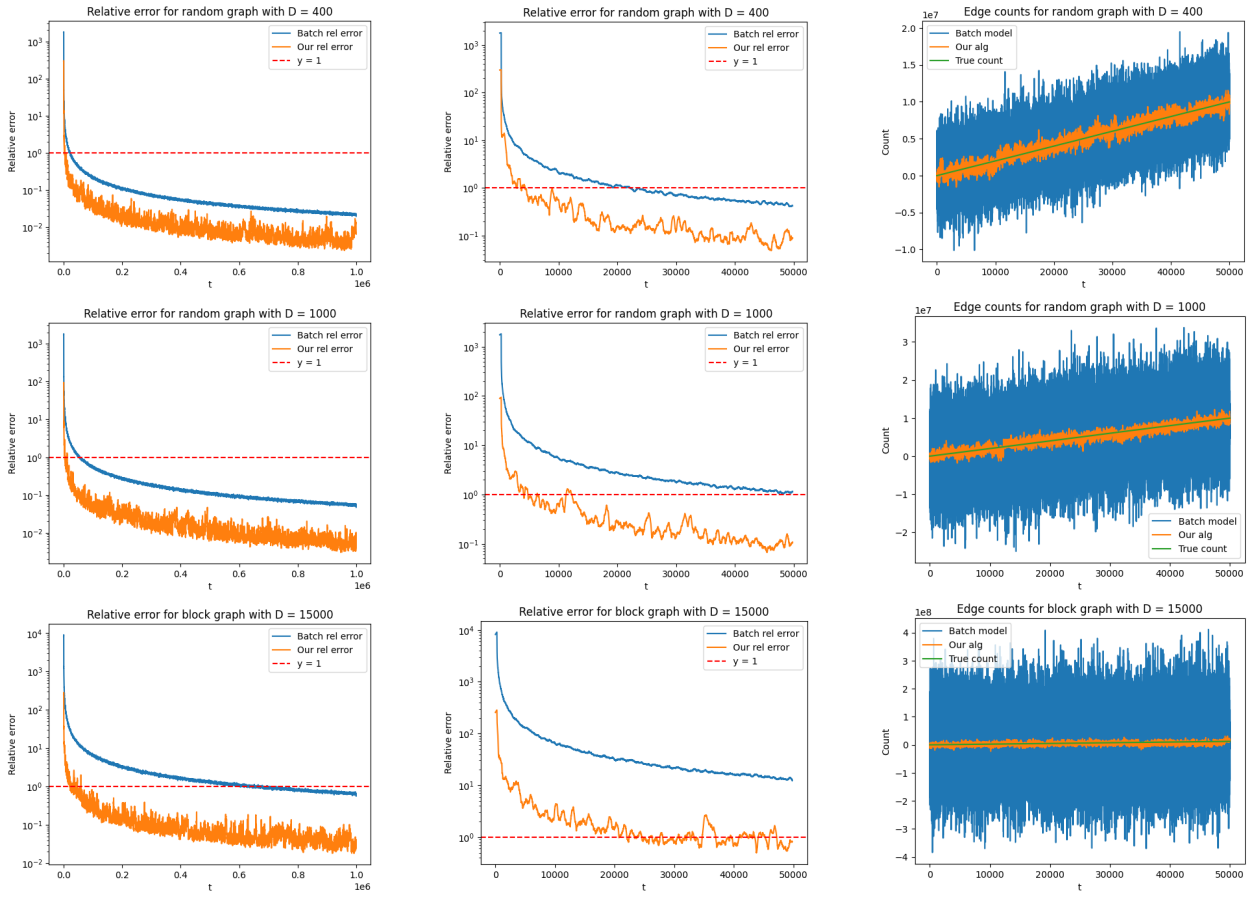

Figure 1. Results from one run of our algorithm (orange) and the batch-model baseline (blue) on three stream/cutoff pairs: a random graph stream with $D = 400$ (top row); a random graph stream with $D = 1{,}000$; middle row); a two-block graph stream with $D = 15{,}000$. For each graph, we provide three plots. Left: Relative error across all 10^6 time steps. Center: Relative error across first $50{,}000$ steps. Right: True edge counts (green) together with reported counts from each algorithm across first $50{,}000$ steps. The left and center plots include a red dashed line at relative error 1 for visual reference.

[DL09] C. Dwork and J. Lei, "Differential privacy and robust statistics," in *STOC*, 2009. DOI: 10.1145/1536414.1536466.

