Scalable Co-Clustering for Large-Scale Data through Dynamic Partitioning and Hierarchical Merging

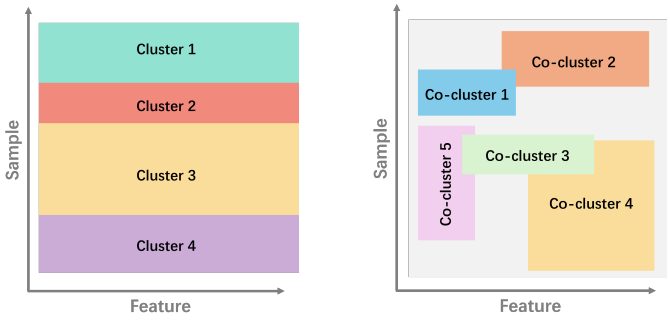
***Abstract*—As a powerful tool to leverage the complex structure of data, co-clustering has attracted increasing attention in data mining and machine learning. However, existing co-clustering methods are not scalable to large-scale data due to the high computational complexity and high memory consumption. Parallel solutions have emerged to address the scalability issue with the help of distributed computing frameworks. To overcome these limitations, parallel solutions employing distributed computing frameworks have been developed. This paper introduces *DPHMCo*, an innovative co-cluster framework designed for efficient and scalable co-clustering. *DPHMCo* comprises three distinct stages: preliminary computations, large matrix partitioning, and hierarchical merging. As illustrated in Figure 1, the preliminary computations prepare the dataset for efficient processing. The large matrix partitioning stage divides the dataset into manageable submatrices, which are processed in parallel, significantly reducing computational overhead. Lastly, the hierarchical merging algorithm efficiently combines the results from individual submatrices, ensuring robustness and maintaining the integrity of the co-clustering process. Our extensive evaluations demonstrate that *DPHMCo* achieves substantial improvements in scalability and efficiency. Experimental results show a significant reduction in computation time, with dense matrices seeing an approximate 83% decrease and sparse matrices up to 30%.**

***Index Terms*—Co-clustering, scalable, distributed computing, dynamic partitioning, hierarchical merging.**

I. Introduction

RECENT advancements in machine learning, driven by large datasets, have underscored the importance of clustering as an unsupervised learning method that plays a vital role in preprocessing training data, enhancing the performance and efficiency of large models [1]–[5]. The emergence of multimodal learning frameworks, such as CLIP [6], ALIGN [7], and DALL-E [8], has further highlighted the importance of clustering techniques in enhancing the training and inference processes of large-scale models due to the vast amount of data and inherent heterogeneity of multimodal data [9]. Despite the promise of clustering techniques in enhancing model performance, they face significant challenges related to data heterogeneity and scalability.

Co-clustering [10]–[12], a variant of traditional clustering, has emerged as a powerful technique to address these challenges by simultaneously grouping rows (samples) and columns (features) to uncover complex correlations between different data types, as shown in Figure 1b. This approach is transformative in scenarios where the relationships between rows and columns are as important as the individual entities themselves. For example, in bioinformatics, co-clustering can identify gene-related patterns leading to biological insights by concurrently analyzing genes and conditions [11], [13]–[16]. In recommendation systems, it can simultaneously discover more fine-grained relationships between users and items [17]–[19]. By extending traditional clustering methods, co-clustering enhances accuracy in pattern detection and broadens the scope of analyses.



(a) Clustering (b) Co-clustering

Fig. 1: An illustration of the differences between (a) Clustering and (b) Co-clustering.

Despite the significant advantages and transformative potential of co-clustering, there are notable challenges that need to be addressed, particularly in terms of scalability and high complexity, resulting co-clustering seen as an impractical solution for large-scale datasets [20]. This complexity arises from the need to simultaneously optimize both row and column clusters, turning the problem into a multi-target optimization challenge. Minimizing multiple loss functions concurrently can lead to conflicting gradients and optimization paths, making convergence difficult [21].

Parallel solutions have emerged to address the scalability issue using distributed computing frameworks. PNMTF [22] attempts parallelization but lacks guaranteed iteration numbers and requires broadcasting a non-adaptive scale matrix, which limits its scalability. Similarly, CoClusterD [20]'s main thread must handle co-cluster statistics S_{cc}, constraining its scalability. Both PNMTF and CoClusterD strive to distribute co-clustering but still rely on the main thread to manage high-complexity tasks.

The challenges associated with existing co-clustering methods can be summarized as follows:

- **Scalability Constraints:** Many existing co-clustering algorithms, such as PNMTF and CoClusterD, rely on a central processing thread to handle complex tasks like managing co-cluster statistics. This centralization creates a bottleneck, significantly limiting their scalability and making them impractical for large-scale datasets.
- **High Computational Complexity:** The dual-focus analysis of co-clustering, which simultaneously optimizes row and column clusters, requires evaluating a vast number of potential relationships. This complexity can grow expo-

nentially with the size of the data, making it impractical for large-scale applications.

To address these challenges, we propose a novel and scalable co-clustering method designed for large-scale datasets. Our approach begins with a dynamic partitioning algorithm that divides the original data matrix into smaller submatrices. This partitioning facilitates parallel processing of co-clustering tasks across submatrices, significantly reducing both processing time and computational and storage demands for each processing unit. Additionally, we design a probabilistic model to determine the optimal number and configuration of these submatrices, ensuring comprehensive data coverage. If the atom co-clustering method is effective, this model guarantees that the entire framework will converge with an iteration number specified only by the scale of the data matrix, providing confidence in the robustness and reliability of our approach across diverse datasets.

Furthermore, we develop a hierarchical co-cluster merging algorithm that iteratively combines the co-clusters from these submatrices. The biggest problem is one cannot find small co-clusters if partitioned.

This process enhances the accuracy and reliability of the final co-clustering results and ensures robust and consistent clustering performance, particularly addressing issues of heterogeneity and model uncertainty. By leveraging a hierarchical merging strategy, we can efficiently integrate co-clusters without overwhelming any single processing unit, thereby maintaining scalability. Our method is modular, allowing any co-clustering method, including those with existing parallel designs, to be integrated seamlessly. This flexibility ensures that our approach can adapt and evolve with advancements in co-clustering techniques.

Our proposed method also includes an MPI (Message Passing Interface) implementation, which allows for effective distribution of the computational load across multiple nodes. The main computation node only needs to compute thresholds based on the size of the matrix, without requiring specialized performance beyond that of other nodes, making our approach more practical and scalable for large-scale applications. Unlike existing methods that require a high-performance central processing unit, our approach does not demand a supercomputer for the main thread. Additionally, our framework is probabilistically guaranteed to work.

