Matrix Sketching Over Sliding Windows

Zhewei Wei[1]*, Xuancheng Liu[2], Feifei Li[3], Shuo Shang[2], Xiaoyong Du[1], Ji-Rong Wen[1]†

[12]School of Information, Renmin University of China
Key Laboratory of Data Engineering and Knowledge Engineering, MOE
Beijing Key Laboratory of Big Data Management and Analysis Methods
[3]School of Computing, University of Utah
[1]{zhewei, duyong, jrwen}@ruc.edu.cn [2]{xcliu94, jedi.shang}@gmail.com [3]lifeifei@cs.utah.edu

ABSTRACT

Large-scale matrix computation becomes essential for many data data applications, and hence the problem of sketching matrix with small space and high precision has received extensive study for the past few years. This problem is often considered in the row-update streaming model, where the data set is a matrix $A \in \mathbb{R}^{n\times d}$, and the processor receives a row $(1 \times d)$ of A at each timestamp. The goal is to maintain a smaller matrix (termed approximation matrix, or simply approximation) $B \in \mathbb{R}^{\ell\times d}$ as an approximation to A, such that the covariance error $\|A^TA - B^TB\|$ is small and $\ell \ll n$.

This paper studies continuous tracking approximations to the matrix defined by a sliding window of most recent rows. We consider both sequence-based and time-based window. We show that maintaining A^TA exactly requires linear space in the sliding window model, as opposed to $O(d^2)$ space in the streaming model. With this observation, we present three general frameworks for matrix sketching on sliding windows. The sampling techniques give random samples of the rows in the window according to their squared norms. The Logarithmic Method converts a *mergeable* streaming matrix sketch into a matrix sketch on time-based sliding windows. The Dyadic Interval framework converts arbitrary streaming matrix sketch into a matrix sketch on sequence-based sliding windows. In addition to proving all algorithmic properties theoretically, we also conduct extensive empirical study with real data sets to demonstrate the efficiency of these algorithms.

*This work was partly supported by the National Key Basic Research Program (973 Program) of China (No. 2014CB340403, No. 2012CB316205), the National Natural Science Foundation of China (NSFC. 61402532, NSFC. 61502501, NSFC. 61502502, NSFC. 61502503, NSFC. 61432006). Feifei Li was supported in part by NSF grants 1251019, 1200792, and NSFC grant 61428204.

†Corresponding author.

SIGMOD'16, June 26-July 01, 2016, San Francisco, CA, USA

 ISBN 978-1-4503-3531-7/16/06…$15.00

DOI: http://dx.doi.org/10.1145/2882903.2915228

1. INTRODUCTION

Modern data sets, such as text documents, image data and social graphs, are often modeled as large matrices [23–25,30]. Such data matrices are often huge in size and generated continuously, it is essential to process them in streaming fashion and maintain an approximating summary. *Matrix sketching* is a general technique for processing these matrices to reduce the volume of data before a more refined analytic task is performed, or to directly reveal information through Principal Component Analysis (PCA), k-means clustering, or Latent Semantic Indexing (LSI) [13, 17, 21, 25, 29, 35]. Most of the matrix sketching work assume the *row-update streaming model*. In this model, the algorithm receives a row of matrix $A \in \mathbb{R}^{n\times d}$ from time to time (n increases by one after receiving a new row), and the goal is to maintain a matrix sketch structure κ that is able to produce the approximation matrix $B \in \mathbb{R}^{\ell\times d}$ with only $\ell \ll n$ rows, but guarantees that $B^TB \approx A^TA$ for the covariance matrix A^TA, i.e., B approximates A well.

In many applications, however, streams are time-sensitive: people are more interested in recent data than those in the far past. The *sliding window model* [11] is the most prominent and intuitive model to capture this essence. Inspired by this observation, we study the problem of *continuous tracking matrix sketch in the sliding window model*. We consider two types of sliding windows. A *sequence-based window* is defined by the N most recent rows. For instance, an application may restrict an analysis to the last million records received by the system. On the other hand, a *timestamp-based window* contains all rows that arrived within a fixed time interval of length Δ that covers the most recent Δ timestamps. For instance, an application may restrict an analysis to tweets that were posted within the last hour.

Motivations. A natural motivation for this problem, as argued in [4, 27, 32], is based on the observation that the sliding window model is a more appropriate model than the unbounded streaming model in many real-world applications. It is particularly the case in the areas of data analysis wherein matrix sketching techniques are widely used, since in general one is not interested in gathering information from outdated data for future predictions. For example, in a large scale text data analysis, each row in the matrix corresponds to a document (e.g. content of a tweet in twitter data) and is associated with a timestamp (e.g. time that the tweet was posted). A streaming matrix sketch is useful for analyzing text data that starts at a particular time in history. How-

ever, a far more interesting task would be analyzing tweets posted for the recent time period, say last 24 hours, or analyzing most recent tweets, say last 100,000 tweets. To meet such demands, we need a matrix sketch for a sliding window of length 24 hours or 100,000 tweets.

A more subtle motivation is due to the hardness of tracking matrix exactly on a sliding window. In the unbounded streaming model, a naive solution for the matrix sketching problem is to maintain A^TA with $O(d^2)$ space and $O(d^2)$ time to process each update. This solution tracks A^TA exactly with efficiency comparable to matrix sketching, as long as the data dimension d is not too large (e.g. less than 1000). In the sliding window model, however, we show that it is not possible to track A^TA exactly unless we store all rows in the matrix from the most recent sliding window. This result has two impacts: 1. Sketching is essential for tracking approximation of matrices in the sliding window model, regardless of the dimension of the data; 2. Matrix sketching over sliding windows requires new techniques.

A concrete application: sliding window PCA. PCA projects high dimensional data into a lower dimensional space, and is widely used for dimensionality reduction, signal denoising, regression, visualization etc. One can use matrix sketching to perform approximate PCA, by applying PCA on the approximation B instead of the original matrix A [7].

One application of PCA is to detect changes and anomalies in multidimensional data streams [31, 33]. In this context, changes and anomalies are detected by comparing the distribution of recent data with previous data. A typical window-based solution is to extract a fixed *reference window* and to update a *test window* with newly coming data. Changes or anomalies are detected by comparing the PCA basis of data in the reference window to the test window. Results in existing literature all assume that all rows in the test window are stored in memory, and PCA is computed via svd(singular value decomposition) over all rows in the test window. This approach is inherently not scalable in applications where the window is too large to fit in memory, or the rows in the stream are too fast to apply offline PCA algorithms (or even streaming matrix sketches) on the whole window. Sliding window matrix sketching allows us to approximate PCA on sliding windows with sub-linear space, thereby giving us the ability to compute PCA for large test windows in a continuous, online fashion.

Terminologies and notations. For a vector $x \in \mathbb{R}^n$, we let $\|x\| = \sqrt{\sum_{i=1}^n x_i^2}$ denote the standard Euclidean norm of x. For a matrix $A \in \mathbb{R}^{n\times d}$, we use $\|A\| = \max_{\|x\|=1} \|Ax\|$ denote the *spectral norm* of A, and $\|A\|_F = \sqrt{\sum_{i=1}^n \|a_i\|^2}$ to denote the *Frobenius norm* of A, where a_i is the ith row of A. Intuitively, the squared spectral norm $\|A\|^2$ represents the maximum influence along any unit direction, and the squared Frobenius norm $\|A\|_F^2$ represents the total "energy" of A. The *singular value decomposition* of A, written $\mathsf{svd}(A)$, produces three matrices $[U, \Sigma, V]$ so that $A = U\Sigma V^T$. U is a $n \times n$ orthogonal matrix that consists of the left singular vectors $[u_1, u_2, \ldots, u_n]$, and V is a $d \times d$ matrix with columns $[v_1, v_2, ..., v_d]$ being the right singular vectors. $\Sigma = \mathrm{diag}(\sigma_1, \ldots, \sigma_d)$ is a $n \times d$ diagonal matrix, where $\sigma_1 \geq, \ldots, \geq \sigma_r$ are the singular values of A and $r \leq d$ is the rank. From the svd, we have the spectral norm $\|A\| = \sigma_1$, and the j-th singular value $\sigma_j = \|Av_j\|$. We also let A_k be the *best rank k approximation* of matrix A, specifically $A_k = \arg\min_{X:\mathsf{rank}(X)\leq k} \|A - X\|_F$. Note that A_k can be computed by $A = U_k\Sigma_k V_k^T$, where Σ_k is a diagonal matrix consisting of the largest k singular values, and U_k and V_k denote the matrices consisting of the first k columns of U and V, respectively.

We model a data stream as an infinite sequence $S = \{(a_i, t_i) \mid i = 1 \ldots \infty\}$, where $a_i \in \mathbb{R}^{1\times d}$ is a row of dimension d, and $t_i \in \mathbb{R}$ denotes the timestamp of a_i. For *time-based sliding windows*, let Δ be the window size Δ and t be the current timestamp. We use the time interval $W = [t - \Delta, t]$ to denote the rows in the window when there is no confusion. We use N_W and A_W to denote the number of rows and the matrix formed by all rows in window W, respectively; or simply use A to denote A_W and N to denote N_W when there is no confusion. A *sequence-based window* W is defined by window size N, which consists of the most recent N rows. All-zero row is not allowed in a sequence-based window. For reasons will become clear later, we assume the squared norms of the rows in the window take value from $[1, R]$, that is, $1 \leq \|a\|^2 \leq R$ for all $a \in W$.

Problem definition. Given window size Δ for time-based sliding windows (or N for sequence-based sliding windows) and error parameter ε, the goal is to maintain a matrix sketch κ such that at current time t, κ can return an approximation B for $A = A_W$, where the approximation quality is measured by the *covariance error*, such that:

$$\textbf{cova-err}(A, B) = \|A^TA - B^TB\|/\|A\|_F^2 \leq \varepsilon.$$

Covariance error can also be written as $\max_{\|x\|=1}(\|Ax\|^2 - \|Bx\|^2)/\|A\|_F^2$. Intuitively, covariance error guarantees that $\|Bx\|$ preserves $\|Ax\|$, the norm (or length) of A in any direction x. For instance, when performing PCA on A, it returns the top k orthogonal directions, measured in this length. Thus by setting an error threshold ε, this bound allows one to approximately retain all important directions.

Our contributions. This work aims at understanding the sliding window matrix sketching problem and providing solutions that are both theoretically sound and practically efficient. We first present two negative results to better understand the hardness of the problem. We show that:

(1) maintaining A^TA exactly on a sequence-based sliding window requires space linear to the sliding window. This result demonstrates the fundamental difference between the unbounded streaming model and the sliding window model.

(2) if the maximum norm of the rows in the window is unbounded, it is not possible to maintain a sketch κ with sub-linear space even we allow large covariance error. This lower bound suggests that we need to limit the maximum possible squared norm of all rows by a upper bound R in order to derive interesting and efficient algorithms. But this is ok in practice, since the largest squared norm for a row of data in most matrix data in real-applications is indeed bounded by some reasonable value of R [20, 21, 25].

That said, we present three techniques for the sliding window matrix sketching problem.

- The Sampling algorithms maintain random samples of rows with probabilities proportional to their squared norms in the sliding windows. Our algorithms can generate randomly sampled rows both *with and without replacement*, denoted as SWR and SWOR respectively. The sampling algorithms work for both *time-based and sequence-based sliding windows.*

sketch κ	update time	sketch size	cova-err	ℓ	window	$B \subset A$?	Need R?
Sampling (SWR)	$(d/\varepsilon^2)\log\log NR$	$(d/\varepsilon^2)\log NR$ (Expected)	$\leq \varepsilon$ w.h.p.	d/ε^2	sequence & time	✓	No
LM-FD	$d\log \varepsilon NR$	$(1/\varepsilon^2)\log \varepsilon NR$	$\leq \varepsilon$	$1/\varepsilon$	sequence & time	×	Yes
DI-FD	$(d/\varepsilon)\log R/\varepsilon$	$(R/\varepsilon)\log R/\varepsilon$	$\leq \varepsilon$	$1/\varepsilon$	sequence	×	Yes

Table 1: Comparison of various sliding-window matrix sketches. "Update cost" denotes the cost for processing a new row; "Sketch size" denotes the number of rows used by the sketch; "Cova-err" denotes the error guarantee; "ℓ" denotes the size of the final approximation B in terms of number of rows; "Window" indicates which types of sliding windows the sketch works for; "$B \subset A$" indicates if the sketch is interpretable; "Need R" indicates if the sketch needs to know the maximum squared norm R a prior.

- The *Logarithmic Method* (LM) converts a streaming matrix sketch into a sliding window matrix sketch, given that the streaming matrix sketch is *mergeable* [2]. It works for *both time-based and sequence-based sliding windows*. We combine the LM framework with Frequent Direction and obtain LM-FD.
- The *Dyadic Interval* (DI) framework converts an arbitrary streaming matrix sketch into a matrix sketch on *sequence-based sliding windows*. It has better error-space tradeoff than LM methods when the maximum squared norm in the window is small. We combine the DI framework with Frequent Direction and obtain a sliding window sketch DI-FD.