[DLL16] W.-Y. Day, N. Li, and M. Lyu, "Publishing graph degree distribution with node differential privacy," in *SIGMOD*, 2016. DOI: 10.1145/2882903.2926745.

[DMNS16] C. Dwork, F. McSherry, K. Nissim, and A. D. Smith, "Calibrating noise to sensitivity in private data analysis," *J. Priv. Confidentiality*, 2016. DOI: 10.29012/jpc.v7i3.405.

[DNPR10] C. Dwork, M. Naor, T. Pitassi, and G. N. Rothblum, "Differential privacy under continual observation," in *STOC*, 2010. DOI: 10.1145/1806689.1806787.

[DNR+09] C. Dwork, M. Naor, O. Reingold, G. N. Rothblum, and S. P. Vadhan, "On the complexity of differentially private data release: Efficient algorithms and hardness results," in *STOC*, 2009. DOI: 10.1145/1536414.1536467.

[DRV10] C. Dwork, G. N. Rothblum, and S. P. Vadhan, "Boosting and differential privacy," in *FOCS*, 2010. DOI: 10.1109/FOCS.2010.12.

[FHO21] H. Fichtenberger, M. Henzinger, and W. Ost, "Differentially private algorithms for graphs under continual observation," *CoRR*, 2021.

[Goo49] L. A. Goodman, "On the estimation of the number of classes in a population," *The Annals of Mathematical Statistics*, 1949.

[HLMJ09] M. Hay, C. Li, G. Miklau, and D. D. Jensen, "Accurate estimation of the degree distribution of private networks," in *ICDM, IEEE International Conference on Data Mining*, 2009. DOI: 10.1109/ICDM.2009.11.

[HR10] M. Hardt and G. N. Rothblum, "A multiplicative weights mechanism for privacy-preserving data analysis," in *FOCS*, 2010. DOI: 10.1109/FOCS.2010.85.

[JM09] C. Jernigan and B. F. T. Mistree, "Gaydar: Facebook Friendships Expose Sexual Orientation," *First Monday*, 2009.

[JRSS23] P. Jain, S. Raskhodnikova, S. Sivakumar, and A. D. Smith, "The price of differential privacy under continual observation," in *ICML*, 2023.

[JSW24] P. Jain, A. Smith, and C. Wagaman, "Time-aware projections: Truly node-private graph statistics under continual observation," *CoRR*, 2024. DOI: 10.48550/ARXIV.2403.04630.

[KNRS13] S. P. Kasiviswanathan, K. Nissim, S. Raskhodnikova, and A. D. Smith, "Analyzing graphs with node differential privacy," in *TCC*, 2013. DOI: 10.1007/978-3-642-36594-2_26.

[KRST23] I. Kalemaj, S. Raskhodnikova, A. D. Smith, and C. E. Tsourakakis, "Node-differentially private estimation of the number of connected components," in *PODS*, 2023. DOI: 10.1145/3584372.3588671.

[LSL17] M. Lyu, D. Su, and N. Li, "Understanding the sparse vector technique for differential privacy," *Proc. VLDB Endow.*, 2017. DOI: 10.14778/3055330.3055331.

[NRS07] K. Nissim, S. Raskhodnikova, and A. D. Smith, "Smooth sensitivity and sampling in private data analysis," in *STOC*, 2007. DOI: 10.1145/1250790.1250803.

[RR10] A. Roth and T. Roughgarden, "Interactive privacy via the median mechanism," in *STOC*, 2010. DOI: 10.1145/1806689.1806794.

[RS16a] S. Raskhodnikova and A. D. Smith, "Differentially private analysis of graphs," in *Encyclopedia of Algorithms*, 2016. DOI: 10.1007/978-1-4939-2864-4_549.

[RS16b] ——, "Lipschitz extensions for node-private graph statistics and the generalized exponential mechanism," in *FOCS*, 2016. DOI: 10.1109/FOCS.2016.60.

[SLM+18] S. Song, S. Little, S. Mehta, S. A. Vinterbo, and K. Chaudhuri, "Differentially private continual release of graph statistics," *CoRR*, 2018.

[Upa13] J. Upadhyay, "Random projections, graph sparsification, and differential privacy," in *ASIACRYPT (1)*, 2013. DOI: 10.1007/978-3-642-42033-7_15.

[UUA21] J. Upadhyay, S. Upadhyay, and R. Arora, "Differentially private analysis on graph streams," in *AISTATS*, 2021.