The main contributions of this paper are as follows:

1) **Dynamic Partitioning Algorithm:** We propose a dynamic partitioning algorithm grounded in our probabilistic model to determine the optimal parameters for dividing large matrices. This algorithm enhances parallel processing and reduces computational complexity.
2) **Hierarchical Merging Algorithm:** We introduce a hierarchical merging algorithm designed to amalgamate co-clusters from submatrices. This method fortifies the co-clustering process, ensuring robustness and preserving data integrity.
3) **MPI Implementation and Performance Evaluation:** We implement our algorithm using MPI (Message Passing Interface) and conduct extensive performance evaluations on large datasets. The results demonstrate significant improvements in computational efficiency and robustness.

The remainder of this paper is organized as follows. Section II discusses related work in the field of co-clustering and parallel computing. Section III presents the mathematical formulation and problem statement of co-clustering. Section IV details our proposed scalable co-clustering method. Section V presents the experimental evaluation of our method. Finally, Section VI concludes the paper and outlines future research directions.

II. Related work

A. Co-clustering Models

The concept of *biclustering*, where both the rows and columns of data are clustered simultaneously, was first introduced by Hartigan [23] to analyze gene expression data. This pioneering approach recognized the importance of simultaneously considering multiple dimensions of data to uncover more nuanced patterns. Since then, biclustering has expanded to various other areas, giving rise to the broader concept of *co-clustering*, which has led to the development of several models designed to address a range of analytical challenges.

Spectral Co-clustering with Singular Value Decomposition (SVD), introduced by Dhillon *et al.* [24], became widely adopted due to its straightforward implementation and effectiveness. This method leverages the mathematical properties of SVD to identify clusters by analyzing the principal components of the data matrix. Spectral co-clustering has since become a foundational technique, inspiring two main branches in co-clustering research: graph-based and matrix factorization-based approaches.

The most widely used graph-based co-clustering method is the Flexible Bipartite Graph Co-clustering (FBGPC) [25]. FBGPC applies a flexible bipartite graph model directly to the original data, allowing for the simultaneous clustering of two distinct sets of objects, such as users and items in a recommendation system. This approach effectively captures the interactions between these sets, providing deeper insights into their relationships.

In contrast, Non-negative Matrix Tri-Factorization (NMTF) [26] represents a leading matrix factorization-based method. NMTF decomposes the data matrix into multiple non-negative matrices, which correspond to the underlying factors of the data. By independently clustering both samples and features, NMTF reveals the latent structures within the data. This method has proven to be particularly useful in applications where the interpretability of factors is crucial, such as bioinformatics and social network analysis. Our method is orthogonal to NMTF, allowing for potential integration to enhance co-clustering efficiency.

Another innovative method, Deep Co-Clustering (DeepCC) [27], combines the power of deep learning with traditional clustering techniques. DeepCC integrates deep autoencoders with Gaussian Mixture Models to improve the clustering of complex data. The autoencoders reduce the dimensionality of the data while preserving essential features, and the Gaussian Mixture Models identify clusters within this reduced space.

Despite its advancements, DeepCC struggles with computational efficiency, particularly with diverse data types and iterative processes dependent on data sparsity. The deep learning components introduce additional layers of computation, which can be resource-intensive and time-consuming.

B. Parallelizing Co-clustering

To address the significant computational challenges posed by big data, parallel co-clustering methods have emerged as a vital solution. Parallelization leverages distributed computing frameworks to divide the computational load across multiple processors or machines, thereby improving scalability and efficiency.

Cheng *et al.* [20] introduced a distributed co-clustering framework, CoClusterD, which utilizes an Alternating Minimization Co-clustering (AMCC) algorithm with sequential updates. CoClusterD partitions the data into smaller chunks, which are processed in parallel, and then combines the results. However, this method faces challenges with guaranteed convergence. The iterative nature of AMCC, coupled with the need for sequential updates, can lead to inefficiencies and bottlenecks, particularly as the size and complexity of the data increase.

While matrix factorization techniques have shown promise for co-clustering large datasets, scaling these algorithms for massive high-dimensional data remains an open challenge. Chen *et al.* [18] proposed a parallel non-negative matrix trifactorization (PNMTF) method that distributes computation across multiple nodes to accelerate factorizations. PNMTF decomposes the original matrix into simpler, non-negative components, which are then processed in parallel. However, even these advanced methods encounter difficulties with extremely large datasets. The need to broadcast intermediate results and synchronize updates across nodes introduces significant communication overhead, which can negate the benefits of parallelization.

Our proposed method adopts a divide-and-conquer strategy to overcome these limitations. By partitioning the input matrix into smaller submatrices, we reduce the dimensionality of each individual task, making them more manageable. Each submatrix is co-clustered in parallel, leveraging distributed computing resources to expedite the process. This technique reduces the complexity imposed by high dimensionality and minimizes the communication overhead associated with broadcasting and synchronization. The separate co-clustering results from each submatrix are then combined using a hierarchical merging algorithm to form the final co-clusters. This novel approach addresses the computational challenges inherent in co-clustering large-scale data and introduces a scalable solution tailored for big data.

III. Mathematical Formulation and Problem Statement

A. Mathematical Formulation of Co-clustering

Co-clustering is a method designed to simultaneously group a set of rows and columns in a data matrix $\mathbf{A} \in \mathbb{R}^{M \times N}$, where M denotes the number of features or variables and N denotes the number of samples or objects. Each element a_{ij} in the matrix at the (i, j)-th position represents the relationship or measurement between the i-th feature and the j-th object. The primary goal of co-clustering is to identify subsets of rows (I) and columns (J) into homogenous submatrices, $\mathbf{A}_{I,J}$, that exhibit distinctive patterns or uniform characteristics, thus forming coherent co-clusters. The objective is to partition $\mathbf{A}$ into k row clusters and d column clusters, resulting in $k \times d$ distinct co-clusters.

When rows and columns are optimally reordered, $\mathbf{A}$ can be visualized as a block-diagonal matrix, where each block represents a co-cluster with higher similarity within than between blocks. We define the row and column label sets as $u \in \{1, \ldots, k\}^M$ and $v \in \{1, \ldots, d\}^N$, respectively. Indicator matrices $R \in \mathbb{R}^{M \times k}$ and $C \in \mathbb{R}^{N \times d}$ are used to assign rows and columns to clusters, with the constraints $\sum_k R_{i,k} = 1$ and $\sum_d C_{j,d} = 1$, ensuring each row and column is assigned to exactly one cluster.