In the unbounded streaming model, it is possible to maintain B as the matrix sketch κ, e.g., the frequent direction (FD) sketch [25]. In the sliding window model, this is difficult and our algorithms maintain a sketch κ that is queryable and whenever κ is queried, it will be able to generate the approximation matrix B for the latest sliding window.

We analyze the behaviors of our algorithms with rigorous theoretical analysis in Section 5, 6 and 7.1, and present thorough experimental studies in Section 8 to verify the effectiveness of our methods in practice. All of our sketches maintain a set of *matrix rows* that are *not necessarily the rows from A*. They differ in: (1) how these rows are constructed, and (2) how these rows are stored and updated.

To describe the strength of each method, we select three algorithms and present the theoretical results in Table 1. The *Sketch size is measured in terms of the number of rows stored by each algorithm, and update cost is measured in terms of the CPU cost*. The sampling algorithms provide sketches that are interpretable, which is a desirable property in some applications [12, 14, 17]. On the other hand, theoretical analysis and experimental studies suggest that the LM-FD algorithm provides better approximation quality and faster processing time with the same space budget. Finally, if we *only* work with sequence-based window, *and* the norms of rows in the stream are bounded (e.g. all rows are normalized and $R = 1$), the DI-FD algorithm is more efficient than LM-FD and the sampling algorithms in terms of the space usage. Note that the LM-FD and the DI-FD algorithms must know the value R a prior, while the sampling algorithms do not require the knowledge of R.

An appealing feature of our sketches is that they are in fact very easy to implement and use in practice. However, the analyses, including both error analysis and space complexity, are nontrivial and quite involved for a general audience. Thus, we only present the main results and high-level ideas of our constructions in the main text, while deferring all details of the proofs and analyses to the appendix.

2. RELATED WORK

To the best of our knowledge, this is the first work on matrix sketching in the sliding window model. There are two classes of prior work that are relevant to our study: matrix sketching algorithms in the unbounded streaming model, and streaming algorithms in the sliding window model.

Streaming matrix sketching. Streaming matrix sketching approaches can be broadly divided into three categories. The first approach is to find a small subset of matrix rows (or columns) that approximate the entire matrix. This problem is known as the *column subset selection problem* [12, 14, 17], or in our model, the *row subset section problem*. A streaming solution to the column subset selection problem is obtained by iteratively sampling rows from the input matrix with probability proportional to their squared norms. Despite this algorithm's apparent simplicity, providing tight bounds for its performance required over a decade of research [15, 17]. We will refer to this algorithm as *sampling*. The second approach is to randomly combine matrix rows via random projection. Several results exist in the literature, including random projections [34] and hashing [1,8,38]. For details of these works, we refer to the survey by Woodruff [39]. The third approach is the deterministic matrix sketching technique [25] which adapts a well-known streaming algorithm for approximating item frequencies, the MG algorithm [28], to sketching a streaming matrix through tracking *frequent directions* (FD). FD was extended to derive streaming sketching results with bounds on relative error [20], and to monitor matrix approximation in distributed setting [21]. We refer readers to recent work [19] for extensive discussion and experimental study of various streaming matrix sketching algorithms and their error bounds.

But as we explained earlier in Section 1, there are fundamental differences between streaming matrix sketching and sliding window matrix sketching (as indicated by the huge difference between the lower bounds of the exact algorithms for the two problems). Hence, all these sketching techniques cannot be directly applied to solve our problem.

Sliding window algorithms. As mentioned earlier, the bulk of existing work on the sliding-window model has focused on algorithms for maintaining simple statistics, such as count, sum, heavy hitters, and quantiles, in space and time that is significantly sublinear (typically, poly-logarithmic) in the sliding-window size. Exponential histograms [11] is a state-of-the-art deterministic technique for maintaining ε-approximate counts and sums over sliding windows. ECM-sketch [32] extends exponential histograms to distributed environment. Arasu and Manku [3] proposed an algorithm that returns ε-approximate quantiles on sliding windows. Various other queries were studied in the sliding window model, including distinct elements [22], frequency count [27],

top-k [32], skyline [37] frequent pattern mining [36]. We refer interested readers to a survey by Cormode [9].

Random sampling over sliding windows is a line of work that is more related to ours. In this problem, the goal is to maintain a set of ℓ elements that is randomly drawn from the window. Babcock, Datar and Motwani [5] were the first to consider the problem, and they proposed an elegant solution by combining priority sampling with the "successors list" techniques. Since then this problem has received considerable attention [10,18]. Efraimidis and Spirakis [16] modified the priority sampling algorithm to sample weighted elements on an unbounded stream with probabilities proportional to the weights. Longbo et. al. [26] considered the weighted sampling problem over sliding windows. However, their sampling algorithm does not provide samples with probabilities proportional to the weights.

Finally, another independent related work is the sliding window sbd [6], which computes the svd over a window sliding through the rows of a matrix. However, it needs to store all rows in the sliding window, hence, is not a sketch.

3. PRELIMINARIES

This section reviews matrix sketching algorithms on an unbounded stream (i.e., streaming matrix sketching). Recall that we adapt the *row update model*, in which the algorithms receive n rows from a matrix $A \in \mathbb{R}^{n\times d}$ one after another, and the goal is to maintain a sketch κ that is able to produce the approximation matrix $B \in \mathbb{R}^{\ell\times d}$ to approximate A with only $\ell \ll n$ rows. The approximation quality is usually characterized by the covariance error $\|A^TA - B^TB\|/\|A\|_F^2$.

Row sampling. Sampling algorithms assign a probability p_i for each row a_i proportional to its squared norm, and elect ℓ rows from A into B. We have two ways for sampling rows. In the *sampling with replacement* scheme, each row a_i is sampled with probability $p_i = \|a_i\|^2/\sum_{a_k\in A}\|a_k\|^2 = \|a_i\|^2/\|A\|_F^2$. Intuitively, the squared norm of a row decides its contribution to the frequent directions for the sliding window it's in. To sample ℓ rows with replacement, we take ℓ independent copies of such random samples. Then we scale each sampled row a_i by a factor of $\sqrt{\ell}\|A\|_F/\|a_i\|$ to form the matrix B. Note that there may be duplicated rows in sampling with replacement scheme.

For *sampling without replacement*, we avoid sampling the same row by excluding sampled rows from future selection. More precisely, let S_i be the set of the first i samples. The $(i+1)$-th sample is selected from $A\setminus S_i$, and each $a_j \in A\setminus S_i$ is sampled with probability $\|a_j\|^2/\sum_{a_k\in A\setminus S_i}\|a_k\|^2$. Assuming S is the set of sampled rows, we scale each sampled row $a_i \in S$ by a factor of $\|A\|_F/\sqrt{\sum_{a_k\in S}\|a_k\|^2}$ to form the matrix B. The row sampling schemes were analyzed in [15,21], and it was shown that with $\ell = O(\frac{1}{\varepsilon^2}\log\frac{1}{\varepsilon})$ samples, both sample with and without replacement schemes achieve covariance error **cova-err** $\le \varepsilon$.

Priority sampling is a general technique for sampling rows in the streaming model. For each row a_i, we assign a random priority $\rho_i = u_i^{1/w_i}$, where u_i is a number drawn from $(0,1)$ uniformly at random. It was shown in [16] that row a_i takes the largest priority with probability $p_i = w_i/w$. To sample ℓ rows without replacement, we simply maintain a reservoir S that consists of the rows with top-ℓ priorities. Note that one can maintain top-ℓ priorities with a priority queue of size ℓ. To sample ℓ rows with replacement, we maintain ℓ independent priority samples, each consists of a single sample.

Frequent direction. In this case, the sketch κ maintains B directly, i.e., $\kappa = B$. To process each row a_i, we first check if there is an all zero row in B_{i-1}. If so, we replace the last empty row of B_i with a_i to form B_i. Otherwise, we perform $\mathsf{svd}(B_{i-1}) = U_{i-1}\Sigma_{i-1}V_{i-1}$. Let $\Sigma_{i-1} = \mathrm{diag}(\sigma_1,\ldots,\sigma_d)$, and $\lambda = \sigma_{\ell/2}^2$ where $\sigma_{\ell/2}$ is the $\ell/2$-largest singular value. We set

$$\Sigma_i = \mathrm{diag}\left(\sqrt{\sigma_1^2-\lambda},\ldots,\sqrt{\sigma_{\ell/2}^2-\lambda},0,\ldots,0\right),$$

and $B_i = \Sigma_i V_{i-1}$. Note that B_i has $\ell/2$ empty rows and can accommodate $\ell/2$ future updates. Setting $\ell = \varepsilon/2$, One can show that FD achieves **cova-err**$(A,B) \le \varepsilon$ using $O(1/\varepsilon)$ space and $O(d/\varepsilon)$ amortized update time. An extension of FD to derive streaming sketch results with bounds on relative error appeared in [20]. FD was also used to monitor matrix approximation in distributed setting [21].

4. LOWER BOUNDS

In this section, we prove two lower bounds that characterize the sliding window matrix sketching problem. We note that the lower bounds in this section will be described in the number of bits used by the algorithms, while all the upper bound results (in the rest of the paper) are presented in terms of number of rows used by the algorithms. This is because for lower bound proofs, it is not reasonable to make assumptions on how an algorithm stores its information. For simplicity, we only present the high level ideas of the proofs, and defer all details to the appendix.

A linear lower bound for exact sketching. We show that maintaining A^TA exactly on a sequence-based sliding window requires $\Omega(Nd)$ space. Note that this is the space we need to store A exactly in the sliding window. Recall that in the unbounded streaming model, we can maintain A^TA with only d^2 space, so this lower bound implies that there is a fundamental difference between streaming model and sliding window model for the matrix sketching problem.

Theorem 4.1 *An algorithm that returns A^TA for any sequence-based sliding window must use $\Omega(Nd)$ bits space.*

The proof of Theorem 4.1 relies on a reduction from the *INDEX* problem in communication complexity theory. For the details of the proof, please refer to Section B.1.

A linear lower bound for unbounded norms. The next lower bound concerns with the norm distribution in the sliding window. We show that if the maximum squared norms of rows in the matrix is unbounded, then we need linear space for the matrix sketching problem, even a constant covariance error is allowed.

Theorem 4.2 *An algorithm that returns B_W such that*

$$\Pr\left[\|A_W^TA_W - B_W^TB_W\| \le \frac{1}{8d}\|A_W\|_F^2\right] > \frac{1}{2}$$

for any sliding window W must use $\Omega(Nd)$ bits space.

We reduce the *d-Majority INDEX* Problem, a variation of the INDEX problem, to the matrix sketching problem. For the details of the proof, please refer to Section B.2.

Remark. By Theorem 4.2, we need to constraint the norms in the window if we aim at sub-linear space. Throughout the

paper, we assume the squared norms of rows in the stream takes value from $[1, R]$, that is, $1 \leq \|a_j\|^2 \leq R$ for any a_j in the window. This is a mild assumption.

5. BASELINE: SAMPLING ALGORITHMS

In this section, we describe a framework for sampling matrix rows on a sliding window. Our algorithms work for both sequence-based and time-based sliding windows. Although it is a fairly common setting in many matrix sketching works to treat the sampling algorithms as the "baseline solutions", we would like to emphasize that sampling rows according to their norms over sliding windows is actually a non-trivial task. More over, analyzing the space and error bounds for such algorithms also needs novel techniques.

5.1 Algorithm Description

Row Sampling With Replacement. We first show how to sample a single row on a sliding window. Intuitively, we make slight modification to the uniform sampling algorithm on sliding windows [5] to support the row sampling schemes. For each new row a_j, we assign to it a priority $\rho_t = u_t^{1/\|a_i\|^2}$, where u_t is a random number uniformly drawn from $(0, 1)$. By the analysis of priority sampling, we need to maintain the top-1 row, that is, the row with largest priority in the window. Simply storing the top-1 row is not sufficient, since it will expire as the window moves. The idea is to maintain a list of *candidate rows*, which are defined to be the rows that could become the top-1 row in the future. More precisely, a_j is a candidate row if and only if its priority ρ_j is the largest priority between its timestamp t_j and current timestamp t_c.

Algorithm 5.1 Update algorithm of SWR at time t

```
1: Remove all (a_j, t_j, ρ_j) in Q with t_j < t − Δ
2: if update = (a_t, t) then
3:    Choose u_t ∈ Unif(0, 1) and set ρ_t ← u_t^{1/‖a_t‖^2}
4:    while ρ < ρ_t do
5:       (a_j, t_j, ρ_j) = Q.back
6:       if ρ < ρ_t then
7:          Remove (a_j, t_j, ρ_j)
8:    Append (a_t, t, ρ_t) to the end of Q
```

For the query process, we find each sampled row a_j and rescale it back by a factor of $\frac{\sqrt{\ell}\|a_j\|}{\|A\|_F}$. One technical issue is that we are not able to track the exact value of $\|A\|_F$ on a sliding window. Recall that $\|A\|_F^2 = \sum_{a \in W} \|a\|^2$, so tracking $\|A\|_F$ is equal to the problem of tracking sum on sliding windows, and is discussed thoroughly in [11]. We can maintain an exponential histogram to approximate $\|A\|_F^2$ with additive error $\varepsilon^2\|A\|_F^2$, such that we can approximate $\|A\|_F$ within $(1+\varepsilon)$ factor. As shown in [11], this requires an addition space of $O(\frac{1}{\varepsilon^2} \log NR)$. As we will show in the analysis, this will not affect the approximation quality asymptotically. We also note that in many real world applications it is possible to maintain the exact value of $\|A\|_F$ by storing the norm of each row in the window, which requires much less space than storing the original rows.

