Approximate Matrix Multiplication over Sliding Windows

Ziqi Yao*
East China Normal University
Shanghai, China
51265902073@stu.ecnu.edu.cn

Lianzhi Li*
East China Normal University
Shanghai, China
51265902102@stu.ecnu.edu.cn

Mingsong Chen
East China Normal University
Shanghai, China
mschen@sei.ecnu.edu.cn

Xian Wei
East China Normal University
Shanghai, China
xwei@sei.ecnu.edu.cn

Cheng Chen†
East China Normal University
Shanghai, China
chchen@sei.ecnu.edu.cn

ABSTRACT

Large-scale streaming matrix multiplication is very common in various applications, sparking significant interest in develop efficient algorithms for approximate matrix multiplication (AMM) over streams. In addition, many practical scenarios require to process time-sensitive data and aim to compute matrix multiplication for most recent columns of the data matrices rather than the entire matrices, which motivated us to study efficient AMM algorithms over sliding windows. In this paper, we present two novel deterministic algorithms for this problem and provide corresponding error guarantees. We further reduce the space and time costs of our methods for sparse matrices by performing an approximate singular value decomposition which can utilize the sparsity of matrices. Extensive experimental results on both synthetic and real-world datasets validate our theoretical analysis and highlight the efficiency of our methods.

CCS CONCEPTS

• **Mathematics of computing → Computations on matrices**.

KEYWORDS

Streaming data, Sliding window, Approximate Matrix Multiplication

ACM Reference Format:
Ziqi Yao, Lianzhi Li, Mingsong Chen, Xian Wei, and Cheng Chen. 2024. Approximate Matrix Multiplication over Sliding Windows. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '24), August 25–29, 2024, Barcelona, Spain.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3637528.3671819

*Both authors contributed equally to this research.
†Corresponding author

KDD '24, August 25–29, 2024, Barcelona, Spain.

ACM ISBN 979-8-4007-0490-1/24/08
https://doi.org/10.1145/3637528.3671819

1 INTRODUCTION

Matrix multiplication is one of the most fundamental subroutines in machine learning and data mining. It has been applied to many machine learning tasks such as regression [26, 35], clustering [8, 12], online learning [1, 14, 21, 33] and canonical component analysis [16, 36]. The ever-increasing size of data matrices poses great challenges to the computational efficiency of matrix multiplication, and reinforces efforts to design efficient approximate matrix multiplication (AMM) methods.

Many recent works [13, 20, 24, 32, 35, 36] consider AMM in the streaming model, where columns of the data matrices arrive sequentially. Among these methods, Co-occurring Directions (COD) and its variants enjoy smaller approximation errors and higher robustness. Despite the success of COD in streaming AMM, it does not fully address the situation where the timeliness of data is very important, i.e., people are more interested in the recent data rather than the outdated data. Such setting is very common in many real-world applications. For example, in social media analysis, each column of the data matrix corresponds to a document (e.g. the content of a Twitter post) along with a corresponding timestamp. Usually, advertisers are more interested in the tweets posted in the most recent week or month. In user behavior analysis, the correlation between user searches and ad clicks in the most recent period are more attractive to researchers.

In these time-sensitive applications, the sliding window model is more appropriate than the unbounded streaming model [3, 6, 7, 23, 27, 34]. The sliding window model [10] only considers a *window* of size N, i.e., the N most recent columns of the data matrices. We call the data in the window *active data*, in contrast to the so-called *expired data*. In this paper, we study AMM algorithms over sliding windows. Specifically, We have two streaming matrices $\mathbf{X} \in \mathbb{R}^{m_x \times n}$ and $\mathbf{Y} \in \mathbb{R}^{m_y \times n}$. The algorithm sequentially receives the columns of matrices $\mathbf{X}$ and $\mathbf{Y}$ and maintains two low-rank sketches $\mathbf{A} \in \mathbb{R}^{m_x \times \ell}$ and $\mathbf{B} \in \mathbb{R}^{m_y \times \ell}$, where the number of columns ℓ is significantly smaller than n. The objective of the algorithm is to guarantee that $\mathbf{AB}^T$ is a good approximation of $\mathbf{XY}^T$ (i.e., $\mathbf{AB}^T \approx \mathbf{XY}^T$) while minimizing the algorithm's space and time cost. Though streaming AMM has been widely studied, few works explore the AMM over sliding windows. Generally, computing matrix sketching in the sliding window model is harder that in the streaming model. Wei et al. [34] showed that if $\mathbf{X} = \mathbf{Y}$, maintaining $\mathbf{XY}^T$ only requires $O(m_x m_y)$ time and $O(m_x m_y)$ space but tracking $\mathbf{XY}^T$ exactly in the sliding window models requires the algorithm

to store all columns in the window. Thus developing algorithms for AMM in the sliding window model requires novel techniques.

In this paper, we provide two algorithms for AMM over sliding windows, called Exponential Histogram Co-occurring Directions (EH-COD) and Dyadic Interval Co-occurring Directions (DI-COD). The EH-COD algorithm, based on the idea of Exponential Histogram [10], stores stream data in blocks and maintains an approximation of the window content with space much smaller than the window size through regular merging. The DI-COD algorithm, based on the dyadic interval structure [2], constructs a tree-like structure from the stream data blocks, and concatenate sketches from each layer to generate the final approximation matrix. The DI-COD algorithm has better approximation quality than the EH-COD algorithm when the maximum column norm of the data matrix is small. We further improve the time and space costs of our methods for sparse matrices by leveraging the subspace power method (SPM) [25, 35] which can utilize the sparsity of matrices. Our main contributions can be summarized as follows:

- We develop two efficient algorithms for AMM in the sliding window models. We provide error analysis for both algorithms as well as their time and space complexities.
- We propose improved algorithm for approximate sparse matrix multiplication in the sliding window models.
- We empirically study the performance of proposed methods on both synthetic and real-world datasets. The experimental results validate the effectiveness of the proposed approaches.

Paper Organization. The rest of the paper is organized as follows. In Section 2, we introduce previous works related to this paper. Then we define the notation used in this paper and introduce the Co-occurring directions algorithm in Section 3. In Section 4, we presents our EH-COD and DI-COD algorithms and show their approximation error bound and complexity analysis. In Section 5, we describe the improved variants of our methods for sparse matrices. We present our empirical results in Section 6 and provide conclusions in Section 7.

2 RELATED WORKS

In this section, we first review prior works on approximate matrix multiplication. Then we introduce sketching methods in the sliding window model.

2.1 Approximate Matrix Multiplication

The AMM problem has been widely studied in recent decades. Existing approaches to the AMM problem can be broadly categorized into two types: randomized algorithms, exemplified by random sampling [13] and random projection [9, 22, 31], and deterministic algorithms, represented by FD-AMM [36] and COD [24]. Randomized algorithms exhibit favorable time complexity and come with theoretical guarantees, yet these methods require larger sketch size to achieve same approximation error as deterministic methods. The deterministic algorithms leverage singular value decomposition (SVD) to generate approximate matrices and usually enjoys smaller approximation errors and better adaptability to the streaming setting. The FD-AMM [36] algorithm employs the Frequent Directions (FD) [15, 18] to process the stacked matrix $\mathbf{Z} = [\mathbf{X}; \mathbf{Y}]$ column by column, where $\mathbf{X}$ and $\mathbf{Y}$ are the input matrices. The COD algorithm computes the correlation sketch by shrinking the singular values of $\mathbf{X}$ and $\mathbf{Y}$ in each iteration. Compared with FD-AMM, the COD algorithm [24] usually have better empirical performance. Recently, Wan and Zhang [32] and Luo et al. [20] proposed sparse variants of COD algorithm, which utilize the sparsity of the input matrices to improve the time and space costs. Blalock and Guttag [5] proposed a learning-augmented methods for AMM problem and achieves better speed-quality tradeoff. However, their method requires training data and is difficult to adapt to the streaming or sliding window model. To the best of our knowledge, none of previous works adapt well to AMM in the sliding window models.

2.2 Sliding Window Algorithms

The sliding window model [11] is designed for analyzing data within a window of most recent elements in the stream. Many existing works studied various queries over sliding windows such as distinct elements, frequency count, top-k and skyline. Datar et al. [10] proposed an effective sliding window framework based on the Exponential Histogram, creating a logarithmic hierarchical structure that can effectively approximate the count, sum, and vector norms of the elements within the window. Arasu and Manku [2] addressed the problems of element counting and quantiles within the window using a structure called dyadic intervals.

Random sampling can generate sketches of the data within the window by maintaining a collection of random samples from the window [3, 19]. Recently, Braverman et al. [7] adapted the random sampling technique to numerical linear algebra over sliding windows. They achieve near optimal space cost in several linear algebra tasks such as spectral approximation and low-rank approximation.

Badeau et al. [4] studied SVD algorithm in the sliding window model, but their method needs to store all data in the window rather than store a sketch. Recently, Wei et al. [34] studied approximating covariance matrix over sliding windows, which can be regard as a special case of AMM, i.e., $\mathbf{X} = \mathbf{Y}$. Their proposed Logarithmic Method and Dyadic Interval framework combined with the FD algorithm can achieve ε-approximation with logarithmic space complexity relative to the window size. Their methods are later generalized to the distributed setting by Zhang et al. [37]. However, few works studied AMM problem in the sliding window model.

3 PRELIMINARIES

In this section, we first introduce the notation used throughout the paper. Then, we present our problem definition, followed by the description of the COD algorithm and its theoretical guarantees.