B. Problem Statement

This paper aims to develop a method that efficiently and accurately identifies co-clusters $\mathbf{A}_{I,J}$ within a matrix $\mathbf{A}$ representing large datasets. These co-clusters should exhibit specific structural patterns such as uniformity across elements, consistency along rows or columns, or patterns demonstrating additive or multiplicative coherence. Properly identifying and categorizing these patterns is crucial for understanding the complex data structures inherent in large datasets. This method is intended to improve the detection capabilities of co-clustering, enhancing both the efficiency and precision necessary for handling large-scale data challenges.

IV. The Scalable Co-clustering Method

A. Overview

This paper presents a novel and scalable co-clustering method specifically designed for large matrices, as shown in Figure 2. This method applies a probabilistic model-based optimal partitioning algorithm, which not only predicts the ideal number and sequence of partitions for maximizing computational efficiency but also ensures the effectiveness of the co-clustering process.

Our method involves partitioning large matrices into smaller, manageable submatrices. This strategic partitioning is meticulously guided by our algorithm to facilitate parallel processing. By transforming the computationally intensive task of co-clustering a large matrix into smaller, parallel tasks, our approach significantly reduces computational overhead and enhances scalability. This partitioning strategy ensures that each submatrix is small enough to be processed efficiently while still retaining the essential co-cluster structures of the original large matrix.

Following the partitioning, each submatrix undergoes a co-clustering process. This is implemented via the application of Singular Value Decomposition (SVD) and k-means clustering on the resulting singular vectors. SVD helps in reducing the dimensionality of each submatrix, capturing the most significant features, while k-means clustering groups these

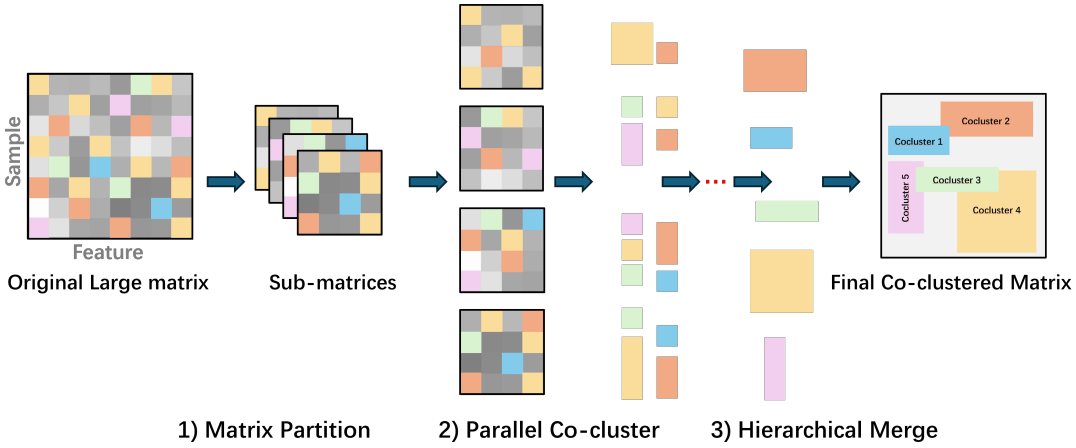


Fig. 2: Workflow of our proposed scalable co-clustering method for large matrices.

features into meaningful clusters. This pivotal step ensures the adaptability of our method, allowing our algorithm to tailor its approach to the unique characteristics of each submatrix, thus optimizing clustering results. The use of SVD and k-means clustering leverages the strengths of both techniques, ensuring a robust and accurate co-clustering outcome.

Furthermore, our method integrates a novel hierarchical merging strategy that combines the co-clustering results from all submatrices. This integration provides more fine-grained insight into each submatrix and enhances the overall accuracy and reliability of the co-clustering results. The hierarchical merging process iteratively combines co-clusters from the submatrices, ensuring that the final result is comprehensive and consistent across the entire dataset. Our method, validated and optimized through a comprehensive process, showed efficiency in handling large-scale datasets that were never reached before. This multi-step process not only improves scalability but also ensures that the final co-clustering results are robust and reliable, even for very large datasets.

B. Large Matrix Partitioning

The primary challenge in co-clustering large matrices is the risk of losing meaningful co-cluster relationships when the matrix is partitioned into smaller submatrices. To address this, we introduce an optimal partitioning algorithm underpinned by a probabilistic model. This model is meticulously designed to navigate the complexities of partitioning, ensuring that the integrity of co-clusters is maintained even as the matrix is divided. The objective of this algorithm is twofold: to determine the optimal partitioning strategy that minimizes the risk of fragmenting significant co-clusters and to define the appropriate number of repartitioning iterations needed to achieve a desired success rate of co-cluster identification.

1) Partitioning and Repartitioning Strategy based on the Probabilistic Model: Our probabilistic model serves as the cornerstone of the partitioning algorithm. It evaluates potential partitioning schemes based on their ability to preserve meaningful co-cluster structures within smaller submatrices. The model operates under the premise that each atom-co-cluster (the smallest identifiable co-cluster within a submatrix) can be identified with a probability p. This probabilistic model allows us to estimate the likelihood of successfully identifying all relevant co-clusters across the partitioned submatrices.

In the scenario where the matrix A is partitioned into $m \times n$ blocks, each block has size $\phi_i \times \psi_j$, that is, $M = \sum_{i=1}^{m} \phi_i$ and $N = \sum_{j=1}^{n} \psi_j$, the joint probability of $M_{(i,j)}^{(k)}$ and $N_{(i,j)}^{(k)}$ is given by:

$$\begin{aligned} &P(M_{(i,j)}^{(k)} < T_m, N_{(i,j)}^{(k)} < T_n) \\ &\quad = \sum_{\alpha=1}^{T_m-1} \sum_{\beta=1}^{T_n-1} P(M_{(i,j)}^{(k)} = \alpha) P(N_{(i,j)}^{(k)} = \beta) \\ &\quad \leq \exp[-2(s_i^{(k)})^2 \phi_i + -2(t_j^{(k)})^2 \psi_j] \end{aligned}$$

where $s_i^{(k)}$ and $t_j^{(k)}$ are the minimum row and column sizes of co-cluster C_k in block $B_{(i,j)}$, the size of the co-cluster C_k is $M^{(k)} \times N^{(k)}$, and $M^{(k)}$ and $N^{(k)}$ are the row and column sizes of co-cluster C_k, respectively.