Row Sampling Without Replacement. Recall that to sample ℓ items without replacement in the streaming setting, we need to maintain top-ℓ rows, that is, the ℓ rows with highest priorities. Consider a row a_j with timestamp t_j and priority ρ_j. A simple observation is that if a_t is not a top-ℓ row in window $[t_j, t]$, it will never become a top-ℓ row from this time forward. Therefore, we only need to store row a_j if and only if it is a top-ℓ row in window from time t_j to current time t.

Algorithm 5.2 Update algorithm for SWOR at time t

```
1: Remove all (a_j, t_j, ρ_j, k_j) in Q with t_j < t − Δ
2: if update = (a_t, t) then
3:    Choose u_t ∈ Unif(0, 1) and set ρ_t ← u_t^{1/‖a_t‖^2}
4:    for (a_j, t_j, ρ_j, k_j) ∈ Q do
5:       if ρ_t > ρ_j then
6:          k_j += 1
7:          if k_j > l then
8:             Remove (a_j, t_j, ρ_j, k_j) from Q
9:    Append (a_t, t, ρ_t, 1) to the end of Q
```

Algorithm 5.2 illustrates the pseudo code of sampling ℓ rows without replacement. We initiate a queue Q to store the candidate rows and their priorities. For each candidate row a_j with timestamp t_j, we also store its rank k_j in window between t_j and current time t. At timestamp t, we first remove all expired tuples (line 1). If there is a new row a_t coming at time t, we first assign its priority $\rho_t = u_t^{1/\|a_t\|^2}$ (line 2-3). We then scan the tuples in Q and update the rank value k_j for each tuple (a_j, t_j, ρ_j, k_j) (line 5-6). If the rank k_j is larger than ℓ, then we remove the tuple from Q (line 7). Finally, we add the tuple $(a_t, t, \rho_t, 1)$ to Q (line 8).

For the query process, we set w to the the square root of $\sum_{a_k \in S} \|a_k\|^2$, the summation of squared norms of all sampled rows. Then we find each sampled row a_j and rescale it back by a factor of $\frac{\sqrt{\ell}\|a_j\|}{\|A\|_F}$ to make up B. Here we will also use an $(1+\varepsilon)$-approximation of $\|A\|_F$ from an exponential histogram if the exact value of $\|A\|_F$ is unknown.

5.2 Analysis

Note that Algorithm 5.1 and Algorithm 5.2 follow the same framework as the uniform priority sampling algorithms in [5]. The only difference is the way we generate the priorities. However, this modification requires non-trivial analysis on the space-error tradeoffs. In the following, we will state the theorem and key lemmas, and defer all proofs in the appendix.

We first show two Lemmas on the expected number of candidate rows in Algorithm 5.1 and Algorithm 5.2 for sampling ℓ rows. We assume the maximum window size is bounded by N, and the squared norms of rows in the stream takes value from $[1, R]$.

Lemma 5.1 *The expected number of candidate rows for Algorithm 5.1 is $O(\log NR)$. Consequently, the expected number of candidate rows for sampling ℓ rows in SWR is bounded by $O(\ell \log NR)$.*

Lemma 5.2 *The expected number of candidate rows to sample ℓ rows in SWOR is bounded by $O(\ell \log NR)$.*

Recall that we need to sample $\ell = O(\frac{d}{\varepsilon^2})$ rows to achieve ε covariance error for both SWR and SWOR. Combining with Lemma 5.1 and 5.2, we have the following space-error tradeoffs for the row sampling algorithms.

Theorem 5.1 *The SWR and SWOR algorithms can return an approximation B with error* **cova-err**$(A, B) \leq \varepsilon$ *using $O(\frac{d}{\varepsilon^2} \log NR)$ space in expectation. The expected update times*

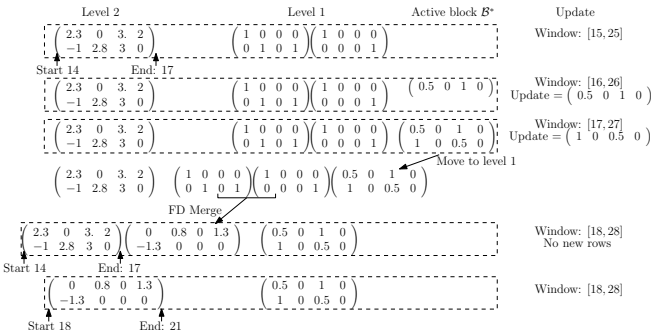


Figure 1: Running example for the LM-FD algorithm

for SWR and SWOR are $O(\frac{d}{\varepsilon^2}\log\log NR)$ *and* $O(\frac{d}{\varepsilon^2}\log NR)$*, respectively.*

6. LOGARITHMIC METHOD

In this section, we present a general framework to convert streaming matrix sketches into sliding window matrix sketches. This framework is inspired by the notion of *mergeable summaries* [2] and the logarithmic level idea of the well-known Exponential Histogram [11]. In this section, we use the Frequent Direction sketch as an example. We will show how the logarithmic method framework can cope with other mergeable sketches in the appendix.

6.1 Mergeability

Given two matrices A_1 and A_2 and their sketches $B_1 = \kappa(A_1, \varepsilon)$ and $B_2 = \kappa(A_2, \varepsilon)$, mergeability states that we can merge two sketches into a sketch for $A = [A_1; A_2]$ without increasing the size or the error. More precisely, recall that we have $\textbf{cova-err}(A_1, B_1) \leq \varepsilon$ and $\textbf{cova-err}(A_2, B_2) \leq \varepsilon$. More over, the space usage for both sketches are bounded by ℓ, respectively. A matrix sketch κ is *mergeable* if there is a merging algorithm such that $B = \mathsf{merge}(B_1, B_2)$ satisfies:

- The space usage for B is at most ℓ;
- The covariance error $\textbf{cova-err}(A, B)$ is at most ε.

Intuitively, if we can merge two sketches without increasing their error or size, then the sketches are mergeable. Mergeability is a very strong property, and is widely used in sensor networks and parallel computations. [2] focuses on summaries on streams of single attribute data.

Liberty [25] showed that Frequent Direction is also mergeable. Given two sketches $B_1 = \mathsf{FD}(A_1, \varepsilon)$ and $B_2 = \mathsf{FD}(A_2, \varepsilon)$, we set $C = [B_1; B_2]$ be the matrix that contains the two sketches B_1 and B_2 vertically stacked. We then run a singular decomposition on $[B_1; B_2] = U\Sigma V^T$, and let δ be the square of the ℓ-largest singular value, where $\ell = 1/\varepsilon$. We then subtract δ from all diagonal values in Σ^2, and set all negative values to be 0. Finally, we set $B = \Sigma V^T$. Liberty shows that the final sketch B can approximate $A = [A_1; A_2]$ with covariance error ε.

6.2 Algorithm Description

High Level Ideas. Figure 1 illustrates the high level idea for the LM framework. Suppose we have a streaming matrix sketch that achieves covariance error $\varepsilon/2$ with ℓ rows. For simplicity, we assume that $\ell \leq R$; We will show how to remove this assumption later. We divide the window into levels of exponentially decreasing sizes. Each level is further divided into $b = \Theta(1/\varepsilon)$ blocks. The block in the leftmost level will have size roughly $\varepsilon\|A\|_F^2$, such that the expiration of some of its rows introduces a covariance error no more ε. We maintain a streaming mergeable sketch of size ℓ, for each block. For a query, we merge the sketches from each block together into an approximation B of size ℓ. By mergeability, B approximates the matrix A with covariance error $\varepsilon/2$. Finally, mergeability also ensures that we can merge two sketches in level i into a sketch in level $i+1$ without increasing the size, which is crucial for maintaining the logarithmic structure.

Data Structures. We divide the rows in the window into blocks, each covering a segment of non-overlapping consecutive rows. Each block $\mathcal{B}$ is associated with an streaming sketch of size ℓ, as well as the start time, the end time and the size of $\mathcal{B}$. We define the size of a block $\mathcal{B}$ to be the summation of the squared norms of the rows covered by $\mathcal{B}$, that is, $\mathcal{B}.size = \sum_{a\in\mathcal{B}}\|a\|^2$. We further group the blocks into levels of exponentially increasing sizes. Levels are numbered sequentially $1, \ldots, L$, with the rightmost (most recent) level to be level 1, and the leftmost (furthest) level to be level L. We maintain two invariants:

1. Each level contains $b-1$ or b blocks;
2. The size of a block in level i is between $2^{i-1}\ell$ and $2^i\ell$.

Note that since each row carries a squared norm at least 1, the second invariant ensures a block at level i covers at most $2^i\ell$ rows. We define 3 states of the block: *active*, *inactive*, *expiring*. An active block is one that receives updates ; an inactive block is one that lies completely inside the window; an expiring block is one that contains timestamp $t - \Delta$. See Figure 1. We note that there are exactly 1 active block in the sketch.

Algorithm 6.1 Update algorithm for LM (at time t)

```
1: Remove all blocks B in L[L] with B.end < t
2: if L[L] is empty then
3:    Remove L[L] and set L ← L − 1
4: if update = (a_t, t) then
5:    Insert a_t to B*.sketch
6:    Set B*.end ← t and B*.size ← B*.size + ||a_t||^2
7:    if B*.size > ℓ then
8:       Append B* to L[1] and initialize an empty B* with
         B*.start = B*.end = t
9:    for i from 1 to L and L[i].length ≥ b + 1 do
10:      Find the rightmost blocks B_1 and B_2 in L[i]
11:      Construct a new block B with B.sketch =
         Merge(B_1.sketch, B_2.sketch)
12:      Set B.start ← B_1.start, B.end ← B_2.end, and
         B.size ← B_1.size + B_2.size
13:      Append B to L_{i+1} and remove B_1 and B_2 from L[i]
```

Update algorithm. Algorithm 6.1 illustrates the pseudo code of updating a LM sketch. Figure 1 use a running example to demonstrate the update process. Consider a time-based window of size 10. We set $\ell = 2$ and $d = 4$ for the LM sketch. At timestamp 25, we assume the sketch consists of 3 blocks, two from level 1 and one from level 2. We also assume the active block $\mathcal{B}^*$ is empty. Note that the rightmost block start at time 15 and end at time 18. As the window moves to $[16, 26]$, a new row $(0.5, 0, 1, 0)$ is inserted to the active block $\mathcal{B}^*$ (line 5-6 of Algorithm 6.1). Next, the window moves to $[17, 27]$. The sketch receives a new row $(1, 0, 0.5, 0)$

and insert it to $\mathcal{B}^*$. Since $\mathcal{B}^*$ is full, we move it to level 1 (line 5-6) and initialize a new active block (line 8). There are 3 blocks in level 1, so we perform the merging algorithm to the leftmost two blocks of level 1 to form a block of level 2 (line 10-14). Finally, when the window moves to $[18, 28]$, the rightmost block is expired and removed (line 2-3).

Algorithm 6.2 Query algorithm for LM (at time t)

```
1: Initialize B = B*
2: for i from 1 to L do
3:    for Each B in L[i] do
4:       B ← Merge(B, B.sketch)
```

Query algorithms. The query algorithm for LM is straightforward, and is illustrated in Algorithm 6.1. We merge all sketches in non-expiring blocks. Note that this is feasible since we require all sketches to have the same size ℓ.

Remark. Finally, we remove the assumption $\ell \leq R$. If a newly received row a_t carries a squared norm $\|a_t\|^2 \geq \ell$, we will maintain it as a single block in level 1. This row will remain untouched until its block reaches level $j = \log(\|a_t\|^2/\ell)$, which is the first level with block size larger than $\|a_t\|^2$. After level j, this row is treated as a common row and is allowed to participate merging process. As we shall see in the analysis, this will not affect the space or time bounds for the LM framework.

6.3 Analysis

We analyze the error guarantees as well as space and time complexity for the LM framework. We first prove a general Theorem that relates the complexity for LM algorithm and the streaming sketch used in each block. Then we investigate how a specific sketch can be integrate into the LM framework. In particular, assume the squared norms in the stream take value from $[1, R]$, and the maximum window size is N. We have the following Theorem for the space usage and update cost of the LM algorithm.

