On Efficient Large Sparse Matrix Chain Multiplication

CHUNXU LIN*, The Chinese University of Hong Kong, Shenzhen, China
WENSHENG LUO*†, The Chinese University of Hong Kong, Shenzhen, China
YIXIANG FANG†, The Chinese University of Hong Kong, Shenzhen, China
CHENHAO MA, The Chinese University of Hong Kong, Shenzhen, China
XILIN LIU, HUAWEI CLOUD, China
YUCHI MA, HUAWEI CLOUD, China

Sparse matrices are often used to model the interactions among different objects and they are prevalent in many areas, including e-commerce, social networks, and biology. As one of the fundamental matrix operations, the *sparse matrix chain multiplication* (SMCM) aims to efficiently multiply a chain of sparse matrices, which has found various real-world applications in areas like network analysis, data mining, and machine learning. The efficiency of SMCM largely hinges on the order of multiplying the matrices, which further relies on the accurate estimation of the sparsity of intermediate matrices. Existing matrix sparsity estimators often struggle with large sparse matrices, because they suffer from the accuracy issue in both theory and practice. To enable efficient SMCM, in this paper, we introduce a novel row-wise sparsity estimator (RS-estimator), a straightforward yet effective estimator that leverages matrix structural properties to achieve efficient, accurate, and theoretically guaranteed sparsity estimation. Based on the RS-estimator, we propose a novel ordering algorithm for determining a good order of efficient SMCM. We further develop an efficient parallel SMCM algorithm by effectively utilizing multiple CPU threads. We have conducted experiments by multiplying various chains of large sparse matrices extracted from five real-world large graph datasets, and the results demonstrate the effectiveness and efficiency of our proposed methods. In particular, our SMCM algorithm is up to three orders of magnitude faster than the state-of-the-art algorithms.

CCS Concepts: • **Theory of computation → Design and analysis of algorithms**; • **Mathematics of computing**;

Additional Key Words and Phrases: Sparse matrix chain multiplication, sparse matrix, sparsity estimator

ACM Reference Format:
Chunxu Lin, Wensheng Luo, Yixiang Fang, Chenhao Ma, Xilin Liu, and Yuchi Ma. 2024. On Efficient Large Sparse Matrix Chain Multiplication. *Proc. ACM Manag. Data* 2, N3 (SIGMOD), Article 156 (June 2024), 27 pages. https://doi.org/10.1145/3654959

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

Authors' addresses: Chunxu Lin, chunxulin1@link.cuhk.edu.cn, The Chinese University of Hong Kong, Shenzhen, Guangdong, China; Wensheng Luo, luowensheng@cuhk.edu.cn, The Chinese University of Hong Kong, Shenzhen, Guangdong, China; Yixiang Fang, The Chinese University of Hong Kong, Shenzhen, Guangdong, China, fangyixiang@cuhk.edu.cn; Chenhao Ma, The Chinese University of Hong Kong, Shenzhen, Guangdong, China, machenhao@cuhk.edu.cn; Xilin Liu, HUAWEI CLOUD, Guangdong, China, liuxilin3@huawei.com; Yuchi Ma, HUAWEI CLOUD, Guangdong, China, mayuchi1@huawei.com.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
© 2024 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM 2836-6573/2024/6-ART156
https://doi.org/10.1145/3654959

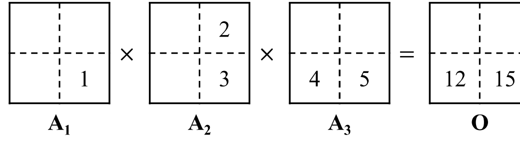


Fig. 1. An example for illustrating SMCM.

1 INTRODUCTION

A sparse matrix is a matrix in which most of the elements are zero. In many real-world areas including e-commerce, social networks, and biology, the interactions among different objects are often represented by sparse matrices. For example, in an e-commerce platform (e.g., Amazon), the shopping records of users buying items are often modeled as a large sparse matrix [13]. That is, if the i-th user purchases the j-th item, then the element in the i-th row and j-th column of the matrix will be 1. Figure 1 presents three sparse matrices, i.e., $\mathbf{A}_1$, $\mathbf{A}_2$, and $\mathbf{A}_3$, where zero elements are omitted.

In this paper, we study the *sparse matrix chain multiplication* (SMCM) problem [12, 32], which aims to efficiently multiply a chain of sparse matrices. For example, in Figure 1, we aim to multiply $\mathbf{A}_1$, $\mathbf{A}_2$, and $\mathbf{A}_3$, and the result is the matrix $\mathbf{O}$. SMCM has served as a fundamental computational kernel in various real-world applications such as network analysis, data mining, and machine learning. For example, its versatility has been demonstrated in numerous network analysis tasks, such as clustering [51, 57], community searching [18, 63, 68], triangle counting [62], shortest path enumeration [7], subgraph matching [58], NoSQL database operations [13, 21, 30, 33], and neural networks [20, 29, 37, 60, 65]. In the following, we present three concrete applications:

- ***Similarity search.*** As a popular similarity metric in heterogeneous information networks (HINs), PathSim [55] uses a symmetric meta-path $\mathcal{P}$ to measure the similarity between two vertices x and y as $sim(x,y) = \frac{2p(x,y)}{p(x,x)+p(y,y)}$, where $p(x,y)$ counts the instances of $\mathcal{P}$ between x and y, and $\mathcal{P}$ captures the semantic relationship, e.g., the meta-path $Author \rightarrow Paper \rightarrow Author$ shows the co-authorship in DBLP network. PathSim [55] computes the values of $p(x,y)$ between all the vertex pairs by $\mathbf{M}(T_x T_{x+1}) \times \mathbf{M}(T_{x+1} T_{x+2}) \times \cdots \times \mathbf{M}(T_{x+1} T_x)$, where $\mathbf{M}(T_x T_{x+1})$ is the sparse adjacency matrix between nodes of types T_x and T_{x+1}.
- ***Node embedding.*** HIN node embedding methods [17, 19, 48, 52, 66] often utilize meta-path-guided random walks to learn the relationships among nodes. These algorithms encompass two key steps: guided random walks employing meta-paths and using node2vec algorithm [25] for node embeddings. The first step can be calculated by $\mathbf{P}(T_i T_{i+1}) = \mathbf{D_i}^{-1} \times \mathbf{M}(T_i T_{i+1})$, where $\mathbf{P}(T_i T_{i+1})$ is the probability transition matrix, $\mathbf{M}(T_i T_{i+1})$ is the adjacency matrix between nodes with types T_i and T_{i+1}, and $\mathbf{D_i}$ is the degree matrix.
- ***Multi-source breadth-first search (BFS).*** Given a graph G and a set of source vertices S, the multi-source BFS aims to find the reachable vertices from vertices in S [31, 56]. It can be formulated as an SMCM problem: $\mathbf{B_k} = \mathbf{A}^k \times \mathbf{X}$, where $\mathbf{A}$ denotes the adjacent matrix of G, $\mathbf{X}$ denotes the source vertices (we first initialize $\mathbf{X} = 0$, and then let $\mathbf{X}[i,i] = 1$ if vertex $v_i \in S$), and $\mathbf{B_k}$ denotes the vertices that are reachable from vertices in S within k hops (if $\mathbf{B_k}[i,j] = 1$, then v_i is reachable to v_j within k hops). Clearly, $\mathbf{A}$ and $\mathbf{X}$ are sparse, so computing $\mathbf{B_k}$ is an SMCM problem.