3.1 Notations

We let $\mathbf{I}_n$ be the $n\times n$ identity matrix, and $\mathbf{0}_{\mathbf{m}\times\mathbf{n}}$ be the $m\times n$ matrix of all zeros. We can denote a $m\times n$ matrix as $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n]$, where $\mathbf{x}_i \in \mathbb{R}^m$ is the i-th column of $\mathbf{X}$. We use $[\mathbf{X}_1, \mathbf{X}_2]$ to denote their concatenation on their column dimensions. For a vector $\mathbf{x} \in \mathbb{R}^d$, we let $\|\mathbf{x}\|_2 = \sqrt{\sum_{i=1}^d x_i^2}$ be its ℓ_2-norm. For a matrix $\mathbf{X} \in \mathbb{R}^{m\times n}$, we let $\|\mathbf{X}\| = \max_{\mathbf{u}:\|\mathbf{u}\|=1} \|\mathbf{X}\mathbf{u}\|$ be its spectral norm and $\|\mathbf{X}\|_F = \sqrt{\sum_{i=1}^n \|\mathbf{x}_i\|^2}$ be its Frobenius norm. The condensed singular value decomposition (SVD) of matrix $\mathbf{X}$, written as $\mathrm{SVD}(\mathbf{X})$, is defined as $\mathbf{U}\Sigma\mathbf{V}^T$ where $\mathbf{U} \in \mathbb{R}^{m\times r}$ and $\mathbf{V} \in \mathbb{R}^{n\times r}$ are column orthonormal

Algorithm 1 Co-occurring Directions(COD)

Input: $\mathbf{X} \in \mathbb{R}^{m_x \times n}$, $\mathbf{Y} \in \mathbb{R}^{m_y \times n}$ and sketch size ℓ.
$\mathbf{A} \leftarrow \mathbf{0}_{m_x \times \ell}$, $\mathbf{B} \leftarrow \mathbf{0}_{m_y \times \ell}$.
for $i = 1, 2, \ldots, n$ **do**
  Insert $\mathbf{x}_i$ into a zero valued column of $\mathbf{A}$.
  Insert $\mathbf{y}_i$ into a zero valued column of $\mathbf{B}$.
  if $\mathbf{A}$ or $\mathbf{B}$ has no zero valued columns **then**
    $[\mathbf{Q}_x, \mathbf{R}_x] \leftarrow \mathrm{QR}(\mathbf{A})$.
    $[\mathbf{Q}_y, \mathbf{R}_y] \leftarrow \mathrm{QR}(\mathbf{B})$.
    $[\mathbf{U}, \Sigma, \mathbf{V}] \leftarrow \mathrm{SVD}(\mathbf{R}_x \mathbf{R}_y^T)$.
    $\delta \leftarrow \sigma_{\ell/2}(\Sigma)$.
    $\hat{\Sigma} \leftarrow \max(\Sigma - \delta \mathbf{I}_\ell, \mathbf{0})$.
    $\mathbf{A} \leftarrow \mathbf{Q}_x \mathbf{U}\sqrt{\hat{\Sigma}}$, $\mathbf{B} \leftarrow \mathbf{Q}_y \mathbf{V}\sqrt{\hat{\Sigma}}$.
  end if
end for
Output: $\mathbf{A}$ and $\mathbf{B}$.

and Σ is a diagonal matrix with nonzero singular values $\sigma_1(\mathbf{X}) \geq \sigma_2(\mathbf{X}) \geq \cdots \geq \sigma_r(\mathbf{X}) > 0$. We use $\mathrm{nnz}(\mathbf{X})$ to denote number of nonzero elements of matrix $\mathbf{X}$.

3.2 Problem Setup

We first provide the definition of correlation sketch as follows:

Definition 3.1 ([24]). Let $\mathbf{X} \in \mathbb{R}^{m_x \times n}$, $\mathbf{Y} \in \mathbb{R}^{m_y \times n}$, $\mathbf{A} \in \mathbb{R}^{m_x \times \ell}$ and $\mathbf{B} \in \mathbb{R}^{m_y \times \ell}$ where $n \geq \max(m_x, m_y)$ and $\ell \leq \min(m_x, m_y)$. We call the pair $(\mathbf{A}, \mathbf{B})$ is an ε-correlation sketch of $(\mathbf{X}, \mathbf{Y})$ if the correlation error satisfies

$$\text{corr-err}\left(\mathbf{X}\mathbf{Y}^T, \mathbf{A}\mathbf{B}^T\right) \triangleq \frac{\left\|\mathbf{X}\mathbf{Y}^T - \mathbf{A}\mathbf{B}^T\right\|_2}{\|\mathbf{X}\|_F \|\mathbf{Y}\|_F} \leq \varepsilon.$$

This paper considers the problem of approximate matrix multiplication (AMM) over sliding windows. At time step t, the algorithm receives column pairs $(\mathbf{x}_t, \mathbf{y}_t)$ from the original matrices $\mathbf{X}$ and $\mathbf{Y}$. Assuming the window size is N, we use W to denote the sliding window, which consists of N most recent columns of $\mathbf{X}$ and $\mathbf{Y}$. We use matrices $\mathbf{X}_W$ and $\mathbf{Y}_W$ to denote the submatrices of $\mathbf{X}$ and $\mathbf{Y}$ in the windows, respectively. The algorithm aims to maintain a pair $(\mathbf{A}, \mathbf{B})$, which is an ε-correlation sketch of $(\mathbf{X}_W, \mathbf{Y}_W)$. Similar to [34], we assume that both $\mathbf{X}$ and $\mathbf{Y}$ do not contain zero columns, and the square norms of the columns in $\mathbf{X}$ and $\mathbf{Y}$ are normalized to $[1, R_\mathbf{X}]$ and $[1, R_\mathbf{Y}]$, respectively. We let $R = \max(R_\mathbf{X}, R_\mathbf{Y})$.

3.3 Co-occurring Directions

Co-occurring directions (COD) [24] is a deterministic algorithm for correlation sketching. COD continuously receives columns from $\mathbf{X} \in \mathbb{R}^{m_x \times n}$ and $\mathbf{Y} \in \mathbb{R}^{m_y \times n}$ in a streaming way and maintains two sketches $\mathbf{A} \in \mathbb{R}^{m_x \times \ell}$ and $\mathbf{B} \in \mathbb{R}^{m_y \times \ell}$ to approximate $\mathbf{X}\mathbf{Y}^T$ with $\mathbf{A}\mathbf{B}^T$. The details of COD are presented in Algorithm 1. COD has following theoretical guarantees.

Lemma 1 ([24]). *The output of co-occurring directions (Algorithm 1) gives a correlation sketch $(\mathbf{A}, \mathbf{B})$ of $(\mathbf{X}, \mathbf{Y})$ which satisfies: for a correlation sketch of length $\ell \leq \min(m_x, m_y)$, we have:*

$$\left\|\mathbf{X}\mathbf{Y}^T - \mathbf{A}\mathbf{B}^T\right\|_2 \leq \frac{2}{\ell}\|\mathbf{X}\|_F \|\mathbf{Y}\|_F.$$

Algorithm 1 runs in $O(n(m_x + m_y + \ell)\ell)$ time and requires a space of $O((m_x + m_y + \ell)\ell)$.

4 APPROXIMATE MATRIX MULTIPLICATION OVER SLIDING WINDOWS

In this section, we propose two algorithms for computing approximate matrix multiplication over sliding windows.

4.1 The EH-COD Algorithm

We first introduce Exponential Histogram Co-occurring Directions (EH-COD), which leverages the ideas of Exponential Histograms [10] and incorporates the COD technique for efficiently approximating matrix multiplication in the sliding window model.

4.1.1 Algorithm Description.

Main Idea. We show the main idea of the EH-COD algorithm in Figure 1 and describe the detailed steps in Algorithm 2. In the EH-COD algorithm, columns within a sliding window are segmented into blocks with different sizes. The blocks are organized into L levels ($\mathcal{L}[1], \mathcal{L}[2], \ldots, \mathcal{L}[L]$ in Algorithm 2) and each level contains up to $b = O(1/\varepsilon)$ blocks. Each block K stores two matrices $\mathbf{A}'$ and $\mathbf{B}'$ as well as the start time, the end time and the size of K. We define the size of a block K as $K.size = \|\mathbf{A}'\|_F \|\mathbf{B}'\|_F$. The sketches $\mathbf{A}'$ and $\mathbf{B}'$ are approximations of columns of the streaming data matrices $\mathbf{X}$ and $\mathbf{Y}$ from the start time to the end time, respectively. The EH-COD algorithm will guarantee that the size of blocks in level i between $2^{i-1}\ell$ and $2^i\ell$, ensuring a structured and efficient data hierarchy.

Update Algorithm. The EH-COD algorithm employs a buffer B^* to collect columns $\mathbf{x}_t, \mathbf{y}_t$ from matrices $\mathbf{X}$ and $\mathbf{Y}$. Once the size of B^* exceeds ℓ , EI-COD will move it to the first level, where the blocks hold raw data columns without approximation. When level 1 fills up and contains b blocks, we combine the two oldest blocks K_1 and K_2 into block K, where $K.\mathbf{A}' = [K_1.\mathbf{A}', K_2.\mathbf{A}']$ and $K.\mathbf{B}' = [K_1.\mathbf{B}', K_2.\mathbf{B}']$ with updated size and timeframe. If the number of columns in $K.\mathbf{A}'$ and $K.\mathbf{B}'$ does not exceeds ℓ, we directly move it to level 2. Otherwise we compress the matrices $K.\mathbf{A}'$ and $K.\mathbf{B}'$ into ℓ columns by the correlation shrinkage (CS) procedure shown in Algorithm 3, which is a variant of the COD algorithm.

This process of merging and compressing also continues in higher levels. The algorithm will remove outdated blocks in the level L whose start time is earlier than $t - N$. To guarantee that the block sizes in level i are between $2^{i-1}\ell$ and $2^i\ell$, blocks whose sizes are larger than $2^i\ell$ in the level i are directly moved up without merging. They are merged only when they reach level j and their sizes are not larger than $2^j\ell$.

Query Algorithm. The query algorithm of EH-COD is presented in Algorithm 4. Notice that the EH-COD algorithm maintains a sequence of blocks with sketches approximating the $\mathbf{X}_W$ and $\mathbf{Y}_W$ matrices. We can generate the approximate matrix multiplication of $\mathbf{X}_W$ and $\mathbf{Y}_W$ by merging the sketches from non-expiring blocks in all levels and then performing the COD algorithm for once.