Theorem 6.1 *Suppose a mergeable matrix sketch κ achieves* **cova-err**$(A, B) \leq \varepsilon/8$ *with ℓ rows and update cost u in the streaming model. If we use κ as the sketch for each block in the LM algorithm and set $b = 8/\varepsilon$, we obtain a matrix sketch that works on time-based sliding window with error guarantee* **cova-err**$(A_W, B) \leq \varepsilon$ *for any given window W. The algorithm uses $O(\frac{\ell}{\varepsilon} \log \varepsilon NR)$ space and process an update in $O(u \log \varepsilon NR)$ time.*

Proof. We first prove the error bound. Recall that the error consists of two parts: the error from the expiration of the expiring block in level L, and the error from the merged sketch for the rows in unexpired blocks. By the construction of the LM framework, there are two possibilities for the sketch in each block $\mathcal{B}$ in level i: if the size of $\mathcal{B}$ is between $2^{i-1}\ell$ and $2^i\ell$, we maintain a streaming sketch for the rows covered by $\mathcal{B}$ with size ℓ. Otherwise, we know $\mathcal{B}$ covers a single row with squared norm larger than $2^i\ell$, and we maintain that row exactly. It follows that the size of a block $\mathcal{B}$ at level i is always lower bounded by $2^{i-1}\ell$, and is upper bounded by $2^i\ell$ unless it only covers a single row. In particular, consider the level $L-1$. There are b block in this level, each with size at least $2^{L-2}\ell$. Using the simple fact that the total size in this level cannot exceeds $\|A\|_F^2$, we have $\|A\|_F^2 \geq 2^{L-2}b \cdot \ell$, and thus

$$L \leq \log \frac{\|A\|_F^2}{\ell b} + 2 \leq \log \frac{\varepsilon \|A\|_F^2}{2\ell}. \quad (1)$$

The last inequality uses that $b \leq 8/\varepsilon$. This suggests that a block in the leftmost level L has size upper bounded by $2^L\ell = \varepsilon\|A\|_F^2/2$, unless it covers a single row. Since the expiring block is at level L, if its size is smaller than $\varepsilon\|A\|_F^2/2$, then the error contributed by it is bounded by $\varepsilon\|A\|_F^2/2$, otherwise it covers a single row and introduces no error. For the error from the merged sketch, we note that for each block $\mathcal{B}$ in level i, if the its size is between $2^{i-1}\ell$ and $2^i\ell$, the $\mathcal{B}.sketch$ uses space ℓ and approximate $\mathcal{B}$ with error bounded by $(\varepsilon/8) \cdot 2^i\ell$. Otherwise, the block covers a single row and its sketch returns no error. Therefore the total error returned by the merged sketch is bounded by

$$\sum_{i=1}^{L} (\varepsilon/8) \cdot 2^i \ell \cdot b \leq \varepsilon 2^{L+1} \ell b/8 \leq \varepsilon \cdot (\varepsilon/2)\|A\|_F^2 \cdot 1/\varepsilon = \frac{\varepsilon\|A\|_F^2}{2}.$$

The last inequality is due to (1). Therefore, the error contributed by the merged sketch is also bounded by $\varepsilon\|A\|_F^2/2$, and thus the total covariance error is bounded by ε.

For the space complexity, we note that each block is maintaining a sketch with size ℓ. There are at most Lb blocks, thus the total space usage is bounded by

$$Lb\ell \leq \log \frac{\varepsilon\|A\|_F^2}{2\ell} \cdot \frac{8}{\varepsilon} \cdot \ell = O\left(\frac{\ell}{\varepsilon} \log \frac{\varepsilon NR}{s}\right) = O\left(\frac{\ell}{\varepsilon} \log \varepsilon NR\right).$$

Finally, for the update cost, note that each newly arriving row is inserted into the the sketch of the active block with $u_{1/\ell}$ update time. Moreover, we have to merge the leftmost two blocks for level i if the sum of squared norms received since last merging process exceeds $2^i\ell$, for $i = 1, \ldots, L$. To amortize this cost, we note that each row is merged at most L times, each merge process takes u time per row, so the amortized update cost is bounded by $Lu = O(u \log \frac{\varepsilon\|A\|_F^2}{\ell}) = O(u \log \varepsilon NR)$. □

Analysis for LM-FD. Recall that this algorithm uses the FD sketch for each block in the LM framework. Recall that given error parameter $\varepsilon/8$, the FD sketch uses $\ell = O(1/\varepsilon)$ rows and process an update with $u = O(d/\varepsilon)$ amortized cost. By Theorem 6.1, FM-FD achieves **cova-err** $\leq \varepsilon$ with space $O(\frac{1}{\varepsilon^2} \log \varepsilon NR)$ and amortized update cost $O(\frac{d}{\varepsilon} \log \varepsilon NR)$.

We note that the update time can be dramatically improved with a simple modification. Observe that the active block $\mathcal{B}^*$ covers at most ℓ rows. Therefore, instead of maintaining a FD sketch as $\mathcal{B}^*.sketch$, we simply store the newly receiving rows in $\mathcal{B}^*.sketch$ until its size exceeds ℓ. This will not affect the space-error tradeoffs for LM_FD, but will improve the amortized update cost to $O(d \log \varepsilon NR)$.

Corollary 6.1 *The LM-FD algorithm achieves* **cova-err** $\leq \varepsilon$ *with $O(\frac{1}{\varepsilon^2} \log \varepsilon NR)$ space and $O(d \log \varepsilon NR)$ update cost.*

7. DYADIC INTERVAL FRAMEWORK

In this section, we present another framework, termed Dyadic Interval (DI), that converts a streaming matrix sketch into a matrix sketch that works for sequence-based sliding windows. This framework is more efficient (space-wise) than the LM framework when the norms are small, and is capable of converting an arbitrary stream sketch into a sliding window matrix sketch. We use the Frequent Direction sketch to illustrate the framework in this section. We will show how the Dyadic Interval framework can cope with other streaming sketches in the appendix.

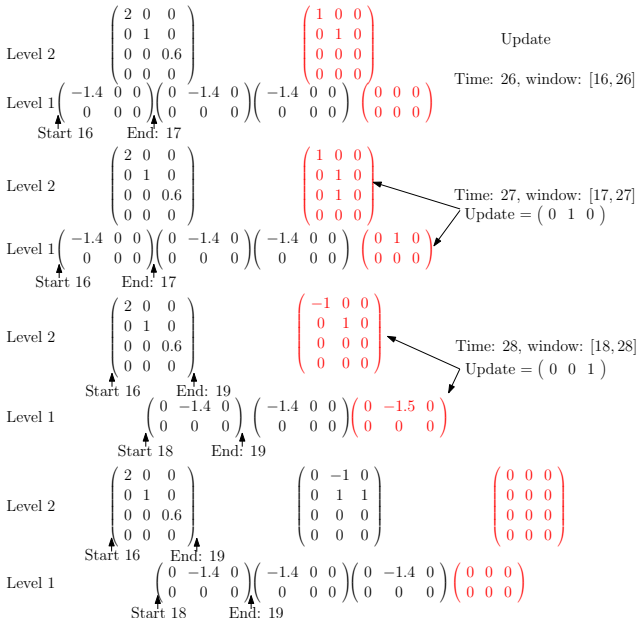


Figure 2: Illustration for Dyadic Interval method

7.1 Decomposability

DI framework is based on the *decomposability* of streaming matrix sketches. Intuitively, decomposability states that if we divide the rows of a matrix into submatrices and construct a sketch for each submatrix, we can concatenate these sketches back together to form a sketch for the original matrix. Note that decomposability is weaker than mergeability since it does not require that sketch size to remain the same after the merge, and thus is a weaker property. In fact, it is easy to show that all matrix sketches with the covariance error measure satisfy decomposability.

Lemma 7.1 (Decomposability) *Given a matrix $A_{n\times d}$, suppose we decompose A into k sub-matrices $A = [A_1; \ldots; A_k]$, in which A_j is an $n_j \times d$ matrix. Suppose we compute a matrix sketch for each sub-matrix A_i, denoted B_i, such that $\|A_i^T A_i - B_i^T B_i\| \le \varepsilon_i \|A_i\|_F^2$. Then $B = [B_1; \ldots; B_k]$ is an approximation for $A = [A_1; \ldots; A_k]$ with error bound*

$$\|A^T A - B^T B\| \le \sum_{i=1}^{k} \varepsilon_i \|A_i\|_F^2.$$

Proof. We observe that $A^T A = \sum_{i=1}^k A_i^T A_i$ and $B^T B = \sum_{i=1}^k B_i^T B_i$. Therefore we have

$$\|A^T A - B^T B\| \le \sum_{i=1}^{k} \|A_i^T A_i - B_i^T B_i\| \le \sum_{i=1}^{k} \varepsilon_i \|A_i\|_F^2,$$

and the Lemma follows. □

7.2 Algorithm Descriptions

High Level Ideas. The DI framework is inspired by the dyadic interval structure of the sliding window quantile structure in [3]. The high level idea is illustrated in Figure 2. We construct logarithmic number (denoted L) of levels, with each level partitions the stream into dyadic blocks. A block at the level $i+1$ contains two blocks at level i. It is easy to see that a sliding window can be decomposed into $2L$ blocks, with each level contributing at most 2 blocks. So if we maintain a streaming sketch for each block, we can concatenate the $2L$ sketches to form an approximation B for the rows in the sliding window, and the error is bounded by the decomposability property. A constraint for this dyadic structure is that it does not allow the window to shrink or expand, and thus only works for sequence-based sliding windows.

Data Structure. Suppose we have a streaming matrix sketch κ that achieves covariance error $\varepsilon/4$ with ℓ_ε rows. The notations of levels and blocks are slightly different from the ones in Section 6. We set the number of levels to be $L = \log R/\varepsilon$. For each level, we partition the stream into non-overlapping blocks, with block covers several consecutive rows in the stream. At the bottom level $\mathcal{L}[1]$, we impose a constraint such that the size of each block is between $NR/2^L$ and $NR/2^{L-1}$. We assume $2^L \le N$. Each block is associated with the following information: the start and end time and the size. We number the block in $\mathcal{L}[1]$ from old to new. For a higher level i, we make sure that each block covers exactly 2^i blocks of level 1. In particular, a block in level i covers block $j2^i$ to $(j+1)2^i - 1$ in level 1. This implies that a block in level i is of size $\Theta(NR/2^{L-i})$. Since level 1 contains at most 2^L blocks.

Following the same definitions as in Section 6, there are 3 states of the block: *active*, *inactive* and *expiring*. See Figure 2. We note that there are exactly 1 active block in each level. For the active block $\mathcal{B}^*$ at level i, we run a streaming sketch with error parameter $\varepsilon_i = 1/(2^i L)$. Once the active block $\mathcal{B}^*$ becomes full, that is, the size of $\mathcal{B}$ exceeds ℓ, the block becomes inactive. Note that this implies that an inactive block in level i stores a sketch with error parameter $\varepsilon_i = 1/(2^i L)$. To return a matrix approximation for the current window, we combine the sketches of some active and inactive blocks to determine the answer.

Algorithm 7.1 Update Algorithm for DI method

```
1: On receiving t-th update (a_t, t).
2: for i ∈ [L] do
3:     Remove all blocks L[i].B[j] with B[j].end < t − N
4:     Insert a_t to L[i].B*.sketch
5:     Set L[i].B*.end ← t
6:     Set L[i].B*.size += ‖a_t‖^2
7: if L[1].B*.size > NR/2^L then
8:     Set v ← number of trailing 0's in L[i].length + 1
9:     for i ∈ [v] do
10:        Mark L[i].B* as inactive and append it to L[i]
11:        Initialize an empty L[i].B* with L[i].B*.start =
           L[i].B*.end = t
```

Update algorithms Algorithm 7.1 illustrates the pseudo code of updating a DI-based sketch. Figure 2 use a running example to demonstrate the update process. We use a sequence-based window of size 10. The sketches consists of 2 levels. For each block at level 1, we set its maximum size to be 2. Suppose at time 26, the sketch consists of 4 blocks from level 1 and 2 blocks from level 2. We use red color to mark the active blocks. Note that the active block of level 1 is empty. As the window moves to $[17, 27]$, we insert the new coming row $(0, 1, 0)$ to the active blocks in level 1 and 2 (line 4 in algorithm 7.1). Next, as the window moves to $[18, 28]$, the leftmost block of level 1 is expired and removed from the sketch (line 3). Then we insert the new coming row $(0, 0, 1)$ to the two active blocks (line 4). Since the active

block of level 1 is full (the squared Frobenius norm ≥ 2), we need to mark some active blocks as inactive. To do so, note that the number of blocks in level 1 is 3, and the number of trailing 0's of (3+1=4) is 2. Thus, we set $v = 2$ (line 8) and mark the active blocks in level 1 and 2 as inactive (line 10). Finally, we initialize an new active block for each level (line 11). The final sketch at time 28 is showed in Figure 2.