Despite its popularity and usage, SMCM is computationally costly, especially when the sparse matrices are large. In the literature, SMCM has received plenty of research attention and several algorithms have been developed [2, 24, 43]. Generally, all these algorithms follow the same framework,

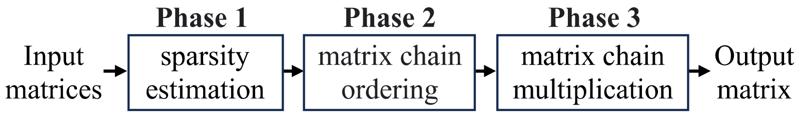


Fig. 2. The workflow of SMCM.

as depicted in Figure 2, involving three phases: (1) *sparsity estimation*, (2) *matrix chain ordering*, and (3) *matrix chain multiplication*. The *sparsity estimation* phase first evaluates the sparsity of the intermediate matrices which are the result matrices of multiplying two matrices. Subsequently, based on sparsity estimation, the *matrix chain ordering* phase derives a good execution order of SMCM to reduce the overall time cost. Finally, the matrices are multiplied by following the order in the *matrix chain multiplication* phase, which ultimately produces the final output matrix.

(1) Sparsity estimation. Given two sparse matrices $\mathbf{A}$ and $\mathbf{B}$, to estimate the sparsity of $\mathbf{O} = \mathbf{A}\times\mathbf{B}$, a few estimators [4, 12, 32, 53, 64] have been developed, where the sparsity is the ratio of non-zero elements in $\mathbf{O}$, as shown in Table 1. Boehm et al. [4] introduced an estimator `MetaAC`, which assumes the matrices are uniformly distributed matrices and estimates output matrix sparsity by calculating the probability of each element being non-zero. Cohen [12] introduced a graph-based method named `Layered-graph`, which estimates the number of non-zero elements by transforming a matrix chain multiplication into a layered graph. The sparsity is then estimated by determining the size of the transitive closure with the length of the chain for vertices in the graph [11]. Kernert et al. [32] introduced the `Density-map` method, which involves decomposing input matrices into individual $b\times b$ blocks. Subsequently, they applied `MetaAC` to estimate the sparsity of the multiplication between two blocks. Yu et al. [64] presented a sampling-based approach that calculates the maximum inner product value of columns and rows from samples to determine the sparsity of the output matrix. Sommer et al. [53] designed a novel estimator `MNC` using matrix non-zero count sketches. However, both `MetaAC` and `Density-map` rely on the assumption of uniformity and independence across the entire matrix structure, which may not hold in practice, and the sampling-based approach only focuses on the multiplication of two matrices, so it is not suitable for SMCM. `Layered-graph` assigns a vector of length r to each leaf node in the graph and computes their transitive closures. Consequently, its computational cost is significant, as indicated in Table 1. Regarding `MNC`, its precision is notably influenced by the matrix chain's length, wherein precision tends to decrease as the matrix chain length increases due to a reduction in the number of non-zero elements in the output matrix. Furthermore, all these estimators provide weak or no theoretical guarantees regarding the error between estimation and exact values.

(2) Matrix chain ordering. To multiply a chain of matrices, there are many different ways to parenthesize the matrices, which are often referred to as the orders of SMCM. Although different orders result in the same output matrix, they have a significant effect on efficiency. For example, in Figure 1, consider two different orders: (1) $(\mathbf{A}_1 \times \mathbf{A}_2) \times \mathbf{A}_3$ and (2) $\mathbf{A}_1 \times (\mathbf{A}_2 \times \mathbf{A}_3)$. Assume that the matrices are multiplied by following the order from left to right. Then, for order (1), we only need to perform 3 numerical multiplication calculations, while for order (2), we have to perform 6 numerical multiplication calculations, so order (1) is better. Note that the calculations for zero elements are skipped in SMCM.

To determine the optimal order of matrix chain multiplication, there is a classic dynamic programming algorithm [14, 24]. The algorithm inherently assumes that the input matrices are dense or full matrices, implying that when multiplying two matrices with sizes $m \times n$ and $n \times l$, we need $m \cdot n \cdot l$ numerical multiplication calculations. This assumption, however, no longer holds for sparse matrices, as we only need to multiply non-zero elements, whose number of numerical

Table 1. Sparsity estimators for multiplying two matrices $\mathbf{A}$ and $\mathbf{B}$, whose sizes are $m\times n$ and $n\times l$, respectively.

Estimator	Time complexity	SupportSMCM	Lowerbound	Upperbound
`MetaAC` [4]	$O(1)$	✓	×	×
`Density-map` [32]	$O(mnl/b^3)$	✓	×	×
`Layered-graph` [12]	$O(r(d+nnz(\mathbf{A},\mathbf{B})))$	✓	×	×
`Sampling` [64]	$O(\lvert\mathcal{I}\rvert(m+l))$	×	✓	×
`MNC` [53]	$O(nnz(\mathbf{A},\mathbf{B}))$	✓	✓	✓
RS-estimator	$O(nnz(\mathbf{A}))$	✓	✓	✓

⋆ $\mathcal{I}$ is the set of sampled instances; $nnz(\mathbf{A})$ is the number of non-zero elements in $\mathbf{A}$; d is the maximum value in $\{m, n, l\}$; b is the block size.

multiplication calculations is much less than $m \cdot n \cdot l$. Thus, existing works [4, 12, 32, 53, 64] often use a sparsity estimator to predict the cost of multiplying two matrices and then determine the order of SMCM by using dynamic programming.

(3) Matrix chain multiplication. After obtaining a good order of SMCM, we then perform a sequence of matrix-matrix multiplications, each of which multiplies two matrices. Although the input matrices of the chain are sparse, the intermediate matrices, which are the result matrices of multiplying two or more matrices, may be very dense. Therefore, matrix-matrix multiplication is also costly, especially when the matrix sizes are large and the intermediate matrices are dense.

To efficiently multiply two matrices, Gustavson et al. [26] introduced a sequential method based on the widely used compressed sparse row (CSR) data structure. However, it faces challenges in fully leveraging the computational capabilities of multi-core processors. To address this limitation, Patway et al. [47] extended the approach to multi-core processors by exploring partitioning schemes. Nonetheless, this extension comes with its own set of limitations. Specifically, the partition schemes for the two input matrices are different, as one uses row partitions and the other uses column partitions, so it necessitates the different storage formats of the matrices, which causes format transformation when dealing with a chain of matrices. When writing the output matrix, a thread responsible for writing the i-th row to memory must wait for all other threads handling rows up to the (i-1)-th row to complete their operations, leading to synchronization issues. Thus, it is desirable to develop faster matrix-matrix multiplication algorithms.

Our technical contributions. To enable efficient SMCM, we aim to develop efficient algorithms by optimizing matrix sparsity estimation, matrix chain ordering, and matrix chain multiplication. We first propose a novel matrix sparsity estimation method, called *row-wise sparsity estimator* (`RS-estimator`), for estimating the result matrix sparsity of two input matrices, by leveraging the structural information of the matrices. Different from existing estimators that analyze entire matrices, `RS-estimator` focuses on individual matrix rows, specifically targeting the row-based structural characteristics within the left matrix of the two input matrices. Contrastingly, both `MetaAC` and `Density-map` neglect row-wise sparsity. `MetaAC` relies on the sparsity of input matrices, while `Density-map` partitions input matrices into smaller blocks, using their sparsity to estimate corresponding block sparsity in the output matrix. Besides, `Layered-graph` assigns r-length vectors, randomly drawn from an exponential distribution with $\lambda = 1$, to all leaf nodes and propagates them upward through the layered graph to estimate the sparsity. On the one hand, `MNC` employs the numbers of non-zero elements in each row and column to estimate the number of non-zero elements in the output matrix, subsequently calculating the overall sparsity directly. However, it does not explicitly compute the row-wise sparsity (i.e., the ratio of non-zero elements in each row). In comparison, our `RS-estimator` directly utilizes input matrix row-wise sparsity to estimate the corresponding row-wise sparsity in the output matrix, eliminating the need for counting non-zero

elements in each column. This not only facilitates extracting valuable structural insights but also enhances parallelization by allowing independent processing of each row. We further analyze the accuracy of `RS-estimator` and theoretically quantify the absolute error gap between the estimated sparsity and the exact sparsity.