4.1.2 Approximation Error Analysis.

The correlation error of the EH-COD algorithm originates from three parts: the error from expiring block, the aggregate errors across all block sketches and

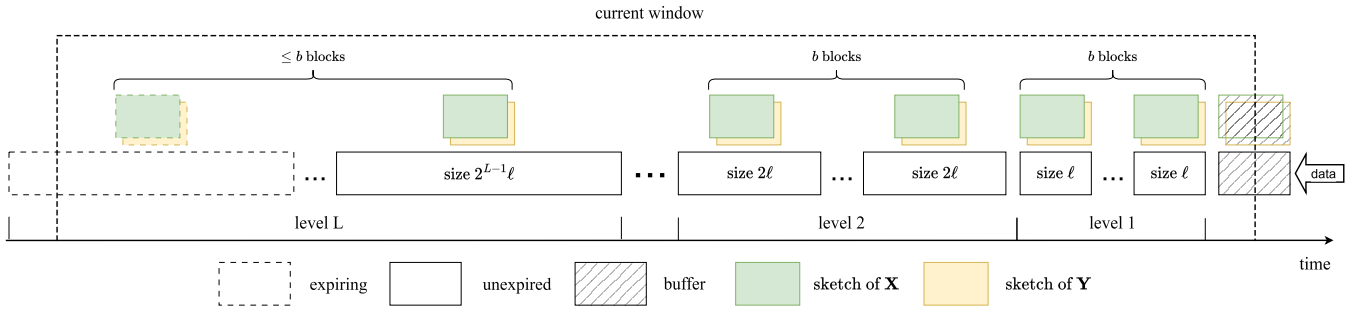


Figure 1: The illustration of the EH-COD Algorithm

Algorithm 2 EH-COD

Input: $\mathbf{X} \in \mathbb{R}^{m_x \times n}, \mathbf{Y} \in \mathbb{R}^{m_y \times n}$, sketch size ℓ.
for $t = 1, 2, \ldots, n$ **do**
  Remove expiring blocks in $\mathcal{L}[L]$.
  B^*.insert$(\mathbf{x}_t, \mathbf{y}_t)$.
  $B^*.end \leftarrow t$.
  if B^*.size $\geq \ell$ **then**
    $\mathcal{L}[1]$.append(B^*).
    initialize an empty B^* with B^*.start = B^*.end = t.
  end if
  for $i = 1, 2, \ldots, L$ **do**
    if length$(\mathcal{L}[i]) \geq b + 1$ **then**
      Find the furthest blocks K_1 and K_2 in $\mathcal{L}[i]$.
      $K \leftarrow$ MERGE(K_1, K_2).
      $\mathcal{L}[i+1]$.append(K).
      Remove K_1 and K_2 from $\mathcal{L}[i]$.
    end if
  end for
end for
procedure MERGE(K_1,K_2)
  $K.\mathbf{A}' \leftarrow [K_1.\mathbf{A}'; K_2.\mathbf{A}'], K.\mathbf{B}' \leftarrow [K_1.\mathbf{B}'; K_2.\mathbf{B}']$.
  K.size $\leftarrow \|K.\mathbf{A}'\|_F \|K.\mathbf{B}'\|_F$.
  K.start $\leftarrow K_1$.start, K.end $\leftarrow K_2$.end.
  if columns$(K.\mathbf{A}') > \ell$ **then**
    $(K.\mathbf{A}', K.\mathbf{B}') \leftarrow \mathrm{CS}(K.\mathbf{A}', K.\mathbf{B}', \ell)$.
  end if
  return K.
end procedure

Algorithm 3 Correlation Shrinkage (CS)

Input: $\mathbf{X} \in \mathbb{R}^{m_x \times n}, \mathbf{Y} \in \mathbb{R}^{m_y \times n}$, sketch size ℓ.
$[\mathbf{Q}_x, \mathbf{R}_x] \leftarrow \mathrm{QR}(\mathbf{X})$, $[\mathbf{Q}_y, \mathbf{R}_y] \leftarrow \mathrm{QR}(\mathbf{Y})$.
$[\mathbf{U}, \Sigma, \mathbf{V}] \leftarrow \mathrm{SVD}(\mathbf{R}_x \mathbf{R}_y^\top)$.
$\delta \leftarrow \sigma_\ell(\Sigma)$, $\hat{\Sigma} \leftarrow \max(\Sigma - \delta \mathbf{I}_n, \mathbf{0})$.
$\mathbf{A} \leftarrow \mathbf{Q}_x \mathbf{U} \sqrt{\hat{\Sigma}}$, $\mathbf{B} \leftarrow \mathbf{Q}_y \mathbf{V} \sqrt{\hat{\Sigma}}$.
Output: $\mathbf{A}$ and $\mathbf{B}$.

the error from merging sketches. The main idea is to bound all parts of errors to be $O(\varepsilon)$. We present our approximation error bound in the following theorem and defer the proof to the appendix.

Algorithm 4 Query for EH-COD

Input: sketch sequence $\mathcal{L}$, sketch size ℓ.
$\mathbf{C} \leftarrow$ empty, $\mathbf{D} \leftarrow$ empty.
for $i = 1, 2, \ldots, L$ **do**
  for $j = 1, 2, \ldots,$ length$(\mathcal{L}[i])$ **do**
    $\mathbf{C} \leftarrow [\mathbf{C}; \mathcal{L}[i][j].\mathbf{A}'], \mathbf{D} \leftarrow [\mathbf{D}; \mathcal{L}[i][j].\mathbf{B}']$.
  end for
end for
$(\mathbf{A}, \mathbf{B}) \leftarrow \mathrm{COD}(\mathbf{C}, \mathbf{D}, \ell)$.
Output: $\mathbf{A}$ and $\mathbf{B}$.

Theorem 1. *If we set $\ell = b = \frac{8}{\varepsilon}$, the EH-COD can generate sketches $\mathbf{A}$ and $\mathbf{B}$ of size $O(\ell)$, with the correlation error upper bounded by*

$$\textit{corr-err}\left(\mathbf{X}_W \mathbf{Y}_W^T, \mathbf{A}\mathbf{B}^T\right) \leq \varepsilon.$$

4.1.3 *Complexity Analysis.*

Space Complexity. The EH-COD algorithm can guarantee that each block only contains at most ℓ columns. Also, the number of blocks is at most Lb. Since we have $\|\mathbf{x}\|\|\mathbf{y}\| \geq 1$ for any column pair $\mathbf{x}, \mathbf{y}$, we know that blocks in level 1 contain no more than ℓ columns. In addition, the CS algorithm guarantee that the blocks at higher levels also contain no more than ℓ columns. Since the number of blocks is at most Lb, the total columns are bounded by $Lb\ell$. According to Lemma 4 in appendix B, we have $L \leq \log\left(\frac{2}{\ell b}\|\mathbf{X}_W\|_F \|\mathbf{Y}_W\|_F\right)$. Thus, the total space usage is bounded as

$$\begin{aligned} Lb\ell(m_x + m_y) &\leq \log\left(\frac{2}{\ell b}\|\mathbf{X}_W\|_F \|\mathbf{Y}_W\|_F\right) \frac{8}{\varepsilon}\ell(m_x + m_y) \\ &= O\left(\frac{m_x + m_y}{\varepsilon^2}\log(\varepsilon NR)\right), \end{aligned}$$

where N is the window size and R is the maximum squared column norm of $\mathbf{X}$ and $\mathbf{Y}$.

Time complexity. In the worst case, each column will be compressed for L times by the CS procedure since we have L levels. The time to run a CS procedure is $O\left((m_x + m_y)\ell^2\right)$ and a CS procedure can compress ℓ new columns. Thus the EH-COD algorithm takes $O((m_x + m_y)\ell)$ to compress a column in average. Therefore, the amortized time cost for processing each column is bounded by $O((m_x + m_y)\ell L) = O\left(\frac{m_x + m_y}{\varepsilon}\log(\varepsilon NR)\right)$.

Algorithm 5 DI-COD

Input: $\mathbf{X} \in \mathbb{R}^{m_x \times n}$, $\mathbf{Y} \in \mathbb{R}^{m_y \times n}$, sketch size ℓ and the size of window N.
$size_{\mathbf{X}} \leftarrow 0$, $size_{\mathbf{Y}} \leftarrow 0$.
for $t = 1, 2, \ldots, n$ **do**
  for $i = 1, 2, \ldots, L$ **do**
    Remove expiring blocks in $\mathcal{L}[i]$.
    B_i^*.insert($\mathbf{x}_t, \mathbf{y}_t$).
    $B_i^*.end \leftarrow t$.
    if columns($B_i^*.\mathbf{A}'$) $= 2\ell_i$ **then**
      $\left(B_i^*.\mathbf{A}', B_i^*.\mathbf{B}'\right) = \mathrm{CS}(B_i^*.\mathbf{A}', B_i^*.\mathbf{B}', \ell_i)$.
    end if
  end for
  $size_{\mathbf{X}} \leftarrow size_{\mathbf{X}} + \|\mathbf{x}_t\|^2$, $size_{\mathbf{Y}} \leftarrow size_{\mathbf{Y}} + \|\mathbf{y}_t\|^2$.
  if $size_{\mathbf{X}} \geq NR_{\mathbf{X}}/2^L$ or $size_{\mathbf{Y}} \geq NR_{\mathbf{Y}}/2^L$ **then**
    $v \leftarrow$ binary trailing zeros($\mathcal{L}[1].length$) + 1.
    for $i = 1, 2, \ldots, v$ **do**
      Append B_i^* to $\mathcal{L}[i]$.
      Initialize empty B_i^* with $B_i^*.start = B_i^*.end = t$.
    end for
  end if
end for

4.2 The DI-COD Algorithm

The second proposed algorithm for sliding window AMM is the Dyadic Interval Co-occurring Directions (DI-COD) algorithm, which utilizes a tree structure to store data within the window. The DI-COD algorithm requires less space usage than the EH-COD algorithm when the column norm of the data matrices is small.

4.2.1 Algorithm Description.

Main Idea. We show the main idea of the DI-COD algorithm in Figure 2 and describe the detailed steps in Algorithm 5. The DI-COD algorithm uses a hierarchical structure with L levels, each of which contains a dynamic number of blocks. Each level contains three types of blocks: expiring, inactive and buffer. Each block contains two matrices $\mathbf{A}'$ and $\mathbf{B}'$. For inactive blocks in level i, we let both $\mathbf{A}'$ and $\mathbf{B}'$ contain ℓ_i columns. This ensures that the approximation error of level i satisfies $\varepsilon_i = \frac{1}{\ell_i} = \frac{1}{2^i L}$. We define the size of a block K as $K.size = \|\mathbf{A}'\|_F \|\mathbf{B}'\|_F$. Notice that for DI-COD, the size of blocks in the same level are the same, thus we let $size_i$ be the size of each blocks in level i. For each level i, we maintain a buffer B_i^* to receive $\mathbf{X}$ and $\mathbf{Y}$. When the number of columns of B_i^* reaches $2\ell_i$, we perform the CS procedure to compress the matrices in B_i^*, so that B_i^* can continue to receive new columns. When $B^*.size$ satisfies the condition in line 13 (here $R_{\mathbf{X}}$ and $R_{\mathbf{Y}}$ are the maximum column norm of $\mathbf{X}$ and $\mathbf{Y}$, respectively), we append it to $\mathcal{L}[i]$ and regard it as an inactive block.