Algorithm 7.2 Query process for DI method on window $(t - N, t]$

```
1: Initialize a empty matrix B
2: Initialize a = t and b = t − N
3: Find the highest level v with at least 1 active block
4: for i from v to 1 do
5:    Find the leftmost block L[i].B[j]
6:    if L[i].B[j].end ≤ a then
7:       Combine B with L[i].B[j].sketch
8:       Set a = L[i].B[j].start
9:    Find the rightmost block L[i].B[k]
10:   if k = j then
11:      Set b = L[i].B[j].end
12:   else if L[i].B[k].start ≥ b and k ≠ j then
13:      Combine B with L[i].B[k].sketch
14:      Set b ← L[i].B[k].end
```

Query algorithm. Algorithm 7.2 illustrates the pseudo code of returning a matrix sketch for window $[t - N, t]$ with a DI-based sketch. We initialize an empty matrix B (line 1) and two indicating timestamp $a = t$ and $b = t - N$. As the algorithm proceeds, the intervals $[t - N, a]$ and $[b, t]$ indicate the rows that are not covered by B. We find the highest level with at least one active block (line 3) and let v denote this level. Then we start the following process in a top-down fashion: at each level i, we find the left most block, denoted $\mathcal{L}[i].\mathcal{B}[j]$ (line 5), and check if it is in intervals $[t - N, a]$ (line 6). If so, we select it to make up the window by combining B with its sketch $\mathcal{L}[i].\mathcal{B}[j].sketch$ (line 7). Then we update a to be $L[i].\mathcal{B}[j].start$, meaning that interval $[t - N, L[i].\mathcal{B}[j].start]$ is uncovered. Then we find the rightmost block $L[i].\mathcal{B}[k]$ (line 9) and check if it is the same block as $\mathcal{L}[i].\mathcal{B}[j]$ (line 10). If so, we simply update b to be $\mathcal{L}[i].\mathcal{B}[j].end$ (line 11), meaning that interval $[\mathcal{L}[i].\mathcal{B}[j].end, t]$ is uncovered. If $\mathcal{L}[i].\mathcal{B}[k] \neq \mathcal{L}[i].\mathcal{B}[j]$, we combine B with $\mathcal{L}[i].\mathcal{B}[k].sketch$ (line 13) and update b to be $\mathcal{L}[i].\mathcal{B}[k].end$ (line 14). The query algorithm will find at most $2L$ blocks that make up the window.

7.3 Analysis

Similar to Section 6, we analyze the space and time complexity of the DI framework based on the parameters of the streaming sketch used in each block. Assume the maximum weight in the stream is R, and the window size is N. We have the following Theorem.

Theorem 7.1 *Suppose a streaming matrix sketch κ achieves* **cova-err**$(A, B) \leq \varepsilon/4$ *with ℓ_ε space and u_ε update cost. By using κ as the sketch for each block in the DI framework, we obtain a matrix sketch that works on sequence-based sliding window and gives error guarantee* **cova-err**$(A_W, B) \leq \varepsilon$ *for any given window W. The sketch uses $O(\frac{R}{\varepsilon}\sum_{i=1}^{L}(\ell_{1/(2^iL)}/2^i))$ space and process an update in $O(u_\varepsilon L)$ time, where $L = \lceil \log R/\varepsilon \rceil$.*

Proof. We consider the error guarantee first. Recall that the error consists of two parts: the error from the expiring block in level 1, and the error from merging at most $2L$ sketches. By the fact that each row carries a squared norm at least 1 in the sequence-based sliding window, we have $\|A\|_F^2 \geq N$, and thus the size of the expiring block in level 1 is bounded by $NR/2^{L-1} \leq \varepsilon N/2 \leq \varepsilon\|A\|_F^2/2$. On the other hand, a block at level i has size at most $NR/2^{L-i}$, and maintains a sketch with error parameter $1/(2^iL)$, so a sketch at level i introduces error at most $\frac{1}{2^iL} \cdot NR/2^{L-i} = \frac{NR}{2^LL} \leq \frac{\varepsilon N}{4L}$. Summing up all $2L$ sketches follows that the error is bounded by $\varepsilon/2$. Therefore, the total covariance error is bounded by ε.

For the space usage, each block at level i maintains a sketch of size $\ell_{1/(2^iL)}$, and there are at most $2^{L-i} = \frac{8R}{2^i\varepsilon}$ blocks in level i. It follows that the space usage for level i is bounded by $8R\ell_{1/(2^iL)}/2^i$, and the total space usage is bounded by $O(\frac{R}{\varepsilon}\sum_{i=1}^{L}(\ell_{1/(2^iL)}/2^i))$.

For the update cost, we need to update L active sketches. This gives us a total update cost $u_\varepsilon L$. □

Note that if κ provides probabilistic bound, that is, κ achieves **cova-err**$(A, B) \leq \varepsilon$ with probability $1 - \delta$, then DI(κ) also achieves **cova-err**$(A_W, B) \leq \varepsilon$ for any given window W with probability $1 - \delta$.

DI-FD. This algorithm uses Frequent Direction sketch for each block in the DI method. Recall that given error parameter ε, the FD sketch uses $\ell_\varepsilon = O(1/\varepsilon)$ space and process an update with $u_\varepsilon = O(d/\varepsilon)$ amortized cost. By Theorem 6.1, we have the following corollary:

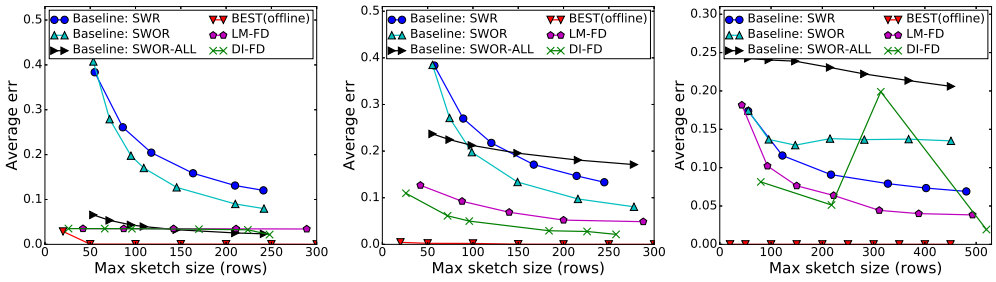
Corollary 7.1 *The LM-FD algorithm use $O(\frac{R}{\varepsilon}\log\frac{R}{\varepsilon})$ space and process an update with $O(\frac{d}{\varepsilon}\log\frac{R}{\varepsilon})$ amortized cost.*

8. EXPERIMENT

Algorithms. We implemented and compared the following algorithms. Note that for all sketches, our theoretical analysis only shows a loose upper bound for the sketch size in relationship to a desirable covariance error threshold ε, and the bad bases that actually meet those loose upper bounds almost never happens in real data sets. Hence, in practice, the performance of theses sketches are always much better, i.e., to meet an error threshold ε, they use much less space than what the theoretical bounds have indicated. That said, instead of setting a sketch's size based on its theoretical upper bound using an error threshold ε, we set the sketch size based on its structure and construction, as explained below, and report the *actual observed error* err in relationship to the actual size of various sketches.

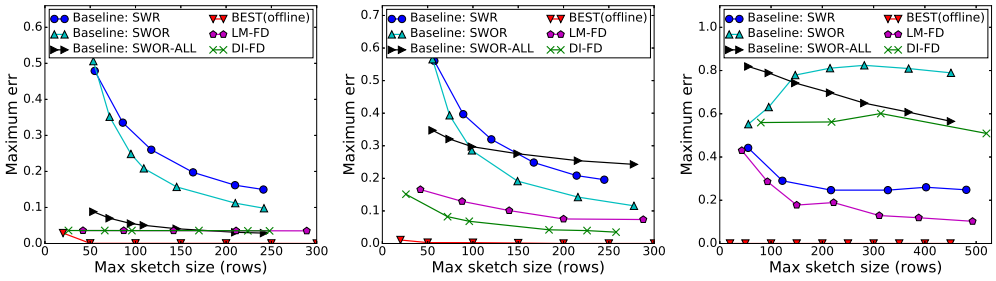
(1) Baseline: Sampling algorithms. We evaluated SWR and SWOR, the row sampling with/without replacement algorithms, respectively. We also implemented a variation of SWOR, denoted as SWOR-ALL, which makes use all the candidate rows for approximating A. The intuition for SWOR-ALL is that since SWOR only returns a small portion of the candidate rows as the random samples, and it is interesting to see if the precision can be improved by using all the candidate rows in the approximation. All sampling algorithms were evaluated on both sequence-based and time-based windows. The space and error bounds were controlled by a parameter ℓ, the number of rows to be sampled. Note that the size of a sampling sketch is larger than ℓ, since it needs to generate also candidate rows in order to construct B.

(2) Logarithmic Method based algorithms. We evaluated LM-FD on both sequence-based and time-based windows.



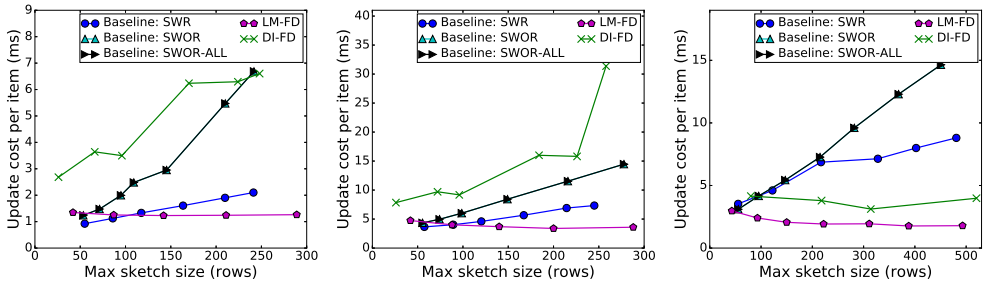
(a) SYNTHETIC (b) BIBD (c) PAMAP

Figure 3: average err vs. maximum sketch size.



(a) SYNTHETIC (b) BIBD (c) PAMAP

Figure 4: maximum err vs. maximum sketch size.



(a) SYNTHETIC (b) BIBD (c) PAMAP

Figure 5: update cost vs. maximum sketch size.

The space and error bounds were tuned by a parameter $b = \frac{1}{\varepsilon}$, the number of blocks in a level, and ℓ the size of the approximation matrix B (number of rows in B). Note that in LM-FD, each block simply stores a approximation matrix.

(3) Dyadic Interval based algorithms. We evaluated DI-FD on sequence-based sliding windows. The space and error bounds were set by the parameter $L = \log \frac{R}{\varepsilon}$, the maximum number of levels in the dyadic interval structure. Note that at the highest level, the streaming sketch has roughly $\ell/2$ rows, so that when B is constructed from all levels it has a size of roughly ℓ rows.

(4) Best rank k approximation. We evaluated BEST(offline), which is the best k-approximation for each sliding window. Note that the error provided by the best rank k approximation is theoretical optimal if the number of rows used by the sketch is bounded by k. It is not known how to compute the best rank k-approximation in the streaming model, so we compute BEST(offline) in an offline fashion.

Metrics. In the experimental study, we changed the parameters for each algorithm to show the the tradeoffs between the sketch size, actual observed error err, and running time.

- Sketch size was measured in terms of the number of rows stored for each algorithm for a sliding window. This is the dominating part of the memory usage, and is a common measurement for evaluating matrix sketching algorithms. Moreover, we measured the *maximum sketch size*, that is, the maximum number of rows stored over all appeared windows for an algorithm. For the best k approximation BEST(offline), we set k with various sizes and report its errors.
- For the approximation quality, we measured the *max error* and the *average error*, the maximum and average observed covariance error (err) over all appeared windows, using the approximation matrix B constructed by various algorithms.
- The update cost was measured by the *average processing time of one stream element (i.e., one matrix row)* for both sequenced based and time based windows. Note that we do not report the update cost of Best since it is an offline algorithm.

Setup. Since most of our sketches are randomized, we run each algorithm over each data set 20 times, and took the average for each metric to report. All algorithms were implemented in Python 2.7.6 using 32-bit addressing. The timing experiments were executed on a single idle core of an Intel Xeon E5-2620, clocked at 2.1 GHz.

Data Set	total rows n	d	N	ratio R
SYNTHETIC	10^6	300	10,000	8.35
BIBD	319,770	231	10,000	1
PAMAP	198,000	35	10,000	90089

Table 2: Data Sets for sequence-based window.

8.1 Sequence-based sliding window

We first evaluate all 6 algorithms on sequence-based sliding windows. We used two publicly available real data sets and one standard synthetic data set in the experiment.

Data Sets. SYNTHETIC is the Random Noisy matrix that has been commonly used for evaluating matrix sketching algorithms [19, 25]. Please see section D for the generating process. We generated a matrix with 300 columns and 1 million rows. All entries in SYNTHETIC are non-zeros.

BIBD[1] is the incidence matrix of a Balanced Incomplete Block Design from Mark Giesbrecht, University of Waterloo. This data set is available from the University of Florida Sparse Matrix Collection. It contains 231 columns and 319,770 rows, and 8,953,560 non-zero entries. Each entry is an 0 or 1 integer that indicates if an edge exists.