Based on the `RS-estimator`, we develop a method for estimating the sparsity of the result matrix by multiplying a chain of matrices and also propose a dynamic programming algorithm to determine a good order for SMCM. To accelerate matrix chain multiplication with a given order, we propose a novel parallel matrix multiplication algorithm, which uses the sparse adjacent list data structure to represent the sparse matrix. Since the sparse adjacent list independently stores rows of the matrix, we can easily perform computation in parallel, by using individual threads' independent cache management to efficiently reduce contention during multi-threaded memory allocation and synchronization costs.

We have conducted experiments by multiplying various large sparse matrices extracted from five real-world large graph datasets, and the results demonstrate the effectiveness and efficiency of our proposed methods. In particular, the accuracy of our `RS-estimator` is much higher than those of existing sparsity estimators, and our SMCM algorithm is up to three orders of magnitude faster than the state-of-the-art approaches. Furthermore, we have demonstrated the utility of our algorithms in real applications.

Outline. We review related work in Section 2, formally introduce the SMCM problem in Section 3, and present our proposed `RS-estimator` along with theoretical analysis in Section 4. Our parallel SMCM algorithm is described in Section 5. Experimental results are reported in Section 6, and we conclude in Section 7.

2 RELATED WORK

In this section, we review the related work of sparsity estimators for multiplying two matrices and sparse matrix multiplication.

• **Sparsity estimator of multiplying two matrices.** To accurately estimate the sparsity of multiplying two matrices, a few estimators have been proposed [4, 12, 32, 53, 64]. Boehm et al. [4] introduced `MetaAC`. It operates under the assumption of uniformly distributed matrices and estimates output sparsity by calculating the probability of an element being non-zero. Leveraging similar propagation techniques as `MetaAC`, Sparso et al. [50] identified structural properties exploitable by subsequent data-dependent operations, including considerations related to symmetric, triangular, and diagonal matrices. Cohen [12] introduced `Layered-graph`, a graph-based method estimating the number of non-zero elements by transforming a matrix chain multiplication into a layered graph. Kernert et al. [32] introduced the `Density-map` method, which involves decomposing input matrices into individual blocks to estimate the sparsity. Yu et al. [64] presented a sampling-based approach that calculates the maximum inner product value of columns and rows from samples to determine the sparsity of the output matrix. Sommer et al. [53] introduced `MNC`, a novel estimator for matrix product chains using matrix non-zero count sketch, providing valuable insights into the matrix data structure.

Nevertheless, the estimators above have severe limitations. `MetaAC` relies on the assumption of uniformity and independence across the entire matrix structure, which is often not met in practice. The sampling-based method is suitable only for multiplying two matrices, so it is not suitable for the SMCM problem. When dealing with SMCM, `MNC` faces challenges in accurately estimating the exact number of non-zero elements, particularly when the largest elements in `MNC` sketch exceed one. Moreover, `MetaAC` does not provide any theoretical guarantee on the accuracy, while the sampling-based estimator only offers lower bounds of accuracy. Hence, they do not offer a strong theoretical guarantee.

• **Sparse matrix multiplication.** We review the works of sparse matrix-matrix multiplication and SMCM respectively. The former one is to multiply two sparse matrices **A** and **B**. Gustavson [26] introduced an algorithm with time complexity proportional to two key factors: the number of non-zero elements in matrix **A** and matrix **B**. When attempting parallelization, this algorithm faces a significant challenge due to the synchronization bottleneck associated with CSR matrices [26, 40]. Patwary et al. [47] expanded upon Gustavson's algorithm to facilitate parallelization on CPUs by partitioning matrix **B** based on its columns. However, this approach introduces format conversion overhead since the two input matrices are stored in different formats, further exacerbating the synchronization bottleneck. A similar approach is employed by MATLAB, which processes one column of the result matrix at a time. MATLAB utilizes a dense vector containing values, indices, and valid flags for accumulating sparse partial results [23]. Buluc and Gilbert tackled the scenario of hyperspace matrices, where the number of non-zero elements is less than the number of columns or rows [6]. Besides, the problem has been studied on various hardware architectures, encompassing GPUs [15, 35, 44, 61], FPGAs [39], ASICs [28, 46, 54], heterogeneous setups [41], and distributed platforms [1, 16, 38].

Although these approaches have demonstrated effectiveness in some applications, they do not specifically target the SMCM which involves a chain of matrices. Recently, people have developed some SMCM algorithms, which often involve two phases of matrix chain ordering and matrix chain multiplication. Chikalov et al. [10] introduced a method for sequential order optimization of matrix chain multiplication by considering various cost functions. Myung et al. [42] implemented SMCM on the MapReduce platform, which represents matrices as (row, column, value) records and translates multiplication into database-style joins. Biswas et al. [3] proposed a GPU-based approach for SMCM, focusing on optimizing memory coalescing within the device. Nevertheless, as mentioned before, there are still many issues in both matrix chain ordering and matrix chain multiplication, calling for faster SMCM algorithms.

3 PRELIMINARIES

In this section, we first formally introduce the SMCM problem and then introduce the basic knowledge of matrix-matrix multiplication.

3.1 Problem definition

In this paper, we use bold uppercase letters (e.g., **A**) to represent matrices. The size of a matrix is the number of rows by the number of columns. Given a matrix $\mathbf{A}$, the i-th row of $\mathbf{A}$ is denoted by $\mathbf{A}[i,*]$, the j-th column is denoted by $\mathbf{A}[*,j]$, and the element in the i-th row and j-th column is denoted by $\mathbf{A}[i,j]$. The numbers of non-zero elements in a matrix $\mathbf{A}$ and one row of $\mathbf{A}$ are denoted by $nnz(\mathbf{A})$ and $nnz(\mathbf{A}[i,*])$ respectively. The frequently used notations are summarized in Table 2.

A matrix is generally called a sparse matrix if most of the elements are zero. Note that there is no standard definition of sparse matrix in the literature. The number of non-zero elements in the matrix reflects its sparsity, and the sparsity of $\mathbf{A}$ is often defined as $\rho(\mathbf{A}) = \frac{nnz(\mathbf{A})}{m \cdot n}$, where the size of $\mathbf{A}$ is $m \times n$.

Problem 1 (SMCM [12, 32])**.** *Given a chain of sparse matrices* $\mathbf{A_1}, \mathbf{A_2}, \cdots, \mathbf{A_p}$*, where* $\mathbf{A_i}$*'s size is* $n_{(i-1)} \times n_i$ *and* $p \geq 3$ $(1 \leq i \leq p)$*, compute the output matrix* $\mathbf{O} = \mathbf{A_1} \times \mathbf{A_2} \times \cdots \times \mathbf{A_p}$.

For example, in Figure 1, the goal is to multiply three sparse matrices $\mathbf{A_1}$, $\mathbf{A_2}$, and $\mathbf{A_3}$, and the output matrix is $\mathbf{O}$.