Query Algorithm. Algorithm 6 presents the query algorithm for DI-COD. As DI-COD uses the tree structure, we need to select appropriate blocks from top to bottom in each level and concatenate them to form the final correlation sketch pair $(\mathbf{A}, \mathbf{B})$. We set two timestamps, $l = t - N$ and $r = t$, to select suitable blocks in each level. The algorithm first identifies the highest level v containing inactive blocks and proceed downwards for selection. Within each

Algorithm 6 Query for DI-COD on window $(t - N, t)$

Input: sketch sequence $\mathcal{L}$ and t.
$\mathbf{A} \leftarrow empty, \mathbf{B} \leftarrow empty$.
$l \leftarrow t - N, r \leftarrow t$.
Find the highest level v with at least 1 inactive block.
for $i = v, v - 1, \ldots, 1$ **do**
  Find the leftmost block $\mathcal{L}[i][j]$.
  if $\mathcal{L}[i][j].end \leq r$ **then**
    $\mathbf{A} \leftarrow [\mathcal{L}[i][j].\mathbf{A}'; \mathbf{A}], \mathbf{B} \leftarrow [\mathcal{L}[i][j].\mathbf{B}'; \mathbf{B}]$.
    $r \leftarrow \mathcal{L}[i][j].start$.
  end if
  Find the rightmost block $\mathcal{L}[i][k]$.
  if $k = j$ **then**
    $l \leftarrow \mathcal{L}[i][k].end$.
  else if $\mathcal{L}[i][k].start \geq l$ **then**
    $\mathbf{A} \leftarrow [\mathbf{A}; \mathcal{L}[i][j].\mathbf{A}'], \mathbf{B} \leftarrow [\mathbf{B}; \mathcal{L}[i][j].\mathbf{B}']$.
    $l \leftarrow \mathcal{L}[i][k].end$.
  end if
end for
Output: $\mathbf{A}$ and $\mathbf{B}$.

level, we locates the furthest block $\mathcal{L}[i][j]$. If the *end* timestamp of this block is less than r, it indicates that this block needs to be combined on the left, and we update r to its *start* timestamp. Then we find the most recent block $\mathcal{L}[i][k]$ in the same level. If it is the same as the leftmost block, we shift l to the right and update it to its *end* timestamp. If $\mathcal{L}[i][k].start$ is greater than or equal to l, it signifies that this block needs to be combined on the right, and we update l to $\mathcal{L}[i][k].end$. We repeat this process until we reach the first level. The blocks selected are highlighted in bold in figure 2.

4.2.2 Approximation Error Analysis.

The approximation error of the DI-COD algorithm can be divided into two parts: the error from expiring block and the aggregate errors across all block sketches. The main idea is to control both parts of errors to be $O(\varepsilon)$. We present our approximation error bound in the following theorem and defer the proof to the appendix.

Theorem 2. *If we set $L = \log \frac{R}{\varepsilon}$ and $\ell_i = 2^i L$, the DI-COD can generate sketches $\mathbf{A}$ and $\mathbf{B}$ of size $O\left(\frac{R}{\varepsilon} \log \frac{R}{\varepsilon}\right)$, with the correlation error upper bounded by*

$$corr\text{-}err\left(\mathbf{X}_W \mathbf{Y}_W^T, \mathbf{A}\mathbf{B}^T\right) \leq \varepsilon.$$

4.2.3 Complexity Analysis.

Space Complexity. Since the window size satisfies $size_W \leq NR$ and $size_i \geq \frac{NR}{2^{L-i+1}}$, the i-th level can have at most 2^{L-i+1} blocks. Thus the space complexity of DI-COD is

$$2\sum_{i=1}^{L} \ell_i 2^{L-i+1} = \sum_{i=1}^{L} 2^i L 2^{L-i+2} = 4\frac{R}{\varepsilon}\log^2\frac{R}{\varepsilon} = O\left(\frac{R}{\varepsilon}\log^2\frac{R}{\varepsilon}\right).$$

Time complexity. The amortized time for each column during the CS process is $O((m_x + m_y)\ell)$. In the worst case, CS operations are performed concurrently across all L levels. Thus, the time complexity amounts to $O((m_x + m_y)\ell L) = O\left(\frac{m_x + m_y}{\varepsilon} \log \frac{R}{\varepsilon}\right)$.

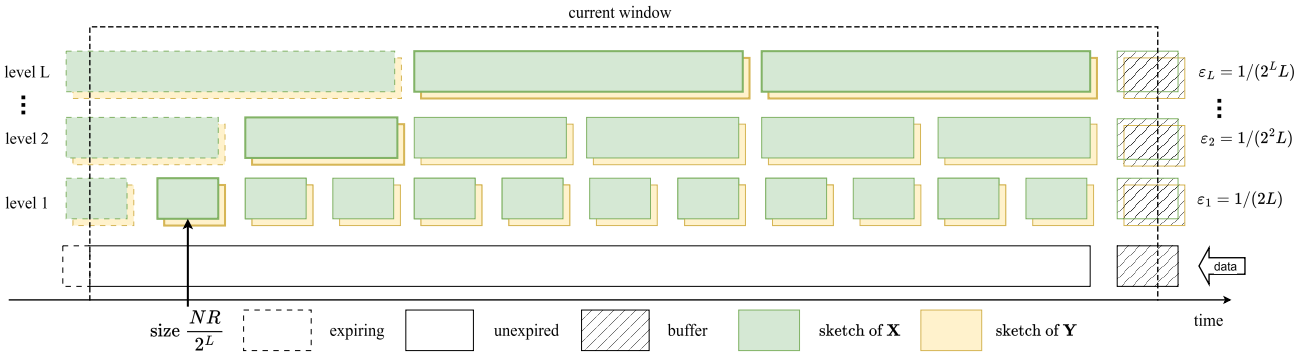


Figure 2: The illustration of the DI-COD Algorithm

Algorithm 7 EH-SCOD

Input: $\mathbf{X} \in \mathbb{R}^{m_x \times n}$, $\mathbf{Y} \in \mathbb{R}^{m_y \times n}$, sketch size ℓ, ratio κ.
for $t = 1, 2, \ldots, n$ **do**
  Remove expiring blocks in $\mathcal{L}[L]$.
  B^*.insert($\mathbf{x}_t, \mathbf{y}_t$).
  if B^*.size $\geq \varepsilon N$ **then**
    $B^* \leftarrow \text{SPM}(B^*, \ell, q)$.
    initialize an empty B^* with B^*.start = B^*.end = t.
  end if
  for $i = 1, 2, \ldots, L$ **do**
    if length($\mathcal{L}[i]$) $\geq b + 1$ **then**
      find the furthest blocks K_1 and K_2 in $\mathcal{L}[i]$.
      $K \leftarrow \text{merge}(K_1, K_2)$.
      $\mathcal{L}[i+1]$.append(K).
      Remove K_1 and K_2 from $\mathcal{L}[i]$.
    end if
  end for
end for

5 IMPROVED ALGORITHMS FOR SPARSE MATRICES

Many data matrices in real-world applications are sparse. In this section we propose variants of EH-COD and DI-COD which enhance the performance over sparse matrices.

5.1 The EH-SCOD Algorithm

We describe details of Exponential Histogram Sparse Co-occurring Directions (EH-SCOD) in algorithm 7, which is an improved version of the EH-COD algorithm. The EH-SCOD algorithm enlarge the capacity of buffer B^* to $O(\varepsilon N)$. When the buffer size exceeds εN, EH-SCOD uses the subspace power method (SPM) algorithm [25] (outlined in Algorithm 8) to compress the matrices within the buffer down to ℓ columns, so as not to exceed the space budget. The compressed block is then placed in level 1, where subsequent merging and compression operations remain consistent with EH-COD. The query method of EH-SCOD is the same as that of EH-COD, producing a final sketch of size ℓ. We have the following theoretical results for the proposed EH-SCOD algorithm.

Algorithm 8 Subspace Power Method (SPM)

Input: block K, target rank ℓ and integer q.
$\mathbf{M} \leftarrow (K.\mathbf{A}')(K.\mathbf{B}')^T \in \mathbb{R}^{m_x \times m_y}$.
Generate $\mathbf{G} = [G_{ij}] \in \mathbb{R}^{m_y \times \ell}$, where $G_{ij} \sim \mathcal{N}(0, 1)$.
$\mathbf{F} \leftarrow (\mathbf{M}\mathbf{M}^T)^q \mathbf{M}\mathbf{G} \in \mathbb{R}^{m_x \times \ell}$.
$\mathbf{Z} \leftarrow$ orthonormal column basis of $\mathbf{F}$.
$[\mathbf{U}, \Sigma, \mathbf{V}] \leftarrow \text{SVD}(\mathbf{Z}^T (K.\mathbf{A}')^T (K.\mathbf{B}'))$.
$K.\mathbf{A}' \leftarrow \mathbf{Z}\mathbf{U}\sqrt{\Sigma}$, $K.\mathbf{B}' \leftarrow \mathbf{V}\sqrt{\Sigma}$.
Output: block K.

Theorem 3. *If we choose $b = \ell = \frac{8}{\varepsilon}$ and $q = \Theta(\log(\ell/\delta))$ for EH-SCOD and SPM algorithms, then with probability $1 - \delta$, EH-SCOD can generate sketches $\mathbf{A}$ and $\mathbf{B}$ of size $O(\ell)$, with the correlation error upper bounded by*

$$\textit{corr-err}\left(\mathbf{X}_W \mathbf{Y}_W^T, \mathbf{A}\mathbf{B}^T\right) \leq \varepsilon,$$

In addition, the EH-SCOD algorithm only requires $O\left(\frac{m_x + m_y}{\varepsilon^2} \log R\right)$ space cost and $O\left(\frac{m_x + m_y}{\varepsilon} \log R\right)$ time cost in each round.

Remark 1. *Since $N \gg \ell = O(\frac{1}{\varepsilon})$, we have $\log R < \log \varepsilon NR$, which means EH-SCOD is more efficient than EH-COD.*

5.2 The DI-SCOD Algorithm

We describe details of the proposed DI-SCOD algorithm in Algorithm 9, an enhanced version of the DI-COD algorithm. In DI-SCOD, data within each buffer B_i^* at every level is stored sparsely without undergoing any CS procedure until it reaches its designated size. Once the buffer hits this size threshold, the SPM is applied to effectively reduce the number of columns in the buffer. Subsequently, the CS procedure is applied just once per buffer, thereby reducing the frequency of CS operations required across inactive blocks. The query steps remain unchanged. We have the following theoretical results for the proposed DI-SCOD algorithm.

Theorem 4. *If we set $L = \log \frac{R}{\varepsilon}$, $\ell_i = 2^i L$ and $q = \Theta(\log(\ell/\delta))$ for DI-SCOD and SPM algorithms, then with probability $1 - \delta$, DI-SCOD can generate sketches $\mathbf{A}$ and $\mathbf{B}$ of size $O\left(\frac{R}{\varepsilon} \log \frac{R}{\varepsilon}\right)$, with the*

Dataset	n	m_x	m_y	density(X)	density(Y)	$R_{\mathbf{X}}$	$R_{\mathbf{Y}}$
DENSE	1×10^4	2000	1000	1.0	1.0	370	719
APR	2.32×10^4	2.80×10^4	4.28×10^4	6.31×10^{-4}	4.53×10^{-4}	773	618
PAN11	8.90×10^4	5.12×10^4	9.96×10^4	4.38×10^{-4}	2.43×10^{-4}	5655	6188
EURO0	4.76×10^5	7.25×10^4	8.77×10^4	3.46×10^{-4}	3.47×10^{-4}	103370	112506

Table 1: The statistics of datasets, where density(X) = $\mathbf{nnz}(\mathbf{X})/(nm_x)$ and density(Y) = $\mathbf{nnz}(\mathbf{Y})/nm_y$.