Thus, the probability of identifying all co-clusters is given by:

$$P(\omega_k) \leq \exp\left\{-2[\phi m (s^{(k)})^2 + \psi n (t^{(k)})^2]\right\}, \tag{1}$$

and

$$\begin{aligned} P &= 1 - P(\omega_k)^{T_p} && (2) \\ &\geq 1 - \exp\left\{-2T_p[\phi m (s^{(k)})^2 + \psi n (t^{(k)})^2]\right\} && (3) \end{aligned}$$

where $P(\omega_k)$ is the probability of the failure of identifying co-cluster C_k, T_p is the number of sampling times, ϕ and ψ are the row and column block sizes, and $s^{(k)}$ and $t^{(k)}$ are the minimum row and column sizes of co-cluster C_k.

Equation (3) is central to our algorithm for partitioning large matrices for co-clustering, providing a probabilistic model that informs and optimizes our partitioning strategy to preserve co-cluster integrity. It mathematically quantifies the likelihood of identifying all relevant co-clusters within partitioned blocks, guiding us to mitigate risks associated with partitioning that might fragment vital co-cluster relationships.

Based on Equation (3), we can establish a constraint between the repartitioning time Tr and the partition solution

$Part$, ensuring that the partitioning strategy adheres to a predetermined tolerance success rate, thereby minimizing the risk of co-cluster fragmentation. The constraints are discussed in the appendix due to space limitations.

2) Optimization and Computational Efficiency: Optimizing the partitioning process for computational efficiency is paramount in both academic and industrial applications, where running time often serves as the primary bottleneck. Thanks to the flexible framework established by our probabilistic model and the constraints derived in Theorem 1, our optimization strategy can be tailored to address the most critical needs of a given context. In this case, we focus on minimizing the running time without compromising the integrity and success rate of co-cluster identification.

Our approach to optimization leverages the probabilistic model to assess various partitioning configurations, balancing the trade-off between computational resource allocation and the need to adhere to theoretical success thresholds. By systematically evaluating the impact of different partitioning schemes on running time, we can identify strategies that not only meet our co-clustering success criteria but also optimize the use of computational resources.

To ensure that our optimization does not sacrifice the quality of co-cluster identification for the sake of efficiency, we introduce a secondary theorem that outlines the conditions under which optimization can be achieved without compromising the success rate of co-cluster discovery. This theorem provides a mathematical basis for optimizing the partitioning algorithm in a manner that maintains a balance between computational efficiency and the fidelity of co-cluster identification.

Under the constraint of maintaining a predetermined success rate P for co-cluster identification, the optimization of the partitioning algorithm with respect to running time must satisfy the following condition:

$$T_p = \text{argmin}_{T_p}\{1 - \exp\{-2T_p[\phi m(s^{(k)})^2 + n(t^{(k)})^2]\} \geq P_{\text{thresh}}\}$$

This condition delineates the parameters within which the partitioning strategy can be optimized for speed without detracting from the algorithm's ability to accurately identify co-clusters. By adhering to this theorem, we ensure that our optimization efforts align with the overarching goal of preserving the integrity and effectiveness of co-cluster discovery. This balance is crucial for developing a partitioning algorithm that is not only fast and efficient but also robust and reliable across various datasets and co-clustering challenges.

C. Co-clustering on Small Submatrices

1) Atom-co-clustering Algorithm: Our framework, which encompasses both partitioning and ensembling, offers remarkable flexibility, allowing it to

be compatible with a wide range of atom-co-clustering methods. For the sake of making this paper comprehensive and self-contained, we provide an introduction to the atom-co-cluster method herein. The only requirement for an atom-co-clustering method to be compatible with our framework is that it must be able to identify co-clusters under a given criterion with a probability p, or more relaxed conditions, has a lower bound estimate of the probability of identifying co-clusters equipped with a validation mechanism.

2) Graph-based Spectral Co-clustering Algorithm: Spectral co-clustering (SCC) stands as one of the most prevalent methods in the realm of co-clustering today [28], primarily due to its adeptness in unraveling the complexities of high-dimensional data. At its core, this method harnesses the power of spectral clustering principles, focusing on the utilization of a graph's Laplacian matrix eigenvectors for effectively partitioning data into distinct clusters. This approach is exceptionally beneficial for analyzing data that naturally forms a bipartite graph structure, including applications in text-document analysis, social network modeling, and gene expression studies.

a) Graph Construction in Co-clustering Expanded: SCC begins with constructing a bipartite graph $G = (U, V, E)$. Here, U and V, both as vertex sets, symbolize the sets corresponding to the rows and columns of the data matrix, respectively. The edges E of this graph are assigned weights reflecting the relationships between rows and columns. Consequently, the graph's weighted adjacency matrix W is defined as:

$$W = \begin{bmatrix} 0 & A \\ A^T & 0 \end{bmatrix},$$

where A denotes the data matrix, also called the adjacency matrix in the graph context. Through this representation, the challenge of co-clustering is reformulated into a graph partitioning task, aiming to segregate the graph into distinct clusters based on the interrelations between the data matrix's rows and columns.

b) Laplacian Matrix: The graph's Laplacian matrix L is computed as $L = D - W$, with D being the graph's degree matrix—a diagonal matrix whose entries equal the sum of the weights of the edges incident to each node. The Laplacian matrix plays a crucial role in identifying the graph's cluster structure. It does so by facilitating the calculation of eigenvectors associated with its smallest positive eigenvalues, which in turn, guide the partitioning of the graph into clusters.

c) Graph Partitioning and Singular Value Decomposition: Theorem 4 in [24] states that the eigenvector corresponding to the second smallest eigenvalue of the following eigenvalue problem gives the generalized partition vectors for the graph:

$$L\mathbf{v} = \lambda D\mathbf{v} \tag{4}$$

And according to Section 4 of [24], the singular value decomposition of the normalized matrix $A_n = D^{-1/2}AD^{-1/2}$

$$A_n = U\Sigma V^T$$

gives the solution to Equation (4). To be more specific, the singular vectors corresponding to the second largest singular value of A_n are the eigenvector corresponding to the second smallest eigenvalue of Equation (4).

The above discussion is under the assumption that the graph has only one connected component. In a more general setting, $\mathbf{u_2}, \mathbf{u_3}, \ldots, \mathbf{u_{l+1}}$ and $\mathbf{v_2}, \mathbf{v_3}, \ldots, \mathbf{v_{l+1}}$ reveal the k-modal

information of the graph, where $\mathbf{u_k}$ and $\mathbf{v_k}$ are the k-th left and right singular vectors of A_n, respectively. And for the last step,

$$Z = \begin{bmatrix} D_1^{-1/2}\hat{U} \\ D_2^{-1/2}\hat{V} \end{bmatrix}$$

is stacked where $\hat{U} = [\mathbf{u_2}; \mathbf{u_3}; \ldots; \mathbf{u_{l+1}}]$ and $\hat{V} = [\mathbf{v_2}; \mathbf{v_3}; \ldots; \mathbf{v_{l+1}}]$. The approximation to the graph partitioning optimization problem is then solved by applying a k-means algorithm to the rows of Z. More details can be found in [24].