Table 2. Notations and meaning.

Notation	Meaning
$\mathbf{A}[i,*]$	The i-th row in matrix $\mathbf{A}$
$\mathbf{A}[*,j]$	The j-th column in matrix $\mathbf{A}$
$\mathbf{A}[i,j]$	The element with index (i,j) in matrix $\mathbf{A}$
$nnz(\mathbf{A})$	The number of non-zero elements in matrix $\mathbf{A}$
$nnz(\mathbf{A}[i,*])$	The number of non-zero elements in a row $\mathbf{A}[i,*]$
$\rho(\mathbf{A})$	The sparsity of matrix $\mathbf{A}$, i.e., $\rho(\mathbf{A}) = \frac{nnz(\mathbf{A})}{m \cdot n}$
$\mathbf{A_{i,j}}$	The result matrix of $\mathbf{A_i} \times \mathbf{A_{i+1}} \cdots \times \mathbf{A_j}$
$\eta(\mathbf{A},i)$	The row-wise sparsity of a row $\mathbf{A}[i,*]$ in matrix $\mathbf{A}$
$\mathbf{r}(\mathbf{A})$	The row-wise sparsity vector of matrix $\mathbf{A}$

3.2 Matrix-matrix multiplication

To multiply two matrices $\mathbf{A}$ and $\mathbf{B}$, there are four methods to access the matrices [22]: row-by-column (inner product), column-by-row (outer product), row-by-row (row-wise product), and column-by-column (column-wise product), as shown in Table 3.

Table 3. Four methods of computing $\mathbf{O} = \mathbf{A} \times \mathbf{B}$.

Method	Calculation formula
Inner product	$\mathbf{O}[i,j] = \sum_{k=1}^{n} \mathbf{A}[i,k] \cdot \mathbf{B}[k,j]$
Outer product	$\mathbf{O} = \sum_{k=1}^{n} \mathbf{A}[*,k] \cdot \mathbf{B}[k,*]$
Row-wise product	$\mathbf{O}[i,*] = \sum_{k=1}^{n} \mathbf{A}[i,k] \cdot \mathbf{B}[k,*]$
Column-wise product	$\mathbf{O}[*,j] = \sum_{k=1}^{n} \mathbf{A}[*,k] \cdot \mathbf{B}[k,j]$

Although the inner product is frequently used in textbooks, the row- and column-wise products [26] are more efficient for parallel computing since the computations for rows and columns are independent of each other. Besides, in contrast to inner and outer products, the row- and column-wise products allow us to use the same format for the input matrices and output matrix since SMCM requires a consistent format of the inputs and outputs [54]. Thus, in this work, we use the row-wise product.

In the row-wise product, each element $\mathbf{A}[i,k]$ of $\mathbf{A}$ is multiplied with each element $\mathbf{B}[k,j]$ in the k-th row of $\mathbf{B}$, and the result $\mathbf{A}[i,k] \cdot \mathbf{B}[k,j]$ will be accumulated into the element $\mathbf{O}[i,j]$ of output matrix $\mathbf{O}$.

Example 1. *Figure 3 shows the process of running row-wise product for* $\mathbf{A} \times \mathbf{B}$*, where* $\mathbf{A}$ *has four elements* $\mathbf{A}[1,1]$*,* $\mathbf{A}[1,2]$*,* $\mathbf{A}[2,1]$*, and* $\mathbf{A}[2,2]$*. For* $\mathbf{A}[1,1]$ *and* $\mathbf{A}[2,1]$*, they need to multiply with the 1-st row of* $\mathbf{B}$*, respectively. For* $\mathbf{A}[1,2]$ *and* $\mathbf{A}[2,2]$*, it needs to multiply with the 2-nd row of* $\mathbf{B}$*, respectively. Finally, we accumulate the results and get the output matrix* $\mathbf{O}$*.*

A seminal feature of the row-wise product is that each row $\mathbf{O}[i,*]$ of the output matrix $\mathbf{O}$ depends only on the i-th row $\mathbf{A}[i,*]$ of $\mathbf{A}$, and is independent of any other row of $\mathbf{A}$. In other words, it accesses both input matrices and the output matrix in row-major order, thereby enabling fast concurrent processing of different rows across distinct CPU threads without the need for locking operations. In Example 1, we can schedule two CPU threads such that each thread processes the multiplication of all the elements in one row of $\mathbf{A}$.

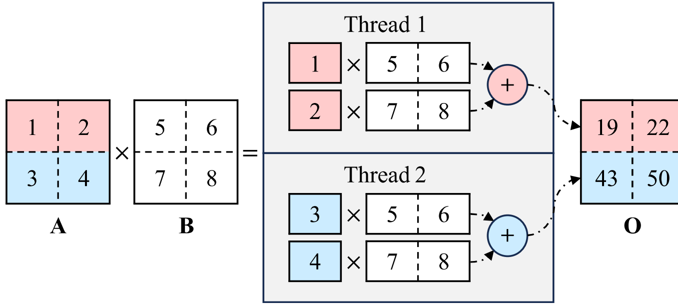


Fig. 3. Illustrating the row-wise product for $\mathbf{A} \times \mathbf{B}$.

4 SPARSITY ESTIMATION

As aforementioned, the order of executing the matrix multiplication has a significant effect on the SMCM efficiency. Determining the optimal order is non-trivial since the number of possible orders is exponentially large [36]. Fortunately, there is a classic dynamic programming algorithm [14, 24] for computing the optimal order. Specifically, denote by $\mathbf{A_{i,j}}$ the result matrix of $\mathbf{A_i} \times \mathbf{A_{i+1}} \times \cdots \times \mathbf{A_j}$, where $1 \leq i \leq j$. Then, the minimum cost of computing $\mathbf{A_{i,j}}$ is

$$cost(\mathbf{A_{i,j}}) = \min_{i \leq k < j} \left\{ cost(\mathbf{A_{i,k}}) + cost(\mathbf{A_{k+1,j}}) + cost(\mathbf{A_{i,k}} \times \mathbf{A_{k+1,j}}) \right\}, \quad (1)$$

where $cost(\mathbf{A_{i,k}} \times \mathbf{A_{k+1,j}})$ represents the cost of multiplying two intermediate matrices $\mathbf{A_{i,k}}$ and $\mathbf{A_{k+1,j}}$. Note that $cost(\mathbf{A_{i,i}}) = 0$.

When the input matrices are full or dense, we can simply let $cost(\mathbf{A_{i,k}} \times \mathbf{A_{k+1,j}}) = n_{(i-1)} \cdot n_k \cdot n_j$. However, for sparse matrices, since most of the elements are zero and their multiplications can be skipped, $cost(\mathbf{A_{i,k}} \times \mathbf{A_{k+1,j}})$ may be much smaller than $n_{(i-1)} \cdot n_k \cdot n_j$. As a result, we cannot directly let $cost(\mathbf{A_{i,k}} \times \mathbf{A_{k+1,j}})$ be $n_{(i-1)} \cdot n_k \cdot n_j$. In the literature, existing algorithms often estimate the value of $cost(\mathbf{A_{i,k}} \times \mathbf{A_{k+1,j}})$ by designing matrix sparsity estimators.

In the following, we propose a novel sparsity estimator in Section 4.1, and further theoretically analyze its accuracy in Section 4.2.