Algorithm 9 DI-SCOD

Input: $\mathbf{X} \in \mathbb{R}^{m_x \times n}$, $\mathbf{Y} \in \mathbb{R}^{m_y \times n}$, sketch size ℓ.
$size_{\mathbf{X}} \leftarrow 0$, $size_{\mathbf{Y}} \leftarrow 0$.
for $t = 1, 2, \ldots, n$ **do**
  for $i = 1, 2, \ldots, L$ **do**
    Remove expiring blocks in $\mathcal{L}[L]$.
    B_i^*.insert($\mathbf{x}_t, \mathbf{y}_t$).
    $B_i^*.end \leftarrow t$.
  end for
  $size_{\mathbf{X}} \leftarrow size_{\mathbf{X}} + \|\mathbf{x}_t\|^2$, $size_{\mathbf{Y}} \leftarrow size_{\mathbf{Y}} + \|\mathbf{y}_t\|^2$.
  if $size_{\mathbf{X}} \geq NR_{\mathbf{X}}/2^L$ or $size_{\mathbf{Y}} \geq NR_{\mathbf{Y}}/2^L$ **then**
    $v \leftarrow$ binary trailing zeros($\mathcal{L}[1].length$) + 1.
    for $i = 1, 2, \ldots, v$ **do**
      $B_i^* = \mathrm{SPM}(B_i^*, 2\ell_i, q)$.
      $\left(B_i^*.\mathbf{A}', B_i^*.\mathbf{B}'\right) = \mathrm{CS}(B_i^*.\mathbf{A}', B_i^*.\mathbf{B}', \ell_i)$.
      Append B_i^* to $\mathcal{L}[i]$.
      Initialize empty B_i^* with $B_i^*.start = B_i^*.end = t$.
    end for
  end if
end for

correlation error upper bounded by

$$corr\text{-}err\left(\mathbf{X}_W\mathbf{Y}_W^T, \mathbf{A}\mathbf{B}^T\right) \leq \varepsilon.$$

In addition, in each round DI-SCOD requires $O\left(\frac{R}{\varepsilon}\log^2\frac{R}{\varepsilon}\right)$ *space cost and* $O\left(\left(\frac{nnz(\mathbf{X}_W)+nnz(\mathbf{Y}_W)}{N\varepsilon} + \frac{m_x+m_y}{N\varepsilon^2}\right)\log\frac{R}{\varepsilon}\right)$ *time cost.*

Remark 2. *We have* $N \gg \frac{1}{\varepsilon}$ *since the size of final sketches should be much less than the window size. Thus we have* $\frac{m_x+m_y}{N\varepsilon^2} \ll \frac{m_x+m_y}{\varepsilon}$. *In addition, for the sparse matrices* $\mathbf{X}_W$ *and* $\mathbf{Y}_W$, *we have* $nnz(\mathbf{X}_W) + nnz(\mathbf{Y}_W) \ll N(m_x+m_y)$. *Thus DI-SCOD requires less computational cost than DI-COD.*

6 EXPRIMENTS

In this section, we empirically study performance of proposed algorithms[1]. We compare proposed algorithms with a baseline algorithm: priority sampling, which randomly select each column pair $(\mathbf{x}_t, \mathbf{y}_t)$ according to its weight $\rho = u_t^{1/(\|\mathbf{x}\|\|\mathbf{y}\|)}$. Here u_t is uniformly sampled from $(0, 1)$.

Datasets. We employ one dense synthetic dataset named DENSE along with three sparse cross-language datasets: APR, PAN11, and EURO0 [17, 28–30] for evaluating our methods. The statistics of the datasets are provided in Table 1.

[1]The code is available at: https://github.com/lilianzhi0/AMMSW

The DENSE dataset consists of two synthetic random matrices where each element is uniformly sampled from $[0, 1]$. The matrices $\mathbf{X}$ and $\mathbf{Y}$ have $2,000$ and $1,000$ rows, respectively. Both matrices have $10,000$ columns.

The Amazon Product Reviews (APR) dataset is a publicly available collection that includes product reviews and related information from the Amazon website. The dataset contains millions of sentences in English and French. We construct it into a review matrix where the $\mathbf{X}$ matrix has $28,017$ rows and $\mathbf{Y}$ has $42,833$ rows, with both matrices sharing $23,235$ columns.

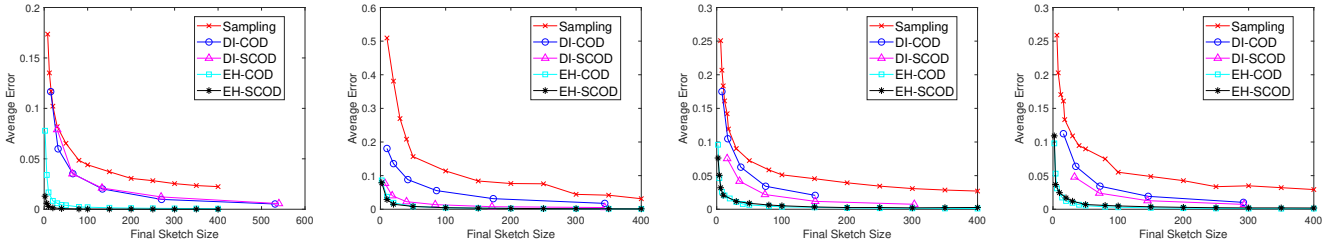
PANPC-11 (PAN11) is a dataset for text analysis, focusing on tasks such as plagiarism detection, author identification, and near-duplicate detection. Its texts are in English and French. The two data matrices $\mathbf{X}$ and $\mathbf{Y}$ contain $51,219$ and $99,595$ rows, respectively. Both of them have $88,977$ columns.

The Europarl (EURO) dataset is a widely used multilingual parallel corpus, composed of the proceedings of the European Parliament. We only utilize parts of its English and French texts to form EURO0. The matrix $\mathbf{X}$ and $\mathbf{Y}$ contains $72,470$ and $87,686$ rows, respectively. They both have $475,834$ columns.

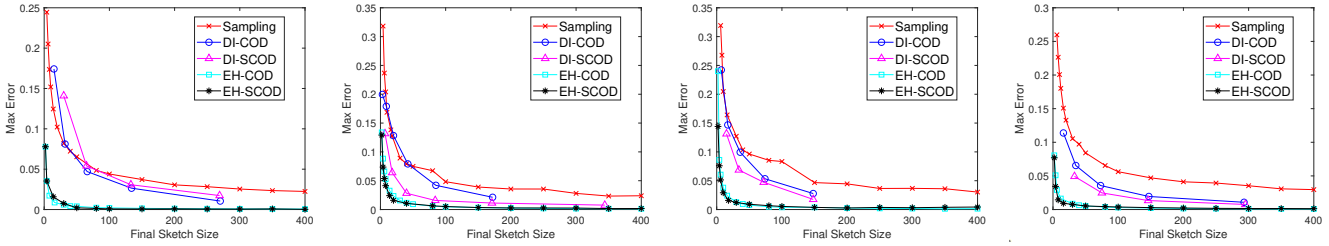
Setup. We set the window size to be $4,000$ for the DENSE dataset and $10,000$ for the larger sparse datasets APR, PAN11, and EURO0. We evaluate all algorithms according to their correlation errors corr-err$(\mathbf{X}_W\mathbf{Y}_W^T, \mathbf{A}\mathbf{B}^T)$, where $\mathbf{X}_W$ and $\mathbf{Y}_W$ are the original matrices in the window, and $\mathbf{A}$ and $\mathbf{B}$ are the generated correlation sketches. We compare both average and maximum errors through all sampled windows. All algorithms are implemented in MATLAB (R2021a). The experiments are executed on a Windows server with 32GB memory and a single core of Intel i9-13900K.

Performance. For the time comparison, we present error vs. final sketch size and error vs. total space usage in Figure 3 and Figure 4, respectively. The final sketch size is the number of columns of the sketches generated by the algorithms for query, and the total space usage is the maximum number of columns required by the algorithms. In these figures, we find that EH-SCOD and DI-SCOD require much less space than EH-COD and DI-COD when the data matrices are sparse. In addition, Figure 4 shows that DI-SCOD outperforms EH-SCOD when the maximum column norm $R_{\mathbf{X}}$ and $R_{\mathbf{Y}}$ are large, and vice versa. This phenomenon matches our theoretical results that the space cost of DI-SCOD is linear to the maximum column norm while the space cost of EH-SCOD is logarithmic to the maximum column norm. Figure 3 compares the final sketch size of proposed algorithms. We can find that EH-COD and EH-SCOD outperforms DI-COD and DI-SCOD on all datasets.

For the time comparison, we present time vs. final sketch size in Figure 5, which shows that EH-SCOD and DI-SCOD are significantly faster than EH-COD and DI-COD on the sparse datasets. We