PAMAP[2] is is a physical activity monitoring data set which contains data of recorded with 8 subjects performing 14 different activities. The data set contains 45 columns, including a timestamp, an activity ID and 43 attributes of raw sensory data. In our experiments, we used a data of subject 1 with 198,000 rows and 35 columns (removing timestamps, activity IDs and columns containing missing values). We note that the gap between two consecutive timestamps of PAMAP is fixed to 0.5 second, so this data set can be easily treated using a sequence-based sliding window.

Table 2 summarizes these data sets.

Results. We set window size to be $N = 10,000$ for all data sets. Figure 3 and 4 illustrate the the tradeoffs between the max sketch size and the average error, and between the max sketch size and the maximum error, respectively. Figure 5 shows the tradeoffs between the max sketch size and update cost. We use the same color and line style for the same algorithm across figures for better illustration. We make the following observations:

(1) The performance of SWR and SWOR varies on different data sets. A theoretical advantage of SWOR over SWR is that it does not return duplicating rows, which is more theoretically "representative". Figures 3a and 3b show that given the same space budget, SWOR provides smaller average error than SWR on SYNTHETIC and BIBD. Figures 4a and 4b indicate the same for the maximum error. However, we observe in Figures 3c and 4c that SWR provides smaller error than SWOR on PAMAP.

(2) An interesting observation from Figure 4c is that the maximum covariance error of SWOR increases with the max sketch size on PAMAP, which is counter-intuitive. To eliminate the convolution effect from the sliding window structures, we located the window where the max error was achieved (row 125,000 to row 135,000), and performed offline sampling algorithms to the rows in that window. The results is shown in Figure 6. We observe that the covariance error for SWOR indeed increases as we increase the number of rows to be sampled. One possible reason is that SWOR is not preferable when the distribution of the norms is skewed. Consider an extreme case, in which the window consists of

[1] http://www.cise.ufl.edu/research/sparse/matrices/JGD_BIBD/bibd_22_8.html

[2] http://www.pamap.org/demo.html

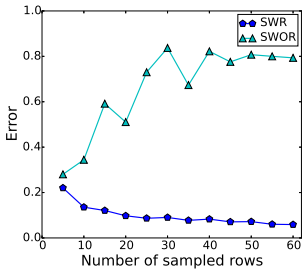


Figure 6: Covariance error vs. number of sampled rows for offline SWR and SWOR on PAMAP

$\ell - 1$ rows with very large norms, and $N - \ell + 1$ rows with very small norms. If we were to sample ℓ rows, one small row will be sampled and rescaled back according to $\|A\|_F^2$. This will over-emphasize the weight on this small row and will introduce large error.

(3) Making use of all candidate rows in the SWOR algorithm does not always give better error guarantee. While Figures 3a and 4a suggest that SWOR-ALL indeed performs better than SWR and SWOR on SYNTHETIC, the advantage becomes less obvious on BIBD (Figures 3b and 4b). Figure 3c shows that SWR or SWOR outperformed SWOR-ALL by a large margin on PAMAP in terms of average error. This can be explained by the fact that SYNTHETIC data set is a random matrix, so a candidate row acts approximately as a random sample. On the other hand, BIBD and PAMAP exhibit less randomness, so treating candidate rows as random samples may hurt the precision of estimation.

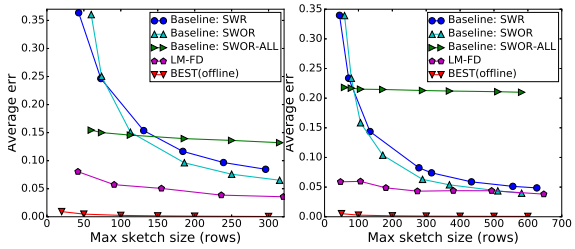
(4) The performance of DI-FD and LM-FD depends on the ratio between maximum squared norm and minimum squared norms in the data set. Figures 3b and 4b show that DI-FD achieves better error-space tradeoff than LM-FD on BIBD, while Figures 3c and 4c show the opposite results on PAMAP. Note that the ratio is 1 for BIBD, and is over 90,000 for PAMAP. This concurs with our theoretical analysis, that is, DI-FD is preferable when the ratio is small, and LM-FD is preferable when the ratio is large.

(5) Figures 3a and 4a show that for LM-FD, DI-FD and SWOR-ALL , the error does not decrease as we increase the space budget. Further investigation shows that by simply returning $B = 0$ as an approximation, one can achieve covariance error 0.0338, and any further improvement over this error may require very large space.

(6) Figure 5 shows that SWR was faster than SWOR on all three data sets. This is due to the fact that SWOR has to scan the queue Q to update the rank of each candidate row, while SWR can terminate the scan process. Figure 5 also suggests that LM-FD achieved the best performance in terms of running time. This is as expected, since the update costs for the LM-FD algorithm is bounded by $O(d \log \varepsilon NR)$, while the update costs for the DI-FD algorithm and the sampling algorithms are polynomial in d/ε. We also note that unlike the DI-FD and sampling algorithms, the running time for the LM-FD has slightly decreased as we increase the sketch size. This fits in the $O(d \log \varepsilon NR)$, since by increasing sketch size, we have effectively reduced the error threshold ε. Finally, note that it takes 20+ ms to update the DI-FD sketch on some data sets, which means it takes several hours to process 1 million rows.

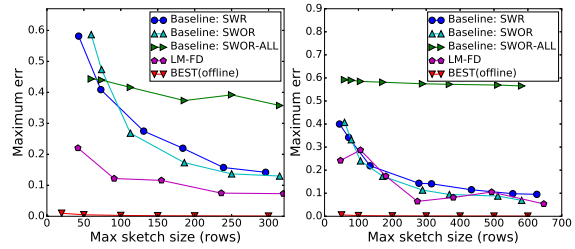
8.2 Time-based sliding window

We used two real world data sets to for evaluating our algorithms on time-based sliding windows.



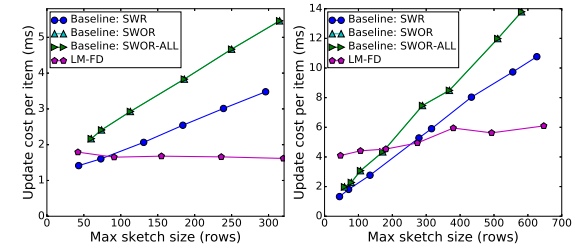
(a) RAIL (b) WIKI

Figure 7: average err vs. max sketch size.



(a) RAIL (b) WIKI

Figure 8: maximum err vs. max sketch size.



(a) RAIL (b) WIKI

Figure 9: update cost vs. sketch size.

WIKI is the text corpus built on the article dump of the Sep 2015 version of English WIKIpedia[3]. We preprocessed the data to remove the stop-words and insignificant articles. We used words occurring at least 1000 times in the entire corpus as features (columns), and selected articles with at least 500 features as rows. The entry at row i and column j is the *tf-idf weighting* of word j in article i. Each row is associated with a timestamp, which is the time that this article is published. This matrix consists of 7047 columns and 68,319 rows. The timestamps of WIKI spans over several years. We view one day as the time unit, set the window size to be 578 such that on average there are approximately 10,000 rows in a window. The maximum number of rows in a window is 62,125.

RAIL[4] is the crew scheduling matrix for the Italian railways. The entry at row i and column j is the integer cost for assigning crew i to cover trip j. This matrix contains 2586 columns and 923,269 rows, with 8,011,362 nonzero entries. We added synthetic timestamps RAIL to create a time-based stream that follows the *Poisson arrival model*, that is, the timestamps of the updates follow Poisson distribution with $\lambda = 0.5$. We set the window size to be 5,000 such that on average there are approximately 10,000 rows in a window. The maximum number of rows in a window is 10,347.

Figures 7 and 8 illustrate the the tradeoffs between the max sketch size and the average error, and between the sketch size and the maximum error, respectively. Figure 9 shows the tradeoffs between sketch size and update cost. The results suggest that the performance of the sampling and LM-FD on time-based windows is consistent with their performance on sequence-based windows. Figures 7 and 8 show that LM-FD achieved the best error-space tradeoff, followed by the two sampling algorithms.

Figure 9 shows that LM-FD outperforms the sampling algorithms in terms of the update cost. This concurs with the results for sequence-based windows. However, Figure 9b suggests that the advantage of LM-FD is less obvious on the WIKI data set. This is due to the fact that in English WIKIpedia, articles are published more frequently in recent time. Thus a fixed time interval many contain very few rows at the beginning, and the queues in the sampling algorithms may be relatively small and efficient to update.

Data Set	rows n	d	Δ	N_W	ratio R
WIKI	68,319	7047	578	$\approx 10,000$	422.81
RAIL	923,269	2586	5000	$\approx 10,000$	12

Table 3: Data Sets for time-based sliding window.

8.3 Remarks

We conclude our experimental evaluations with a few remarks. In general, the sampling algorithms performs not so well comparing to the DI-FD and the LM-FD algorithm. However, the sampling algorithms provide the desirable property that the final sketch is interpretable. Inside the family of sampling algorithms, SWR is preferred over SWOR and SWOR-ALL. We recommend LM-FD for other general analytic task, due to its overall superior efficiency in terms of both space and update cost, and to its applicability on both sequence-based and time-based windows. Finally, if the space efficiency is the main concern, the norms are strictly bounded (e.g. normalized stream), and sequenced-based window is used, DI-FD has the best performance.

We also note that the performance of our sketches is relatively stable over time, which means the overall data set size is not a critical factor in evaluating the performance of our algorithms. What really matters is the ratio R between max row norm and min row norm in a sliding window. That said, in practice, the sketch size of our design is typically much smaller than our theoretical bounds' dependence on R or $\log R$ (See the R value of PAMAP in Table 2). Finally, improving the update cost for the DI-FD algorithm is an interesting open problem.

9. CONCLUSION

This paper gives the first treatment to the sliding window sketching problem. We presented efficient algorithms for both time-based and sequence-based windows, and explored various constructions based on either sampling or embedding streaming matrix sketches (such as frequent direction) into a sliding-window data summary. We provided both formal theoretical guarantees, in terms of update cost and space–approximation error tradeoff, and extensive experimental evaluations over a large collection of real and synthetic data sets that have demonstrated the excellent performance of our algorithms in practice. Directions for future work include, but not limited to, deriving tighter theoretical bounds for some of our sketching algorithms, extending them to handle distributed data, and understanding their behaviors in different error metrics.

[3] https://en.wikipedia.org/wiki/WIKIpedia:Database_download

[4] http://www.cise.ufl.edu/research/sparse/matrices/Mittelmann/rail2586.html

10. REFERENCES

[1] D. Achlioptas. Database-friendly random projections. In *PODS*, pages 274–281. ACM, 2001.

[2] P. K. Agarwal, G. Cormode, Z. Huang, J. M. Phillips, Z. Wei, and K. Yi. Mergeable summaries. *TODS*, 38(4):26, 2013.

[3] A. Arasu and G. S. Manku. Approximate counts and quantiles over sliding windows. In *PODS*, pages 286–296. ACM, 2004.

[4] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom. Models and issues in data stream systems. In *PODS*, pages 1–16. ACM, 2002.

[5] B. Babcock, M. Datar, and R. Motwani. Sampling from a moving window over streaming data. In *SODA*, pages 633–634. SIAM, 2002.

[6] R. Badeau, G. Richard, and B. David. Sliding window adaptive svd algorithms. *IEEE Transactions on Signal Processing*, 52(1):1–10, 2004.

[7] C. Boutsidis, D. Garber, Z. Karnin, and E. Liberty. Online principal components analysis. In *SODA*, pages 887–901. SIAM, 2015.

[8] K. L. Clarkson and D. P. Woodruff. Low rank approximation and regression in input sparsity time. In *STOC*, pages 81–90, 2013.

[9] G. Cormode, M. Garofalakis, P. J. Haas, and C. Jermaine. Synopses for massive data: Samples, histograms, wavelets, sketches. *Foundations and Trends in Databases*, 4(1–3):1–294, 2012.

[10] G. Cormode, S. Muthukrishnan, K. Yi, and Q. Zhang. Continuous sampling from distributed streams. *JACM*, 59(2):10, 2012.

[11] M. Datar, A. Gionis, P. INDYK, and R. Motwani. Maintaining Stream Statistics over Sliding Windows. *SIAM Journal on Computing*, 31(6):1794–1813, Jan. 2002.

[12] A. Deshpande and L. Rademacher. Efficient volume sampling for row/column subset selection. In *FOCS*, pages 329–338, 2010.

[13] P. Drineas, A. Frieze, R. Kannan, S. Vempala, and V. Vinay. Clustering large graphs via the singular value decomposition. *Machine learning*, 56(1-3):9–33, 2004.

[14] P. Drineas and R. Kannan. Pass efficient algorithms for approximating large matrices. In *SODA*, pages 223–232, 2003.

[15] P. Drineas, R. Kannan, and M. W. Mahoney. Fast monte carlo algorithms for matrices i: Approximating matrix multiplication. *SIAM Journal on Computing*, 36(1):132–157, 2006.

[16] P. S. Efraimidis and P. G. Spirakis. Weighted random sampling with a reservoir. *Information Processing Letters*, 97(5):181–185, 2006.