D. Hierarchical Co-cluster Merging

Hierarchical co-cluster merging is a novel approach that combines the results of co-clustering on submatrices to produce a final co-clustered result. The merging method is designed to enhance the accuracy and robustness of the co-clustering outcome by leveraging the design of the partitioning algorithm. The hierarchical merging process iteratively combines the co-clusters from each submatrix, ensuring that the final co-clustered result is comprehensive and consistent across all submatrices. This iterative merging process is crucial for addressing issues of heterogeneity and model uncertainty, ensuring that the final co-clustering results are reliable and robust.

E. Algorithmic Description

Our proposed Optimal Matrix Partition and Hierarchical Co-cluster Merging Method is outlined in Algorithm 2. The algorithm is an advanced algorithm designed for efficient co-clustering of large data matrices. The algorithm begins by initializing a block set based on predetermined block sizes. For each co-cluster in the given set, the algorithm calculates specific values $s^{(k)}$ and $t^{(k)}$, which are then used to determine the probability $P(\omega_k)$ of each co-cluster. If this probability falls below a predefined threshold P_{thresh}, the algorithm partitions the data matrix A into blocks B and performs co-clustering on these blocks. This step is crucial for managing large datasets by breaking them down into smaller, more manageable units. After co-clustering, the results from each block are aggregated to form the final co-clustered result $\mathcal{C}$. The algorithm's design allows for a flexible and efficient approach to co-clustering, particularly suited to datasets with high dimensionality and complexity.

In conclusion, our proposed scalable co-clustering method effectively addresses the challenges associated with large-scale data analysis. By leveraging a probabilistic model for optimal partitioning and a hierarchical merging strategy, we ensure that our method is both efficient and robust. The detailed algorithmic framework and theoretical underpinnings provide a solid foundation for further research and development in this field.

V. Experiment Evaluation

This section presents the empirical evaluation of the proposed co-cluster method. The experiments aim to assess the method's efficiency, scalability, and accuracy when applied to large data matrices.

Algorithm 1 Optimal Matrix Partition and Hierarchical Co-cluster Merging Method

Require: Data matrix $A \in \mathbb{R}^{M \times N}$, Co-cluster set $C = \{C_k\}_{k=1}^{K}$, Block sizes $\{\phi_i\}_{i=1}^{m}$, $\{\psi_j\}_{j=1}^{n}$, Thresholds T_m, T_n, Sampling times T_p, Probability threshold P_{thresh};
Ensure: Co-clustered result $\mathcal{C}$;
Initialize block set $B = \{B_{(i,j)}\}_{i=1,j=1}^{m,n}$ based on ϕ_i and ψ_j
Calculate $s^{(k)}$ and $t^{(k)}$ for each co-cluster C_k
for $k = 1$ to K **do**
  Calculate $P(\omega_k)$ for co-cluster C_k
  if $P(\omega_k) < P_{thresh}$ **then**
    Partition matrix A into blocks B and perform co-clustering
    Aggregate co-clustered results from each block
  end if
end for
return Aggregated co-clustered result $\mathcal{C}$

Algorithm 2 Optimal Matrix Partition and Hierarchical Co-cluster Merging Method

Require: Data matrix $A \in \mathbb{R}^{M \times N}$, Co-cluster set $C = \{C_k\}_{k=1}^{K}$, Block sizes $\{\phi_i\}_{i=1}^{m}$, $\{\psi_j\}_{j=1}^{n}$, Thresholds T_m, T_n, Sampling times T_p, Probability threshold P_{thresh};
Ensure: Co-clustered result $\mathcal{C}$;
Initialize block set $B = \{B_{(i,j)}\}_{i=1,j=1}^{m,n}$ based on ϕ_i and ψ_j
Calculate $s^{(k)}$ and $t^{(k)}$ for each co-cluster C_k
for $k = 1$ to K **do**
  Calculate $P(\omega_k)$ for co-cluster C_k
  if $P(\omega_k) < P_{thresh}$ **then**
    Partition matrix A into blocks B and perform co-clustering
    Aggregate co-clustered results from each block
  end if
end for
return Aggregated co-clustered result $\mathcal{C}$

A. Experiment Setup

Datasets. The experiments were conducted using three distinct datasets to demonstrate the versatility and robustness of our method:

- Amazon 1000: Comprising 1000 Amazon reviews; each represented as a 1000-dimensional vector, this dataset is designed to mimic customer behavior analysis.
- CLASSIC4: Containing 18000 documents from 20 newsgroups; each document is represented as a 1000-dimensional vector, this dataset is suitable for text analysis and topic discovery.
- RCV1-Large: A larger dataset used to test the scalability of our method, it includes a vast array of document vectors for high-dimensional data analysis.

Implementation details. All experiments were performed on a computing cluster with the following specifications: Intel Xeon E5-2670 v3 @ 2.30GHz processors, 128GB RAM, and

TABLE I: Comparison of Running Times (in seconds) for Various Co-clustering Methods on Selected Datasets.

Dataset	SCC [24]	NMTF [26]	ONMTF [29]	WC-NMTF [30]	FNMTF [31]	PNMTF [18]	**MPHM-SCC**	**MPHM-PNMTF**
Amazon 1000	64545.2	298.7	498.4	198.8	275.1	303.7	112.5	242.8
CLASSIC4	*	75,530	68,793	27,114	189,47	17,810	22,894	3,028
RCV1-Large	*	*	*	*	*	277,092	*	208,048

Notes: * indicates that the method cannot process the dataset because the dataset size exceeds the processing limit.

TABLE II: NMIs and ARIs Scores for Various Co-clustering Methods on Selected Datasets.

Dataset	Metric	Compared Methods			
		SCC [24]	PNMTF [18]	**MPHM-SCC**	**MPHM-PNMTF**
Amazon 1000	NMI	0.9223	0.6894	0.8650	0.6609
	ARI	0.7713	0.6188	0.7763	0.6057
CLASSIC4	NMI	*	0.5944	0.7676	0.6073
	ARI	*	0.4523	0.5845	0.4469
RCV1-Large	NMI	*	0.6519	0.8349	0.6348
	ARI	*	0.5383	0.7576	0.5298

Notes: * indicates that the method cannot process the dataset because the dataset size exceeds the processing limit.