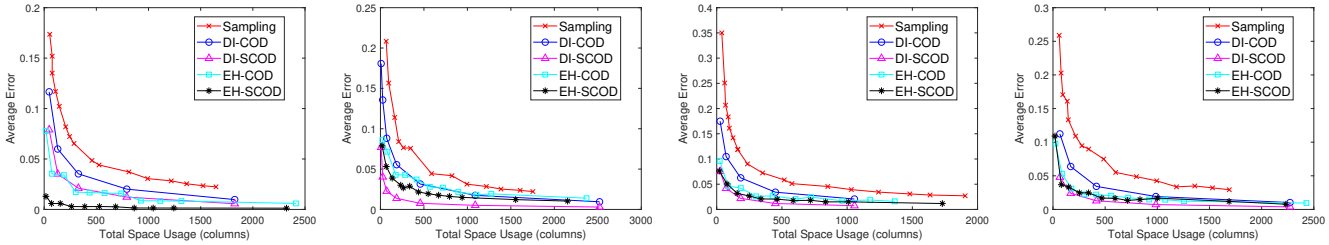


(a) DENSE (b) APR (c) PAN11 (d) EURO0

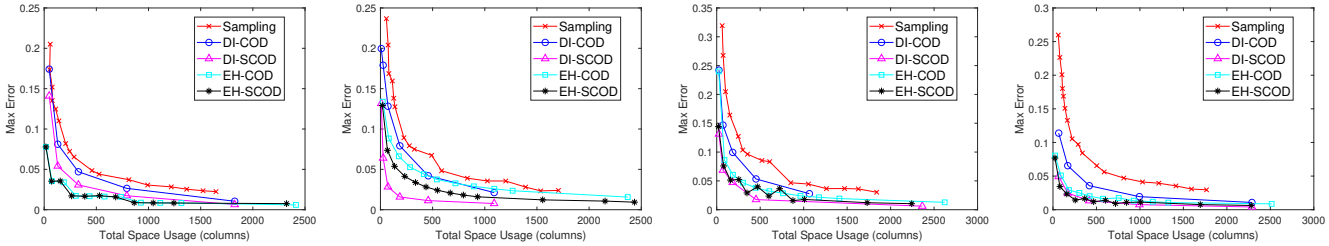


(e) DENSE (f) APR (g) PAN11 (h) EURO0

Figure 3: error vs. final sketch size

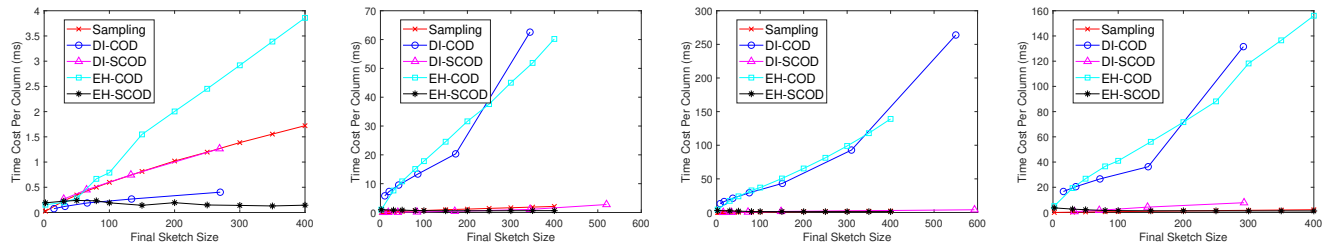


(a) DENSE (b) APR (c) PAN11 (d) EURO0



(e) DENSE (f) APR (g) PAN11 (h) EURO0

Figure 4: error vs. total space usage



(a) DENSE (b) APR (c) PAN11 (d) EURO0

Figure 5: time cost per column vs. final sketch size

surprisingly find that EH-SCOD is faster than EH-COD on dense data while DI-SCOD is faster than DI-COD. We think this is because EH-SCOD increases the buffer capacity and enables each block to store more data than EH-COD. Thus EH-SCOD performs fewer CS procedures than EH-COD and thus requires less amortized running time. On the other hand, DI-SCOD needs to perform more SPM procedures than EI-SCOD. Since SPM is expensive on dense matrices, DI-SCOD is not as efficient as EH-SCOD on dense datasets, and is even slower than DI-COD.

7 CONCLUSION

In this paper, we introduce two novel algorithms, called EH-COD and DI-COD, for approximate matrix multiplication over sliding windows. We also provide their error analysis as well as the computational complexities. In addition, we propose sparse variants of EH-COD and DI-COD which enjoy better performance when the data matrices are sparse. The experimental results show that the proposed algorithms have better performance than baseline. In the future, we will explore the lower bound of the space complexity of the AMM problem in the sliding window model and study the optimality of the proposed algorithms.

ACKNOWLEDGMENTS

Cheng Chen is supported by National Natural Science Foundation of China (62306116). Mingsong Chen is supported by National Natural Science Foundation of China (62272170) and "Digital Silk Road" Shanghai International Joint Lab of Trustworthy Intelligent Software (22510750100).

REFERENCES

[1] Naman Agarwal, Brian Bullins, Xinyi Chen, Elad Hazan, Karan Singh, Cyril Zhang, and Yi Zhang. 2019. Efficient full-matrix adaptive regularization. In *International Conference on Machine Learning*. PMLR, 102–110.

[2] Arvind Arasu and Gurmeet Singh Manku. 2004. Approximate counts and quantiles over sliding windows. In *Proceedings of the twenty-third ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*. 286–296.

[3] Brian Babcock, Shivnath Babu, Mayur Datar, Rajeev Motwani, and Jennifer Widom. 2002. Models and issues in data stream systems. In *Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*. 1–16.

[4] Roland Badeau, Gaël Richard, and Bertrand David. 2004. Sliding window adaptive SVD algorithms. *IEEE Transactions on Signal Processing* 52, 1 (2004), 1–10.

[5] Davis Blalock and John Guttag. 2021. Multiplying matrices without multiplying. In *International Conference on Machine Learning*. PMLR, 992–1004.

[6] Michele Borassi, Alessandro Epasto, Silvio Lattanzi, Sergei Vassilvitskii, and Morteza Zadimoghaddam. 2020. Sliding window algorithms for k-clustering problems. *Advances in Neural Information Processing Systems* 33 (2020).

[7] Vladimir Braverman, Petros Drineas, Cameron Musco, Christopher Musco, Jalaj Upadhyay, David P Woodruff, and Samson Zhou. 2020. Near optimal linear algebra in the online and sliding window models. In *2020 IEEE 61st Annual Symposium on Foundations of Computer Science (FOCS)*. IEEE, 517–528.

[8] Michael B Cohen, Sam Elder, Cameron Musco, Christopher Musco, and Madalina Persu. 2015. Dimensionality reduction for k-means clustering and low rank approximation. In *Proceedings of the forty-seventh annual ACM symposium on Theory of computing*. 163–172.

[9] Michael B Cohen, Jelani Nelson, and David P Woodruff. 2015. Optimal approximate matrix product in terms of stable rank. *arXiv preprint arXiv:1507.02268* (2015).

[10] Mayur Datar, Aristides Gionis, Piotr Indyk, and Rajeev Motwani. 2002. Maintaining stream statistics over sliding windows. *SIAM journal on computing* 31, 6 (2002), 1794–1813.

[11] Mayur Datar and Rajeev Motwani. 2016. The sliding-window computation model and results. In *Data Stream Management: Processing High-Speed Data Streams*. Springer, 149–165.

[12] Inderjit S Dhillon. 2001. Co-clustering documents and words using bipartite spectral graph partitioning. In *Proceedings of the 7th ACM SIGKDD international conference on Knowledge discovery and data mining*. 269–274.

[13] Petros Drineas, Ravi Kannan, and Michael W Mahoney. 2006. Fast Monte Carlo algorithms for matrices I: Approximating matrix multiplication. *SIAM J. Comput.* 36, 1 (2006), 132–157.

[14] John Duchi, Elad Hazan, and Yoram Singer. 2011. Adaptive subgradient methods for online learning and stochastic optimization. *The Journal of Machine Learning Research* 12, 7 (2011).

[15] Mina Ghashami, Edo Liberty, Jeff M Phillips, and David P Woodruff. 2016. Frequent directions: Simple and deterministic matrix sketching. *SIAM J. Comput.* 45, 5 (2016), 1762–1792.

[16] Harold Hotelling. 1936. Relations between two sets of variates. *Biometrika* (1936).

[17] Philipp Koehn. 2005. Europarl: A parallel corpus for statistical machine translation. In *Proceedings of machine translation summit x: papers*. 79–86.

[18] Edo Liberty. 2013. Simple and deterministic matrix sketching. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*. 581–588.

[19] Zhang Longbo, Li Zhanhuai, Zhao Yiqiang, Yu Min, and Zhang Yang. 2007. A priority random sampling algorithm for time-based sliding windows over weighted streaming data. In *Proceedings of the 2007 ACM symposium on Applied computing*. 453–456.

[20] Luo Luo, Cheng Chen, Guangzeng Xie, and Haishan Ye. 2021. Revisiting Co-Occurring Directions: Sharper Analysis and Efficient Algorithm for Sparse Matrices. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 35. 8793–8800.

[21] Luo Luo, Cheng Chen, Zhihua Zhang, Wu-Jun Li, and Tong Zhang. 2019. Robust frequent directions with application in online learning. *The Journal of Machine Learning Research* 20, 1 (2019).

[22] Avner Magen and Anastasios Zouzias. 2011. Low rank matrix-valued Chernoff bounds and approximate matrix multiplication. In *Proceedings of the twenty-second annual ACM-SIAM symposium on Discrete Algorithms*. SIAM, 1422–1436.

[23] Gurmeet Singh Manku and Rajeev Motwani. 2002. Approximate frequency counts over data streams. In *VLDB'02: Proceedings of the 28th International Conference on Very Large Databases*. Elsevier, 346–357.

[24] Youssef Mroueh, Etienne Marcheret, and Vaibahava Goel. 2017. Co-occurring directions sketching for approximate matrix multiply. In *Artificial Intelligence and Statistics*. PMLR, 567–575.

[25] Cameron Musco and Christopher Musco. 2015. Randomized block krylov methods for stronger and faster approximate singular value decomposition. *Advances in Neural Information Processing Systems* 28 (2015).

[26] Imran Naseem, Roberto Togneri, and Mohammed Bennamoun. 2010. Linear regression for face recognition. *IEEE transactions on pattern analysis and machine intelligence (TPAMI)* 32, 11 (2010), 2106–2112.

[27] Odysseas Papapetrou, Minos Garofalakis, and Antonios Deligiannakis. 2015. Sketching distributed sliding-window data streams. *The VLDB Journal* 24 (2015), 345–368.

[28] Martin Potthast, Alberto Barrón-Cedeno, Benno Stein, and Paolo Rosso. 2011. Cross-language plagiarism detection. *Language Resources and Evaluation* 45 (2011), 45–62.

[29] Martin Potthast, Benno Stein, Alberto Barrón-Cedeño, and Paolo Rosso. 2010. An evaluation framework for plagiarism detection. In *Coling 2010: Posters*. 997–1005.

[30] Peter Prettenhofer and Benno Stein. 2010. Cross-language text classification using structural correspondence learning. In *Proceedings of the 48th annual meeting of the association for computational linguistics*. 1118–1127.

[31] Tamas Sarlos. 2006. Improved approximation algorithms for large matrices via random projections. In *2006 IEEE 47th Annual Symposium on Foundations of Computer Science (FOCS)*. IEEE, 143–152.

[32] Yuanyu Wan and Lijun Zhang. 2021. Approximate Multiplication of Sparse Matrices with Limited Space. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 35. 10058–10066.

[33] Yuanyu Wan and Lijun Zhang. 2022. Efficient Adaptive Online Learning via Frequent Directions. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)* 44, 10 (2022), 6910–6923.

[34] Zhewei Wei, Xuancheng Liu, Feifei Li, Shuo Shang, Xiaoyong Du, and Ji-Rong Wen. 2016. Matrix sketching over sliding windows. In *Proceedings of the 2016 International Conference on Management of Data*. 1465–1480.

[35] David P Woodruff et al. 2014. Sketching as a tool for numerical linear algebra. *Foundations and Trends® in Theoretical Computer Science* 10, 1–2 (2014), 1–157.

[36] Qiaomin Ye, Luo Luo, and Zhihua Zhang. 2016. Frequent direction algorithms for approximate matrix multiplication with applications in CCA. *Computational Complexity* 1, m3 (2016), 2.

[37] Haida Zhang, Zengfeng Huang, Zhewei Wei, Wenjie Zhang, and Xuemin Lin. 2017. Tracking matrix approximation over distributed sliding windows. In *2017 IEEE 33rd International Conference on Data Engineering (ICDE)*. IEEE, 833–844.