4.1 A row-wise sparsity estimator

To estimate the computational cost of sparse matrix-matrix multiplication, Kernert et al. [32] designed an effective cost model for multiplying two sparse matrices, which we will also use:

Definition 1 (Cost model [32]). *Let* $\mathbf{A}$ *and* $\mathbf{B}$ *be two sparse matrices with sizes* $m \times n$ *and* $n \times l$ *respectively. The cost of computing* $\mathbf{O} = \mathbf{A} \times \mathbf{B}$ *can be modeled by*

$$cost(\mathbf{A} \times \mathbf{B}) \approx \alpha \underbrace{\Big(m \cdot n \cdot \rho(\mathbf{A})\Big)}_{nnz(\mathbf{A})} + \beta \underbrace{\Big(m \cdot n \cdot \rho(\mathbf{A}) \cdot l \cdot \rho(\mathbf{B})\Big)}_{\widehat{N_{op}}} + \gamma \underbrace{\Big(m \cdot l \cdot \widehat{\rho}(\mathbf{O})\Big)}_{\widehat{nnz}(\mathbf{O})}, \quad (2)$$

where the coefficients α, β, *and* γ *are constant values associated with hardware read and write operations during matrix multiplication, and* $\widehat{\rho}(\mathbf{O})$ *indicates the estimated sparsity of* $\mathbf{O}$. $\widehat{N_{op}}$ *denotes the estimated number of numerical multiplication calculations, and* $\widehat{nnz}(\mathbf{O})$ *denotes the estimated count of non-zero elements in* $\mathbf{O}$.

From Equation (2), we see that accurately estimating the sparsity of the result matrix $\mathbf{O}$, $\widehat{\rho}(\mathbf{O})$, plays a pivotal role in estimating the cost of sparse matrix-matrix multiplication.

In the following, we propose a row-wise sparsity estimator, also called `RS-estimator`, for estimating $\rho(\mathbf{O})$, where $\mathbf{O}$ is the output matrix of multiplying two sparse matrices $\mathbf{A}$ and $\mathbf{B}$. We first introduce a widely used assumption in sparse matrix multiplication [12]. Given the potential presence of negative values within the matrix, certain computations may yield zero as a result of the cancellation of positive and negative values. In the context of matrix multiplication, these elements are considered non-zero, as they still entail a computational overhead in their acquisition.

Assumption 1. *In sparse matrix multiplication, zero elements do not arise during the element aggregation process [12].*

Then we put forth a novel assumption for sparsity estimation.

Assumption 2. *All the matrices satisfy the following properties:*

(1) *The locations of non-zero elements are randomly and independently distributed across the rows and columns;*
(2) *Within any given row, all non-zero elements exhibit a uniform and independent distribution;*
(3) *The probability of an element being a non-zero element is directly proportional to the sparsity of the row in which it is situated.*

Our `RS-estimator` is mainly grounded in these two key assumptions, which have been substantiated through subsequent experiments. To leverage the structural information within matrices, we introduce the concept of row-wise sparsity.

Definition 2 (Row-wise sparsity). *Given an $m \times n$ matrix $\mathbf{A}$, the row-wise sparsity of the i-th ($1 \leq i \leq m$) row is defined as*

$$\eta(\mathbf{A}, i) = \frac{nnz(\mathbf{A}[i, *])}{n}. \quad (3)$$

By considering the row-wise sparsity of all the rows in $\mathbf{A}$, we then obtain a *row-wise sparsity vector* of $\mathbf{A}$

$$\mathbf{r}(\mathbf{A}) = (\eta(\mathbf{A}, 1), \eta(\mathbf{A}, 2), \cdots, \eta(\mathbf{A}, m)). \quad (4)$$

Table 4. Illustrating sparsity and row-wise sparsity.

	$\mathbf{A}_1$	$\mathbf{A}_2$	$\mathbf{A}_3$	$\mathbf{O}$
Matrix sparsity	0.25	0.5	0.5	0.5
Row-wise sparsity vector	(0, 0.5)	(0.5, 0.5)	(0, 1)	(0, 1)

Example 2. *Table 4 reports the sparsity and row-wise sparsity vectors of the four matrices in Figure 1. For instance, the row-wise sparsity vector of $\mathbf{A}_1$ is $(0, 0.5)$, and the sparsity of $\mathbf{A}_1$ is $(0 + 0.5)/2 = 0.25$. Clearly, the overall matrix sparsity is actually the mean value of the row-wise sparsity of all rows.*

Given a matrix-matrix multiplication $\mathbf{O} = \mathbf{A} \times \mathbf{B}$ where the sizes of $\mathbf{A}$ and $\mathbf{B}$ are $m \times n$ and $n \times l$ respectively, we extend the sparsity analysis from the entire matrix to row-wise sparsity. Leveraging Lemma 4.1 in [32], our `RS-estimator` estimates the row-wise sparsity of $\mathbf{O}$ by Lemma 1.

Lemma 1. *The row-wise sparsity of the i-th row of matrix $\mathbf{O} = \mathbf{A} \times \mathbf{B}$, $\eta(\mathbf{O}, i)$, can be estimated as follows:*

$$\widehat{\eta}(\mathbf{O}, i) = \begin{cases} 0, & \text{if } \eta(\mathbf{A}, i) = 0; \\ 1 - \prod_{\mathbf{A}[i,k] \neq 0} \left(1 - \eta(\mathbf{B}, k)\right), & \text{otherwise}, \end{cases} \quad (5)$$

where $i \in [1, m]$ and $k \in [1, n]$.

The proof of Lemma 1 aligns with Lemma 4.1 in [32]; therefore, we omit it.

Example 3. *In Figure 1, let* $\mathbf{A}_{1,2} = \mathbf{A}_1 \times \mathbf{A}_2$. *Then, we know that* $\eta(\mathbf{A}_{1,2}, 1) = 0$ *and* $\eta(\mathbf{A}_{1,2}, 2) = 0.5$. *We can use Lemma 1 to estimate their values, i.e.,* $\widehat{\eta}(\mathbf{A}_{1,2}, 1) = 0$, *and* $\widehat{\eta}(\mathbf{A}_{1,2}, 2) = 1 - (1 - 0.5) = 0.5$.

Algorithm 1 summarizes the steps of our `RS-estimator`. Given input matrices $\mathbf{A}$ and $\mathbf{B}$, we first compute the row-wise sparsity vector of $\mathbf{B}$ (line 1). Then, we initialize $\widehat{r}(\mathbf{O})$ (line 2). Next, for each row in $\mathbf{O}$, we compute its row-wise sparsity using Lemma 1. Specifically, $\widehat{\eta}(\mathbf{O}, i)$ relies on the non-zero elements in row $\mathbf{A}[i, *]$, and the variable *temp* is used to compute $\widehat{\eta}(\mathbf{O}, i)$ (lines 3-7). This process can be efficiently parallelized as each $\widehat{\eta}(\mathbf{O}, i)$ is independent.

Algorithm 1: `RS-estimator`

```
Input: two sparse matrices A and B
Output: the estimated row-wise sparsity vector of O=A×B
1 compute r(B) ← (η(B, 1), η(B, 2), ··· , η(B, n));
2 initialize a vector r̂(O) ← 0;
3 for each row A[i, *] ∈ A in parallel do
4     temp ← 1;
5     for each non-zero element A[i, k] ∈ A[i, *] do
6         temp ← temp · (1 − η(B, k));
7     η̂(O, i) ← 1 − temp;
8 return r̂(O) = (η̂(O, 1), η̂(O, 2), ··· , η̂(O, m));
```

Time complexity. The time cost of Algorithm 1 is $O(nnz(\mathbf{A}))$, as it processes the non-zero elements in matrix $\mathbf{A}$ and matches them with the respective row-wise sparsity in $\mathbf{r}(\mathbf{B})$.