[17] A. Frieze, R. Kannan, and S. Vempala. Fast monte-carlo algorithms for finding low-rank approximations. *JACM*, 51(6):1025–1041, 2004.

[18] R. Gemulla and W. Lehner. Sampling time-based sliding windows in bounded space. In *SIGMOD*, pages 379–392. ACM, 2008.

[19] M. Ghashami, A. Desai, and J. M. Phillips. Improved practical matrix sketching with guarantees. In *ESA*, pages 467–479. Springer, 2014.

[20] M. Ghashami and J. M. Phillips. Relative errors for deterministic low-rank matrix approximations. In *SODA*, pages 707–717. SIAM, 2014.

[21] M. Ghashami, J. M. Phillips, and F. Li. Continuous matrix approximation on distributed data. *VLDB*, 7(10):809–820, 2014.

[22] Y. Jiao. Maintaining stream statistics over multiscale sliding windows. *TODS*, 31(4):1305–1334, 2006.

[23] A. Lakhina, M. Crovella, and C. Diot. Diagnosing network-wide traffic anomalies. In *ACM SIGCOMM Computer Communication Review*, pages 219–230. ACM, 2004.

[24] A. Lakhina, K. Papagiannaki, M. Crovella, C. Diot, E. D. Kolaczyk, and N. Taft. Structural analysis of network traffic flows. In *SIGMETRICS*, 2004.

[25] E. Liberty. Simple and deterministic matrix sketching. In *SIGKDD*, pages 581–588. ACM, 2013.

[26] Z. Longbo, L. Zhanhuai, Z. Yiqiang, Y. Min, and Z. Yang. A priority random sampling algorithm for time-based sliding windows over weighted streaming data. In *SAC*, pages 453–456. ACM, 2007.

[27] G. S. Manku and R. Motwani. Approximate frequency counts over data streams. In *VLDB*, pages 346–357, 2002.

[28] J. Misra and D. Gries. Finding repeated elements. *Science of computer programming*, 2:143–152, 1982.

[29] C. Papadimitriou, P. Drineas, and M. Magdon-Ismail. Near optimal column-based matrix reconstruction. In *FOCS*, pages 305–314. IEEE, 2011.

[30] S. Papadimitriou, J. Sun, and C. Faloutsos. Streaming pattern discovery in multiple time-series. In *VLDB*, pages 697–708, 2005.

[31] S. Papadimitriou and P. Yu. Optimal multi-scale patterns in time series streams. In *SIGMOD*, pages 647–658. ACM, 2006.

[32] O. Papapetrou, M. Garofalakis, and A. Deligiannakis. Sketching distributed sliding-window data streams. *The VLDB Journal*, 24(3):345–368, 2015.

[33] A. A. Qahtan, B. Alharbi, S. Wang, and X. Zhang. A pca-based change detection framework for multidimensional data streams: Change detection in multidimensional data streams. In *SIGKDD*, pages 935–944. ACM, 2015.

[34] T. Sarlos. Improved approximation algorithms for large matrices via random projections. In *FOCS*, pages 143–152. IEEE, 2006.

[35] Q. Song, J. Cheng, and H. Lu. Incremental matrix factorization via feature space re-learning for recommender system. In *ACM Conference on Recommender Systems*, pages 277–280. ACM, 2015.

[36] S. K. Tanbeer, C. F. Ahmed, B.-S. Jeong, and Y.-K. Lee. Sliding window-based frequent pattern mining over data streams. *Information sciences*, 179(22):3843–3865, 2009.

[37] Y. Tao and D. Papadias. Maintaining sliding window skylines on data streams. *TKDE*, 18(3):377–391, 2006.

[38] K. Weinberger, A. Dasgupta, J. Langford, A. Smola, and J. Attenberg. Feature hashing for large scale multitask learning. In *ICML*, pages 1113–1120. ACM, 2009.

[39] D. P. Woodruff. Sketching as a tool for numerical linear algebra. *Theoretical Computer Science*, 10(1-2):1–157, 2014.

APPENDIX

A. OTHER LM AND FD SKETCHES

In this section, we show how the Logarithmic Method and Dyadic Interval frameworks can cope with other streaming matrix sketches. In particular, we combine the *random projection* and *Hashing* algorithms with the DI framework to obtain the DI-RP and DI-HASH algorithms, and the *Hashing* algorithms with the LM framework to obtain the LM-HASH algorithm.

Random projection. Let R be an $\ell \times n$ random projection matrix such that each cell is a random value $r_{i,j}$ selected from $\{-1/\sqrt{\ell}, 1/\sqrt{\ell}\}$ uniformly. The approximation matrix B is equivalent to the matrix RA. Note that B contains the d columns of A randomly projected from dimension n to dimension ℓ. This is easily computed in a streaming fashion with the following algorithm. On receiving the i-th row a_i, we construct a random vector $r \in R^{d\times 1}$ such that the j-th entry $r(j)$ of r is randomly selected from $\{-1/\sqrt{\ell}, 1/\sqrt{\ell}\}$. Then we set $B \leftarrow B + r \cdot a_i$. By setting $\ell = d/\varepsilon^2$, RP achieves covariance error **cova-err**$(A, B) \leq \varepsilon$ with space

usage $O(d/\varepsilon^2)$ space and $O(d^2/\varepsilon^2)$ update cost. For proofs of correctness and space usage see [29, 39]

Hashing. HASH is often used in practice by the machine learning community and is referred to as "feature hashing" or "hashing trick" [31] and we simply denote it as HASH. For initiation, we choose a 2-wise independent hash function $h : [n] \to [\ell]$ and a pair-wise independent hash function $g : [n] \to \{-1, +1\}$. Set $S \in R^{\ell \times n}$ to be a random sparse matrix with exactly 1 non-zero entry in each column. More precisely, S satisfies $s_{ij} = g(j)$ for $i = h(j)$, and $s_{ij} = 0$ otherwise. The approximation matrix B equals SA. HASH be be computed in a streaming fashion with the following algorithm. On receiving row a_i, we set the $h(i)$-th row of B to be $b_{h(i)} \leftarrow b_{h(i)} + g(i)a_i$. By setting $\ell = d^2/\varepsilon^2$, HASH achieves covariance error **cova-err**$(A, B) \le \varepsilon$ with space usage d^2/ε^2 space and d update cost. For proofs of correctness and space usage see [8, 39].

It is easy to see that HASH is mergeable under sliding window model. Given two HASH sketches $B_1 = \mathsf{HASH}(A_1)$ and $B_2 = \mathsf{HASH}(A_2)$ with the same error parameter ε and the same hash function h and g, the merging algorithm is $\mathsf{merge}(B_1, B_2) = B_1 + B_2$. It is easy to see that $C = \mathsf{merge}(B_1, B_2)$ is a HASH sketch for $[B_1, B_2]$ with the same error parameter and space.

LM-HASH. This algorithm uses the HASH sketch for each block in the LM framework. Recall that given error parameter ε, the HASH sketch achieves **cova-err** $\le \varepsilon/8$ with high probability using $\ell = O(d^2/\varepsilon^2)$ rows and $u = O(d)$ update cost. By Theorem 6.1, we have:

Corollary A.1 *The LM-HASH algorithm achieves* **cova-err** $\le \varepsilon$ *with high probability on any given window using* $O(\frac{d^2}{\varepsilon^3} \log \varepsilon NR)$ *space and* $O(d \log \varepsilon NR)$ *amortized update cost.*

DI-RP. This algorithm uses the random projection sketch for each block in the DI framework. Recall that given error parameter ε, the PR sketch uses $\ell_\varepsilon = O(d/\varepsilon^2)$ rows and process an update with $u_\varepsilon = O(d^2/\varepsilon^2)$ amortized cost. By Theorem 7.1, we have the following corollary:

Corollary A.2 *The DI-RP algorithm use* $O(\frac{Rd}{\varepsilon^2} \log \frac{R}{\varepsilon})$ *space and process an update with* $O(\frac{d^2}{\varepsilon^2} \log \frac{R}{\varepsilon})$ *amortized cost.*

DI-HASH. This algorithm uses hash for each block in the DI framework. Recall that given error parameter ε, the HASH sketch achieves **cova-err** $\le \varepsilon/8$ with high probability using $\ell_\varepsilon = O(\frac{d^2}{\varepsilon^2})$ rows and $u = O(d)$ update cost. By Theorem 7.1, we have the following corollary:

Corollary A.3 *The DI-RP algorithm use* $O(\frac{Rd^2}{\varepsilon^2} \log \frac{R}{\varepsilon})$ *space and process an update with* $O(d \log \frac{R}{\varepsilon})$ *amortized cost.*

B. PROOFS OF TWO NEGATIVE RESULTS

B.1 Proof of Theorem 4.1

We reduce the INDEX problem to the matrix sketching problem. In an instance in the INDEX problem, a party Alice receives an n-bit string x and another party Bob receives an index $i \in [n]$. Alice sends some information to Bob, and Bob computes the i-th bit of x using these information. The *communication cost* of this problem is the number of bits sent by Alice. It is easy to see that INDEX problem can be trivially solved with $O(n)$ communication cost by sending x to Bob. A well-know lower bound in communication complexity literature is that INDEX problem also requires $\Omega(n)$ communication cost, which indicates that Alice essentially has to send all bits of x to Bob. The lower bound also holds for probabilistic protocol, that is, even if Bob is only required to compute the correct answer with probability 2/3, the communication cost is still $\Omega(n)$.

The basic idea of the reduction is to set $n = Nd$, and construct a row-update stream based on the instance in the INDEX problem. We then show that if we have a matrix sketch that uses $o(Nd)$ space and maintains $A^T A$ exactly on any window, then we can use this sketch to solve the INDEX problem with $o(Nd)$ communication cost. This would contradict the communication lower bound of INDEX problem, which implies that we need $\Omega(Nd)$ space for the matrix sketching problem in the sliding window model.

Proof of Theorem 4.1. Suppose there is a matrix sketching algorithm κ that tracks $A^T A$ exactly on a sliding window with $o(Nd)$ space. Given an instance of the INDEX problem with bit string x and index i, Alice will construct a stream A of size N as follow. She divides x into N chunks, such that the j-th chunk $x_{(j)}$ is a bit vector of dimension d, for $j = 1, \ldots, N$. Then we let j-th row of A to be $x_{(j)}$.

Then, Alice runs the sketching algorithm κ on A, and send the memory state $\kappa(A)$ to Bob after all N rows are processed. Given the index i, Bob will first compute $k = \lceil i/N \rceil$ and $r = i - N(k-1)$. In other word, index i the entry $A_{k,r}$. To compute the value of this entry, Bob will continue running $\mathcal{A}$ with N new updates, with each update being $e_{r+1} \in \mathbb{R}^d$ is the $(r+1)$-th unit vector. Let $A(j, N+j-1)$ denote the matrix that is made up by row j to $N+j-1$. Since κ can maintain $A^T A$ exactly, Bob can compute $\|A(j, N+j-1)x\|^2 = x^T A(j, N+j-1)^T A(j, N+j-1)x$ exactly for any vector x, at the time of the j-th update. In particular, Bob will compute $y_j = \|A(j, N+j-1)e_r\|^2$, for $j = 1, \ldots, N$. Since $\|A(j, N+j-1)e_r\|^2$ is the sum of the r-th column of $A(j, N+j-1)$, and the r-th column has not been updated since Bob received it (all updates were e_{r+1}), it follows that $y_j = \sum_{m=1}^{N-j+1} A_{mr}$, where A_{mr} denotes the entry of A at row m and column r. Therefore Bob can recover x_i by $x_i = A_{k+1,r} = y_{N-k} - y_{N-k-1}$. This proves that a sliding window matrix sketch that maintains $A^T A$ exactly with $o(Nd)$ space can be used to construct a communication protocol to solve the INDEX problem with sublinear space, which leads to a contradiction. □

Remark. It is unclear if the $\Omega(Nd)$ lower bound holds for $\varepsilon > 1/d$. In fact, for $\varepsilon > 1/\sqrt{d}$, there exists an $O(N/\varepsilon^2)$ method as follows: We compute an AMS sketch S_i of size $1/\varepsilon^2$ for each row A_i in the window. We can view S_i as a vector, such that $\left| \|S_i\|^2 - \|A_i\|^2 \right| \le \varepsilon \|A_i\|^2$. This implies that we can use S_i to approximate A_i with additive error $\varepsilon \|A_i\|^2$. Hence, the total error for using $S = [S_1; \ldots; S_N]$ to approximate the sliding window matrix A is bounded by $\varepsilon |A|_F^2$. (due to the fact that $\|A\|_F^2 = \sum_{i=1}^N \|A_i\|^2$). So when $\varepsilon \ge 1/\sqrt{d}$, the lower bound in Theorem 4.2 does not hold. It is unclear if the lower bound holds for $1/d < \varepsilon < 1/\sqrt{d}$. Proving an $\Omega(Nd)$ lower bound or coming up with a better upper bound for this range is an interesting open problem.