A AUXILIARY LEMMAS FOR THE COD ALGORITHM

We present some properties of the COD algorithm, which are used in our proofs.

Lemma 2. *Co-occurring directions algorithm is parallelizable. Let* $\mathbf{X} = [\mathbf{X}_1; \mathbf{X}_2] \in \mathbb{R}^{m_x \times (n_1+n_2)}$, *and* $\mathbf{Y} = [\mathbf{Y}_1; \mathbf{Y}_2] \in \mathbb{R}^{m_y \times (n_1+n_2)}$. $(\mathbf{A}_1, \mathbf{B}_1)$ *is the correlation sketch of* $(\mathbf{X}_1, \mathbf{Y}_1)$, *and* $(\mathbf{A}_2, \mathbf{B}_2)$ *is the correlation sketch of* $(\mathbf{X}_2, \mathbf{Y}_2)$. *Then, the correlation sketch* $(\mathbf{C}, \mathbf{D})$ *of* $([\mathbf{A}_1; \mathbf{A}_2], [\mathbf{B}_1; \mathbf{B}_2])$ *is a correlation sketch of* $(\mathbf{X}, \mathbf{Y})$, *and satisfies*

$$\left\|\mathbf{X}\mathbf{Y}^T - \mathbf{C}\mathbf{D}^T\right\| \leq \frac{2}{\ell} \|\mathbf{X}\|_F \|\mathbf{Y}\|_F .$$

Lemma 3. *Given matrices* $\mathbf{X} \in \mathbb{R}^{m_x \times n}$ *and* $\mathbf{Y} \in \mathbb{R}^{m_y \times n}$, *suppose we decompose them into k sub-matrices* $\mathbf{X} = [\mathbf{X}_1; ...; \mathbf{X}_k]$ *and* $\mathbf{Y} = [\mathbf{Y}_1; ...; \mathbf{Y}_k]$, *in which* $\mathbf{X}_j$ *and* $\mathbf{Y}_j$ *both have* n_j *columns. Let* $(\mathbf{A}_i, \mathbf{B}_i) = COD(\mathbf{X}_i, \mathbf{Y}_i, \ell_i)$, *such that* $\|\mathbf{X}_i\mathbf{Y}_i^T - \mathbf{A}_i\mathbf{B}_i^T\| \leq \varepsilon_i \|\mathbf{X}_i\|_F \|\mathbf{Y}_i\|_F$. *Then* $\mathbf{A} = [\mathbf{A}_1; ...; \mathbf{A}_k]$ *and* $\mathbf{B} = [\mathbf{B}_1; ...; \mathbf{B}_k]$ *is an approximation for* $\mathbf{X}$ *and* $\mathbf{Y}$ *with error bound*

$$\left\|XY^T - AB^T\right\|_2 \leq \sum_{i=1}^{k} \varepsilon_i \|\mathbf{X}_i\|_F \|\mathbf{Y}_i\|_F.$$

B THE PROOF OF THEOREM 1

We first bound the number of levels during the algorithm's operation in the following lemma.

Lemma 4. *In EH-COD, the number of levels can be upper bounded by*

$$L \leq \log\left(\frac{2}{\ell b}\|\mathbf{X}_W\|_F \|\mathbf{Y}_W\|_F\right).$$

Proof. Consider the level L-1, which is already filled with b blocks. All these blocks cover columns that are subsets of the entire window, and the combined size of these blocks certainly does not exceed $\|\mathbf{X}_W\|_F \|\mathbf{Y}_W\|_F$. Therefore, we have $b2^{L-1}\ell \leq \|\mathbf{X}_W\|_F \|\mathbf{Y}_W\|_F$. Taking the logarithm of this inequality yields an upper bound for L. □

We partition the matrices $\mathbf{X}_W$ and $\mathbf{Y}_W$ into two submatrices, denoted as $\mathbf{X}_W = [\mathbf{X}_0, \tilde{\mathbf{X}}]$ and $\mathbf{Y}_W = [\mathbf{Y}_0, \tilde{\mathbf{Y}}]$. Here, $\mathbf{X}_0$ and $\mathbf{Y}_0$ represent the submatrices formed by columns covered by expiring blocks in the last level, while $\tilde{\mathbf{X}}$ and $\tilde{\mathbf{Y}}$ are the rest parts that have been effectively approximated. The final correlation sketches are $\mathbf{A}$ and $\mathbf{B}$. Assuming we approximate $\mathbf{X}_0$ and $\mathbf{Y}_0$ with zero matrices, we can derive the covariance error

$$\begin{aligned}\left\|\mathbf{X}_W\mathbf{Y}_W^T - \mathbf{A}\mathbf{B}^T\right\|_2 &= \left\|\left(\mathbf{X}_0\mathbf{Y}_0^T + \tilde{\mathbf{X}}\tilde{\mathbf{Y}}^T\right) - \left(\mathbf{0} + \mathbf{A}\mathbf{B}^T\right)\right\|_2 \\ &\leq \left\|\mathbf{X}_0\mathbf{Y}_0^T\right\|_2 + \left\|\tilde{\mathbf{X}}\tilde{\mathbf{Y}}^T - \mathbf{A}\mathbf{B}^T\right\|_2,\end{aligned}$$

where $\|\mathbf{X}_0\mathbf{Y}_0^T\| \leq \|\mathbf{X}_0\|_F \|\mathbf{Y}_0\|_F$, this denotes the size of the expiring block. The maximum size of the expiring block at level L is $2^L\ell$. From Lemma 3, we can deduce the error associated with the expiring block

$$\left\|\mathbf{X}_0\mathbf{Y}_0^T\right\|_2 \leq 2^L\ell \leq \frac{2}{b}\|\mathbf{X}_W\|_F \|\mathbf{Y}_W\|_F = \frac{\varepsilon}{4}\|\mathbf{X}_W\|_F \|\mathbf{Y}_W\|_F.$$

Then, we consider the approximation error in the part $\left\|\tilde{\mathbf{X}}\tilde{\mathbf{Y}}^T - \mathbf{A}\mathbf{B}^T\right\|_2$. Recalling the process of obtaining $\mathbf{A}$ and $\mathbf{B}$, we approximated $\tilde{\mathbf{X}}$ and $\tilde{\mathbf{Y}}$ in blocks, maintaining a sequence of correlation sketches $(\mathbf{A}_1, \mathbf{B}_1), (\mathbf{A}_2, \mathbf{B}_2), ..., (\mathbf{A}_k, \mathbf{B}_k)$. During the approximation of matrix multiplication within the query window, we merged all the sketches and performed a final COD, resulting in

$$A, B = COD([\mathbf{A}_1, ..., \mathbf{A}_k], [\mathbf{B}_1, ..., \mathbf{B}_k], \ell) = COD(\mathbf{C}, \mathbf{D}, \ell).$$

Precisely calculating the error in this process is complex, but we can roughly view the approximation error as consisting of two parts: the sum of covariance errors from the individual submatrices and the error introduced by the COD after merging. We now explain why the error can be computed in this manner:

$$\begin{aligned}\left\|\tilde{\mathbf{X}}\tilde{\mathbf{Y}}^T - \mathbf{A}\mathbf{B}^T\right\|_2 &= \left\|\tilde{\mathbf{X}}\tilde{\mathbf{Y}}^T - \mathbf{A}\mathbf{B}^T + \mathbf{C}\mathbf{D}^T - \mathbf{C}\mathbf{D}^T\right\|_2 \\ &\leq \left\|\tilde{\mathbf{X}}\tilde{\mathbf{Y}}^T - \mathbf{C}\mathbf{D}^T\right\|_2 + \left\|\mathbf{C}\mathbf{D}^T - \mathbf{A}\mathbf{B}^T\right\|_2 .\end{aligned}$$

Here, the first part can be further expanded and calculated in detail

$$\begin{aligned}\left\|\tilde{\mathbf{X}}\tilde{\mathbf{Y}}^T - \mathbf{C}\mathbf{D}^T\right\|_2 &= \left\|\left(\tilde{\mathbf{X}}_1\tilde{\mathbf{Y}}_1^T + ... + \tilde{\mathbf{X}}_k\tilde{\mathbf{Y}}_k^T\right) - \left(\mathbf{A}_1\mathbf{B}_1^T + ... + \mathbf{A}_k\mathbf{B}_k^T\right)\right\|_2 \\ &\leq \sum_{j=1}^{k}\left\|\tilde{\mathbf{X}}_j\tilde{\mathbf{Y}}_j^T - \mathbf{A}_j\mathbf{B}_j^T\right\|_2 .\end{aligned}$$

We assume that $\left\|\tilde{\mathbf{X}}_i\tilde{\mathbf{Y}}_i^T - \mathbf{A}_i\mathbf{B}_i^T\right\|_2$ represents the covariance error of a block located at level i. Due to the parallelizability, the error for each block adheres to the upper bound introduced by COD, that is, $\left\|\tilde{\mathbf{X}}_i\tilde{\mathbf{Y}}_i^T - \mathbf{A}_i\mathbf{B}_i^T\right\|_2 \leq \frac{\varepsilon}{8}\|\tilde{\mathbf{X}}_i\|_F \|\tilde{\mathbf{Y}}_i\|_F \leq \frac{\varepsilon}{8}2^i\ell$. Therefore, the first part of the approximation error can be bounded by

$$\begin{aligned}\sum_{i=1}^{L} b\left\|\tilde{\mathbf{X}}_i\tilde{\mathbf{Y}}_i^T - \mathbf{A}_i\mathbf{B}_i^T\right\|_2 &\leq \sum_{i=1}^{L} b\frac{\varepsilon}{8}2^i\ell \\ &= \frac{\varepsilon}{4}b\ell(2^L - 1) \\ &\leq \frac{\varepsilon}{2}\|\mathbf{X}_W\|_F \|\mathbf{Y}_W\|_F - \frac{\varepsilon}{4}b\ell \\ &\leq \frac{\varepsilon}{2}\|\mathbf{X}_W\|_F \|\mathbf{Y}_W\|_F.\end{aligned}$$

We can easily derive the second part of the approximation error

$$\left\|\mathbf{C}\mathbf{D}^T - \mathbf{A}\mathbf{B}^T\right\|_2 \leq \frac{\varepsilon}{8}\|\mathbf{C}\|_F \|\mathbf{D}\|_F \leq \frac{\varepsilon}{8}\|\mathbf{X}_W\|_F \|\mathbf{Y}_W\|_F.$$