Ubuntu 20.04 LTS operating system. The algorithms were implemented in Rust and compiled with the latest stable version of the Rust compiler.

Compared Methods. The experiments followed the procedure outlined in Algorithm 2. The proposed method was compared with the following state-of-the-art co-clustering methods:

- Spectral Co-Clustering (SCC) [24]
- Parallel Non-negative Matrix Tri-Factorization (PNMTF) [18]
- Deep Co-Clustering (DeepCC) [27]

Notably, 1) PNMTF is one of the most efficient co-clustering algorithms in the state-of-art. 2) All our experiments show that DeepCC cannot process all selected datasets due to the dataset size exceeds DeepCC processing limit.

Our Methods. Our proposed scalable co-cluster method is applied along with the SCC and PNMTF to demonstrate the enhanced performance and capability of handling large datasets:

- Matrix Partitioned and Hierarchical Co-Cluster Merging with Spectral Co-Clustering (MPHM-SCC)
- Matrix Partitioned and Hierarchical Co-Cluster Merging with Parallel Non-negative Matrix Tri-Factorization (MPHM-PNMTF)

Evaluation Metrics. The effectiveness of the co-clustering was measured using two widely accepted metrics:

- Normalized Mutual Information (NMI): Quantifies the mutual information between the co-clusters obtained and the ground truth, normalized to [0, 1] range, where 1 indicates perfect overlap.
- Adjusted Rand Index (ARI): Adjusts the Rand Index for chance, providing a measure of the agreement between two clusters, with values ranging from -1 (complete disagreement) to 1 (perfect agreement).

B. Results

The results of the experiments are presented in Tables I and II, showcasing the proposed scalable co-clustering methods, MPHM-SCC and MPHM-PNMTF, in comparison with traditional methods such as SCC and PNMTF.

1) Handling Large-scale Datasets: The results first confirm the limitations of some existing methods, such as SCC and DeepCC, in handling large datasets due to their processing constraints. As indicated in Table I, these methods could not process certain datasets (denoted by "*"), highlighting the challenge of scalability in traditional co-clustering methods.

2) Improved Performance: On the other hand, our methods, MPHM-SCC and MPHM-PNMTF, not only succeeded in processing all datasets but also significantly outperformed the traditional methods in terms of efficiency. For instance, the running time for the Amazon 1000 dataset was reduced from 64545.2 seconds for SCC to 112.5 seconds for MPHM-SCC, illustrating a dramatic increase in speed.

3) Quantitative Metrics: According to Table II, our methods also showed enhanced accuracy and robustness across different evaluation metrics like NMI and ARI. For example, in the CLASSIC4 dataset, while SCC was unable to process the dataset, MPHM-SCC achieved an NMI of 0.7676 and an ARI of 0.5845, significantly improving upon PNMTF's performance.

These experiments validate our proposed scalable co-clustering method as more efficient and capable of handling diverse and large-scale datasets without sacrificing the quality of co-cluster identification. The adaptability of our method to different data characteristics and its capacity for parallel processing demonstrate its potential as a robust tool for applications in domains requiring the analysis of large data matrices, such as text and biomedical data analyses and financial pattern recognition.

C. Parallel Efficiency Evaluation

To evaluate the efficiency of our proposed co-clustering method, we conducted experiments varying the number of processing nodes. The datasets used for this evaluation included Amazon 1000, Classic4, and RCV1-Large. Each dataset was subjected to co-clustering using 1, 4, 8, 16, and 24 processing nodes. The efficiency metric was normalized to 1 for a

single node and measured for the increased number of nodes. The results, plotted in Figure 3, demonstrate the efficiency improvements achieved by leveraging parallel processing.

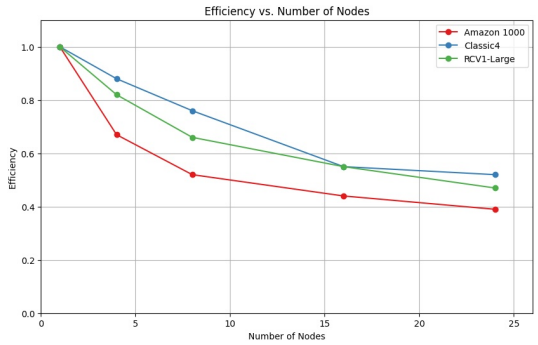


Fig. 3: Enhanced efficiency of the proposed method in handling large-scale datasets.

Figure 3 presents the efficiency of our proposed co-clustering method across different datasets as the number of nodes increases. The datasets considered are Amazon 1000, Classic4, and RCV1-Large. The x-axis represents the number of nodes, while the y-axis indicates the efficiency, normalized to the baseline (single node) efficiency of 1.

1) Amazon 1000 Dataset: The efficiency starts at 1 for a single node and gradually decreases to 0.39 as the number of nodes increases to 24. The decrease in efficiency is relatively smooth, indicating that our method scales well but exhibits some overhead as more nodes are added. Despite the reduction in efficiency, the method maintains more than 39% efficiency with 24 nodes, highlighting the robustness of the algorithm even at higher parallelization levels.

2) Classic4 Dataset: The efficiency for Classic4 begins at 1 and decreases to 0.52 with 24 nodes. The drop in efficiency is more pronounced compared to the Amazon 1000 dataset, especially between 4 and 8 nodes, indicating a higher overhead for this dataset as the number of nodes increases. The method still retains more than half of the efficiency at 24 nodes, demonstrating good scalability.

3) RCV1-Large Dataset: Starting from an efficiency of 1, it decreases to 0.47 at 24 nodes. The decrease is relatively smooth, similar to the Amazon 1000 dataset, but with a slightly steeper decline. The method shows a significant efficiency retention at higher node counts, indicating effective parallelization.

D. Detailed Analysis

1) Scalability: The results demonstrate that our method effectively scales across multiple nodes, maintaining a reasonable level of efficiency even as the number of nodes increases. This is crucial for large-scale data processing, where distributing the computational load is necessary to handle extensive datasets without incurring prohibitive computational costs.

2) Dataset Characteristics: The differences in efficiency reduction between datasets highlight the influence of dataset characteristics on parallelization. For instance, Classic4 shows a more significant drop, which could be due to its specific data structure or inherent complexity that introduces more overhead when partitioned across nodes. The smoother efficiency decline for Amazon 1000 and RCV1-Large suggests that these datasets are more amenable to parallel processing, possibly due to less inter-node communication or better partitioning.