[Vad17] S. P. Vadhan, "The complexity of differential privacy," in *Tutorials on the Foundations of Cryptography*, 2017. DOI: 10.1007/978-3-319-57048-8_7.

[YJM+13] A. M. Young, A. B. Jonas, U. L. Mullins, D. S. Halgin, and J. R. Havens, "Network structure and the risk for HIV transmission among rural drug users," *AIDS Behav.*, 2013.

Appendix A.

A.1. Properties of Differential Privacy

The definition of differential privacy extends to groups of individuals. If an algorithm is DP for one individual, it also offers a (differently parameterized) privacy guarantee for a collection of individuals. We borrow the formulation of this property from [Vad17].

Lemma A.1 (DP offers group privacy [DMNS16])**.** *Let $\mathcal{M} : \mathcal{X} \to \mathcal{Y}$ be a randomized algorithm that is (ε, δ)-DP. Then, where $x, x' \in \mathcal{X}$ differ in the data of k individuals, $\mathcal{A}(x)$ and $\mathcal{A}(x')$ are $(k \cdot \varepsilon, k \cdot e^{k\varepsilon} \cdot \delta)$-indistinguishable. That is,*

$$\mathcal{A}(x) \approx_{k\cdot\varepsilon, k\cdot e^{k\varepsilon}\cdot\delta} \mathcal{A}(x').$$

This follows from a well-known "weak triangle inequality" for (ε, δ)-indistinguishability:

Lemma A.2 (Weak triangle inequality)**.** *For all $\varepsilon_1, \varepsilon_2, \delta_1, \delta_2 \geq 0$: If random variables A, B, C satisfy $A \approx_{\varepsilon_1,\delta_1} B$ and $B \approx_{\varepsilon_2,\delta_2} C$, then $A \approx_{\varepsilon',\delta'} C$ for $\varepsilon' = \varepsilon_1 + \varepsilon_2$ and $\delta' = \max(\delta_1 + e^{\varepsilon_1}\delta_2,\ \delta_2 + e^{\varepsilon_2}\delta_1) \leq e^{\varepsilon_2}\delta_1 + e^{\varepsilon_1}\delta_2$.*

Differential privacy is robust to post-processing.

Lemma A.3 (DP is robust to post-processing [DMNS16])**.** *Let $\mathcal{M} : \mathcal{X} \to \mathcal{Y}$ be a randomized algorithm that is (ε, δ)-DP. Let $f : \mathcal{Y} \to \mathcal{Z}$ be an arbitrary, randomized mapping. Then $f \circ \mathcal{M} : \mathcal{X} \to \mathcal{Z}$ is (ε, δ)-DP.*

A.2. Edge-DP under Continual Observation

We can use the edge-to-edge stability of Π_D^{BBDS} to transform algorithms that satisfy D-restricted edge-DP under continual observation that satisfy (pure or approximate) edge-DP under continual observation. Up to logarithmic factors, the algorithms we obtain are asymptotically optimal for $f_{\text{triangles}}, f_{\text{CC}}, f_{\text{k-stars}}$.

The errors for our edge-private algorithms follow from item (1a) of Theorem 3.3, which says that we can achieve $(\varepsilon, 0)$-edge-DP by running the algorithms of [FHO21] for $f_{\text{triangles}}, f_{\text{k-stars}}, f_{\text{degree-hist}}$, and our algorithm for f_{CC}, with privacy parameter $\varepsilon' = \varepsilon/3$. (Because item (1a) of Theorem 3.3 holds unconditionally, there is no need to use the PTR framework.) In contrast with previous work [FHO21; SLM+18], this allows us to offer privacy on all graph streams, and we maintain the accuracy guarantees of the restricted private algorithm (up to constant factors) for graph streams with maximum degree at most D.

Theorem A.4 (Accuracy of edge-private algorithms)**.** *Let $\varepsilon > 0$, $D \in \mathbb{N}$, $T \in \mathbb{N}$, and S be a length-T graph stream. There exist $(\varepsilon, 0)$-edge-DP algorithms for the following problems whose error is at most α, with probability 0.99, for all times steps $t \in [T]$ where $S_{[t]}$ has maximum degree at most D:*

1) $f_{\text{triangles}}$, $\alpha = O\left(\dfrac{D \log^{3/2} T}{\varepsilon}\right)$.

2) f_{CC}, $\alpha = O\left(\dfrac{\log^{3/2} T}{\varepsilon}\right)$.