Consequently, there is an established upper bound for the correlation error between the matrix within the window and the final correlation sketch. Then we prove Theorem 1 as follows:

$$\begin{aligned}\left\|\mathbf{X}_W\mathbf{Y}_W^T - \mathbf{A}\mathbf{B}^T\right\|_2 &\leq \left\|\mathbf{X}_0\mathbf{Y}_0^T\right\|_2 + \left\|\tilde{X}\tilde{\mathbf{Y}}^T - \mathbf{C}\mathbf{D}^T\right\|_2 + \left\|\mathbf{C}\mathbf{D}^T - \mathbf{A}\mathbf{B}^T\right\|_2 \\ &\leq \left(\frac{\varepsilon}{4} + \frac{\varepsilon}{2} + \frac{\varepsilon}{8}\right)\|\mathbf{X}_W\|_F \|\mathbf{Y}_W\|_F \\ &\leq \varepsilon\|\mathbf{X}_W\|_F \|\mathbf{Y}_W\|_F.\end{aligned}$$

C PROOF OF THEOREM 2

We divide the entire window N into two parts: expired blocks $\mathbf{X}_0, \mathbf{Y}_0$ and non-expiring blocks $\tilde{\mathbf{X}}, \tilde{\mathbf{Y}}$, then $\mathbf{X}_W = [\mathbf{X}_0; \tilde{\mathbf{X}}]$, $\mathbf{Y}_W = [\mathbf{Y}_0; \tilde{\mathbf{Y}}]$ and $\tilde{\mathbf{X}} = [\tilde{\mathbf{X}}_1; \ldots; \tilde{\mathbf{X}}_{last}]$, $\tilde{\mathbf{Y}} = [\tilde{\mathbf{Y}}_1; \ldots; \tilde{\mathbf{Y}}_{last}]$. Let $\mathbf{A}$ and $\mathbf{B}$ represent sketches of $\tilde{\mathbf{X}}$ and $\tilde{\mathbf{Y}}$, respectively. Then, $\mathbf{A} = [\mathbf{A}_1; \ldots; \mathbf{A}_{last}]$, $\mathbf{B} = [\mathbf{B}_1; \ldots; \mathbf{B}_{last}]$.

From lemma 2, we can get

$$\left\|\tilde{\mathbf{X}}\tilde{\mathbf{Y}}^T - \mathbf{A}\mathbf{B}^T\right\| \le \sum_{i=1}^{last} \left\|\tilde{\mathbf{X}}_i\tilde{\mathbf{Y}}_i^T - \mathbf{A}_i\mathbf{B}_i^T\right\| \le \sum_{i=1}^{last} \varepsilon_i \left\|\tilde{\mathbf{X}}_i\right\|_F \left\|\tilde{\mathbf{Y}}_i\right\|_F.$$

The approximation error can be proved as follows:

$$\begin{aligned}\left\|\mathbf{X}_W\mathbf{Y}_W^T - \mathbf{A}\mathbf{B}^T\right\| &= \left\|\mathbf{X}_0\mathbf{Y}_0^T + \tilde{\mathbf{X}}\tilde{\mathbf{Y}}^T - \mathbf{A}\mathbf{B}^T\right\| \\ &\le \left\|\mathbf{X}_0\mathbf{Y}_0^T\right\| + \left\|\tilde{\mathbf{X}}\tilde{\mathbf{Y}}^T - \mathbf{A}\mathbf{B}^T\right\| \\ &\le \left\|\mathbf{X}_0\mathbf{Y}_0^T\right\| + \sum_{i=1}^{last} \varepsilon_i \left\|\tilde{\mathbf{X}}_i\right\|_F \left\|\tilde{\mathbf{Y}}_i\right\|_F.\end{aligned}$$

We can easily deduce that covering the entire window requires selecting at most two blocks in each level. Furthermore, the maximum number of expired blocks is at most one block from the first level. Then $last \le 2L$, $\left\|\mathbf{X}_0\mathbf{Y}_0^T\right\| \le size_1 = \frac{NR}{2^{L-1}}$ and $\left\|\tilde{\mathbf{X}}_i\right\|_F \left\|\tilde{\mathbf{Y}}_i\right\|_F = \sqrt{size_i}^2 = size_i \le \frac{NR}{2^{L-i}}$, thus

$$\begin{aligned}&\left\|\mathbf{X}_0\mathbf{Y}_0^T\right\| + \sum_{i=1}^{last} \varepsilon_i \left\|\tilde{\mathbf{X}}_i\right\|_F \left\|\tilde{\mathbf{Y}}_i\right\|_F \\ \le& \frac{NR}{2^{L-1}} + \sum_{i=1}^{last} \frac{1}{2^i L} \frac{NR}{2^{L-i}} \\ \le& \frac{NR}{2^{L-1}} + 2L\frac{NR}{2^L L} \\ =& \frac{NR}{2^L} \\ =& \varepsilon N,\end{aligned}$$

We constrain that $1 \le \|\mathbf{x}\|^2 \le R$ and $1 \le \|\mathbf{y}\|^2 \le R$. Then $N \le \|\mathbf{X}_W\|_F \|\mathbf{Y}_W\|_F$, thus $\varepsilon N \le \varepsilon \|\mathbf{X}_W\|_F \|\mathbf{Y}_W\|_F$.

D PROOF OF THEOREM 3

During the move from buffer to Level 1, EH-SCOD handles more columns without increasing the space budget, and simultaneously expands the size range of all blocks by a multiple of κ, where $\kappa = O(\frac{\varepsilon N}{\ell})$. For a block in level i, its size satisfies $2^{i-1}\ell\kappa \le size \le 2^i\ell\kappa$. Similar to Lemma 2, we obtain $L \le \log\left(\frac{2}{\ell\kappa b}\|\mathbf{X}_W\|_F\|\mathbf{Y}_W\|_F\right)$. Since the maximum number of blocks per level remains b, EH-SCOD reduces the total number of blocks that need to be maintained for the same window size. Assuming any column pair $\mathbf{x}$ and $\mathbf{y}$ meets $\|\mathbf{x}\|\|\mathbf{y}\| \le R$, it is evident that $\|\mathbf{X}_W\|_F\|\mathbf{Y}_W\|_F \le NR$. We thus derive the new space complexity:

$$\begin{aligned}Lb\ell(m_x + m_y) &\le \log\left(\frac{2}{\ell\kappa b}\|\mathbf{X}_W\|_F\|\mathbf{Y}_W\|_F\right)\frac{8}{\varepsilon}\ell(m_x + m_y) \\ &\le \frac{8\ell}{\varepsilon}\log\left(\frac{\|\mathbf{X}_W\|_F\|\mathbf{Y}_W\|_F}{4N}\right)(m_x + m_y) \\ &= O\left((m_x + m_y)\ell^2\log R\right).\end{aligned}$$

Not only is there a substantial reduction in space overhead, but the decrease in merge operations also significantly cuts down on time costs. The amortized time expense for maintaining the sequence of sketches is $O((m_x + m_y)\ell L) = O((m_x + m_y)\ell\log R)$.

We conduct a brief analysis of the error introduced by EH-SCOD. Let $\bar{\mathbf{X}}_i, \bar{\mathbf{Y}}_i = SPM(\tilde{\mathbf{X}}_i, \tilde{\mathbf{Y}}_i)$, and assume that the error introduced by the SPM compression can be bounded by $\left\|\tilde{\mathbf{X}}_i\tilde{\mathbf{Y}}_i^T - \bar{\mathbf{X}}_i\bar{\mathbf{Y}}_i^T\right\|_2 \le \sigma\|\tilde{\mathbf{X}}_i\|_F\|\tilde{\mathbf{Y}}_i\|_F$. The error from expiring block is still defined by its size: $2^L\ell\kappa \le \frac{\varepsilon}{4}\|\mathbf{X}_W\|_F\|\mathbf{Y}_W\|_F$. The upper bound for the approximation error of blocks in level i is $\left\|\tilde{\mathbf{X}}_i\tilde{\mathbf{Y}}_i^T - \mathbf{A}_i\mathbf{B}_i^T\right\|_2 \le \left\|\tilde{\mathbf{X}}_i\tilde{\mathbf{Y}}_i^T - \bar{\mathbf{X}}_i\bar{\mathbf{Y}}_i^T\right\|_2 + \left\|\tilde{\mathbf{X}}_i\bar{\mathbf{Y}}_i^T - \mathbf{A}_i\mathbf{B}_i^T\right\|_2 \le \left(\sigma + \frac{\varepsilon}{8}\right)2^i\ell\kappa$. Therefore, the second part of the error is $\sum_{i=1}^{L} b\left(\sigma + \frac{\varepsilon}{8}\right)2^i\ell\kappa \le \left(4\sigma + \frac{\varepsilon}{2}\right)\|\mathbf{X}_W\|_F\|\mathbf{Y}_W\|_F$. And the third part, the sketch merging error remains unchanged at $\frac{\varepsilon}{8}\|\mathbf{X}_W\|_F\|\mathbf{Y}_W\|_F$. By the choice of q and the property of the SPM method [25], we have $\sigma = \frac{\varepsilon}{32}$ and thus achieve the conclusion.

E PROOF OF THEOREM 4

We conduct a brief analysis of the error introduced by DI-SCOD. By the property of the SCOD method [20], we can still guarantee $\left\|\tilde{\mathbf{X}}_i\tilde{\mathbf{Y}}_i^T - \mathbf{A}_i\mathbf{B}_i^T\right\| \le \varepsilon_i \left\|\tilde{\mathbf{X}}_i\right\|_F \left\|\tilde{\mathbf{Y}}_i\right\|_F$. Thus, we also have

$$\left\|\mathbf{X}_W\mathbf{Y}_W^T - \mathbf{A}\mathbf{B}^T\right\| \le \left\|\mathbf{X}_0\mathbf{Y}_0^T\right\| + \sum_{i=1}^{last} \varepsilon_i \left\|\tilde{\mathbf{X}}_i\right\|_F \left\|\tilde{\mathbf{Y}}_i\right\|_F.$$

Similar to appendix C, we can get

$$\left\|\mathbf{X}_W\mathbf{Y}_W^T, \mathbf{A}\mathbf{B}^T\right\|_2 \le \varepsilon \|\mathbf{X}\|_F \|\mathbf{Y}\|_F.$$

For space complexity, DI-SCOD dose not use additional space. Thus, the DI-SCOD algorithm only requires $O(\ell R\log^2(\ell R))$ space cost.

For time complexity, the amortized time for the SPM and CS procedures to process a column is

$$O\left(\frac{nnz(\mathbf{X}_W) + nnz(\mathbf{Y}_W)}{N\varepsilon} + \frac{m_x + m_y}{N\varepsilon^2}\right).$$

In the worst-case scenario, SPM and CS operations are performed concurrently across all L levels. Thus, the time complexity amounts to $O\left(\left(\frac{nnz(\mathbf{X}_W)+nnz(\mathbf{Y}_W)}{N\varepsilon} + \frac{m_x+m_y}{N\varepsilon^2}\right)\log\frac{R}{\varepsilon}\right)$.