3) Parallel Processing Overhead: The observed decrease in efficiency is expected due to the overhead introduced by parallel processing, such as inter-node communication, synchronization, and data partitioning. Our method's design, which includes dynamic partitioning and hierarchical merging, helps mitigate these overheads to a significant extent, as evidenced by the relatively high efficiency retention.

4) Practical Implications: Maintaining around 40-50% efficiency with up to 24 nodes is a strong indicator that our method can handle very large datasets in a distributed computing environment effectively. This efficiency retention is critical for real-world applications where computational resources are distributed across many nodes, and maintaining high performance is essential for timely data processing.

E. Optimal Partitioning

We further analyzed the optimization of partition settings to determine the ideal balance between the number of partitions, the number of repetitions, and the computation time. The datasets were divided into varying numbers of partitions: 25, 36, 49, 81, 100, and 121. For each partition setting, the required number of repetitions and the corresponding computation time (in seconds) were recorded. The experiment aimed to identify the optimal partition setting that minimizes computation time while maintaining a feasible number of repetitions. The results, shown in Figure 4, highlight the optimal setting at 100 partitions, where computation time is minimized, and the number of repetitions remains stable.

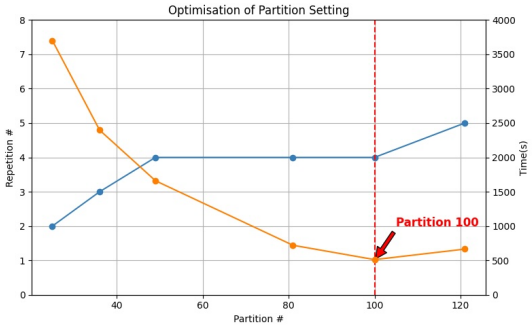


Fig. 4: Optimization of the partitioning algorithm for computational efficiency.

Figure 4 illustrates the relationship between the number of partitions, the number of repetitions, and the computation time (in seconds). The x-axis represents the number of partitions, while the primary y-axis indicates the number of repetitions, and the secondary y-axis shows the computation time in seconds. The plot provides valuable insights into how our proposed method performs under varying partition settings.

1) Repetition Count: The blue line represents the number of repetitions needed for different partition counts. As the number of partitions increases from 25 to 121, the number of repetitions initially rises, peaking at 4 for partitions 49, 81, and 100 before slightly increasing to 5 at 121 partitions. This trend suggests that more partitions generally require a higher number of repetitions to achieve optimal co-clustering, but there is a point of diminishing returns.

2) Computation Time: The orange line shows the computation time corresponding to different partition counts. There is a noticeable decrease in computation time as the number of partitions increases from 25 to 100, with the time dropping from 3701 seconds to 512 seconds. However, the computation time slightly increases to 665 seconds at 121 partitions. This indicates that while increasing the number of partitions initially improves computational efficiency, beyond a certain point, the overhead of managing more partitions outweighs the benefits.

3) Optimal Partition Setting: The red dashed line at 100 partitions highlights the theoretical optimal setting, where the computation time is minimized. At this setting, the number of repetitions required is stable at 4, and the computation time is at its lowest. This confirms that our theoretical analysis aligns with empirical results, validating the effectiveness of our partitioning strategy.

F. Detailed Analysis

1) Balancing Partitions and Repetitions: The results indicate that there is an optimal balance between the number of partitions and the number of repetitions required. Too few partitions result in higher computation times due to the increased complexity within each partition, while too many partitions lead to increased overhead from managing numerous small partitions.

2) Efficiency and Scalability: The significant reduction in computation time as the number of partitions increases up to 100 demonstrates the scalability of our method. This efficiency gain is crucial for processing large-scale datasets, where computational resources and time are often limiting factors. The slight increase in computation time beyond 100 partitions suggests that there is an optimal range for partition counts that maximizes efficiency without incurring excessive overhead.

3) Practical Implications: The practical implication of these findings is that our proposed method can be effectively tuned for various datasets by adjusting the number of partitions. The theoretical optimal setting at 100 partitions provides a benchmark for achieving the best performance, but the method's flexibility allows for adjustments based on specific dataset characteristics and computational constraints.

VI. Conclusion

This paper presented a novel, scalable co-clustering method designed to process large matrices, specifically addressing the computational challenges associated with high-dimensional data analysis. Our two-step method begins with a partitioning algorithm that divides large matrices into smaller, parallel-processed submatrices, significantly reducing processing time. Then, a hierarchical co-cluster merging algorithm integrates the results from these submatrices, ensuring the accuracy and consistency of the final co-clustering results. Extensive evaluations show that our method outperforms existing solutions in handling large-scale datasets, proving its effectiveness, efficiency, and scalability. This work sets a new benchmark for future research in scalable data analysis technologies.

References

[1] B. Raskutti, H. Ferrá, and A. Kowalczyk, "Combining clustering and co-training to enhance text classification using unlabelled data," in *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '02, New York, NY, USA: Association for Computing Machinery, Jul. 23, 2002, pp. 620–625, DOI: 10.1145/775047.775139.

[2] Z. Li, J. Liu, Y. Yang, X. Zhou, and H. Lu, "Clustering-guided sparse structural learning for unsupervised feature selection," *IEEE Transactions on Knowledge and Data Engineering*, vol. 26, pp. 2138–2150, 2014, DOI: 10.1109/TKDE.2013.65.

[3] H. Ghimatgar, K. Kazemi, M. Helfroush, and A. Aarabi, "An improved feature selection algorithm based on graph clustering and ant colony optimization," *Knowl. Based Syst.*, vol. 159, pp. 270–285, 2018, DOI: 10.1016/j.knosys.2018.06.025.

[4] J. Li, W. Wang, W. Abbas, and X. Koutsoukos, "Distributed clustering for cooperative multi-task learning networks," *IEEE Transactions on Network Science and Engineering*, vol. 10, pp. 1–10, 2023, DOI: 10.1109/TNSE.2023.3276854.

[5] D. Bertsimas, A. Orfanoudaki, and H. Wiberg, "Interpretable clustering: An optimization approach," *Machine Learning*, vol. 110, pp. 89–138, 2020, DOI: 10.1007/s10994-020-05896-2.

[6] A. Radford, J. W. Kim, C. Hallacy, *et al.*, "Learning transferable visual models from natural language supervision," in *Proceedings of the 38th International Conference on Machine Learning*, PMLR, Jul. 1, 2021, pp. 8748–8763.

[7] J. Li, D. Li, C. Xiong, and S. Hoi, "BLIP: Bootstrapping language-image pre-training for unified vision-language understanding and generation," in *Proceedings of the 39th International Conference on Machine Learning*, PMLR, Jun. 28, 2022, pp. 12 888–12 900.