4.2 Theoretical analysis for RS-estimator

We now theoretically analyze the accuracy of `RS-estimator` by considering the gap between $\eta(\mathbf{O}, i)$ and $\widehat{\eta}(\mathbf{O}, i)$. When $\eta(\mathbf{A}, i) = 0$, then $\widehat{\eta}(\mathbf{O}, i) = \eta(\mathbf{O}, i) = 0$. For the case that $\eta(\mathbf{A}, i) \neq 0$, we can derive the lower and upper bounds of $\eta(\mathbf{O}, i)$ and $\widehat{\eta}(\mathbf{O}, i)$ as follows.

Lemma 2. *Given* $\mathbf{O} = \mathbf{A} \times \mathbf{B}$, *if* $\eta(\mathbf{A}, i) \neq 0$, *then the lower and upper bounds of* $\eta(\mathbf{O}, i)$ *can be stated by*

$$\max_{\mathbf{A}[i,k]\neq 0} \Big\{\eta(\mathbf{B}, k)\Big\} \leq \eta(\mathbf{O}, i) \leq \min\Big\{1, \sum_{\mathbf{A}[i,k]\neq 0} \eta(\mathbf{B}, k)\Big\}, \tag{6}$$

where $i \in [1, m]$ *and* $k \in [1, n]$.

Proof. According to the definition of row-wise product, $\eta(\mathbf{O}, i) = \frac{1}{l} \cdot nnz(\mathbf{O}[i, *])$. We can further derive

$$\begin{aligned}
\eta(\mathbf{O}, i) &\geq \frac{1}{l} \cdot \max_{\mathbf{A}[i,k]\neq 0} \Big\{nnz(\mathbf{B}[k, *])\Big\} \\
&= \max_{\mathbf{A}[i,k]\neq 0} \Big\{\eta(\mathbf{B}, k)\Big\}, \\
\eta(\mathbf{O}, i) &\leq \frac{1}{l} \cdot \min\Big\{l, \sum_{\mathbf{A}[i,k]\neq 0} nnz(\mathbf{B}[k, *])\Big\} \\
&= \min\Big\{1, \sum_{\mathbf{A}[i,k]\neq 0} \eta(\mathbf{B}, k)\Big\}.
\end{aligned} \tag{7}$$

Therefore, the lemma holds. □

Before showing the upper and lower bounds of $\widehat{\eta}(\mathbf{O}, i)$, we introduce an auxiliary function as follows.

Lemma 3. *Let* $f(x_1, x_2, \cdots, x_n) =$

$$\sum_{i=1}^{n} x_i + \prod_{i=1}^{n} \Big(1 - x_i\Big) - 1. \tag{8}$$

Then, $f(x_1, x_2, \cdots, x_n)$ *is a non-decreasing function if* $0 \leq x_i \leq 1$ *for all* $i \in [1, n]$.

Proof. The partial derivative of f with respect to each x_k is computed as follows:

$$\frac{\partial f}{\partial x_k} = 1 - \prod_{i \neq k} \Big(1 - x_i\Big) \geq 0. \tag{9}$$

Hence, $f(x_1, x_2, \cdots, x_n)$ is a non-decreasing function. □

Since $f(x_1, x_2, \cdots, x_n)$ is a non-decreasing function, it achieves the minimum value 0 when $x_i = 0$ for all $i \in [1, n]$.

Lemma 4. *The lower and upper bounds of* $\eta(\mathbf{O}, i)$ *in Lemma 2 can also be applied for* $\widehat{\eta}(\mathbf{O}, i)$*, i.e., they can share identical bounds.*

Proof. We sequentially prove that the lower and upper bounds of $\eta(\mathbf{O}, i)$ can be applied to $\widehat{\eta}(\mathbf{O}, i)$.

Lower bound: From Equation (5), we can easily see that with the increase of $\eta(\mathbf{A}, i)$, the value of $\widehat{\eta}(\mathbf{O}, i) = 1 - \prod_{\mathbf{A}[i,k] \neq 0} (1 - \eta(\mathbf{B}, k))$ never decreases. Hence, $\widehat{\eta}(\mathbf{O}, i) \geq \max_{\mathbf{A}[i,k] \neq 0} \{\eta(\mathbf{B}, k)\}$, which is the same as the lower bound of $\eta(\mathbf{O}, i)$ stated by Lemma 2.

Upper bound: Obviously, $\widehat{\eta}(\mathbf{O}, i) \leq 1$, since the value of row-wise sparsity is always at most 1. Besides, in Lemma 3, if let $x_k = \eta(\mathbf{B}, k)$, then we have:

$$\sum_{\mathbf{A}[i,k] \neq 0} \eta(\mathbf{B}, k) + \prod_{\mathbf{A}[i,k] \neq 0} \Big(1 - \eta(\mathbf{B}, k)\Big) - 1 \geq 0. \tag{10}$$

By Equation (5), we have

$$\sum_{\mathbf{A}[i,k] \neq 0} \eta(\mathbf{B}, k) - \widehat{\eta}(\mathbf{O}, i) \geq 0. \tag{11}$$

Therefore, we conclude $\widehat{\eta}(\mathbf{O}, i) \leq \min \Big\{1, \sum_{\mathbf{A}[i,k] \neq 0} \eta(\mathbf{B}, k)\Big\}$, which is the same as the upper bound of $\eta(\mathbf{O}, i)$ stated by Lemma 2. Hence, Lemma 4 holds. □

Next, we delve into the error analysis concerning the estimated row-wise sparsity of the output matrix compared to its exact value. For each estimated row-wise sparsity $\widehat{\eta}(\mathbf{O}, i)$ of i-th row in the output matrix $\mathbf{O}$, its absolute error $\epsilon(\mathbf{O}, i) = |\eta(\mathbf{O}, i) - \widehat{\eta}(\mathbf{O}, i)|$ can be determined using the follow theorem.

Theorem 1. *Given* $\mathbf{O} = \mathbf{A} \times \mathbf{B}$*, we have*

$$\epsilon(\mathbf{O}, i) < \begin{cases} e^{-1}, & \text{if } \eta(\mathbf{O}, i) > \widehat{\eta}(\mathbf{O}, i); \\ 1, & \text{otherwise.} \end{cases} \tag{12}$$

Proof. We prove the two cases sequentially.