3) $f_{\text{k-stars}}$, $\alpha = O\left(\dfrac{D^{k-1} \log^{3/2} T}{\varepsilon}\right)$.

4) $f_{\text{degree-hist}}$, $\alpha = \widetilde{O}\left(\dfrac{D \log^{3/2} T}{\varepsilon}\right)$, *where $\widetilde{O}$ hides $\log D$ and $\log(\log T/\varepsilon)$ factors.*

Proof of Theorem A.4. This proof follows immediately from combining item (1a) of Theorem 3.3, which describes the edge-to-edge stability of Algorithm 1 that includes edges in the projection according to original degrees (which we call Π_D^{BBDS}), with the edge-private algorithms of [FHO21] and with our edge-private algorithm for f_{CC}. In particular, for f_{edges} and $f_{\text{triangles}}$ we can run the restricted edge-private algorithms of [FHO21] with $\varepsilon' = \varepsilon/3$ on outputs from the projection Π_D^{BBDS}; and for f_{CC} we do the same, except with our algorithm. □

Appendix B. The Sparse Vector Technique

We use the sparse vector technique (SVT), introduced by [DNR+09] and refined by [RR10; HR10; LSL17], to continually check that the input graph satisfies the conditions of Theorem 3.3. In Algorithm 4, we provide a version of the sparse vector technique described in [LSL17, Algorithm 1].

Algorithm 4 Mechanism SVT for answering threshold queries with the sparse vector technique.

Input: Stream $S \in \mathcal{S}^T$; queries $q_1, q_2, \ldots$ of sensitivity 1; cutoff $c \in \mathbb{N}$; privacy parameter $\varepsilon > 0$; threshold $\tau \in \mathbb{R}$.
Output: Stream of answers in $\{\perp, \top\}$.

$\varepsilon_1 = \varepsilon_2 = \varepsilon/2$
count $= 0$
Draw $Z \sim Lap(1/\varepsilon_1)$
for each time $t \in 1, 2, \ldots, T$ **do**
    Draw $Z_t \sim Lap(2c/\varepsilon_2)$
    if $q_t(S) + Z_t \geq \tau + Z$ and count $< c$ **then**
        Output $\perp$
        count $+= 1$
    else output $\top$

Theorem B.1 (Privacy of SVT [LSL17])**.** *Algorithm 4 is $(\varepsilon, 0)$-DP under continual observation.*

C. Meta-Review

The following meta-review was prepared by the program committee for the 2024 IEEE Symposium on Security and Privacy (S&P) as part of the review process as detailed in the call for papers.

C.1. Summary

The paper develops privacy preserving algorithms to continually release graph statistics (edge count, degree histogram, triangle count) as the graph is updated over time. There are two notions of differential privacy (DP) for graphs: edge privacy and node privacy. While node-DP provides a much more compelling privacy guarantee to individuals, it is quite challenging to develop node-DP algorithms as most graph statistics are highly sensitive to the addition or removal of a single node with high degree. Because of this challenge prior work had only achieved a weaker notion of "degree restricted" node-DP where the DP privacy guarantees only hold if one makes the (implausible) assumption that every node in the graph/stream has degree at most D. The paper uses graph projections to transform a "degree restricted" node-DP algorithm (or a "degree restricted" edge-DP algorithm) into an algorithm that satisfies node-DP unconditionally. The node-DP algorithms are shown to be accurate as long as the original graph/stream was "degree restricted" i.e., the maximum degree of any node in the graph is at most D for some suitable parameter $D < n$.

C.2. Scientific Contributions

- Creates a New Tool to Enable Future Science
- Provides a Valuable Step Forward in an Established Field

C.3. Reasons for Acceptance

1) Creates a New Tool to Enable Future Science: The paper introduces the first node-DP algorithm to continually release fundamental graph statistics (edge count, degree histogram, triangle count) as the graph is updated over time.
2) Provides a Valuable Step Forward in an Established Field: Prior work (e.g., [BBDS13]) used graph projections as a tool to preserve node-DP in the static setting. The paper shows how these graph projections can be extended to work in the continual release setting where the graph is updated over time.

C.4. Noteworthy Concerns

1) The node-DP algorithm in the paper is only shown to release accurate statistics if the underlying graph has maximum degree D. It is not clear whether the statistics will be accurate in the more plausible setting that the graph is (D, ℓ)-bounded i.e., there are at most ℓ nodes whose degree is larger than D.
2) The empirical evaluation only considers synthetic graph streams.