B.2 Proof of Theorem 4.2

We reduce the d-Majority INDEX problem to sliding window sketching problem with unbounded norms. Setting $n =$

Nd, the d-Majority INDEX problem is identical to the INDEX problem with only one modification: if we divide the bit vector x into N consecutive chunks, then each chunk must contain at least a half of 1 bits. In other word, for $i = 1, \ldots, N$, at least $d/2$ bits in $x_i, x_{i+1}, \ldots, x_{i+d-1}$ are 1's. We first show that the d-Majority INDEX Problem also requires $\Omega(Nd)$ communication complexity to solve.

Lemma B.1 *A one-way communication protocol that solves the d-majority index problem must use $\Omega(Nd)$ communication cost, for d sufficiently large.*

PROOF. We reduce the d-majority index problem to index problem. Assume that we have a communication protocol $\mathcal{P}$ that solves the d-majority index problem with $o(Nd)$ communication cost, we will use it to solve the index problem with sublinear communication cost. Given an instance of the index problem, we can construct an instance of the d-majority problem as follow. We set d sufficiently large and divide x into n/d chucks, each of size d. Let $x_{(j)}$ denote the bit vector corresponding to the j-th chuck. We notice that the number of 1's in $x_{(j)}$ is either larger than $d/2$ or at most $d/2$. Alice will generate a new bit vector $x'_{(j)}$ such that $x'_{(j)} = \mathbf{1} - x_{(j)}$ if the number of 1's in $x_{(j)}$ is more than $d/2$, and $x'_{(j)} = x_{(j)}$ otherwise. Alice will also generate a bit vector z such that $z_j = 1$ if $x'_{(j)} = \mathbf{1} - x_{(j)}$ and $z_j = 0$ otherwise. Finally, Alice will concatenate all $x'_{(j)}$ to generate a bit vector x', and send $\mathcal{P}(x')$ and z to Bob. Since x' is an instance of the d-Majority INDEX problem, Bob is able to figure out the i-th component x'_i of x' using $\mathcal{P}(x')$. Then Bob will return $x_i = x'_i$ if $z_{\lceil i/d \rceil} = 0$ and $x_i = 1 - x'_i$ if $z_{\lceil i/d \rceil} = 1$. This protocol correctly solves the INDEX problem using communication cost $C(\mathcal{P}) + O(n/d) = o(n/d * d) + O(n/d) = o(n)$. This contradicts with the $\Omega(n)$ communication complexity lower bound for the INDEX problem. □

PROOF OF THEOREM 4.2. Assume that κ is a sliding window matrix sketch that uses $o(Nd)$ space. Then for any window W, κ cannot produce a matrix B_W such that

$$\mathbf{Pr}\left[\|A_W^T A_W - B_W^T B_W\| \leq \frac{1}{4d}\|A\|_F^2\right] > \frac{2}{3}.$$

We will show how to use κ to construct a communication protocol that solves d-Majority INDEX efficiently.

Assume that x is an instance of the d-Majority INDEX problem. Recall that x is a bit vector of size Nd divided into N chucks, such that the i-th chuck $x_{(i)}$ is a bit vector of size d, and consists of at least $d/2$ 1's. Alice will construct a matrix A with N rows, such that the i-th row $A_{(i)} = x_{(i)}/8^i$, where $x_{(i)}$ is the i-th chuck of x. Recall that $x_{(i)}$ is a bit vector of size d, with at least $d/2$ 1's. Alice will run κ on A and send the memory content $\kappa(A)$ to Bob.

Given the index i, Bob will compute $k = \lceil i/N \rceil$ and $r = i - N(k-1)$ such that $x_i = 8^k A_{k,r}$. To compute this entry, Bob will continue running κ with N new updates, with the $(N + j) - th$ update being $e_{r+1}/8^{N+j}$, for $j = 1, \ldots, N$. We let A_k denote the matrix that is made up by row k to $N + k - 1$. Since κ can provide an approximation B for A_k with error guarantee

$$\mathbf{Pr}\left[\|A_k^T A_k - B^T B\| \leq \frac{1}{8d}\|A_k\|_F^2\right] > \frac{2}{3},$$

it follows that for any fixed $y \in \mathcal{R}^d$,

$$\mathbf{Pr}\left[|\|A_k y\|^2 - \|By\|^2| \leq \frac{1}{8d}\|A_k\|_F^2\right] > \frac{2}{3}.$$

In particular, Bob will set $y = e_r$ so that he can use $\|Be_r\|^2$ to approximate $\|A_k e_r\|^2$ with additive error at most $\frac{1}{8d}\|A_k\|_F^2$, with probability 2/3. Observe that

$$\left|\|A_k e_r\|^2 - A_{k,r}\right| = \left|\sum_{j=k}^{N} A_{j,r} - A_{k,r}\right| = \left|\sum_{j=k+1}^{N} A_{j,r}\right|.$$

Since each $|A_{j,r}|$ is bounded by 8^{-j}, we have

$$\left|\|A_k e_r\|^2 - A_{k,r}\right| \leq \sum_{j=k+1}^{N} 8^{-j} \leq \frac{8^{-k}}{7}.$$

This implies that $\|A_k e_r\|^2$ can approximate $A_{k,r}$ with additive error at most $8^{-k}/7$. On the other hand, $\|Be_r\|^2$ can approximate $\|A_k e_r\|^2$ with probability 2/3 and additive error

$$\frac{1}{8d}\|A_k\|_F^2 \leq \frac{1}{8d}\sum_{j=k}^{\infty} d/8^j = \frac{8^{-k}}{7}.$$

It follows that with probability 2/3, $\|Be_r\|^2$ is able to approximate $A_{k,r}$ with additive error $(2/7)8^{-k}$. Since $A_{k,r} = x_i/8^k$ is either 0 or 8^{-k}, Bob is able to recover $A_{k,r}$ with probability 2/3. This contradicts to the probabilistic lower bound for d-Majority INDEX problem, and thus the Theorem follows. □

C. PROOF FOR SAMPLING ALGORITHMS

Consider a window W with row $a_1, \ldots, a_N$ in it. Let $w_1 = \|a_1\|^2, \ldots, w_N = \|a_N\|^2$ denote the squared norms in the window, with their order sorted according the receiving time. We have $1 \leq w_i \leq R$ for $i \in [N]$. Let $w_\ell(j)$ denote the summation of the top -$(N - j - \ell + 1)$ smallest weights from $\{w_{j+1}, \ldots, w_N\}$, that is, $w_j(\ell)$ is the summation of $w_{j+1}, \ldots, w_N$, excluding the top-$(\ell - 1)$ weights. We define $w(j) = w_1(j) = \sum_{k=j+1}^{N} w_k$. We first present a technical Lemma.

Lemma C.1 *Suppose $w_i \leq R$ for $i = 1, \ldots, N$, then the following inequality holds*

$$\sum_{i=1}^{N-\ell+1} \frac{w_i}{w_i + w_\ell(i)} = O(\ell \log NR).$$

PROOF. We first claim that that once ℓ is fixed, $w_\ell(i)$ is strictly monotonically decreasing with i for $1 \leq i \leq N-\ell+1$. For a proof consider $w_\ell(i) - w_\ell(i+1)$. If w_{i+1} is not one of the top-$(\ell-1)$ weights in $w_{j+1}, \ldots, w_N$, then $w_\ell(i) - w_\ell(i+1) = w_{j+1} > 0$. Otherwise, there is a weight $w_t < w_{j+1}$ that is in top-$(\ell-1)$ for $w_{j+2}, \ldots, w_N$, and $w_\ell(i) - w_\ell(i+1) = w_t > 0$. Either way we have $w_\ell(i) - w_\ell(i+1) > 0$, and $w_\ell(i)$ is monotonically decreasing with i.

By the monotonicity of $w_\ell(i)$, we define i_k to be largest index such that $w_\ell(i_k) \geq 2^k$, and let L be the number of different k's. Since $w_\ell(0) \leq NR$, we have $L \leq \log NR + 1$.

Therefore

$$\sum_{i=1}^{N-\ell+1} \frac{w_i}{w_i + w_\ell(i)} \le \sum_{k=0}^{L} \sum_{i=i_k+1}^{i_{k-1}} \frac{w_i}{w_i + w_\ell(i)} \le \sum_{k=0}^{L} \sum_{i=i_k+1}^{i_{k-1}} \frac{w_i}{w_i + 2^{k-1}}.$$

The last inequality uses the fact that $w_\ell(i) \ge 2^{k-1}$ for i between $i_k + 1$ and i_{k-1}. We have

$$\sum_{i=1}^{N-\ell+1} \frac{w_i}{w_i + w_\ell(i)} \le \sum_{k=1}^{L} \frac{\sum_{i=i_k+1}^{i_{k-1}} w_i}{2^{k-1}} \le \log \ell R + \sum_{k=\log \ell R+1}^{L} \frac{\sum_{i=i_k+1}^{N} w_i}{2^{k-1}}.$$

Finally, note that $w_\ell(i) \le 2^k$ for $i \ge i_k$, and $\sum_{i=i_k+1}^{N} w_i \le w_\ell(i) + \ell R$, it follows that

$$\sum_{i=1}^{N-\ell+1} \frac{w_i}{w_i + w_\ell(i)} \le \log \ell R + \sum_{k=\log \ell R+1}^{L} \frac{2^{k+1} + \ell R}{2^k} \le \log \ell R + L + 1 = O(\log NR),$$

and the Lemma follows. □

Lemma C.2 *The probability that ρ_j is among the top-ℓ largest priorities in $\{\rho_j, \ldots, \rho_N\}$ is at most $\frac{\ell w_j}{w_j + w_j(\ell)}$*

PROOF. Consider the sampling without replacement process on $\{j, \ldots, N\}$. At each round before j is sampled, let S be the set of indices that gets sampled. Since $w_j(\ell)$ is the summation of $w_{j+1}, \ldots, w_N$ excluding the top-$(\ell - 1)$ weights, the probability that j gets sampled in this round is

$$\frac{w_j}{\sum_{s=j}^{N} w_s - \sum_{s \in S} w_s} \le \frac{w_j}{w_j + w_j(\ell)}.$$

Therefore, the probability that j does not get sampled is lower bounded by

$$\left(1 - \frac{w_j}{w_j + w_j(\ell)}\right)^k \ge 1 - \frac{k w_j}{w_j + w_j(\ell)},$$

which implies that the probability the j gets sampled is upper bounded by $\frac{k w_j}{w_j + w_j(\ell)}$. □

C.1 Proof of Lemma 5.1

PROOF. For sampling with replacement, we only need to prove that the expected number of rows in a sampler is $O(\log NR)$. Recall that a_i is in Q if and only if w_i is the largest weight among $w_i, \ldots, w_N$. By Lemma C.2, the probability that w_i is the largest weight among $w_i, \ldots, w_N$ is $p_i = w_i / \sum_{j=i}^{N} w_j$. Therefore the expected number of rows in the priority queue Q is

$$\sum_{i=1}^{N} p_i = \sum_{i=1}^{N} \frac{w_i}{\sum_{j=i}^{N} w_j} \le O(\log NR). \quad (2)$$

The last inequality is by setting $\ell = 1$ in Lemma C.1. □

C.2 Proof of Lemma 5.2

PROOF. By Lemma C.2, the probability that a row a_j is a candidate row is at most $\frac{\ell w_j}{w_j + w_j(\ell)}$. Thus the expected number of candidate rows is at most

$$\ell + \sum_{j=1}^{N-\ell+1} \frac{\ell w_j}{w_j + w_j(\ell)} = O(\ell \log NR).$$

The last inequality is by Lemma C.1. □

C.3 Proof of Theorem 5.1

PROOF. Assuming the algorithms know the value of $\|A\|_F^2$, the bounds of the space and update bounds directly follow from Lemma 5.1 and Lemma 5.2. Now suppose we use the approximation of $\|A\|_F^2$ from an Exponential Histogram with relative error ε. Let B be the matrix that uses $\|A\|_F^2$ for rescaling parameter, and B' be the matrix that uses the approximation of EH for rescaling parameter. It is easy to see that $B' = (1 + \alpha)B$, where $\alpha \in (-\varepsilon, \varepsilon)$. Since $\|B^T B - A^T A\| \le \varepsilon \|A\|_F^2$ and $\|B\|_F^2 = \|A\|_F^2$, it follows that

$$\|B'^T B' - A^T A\| \le \|B'^T B' - B^T B\| + \|B^T B - A^T A\| \le 3\varepsilon \|A\|_F^2,$$

and the Theorem follows. □

D. GENERATION OF SYNTHETIC

The matrix of SYNTHETIC was generated by formula $A = SDU + N/\zeta$. S is a $n \times d$ signal co-efficients matrix with each entry drawn from standard normal distribution. D is a diagonal matrix with $D_{i,i} = 1 - (i-1)/d$. U is a signal row space matrix that satisfies $UU^T = I_d$. The matrix N contributes additive Gaussian noise with $N_{i,j}$ drawn from $\mathcal{N}(0, 1)$. We set $\zeta = 10$ so that the signal SDU is recoverable.