[8] A. Ramesh, M. Pavlov, G. Goh, *et al.*, "Zero-shot text-to-image generation," in *Proceedings of the 38th International Conference on Machine Learning*, PMLR, Jul. 1, 2021, pp. 8821–8831.

[9] B. Chen, A. Rouditchenko, K. Duarte, *et al.*, "Multimodal clustering networks for self-supervised learning from unlabeled videos," *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 7992–8001, 2021, DOI: 10.1109/ICCV48922.2021.00791.

[10] Y. Cheng and G. Church, "Biclustering of Expression Data," in *Proceedings. International Conference on Intelligent Systems for Molecular Biology*, 2000.

[11] Y. Kluger, R. Basri, J. T. Chang, and M. Gerstein, "Spectral Biclustering of Microarray Data: Coclustering Genes and Conditions," *Genome Research*, vol. 13, no. 4, pp. 703–716, Apr. 1, 2003, DOI: 10.1101/gr.648603pmid: 12671006.

[12] H. Yan, "Coclustering of Multidimensional Big Data: A Useful Tool for Genomic, Financial, and Other Data Analysis," *IEEE Systems, Man, and Cybernetics Magazine*, vol. 3, no. 2, pp. 23–30, Apr. 2017, DOI: 10.1109/msmc.2017.2664218.

[13] D. J. Higham, G. Kalna, and M. Kibble, "Spectral clustering and its use in bioinformatics," *Journal of Computational and Applied Mathematics*, Special Issue Dedicated to Professor Shinnosuke Oharu on the Occasion of His 65th Birthday, vol. 204, no. 1, pp. 25–37, Jul. 1, 2007, DOI: 10.1016/j.cam.2006.04.026.

[14] S. C. Madeira and A. L. Oliveira, "Biclustering algorithms for biological data analysis: A survey," *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 2004, DOI: 10.1109/tcbb.2004.2pmid: 17048406.

[15] H. Zhao, A. Wee-Chung Liew, D. Z. Wang, and H. Yan, "Biclustering analysis for pattern discovery: Current techniques, comparative studies and applications," *Current Bioinformatics*, vol. 7, no. 1, pp. 43–55, Mar. 1, 2012, DOI: 10.2174/157489312799304413.

[16] M. Golchev, S. H. Davarpanah, and A. W.-C. Liew, "Biclustering analysis of gene expression data using multi-objective evolutionary algorithms," *2015 International Conference on Machine Learning and Cybernetics (ICMLC)*, vol. 2, pp. 505–510, 2015, DOI: 10.1109/ICMLC.2015.7340608.

[17] I. S. Dhillon, Y. Guan, and B. Kulis, "Weighted graph cuts without eigenvectors a multilevel approach," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 29, no. 11, pp. 1944–1957, Nov. 2007, DOI: 10.1109/TPAMI.2007.1115.

[18] Y. Chen, Z. Lei, Y. Rao, *et al.*, "Parallel non-negative matrix tri-factorization for text data co-clustering," *IEEE Transactions on Knowledge and Data Engineering*, vol. 35, no. 5, pp. 5132–5146, May 2023, DOI: 10.1109/TKDE.2022.3145489.

[19] A. Bouchareb, M. Boullé, F. Clérot, and F. Rossi, "Model based co-clustering of mixed numerical and binary data," in *Advances in Knowledge Discovery and Management*, Springer, Cham, 2019, pp. 3–22, DOI: 10.1007/978-3-030-18129-1_1.

[20] X. Cheng, S. Su, L. Gao, and J. Yin, "Co-ClusterD: A distributed framework for data co-clustering with sequential updates," *IEEE Transactions on Knowledge and Data Engineering*, vol. 27, no. 12, pp. 3231–3244, Dec. 2015, DOI: 10.1109/TKDE.2015.2451634.

[21] C. A. C. Coello, G. B. Lamont, and D. A. V. Veldhuizen, *Evolutionary Algorithms for Solving Multi-Objective Problems (Genetic and Evolutionary Computation)*. Springer, Sep. 7, 2007, 800 pp., ISBN: 978-0-387-33254-3.

[22] Y. Chen, Z. Lei, Y. Rao, *et al.*, "Parallel non-negative matrix tri-factorization for text data co-clustering," *IEEE Transactions on Knowledge and Data Engineering*, vol. 35, no. 5, pp. 5132–5146, May 2023, DOI: 10.1109/TKDE.2022.3145489.

[23] J. A. Hartigan, "Direct clustering of a data matrix," *Journal of the American Statistical Association*, vol. 67, no. 337, pp. 123–129, Mar. 1, 1972, DOI: 10.1080/01621459.1972.10481214.

[24] I. S. Dhillon, "Co-clustering documents and words using bipartite spectral graph partitioning," in *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, San Francisco California: ACM, Aug. 26, 2001, pp. 269–274, DOI: 10.1145/502512.502550.

[25] W. Chen, H. Wang, Z. Long, and T. Li, "Fast Flexible Bipartite Graph Model for Co-Clustering," *IEEE Transactions on Knowledge and Data Engineering*, vol. 35, no. 7, pp. 6930–6940, Jul. 2023, DOI: 10.1109/TKDE.2022.3194275.

[26] B. Long, Z. Zhang, and P. S. Yu, "Co-clustering by block value decomposition," in *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, 2005, pp. 635–640, DOI: 10.1145/1081870.1081949.

[27] D. Xu, W. Cheng, B. Zong, *et al.*, "Deep co-clustering," in *Proceedings of the 2019 SIAM International Conference on Data Mining*, SIAM, 2019, pp. 414–422.

[28] U. von Luxburg, "A tutorial on spectral clustering," *Statistics and Computing*, vol. 17, no. 4, pp. 395–416, Dec. 1, 2007, DOI: 10.1007/s11222-007-9033-z.

[29] C. Ding, T. Li, W. Peng, and H. Park, "Orthogonal nonnegative matrix t-factorizations for clustering," in *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '06, New York, NY, USA: Association for Computing Machinery, Aug. 20, 2006, pp. 126–135, DOI: 10.1145/1150402.1150420.

[30] A. Salah, M. Ailem, and M. Nadif, "Word co-occurrence regularized non-negative matrix tri-factorization for text data co-clustering," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, Apr. 29, 2018, DOI: 10.1609/aaai.v32i1.11659.

[31] J. Kim and H. Park, "Fast nonnegative matrix factorization: An active-set-like method and comparisons," *SIAM Journal on Scientific Computing*, vol. 33, no. 6, pp. 3261–3281, Jan. 2011, DOI: 10.1137/110821172.