Case I: $\eta(\mathbf{O}, i) > \widehat{\eta}(\mathbf{O}, i)$. The absolute error $\epsilon(\mathbf{O}, i)$ of row $\mathbf{O}[i, *]$ is less than the function g as follows:

$$g = \min\left\{1, \sum_{\mathbf{A}[i,k]\neq 0} \eta(\mathbf{B}, k)\right\} - 1 + \prod_{\mathbf{A}[i,k]\neq 0} \Big(1 - \eta(\mathbf{B}, k)\Big). \tag{13}$$

Then, the partial derivative of g with respect to each $\eta(\mathbf{B}, x)$ is

$$\frac{\partial g}{\partial \eta(\mathbf{B}, x)} = \begin{cases} 1 - \prod_{\mathbf{A}[i,k]\neq 0 \wedge k \neq x} \Big(1 - \eta(\mathbf{B}, k)\Big), & \text{if } \sum_{\mathbf{A}[i,k]\neq 0} \eta(\mathbf{B}, k) \leq 1; \\ -\prod_{\mathbf{A}[i,k]\neq 0 \wedge k \neq x} \Big(1 - \eta(\mathbf{B}, k)\Big), & \text{otherwise.} \end{cases} \tag{14}$$

Thus, to maximize g, it necessitates $\sum_{\mathbf{A}[i,k]\neq 0} \eta(\mathbf{B}, k) \geq 1$. Utilizing the Lagrange multipliers method, we obtain that g attains its maximum, when $\eta(\mathbf{B}, k) = \frac{1}{nnz(\mathbf{A}[i,*])}$, where $nnz(\mathbf{A}[i, *]) \neq 0$. Consequently, g satisfies the following equation:

$$g < \lim_{nnz(\mathbf{A}[i,*])\to\infty} \left(1 - \frac{1}{nnz(\mathbf{A}[i, *])}\right)^{nnz(\mathbf{A}[i,*])} = e^{-1}. \tag{15}$$

Case II: $\eta(\mathbf{O}, i) \leq \widehat{\eta}(\mathbf{O}, i)$. We consider the densest row of matrix $\mathbf{B}$ corresponding to non-zero elements in row $\mathbf{A}[i, *]$ of $\mathbf{A}$, and denote it as the k'-th row, where $k' = \arg\max_{\mathbf{A}[i,k]\neq 0} \{\eta(\mathbf{B}, k)\}$.

According to Lemma 4, the absolute error $\epsilon(\mathbf{O}, i)$ is less than the function h which can be described as:

$$h = 1 - \prod_{\mathbf{A}[i,k]\neq 0} \Big(1 - \eta(\mathbf{B}, k)\Big) - \eta(\mathbf{B}, k'). \tag{16}$$

The partial derivative of h for each $\eta(\mathbf{B}, x)$ is computing as follows:

$$\frac{\partial h}{\partial \eta(\mathbf{B}, x)} = \begin{cases} \prod_{\mathbf{A}[i,k]\neq 0 \wedge k \neq x} (1 - \eta(\mathbf{B}, k)), & \text{if } x \neq k'; \\ \prod_{\mathbf{A}[i,k]\neq 0 \wedge k \neq x} (1 - \eta(\mathbf{B}, k)) - 1, & \text{otherwise.} \end{cases} \tag{17}$$

The function h exhibits monotonic growth as $\eta(\mathbf{B}, k')$ decreases or $\eta(\mathbf{B}, x)$ increases since $0 < \prod_{\mathbf{A}[i,k]\neq 0 \wedge k \neq x} (1 - \eta(\mathbf{B}, k)) < 1$. Besides, h monotonically increases as $\eta(\mathbf{A}, i)$ increases. Therefore, the maximum of h occurs when $\eta(\mathbf{A}, i) = 1$ and $\eta(\mathbf{B}, k') = \eta(\mathbf{B}, k)$ for $k = 1, \cdots, n$. Applying the Lagrange multipliers method, h attains its maximum when $\eta(\mathbf{B}, k) = 1 - n^{\frac{1}{1-n}}$. Conclusively, we have

$$h < \lim_{n\to\infty} \left(n^{\frac{1}{1-n}} - n^{\frac{n}{1-n}}\right) = 1. \tag{18}$$

Therefore, the theorem holds. □

In addition, `RS-estimator` is able to estimate the exact row-wise sparsity of the output matrix $\mathbf{O}$ when the input matrices $\mathbf{A}$ and $\mathbf{B}$ satisfy some conditions, as stated by Lemma 5.

Lemma 5. *Given* $\mathbf{O}$=$\mathbf{A}\times\mathbf{B}$, *if any of the following conditions is met,*

(1) *there is at most one non-zero element in the i-th row of matrix* $\mathbf{A}$, *i.e.,* $\eta(\mathbf{A}, i) \leq \frac{1}{m}$ *[53],*
(2) *there is one non-zero element* $\mathbf{A}[i, k]$ *of the i-th row of matrix* $\mathbf{A}$ *and its corresponding k-th row of* $\mathbf{B}$ *having* $\eta(\mathbf{B}, k) = 1$,

then we can claim $\eta(\mathbf{O}, i) = \widehat{\eta}(\mathbf{O}, i)$.

Proof. The proof of condition (1) is presented in Theorem 3.1 in [53]. For condition (2), if there is one non-zero element $\mathbf{A}[i, k]$ in $\mathbf{A}[i, *]$ whose corresponding row in $\mathbf{B}$ is full, then the row $\mathbf{O}[i, *]$ is full, i.e., $\eta(\mathbf{O}, i) = 1$. By applying Equation (5) again, we can claim $\widehat{\eta}(\mathbf{O}, i) = 1 - 0 = 1 = \eta(\mathbf{O}, i)$. Conclusively, the lemma holds. □

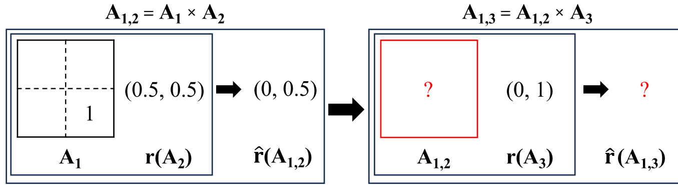


Fig. 4. An attempt to use RS-estimator to estimate the row-wise sparsity vector of $\mathbf{A}_{1,3} = (\mathbf{A}_1 \times \mathbf{A}_2) \times \mathbf{A}_3$ in Figure 1.

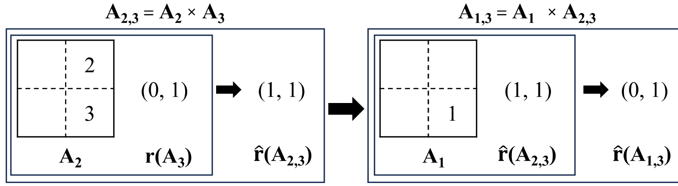


Fig. 5. Estimating the row-wise sparsity vector of $\mathbf{A}_{1,3} = \mathbf{A}_1 \times (\mathbf{A}_2 \times \mathbf{A}_3)$.

Unlike existing estimators [4, 32, 53, 64] which often estimate the sparsity of the whole matrix, RS-estimator focuses on estimating the row-wise sparsity. In other words, existing estimators produce coarse-grained estimated results, while our RS-estimator offers a more fine-grained sparsity, so our estimated results tend to be more accurate. Moreover, we provide a vigorous theoretical analysis of the accuracy, by proving both the lower and upper bounds of the gap between ground-truth sparsity and estimated sparsity. In addition, as shown in Algorithm 1, the estimation process of RS-estimator is efficient, since it can be easily parallelized.

5 OUR SMCM ALGORITHM

In this section, we first present the ordering algorithm for performing SMCM, then propose a parallel algorithm for matrix-matrix multiplication, and finally present our overall SMCM algorithm.

5.1 Matrix chain ordering

Recall that RS-estimator only estimates the sparsity of the output matrix by multiplying two sparse matrices. We now extend it to the case of multiplying a chain of matrices. Given a chain of p matrices, there are $p \cdot (p+1)/2$ non-empty sub-chains in total. To derive a good order, we need to estimate the sparsity of each sub-chain.

In existing SMCM methods, they estimate the sparsity of a sub-chain following an arbitrary order. However, our approach necessitates estimating the sparsity of the result matrices following a right-to-left order in the sub-chain, since RS-estimator needs the positions of non-zero elements in the left matrix. If we attempt to use a left-to-right order to estimate the sparsity of intermediate result matrices, we will not be able to obtain the positions of non-zero elements in the intermediate result matrices, which serve as the left matrices later, since we do not compute the intermediate result matrices during the estimation. Example 4 illustrates this.

Example 4. *Reconsider multiplying $\mathbf{A}_1 \times \mathbf{A}_2 \times \mathbf{A}_3$ in Figure 1. Suppose we estimate the row-wise sparsity of the output matrix of $\mathbf{A}_1 \times \mathbf{A}_2 \times \mathbf{A}_3$ by following the order $(\mathbf{A}_1 \times \mathbf{A}_2) \times \mathbf{A}_3$. Then, we can first estimate $\widehat{\mathbf{r}}(\mathbf{A}_{1,2})$ by* RS-estimator. *Afterwards, we cannot estimate the sparsity of $\mathbf{A}_{1,3} = \mathbf{A}_{1,2} \times \mathbf{A}_3$, because the positions of non-zero elements in $\mathbf{A}_{1,2}$ are unknown. Thus, the estimation cannot be done by following a left-to-right order, as depicted in Figure 4.*

However, by using a right-to-left order $\mathbf{A_1} \times (\mathbf{A_2} \times \mathbf{A_3})$*, we can obtain* $\widehat{\mathbf{r}}(\mathbf{A_{1,3}})$*. First, we estimate* $\widehat{\mathbf{r}}(\mathbf{A_{2,3}})$ *by* `RS-estimator`*. Next, we can estimate* $\widehat{\mathbf{r}}(\mathbf{A_{1,3}})$ *by using matrix* $\mathbf{A_1}$ *and* $\widehat{\mathbf{r}}(\mathbf{A_{2,3}})$*, since the positions of non-zero elements in* $\mathbf{A_1}$ *are known. Thus, the estimation can be done by using a right-to-left order, as depicted in Figure 5.*

Based on the discussions above, we propose an algorithm to estimate the sparsity of all sub-chains generated within the SMCM process, as illustrated in Algorithm 2. Given an input matrix chain $\mathbf{A_1}, \cdots, \mathbf{A_p}$, we initialize a 3D array $\widehat{R}$ for tracking the row-wise sparsity of each sub-chain, and a matrix $\widehat{\mathrm{S}}$ to keep the sparsity of each sub-chain (line 1). Next, we use two nested for-loops to compute the row-wise sparsity of each sub-chain following a reverse order (lines 2-7). Specifically, we set $\widehat{R}[i, i]$ by the exact row-wise sparsity vector $\mathbf{r}(\mathbf{A_i})$ of $\mathbf{A_i}$ (line 3), and store the sparsity of $\mathbf{A_i}$ in $\widehat{\mathrm{S}}[i, i]$ (line 4). Afterwards, in the inner for-loop, we invoke Algorithm 1 to estimate the row-wise sparsity vector $\widehat{\mathbf{r}}(\mathbf{A_{i,j}})$ of the sub-chain $\mathbf{A_{i,j}}$, where the input $\mathbf{r}(\mathbf{A_{i+1,j}})$ is replaced by its estimated vector $\widehat{R}[i+1, j]$ (line 6). The sparsity of $\mathbf{A_{i,j}}$ is also derived by the mean of its estimated row-wise sparsity vector (line 7). Finally, we return the sparsity of all sub-chains (line 8).

Algorithm 2: Sub-chain sparsity estimation

Input: a matrix chain $\mathbf{A_1}, \cdots, \mathbf{A_p}$

Output: a matrix $\widehat{\mathrm{S}}$

1 initialize a 3D array $\widehat{R} \leftarrow \emptyset$, and a $p \times p$ matrix $\widehat{\mathrm{S}} \leftarrow \emptyset$;
2 **for** $i = p$ *to* 1 **do**
3     $\widehat{R}[i, i] \leftarrow \mathbf{r}(\mathbf{A_i})$;
4     $\widehat{\mathrm{S}}[i, i] \leftarrow$ the mean of values in $\widehat{R}[i, i]$;
5   **for** $j = p$ *to* $i + 1$ **do**
6         $\widehat{R}[i, j] \leftarrow$ `RS-estimator`$(\mathbf{A_i}, \widehat{R}[i+1, j])$;
7         $\widehat{\mathrm{S}}[i, j] \leftarrow$ the mean of values in $\widehat{R}[i, j]$;
8 **return** $\widehat{\mathrm{S}}$;

Example 5. *Consider the SMCM of* $\mathbf{A_1} \times \mathbf{A_2} \times \mathbf{A_3}$ *in Figure 1. Following Algorithm 2, when i = 3, we simply initialize* $\widehat{R}[3, 3]$ *as* $\mathbf{r}(\mathbf{A_3})$ *= (0, 1). When i = 2, we first initialize* $\widehat{R}[2, 2]$ *as* $\mathbf{r}(\mathbf{A_2})$ *= (0.5, 0.5), and then estimate* $\widehat{\mathbf{r}}(\mathbf{A_{2,3}})$ *= (1, 1). When i = 1, we first initialize* $\widehat{R}[1, 1]$ *as* $\mathbf{r}(\mathbf{A_1})$ *= (0, 0.5), then estimate* $\widehat{\mathbf{r}}(\mathbf{A_{1,3}})$ *= (0, 1), and finally estimate* $\widehat{\mathbf{r}}(\mathbf{A_{1,2}})$ *= (0, 0.5). Similarly, we can compute the estimated sparsity of the output matrix of each sub-chain by using* $\widehat{R}$*.*

Based on the above discussions, we can derive a good execution order for SMCM using dynamic programming, as outlined in Algorithm 3. The high-level idea is that to derive a good order for multiplying a chain of input matrices, we compute the estimated multiplication cost of all the possible sub-chains, and then parenthesize the sub-chains such that the overall cost is minimized.

Recall that $\widehat{\mathrm{S}}$ is a $p \times p$ matrix, where $\widehat{\mathrm{S}}[i, j]$ denotes the estimated sparsity of the matrix $\mathbf{A_{i,j}}$. We use two auxiliary matrices, $\widehat{\mathbf{C}}$ and $\mathbf{T}$, to facilitate the computation of a good order. Specifically, $\widehat{\mathbf{C}}[i, j]$ stores the minimal cost of the sub-chain matrix multiplication $\mathbf{A_{i,j}} = \mathbf{A_{i,k}} \times \mathbf{A_{k+1,j}}$, and $\mathbf{T}[i, j]$ records the dividing index k . According to Equation (2), the cost $\widehat{cost}(\mathbf{A_{i,k}} \times \mathbf{A_{k+1,j}})$ can be calculated via the sparsity $\widehat{\mathrm{S}}[i, k]$, $\widehat{\mathrm{S}}[k+1, j]$ and $\widehat{\mathrm{S}}[i, j]$ (lines 7). Subsequently, the current cost t_k with dividing index k is computed with the optimal time of corresponding sub-chain matrix multiplication (lines 8). The estimated minimal cost for multiplying each matrix sub-chain is kept

Algorithm 3: SMCM ordering

```
  Input: a matrix chain A_1, ..., A_p and a p × p matrix Ŝ
  Output: an order Ψ to execute A_1 × A_2 × ··· A_p
1 initialize two matrices, Ĉ and T, whose sizes are p × p;
2 for l = 1 to p − 1 do
3     for i = 1 to p − l do
4         j ← i + l;
5         Ĉ[i, j] ← +∞;
6         for k = i to j − 1 do
7             compute cost^(A_{i,k} × A_{k+1,j}) using Ŝ with Eq. (2);
8             t_k ← Ĉ[i, k] + Ĉ[k + 1, j] + cost^(A_{i,k} × A_{k+1,j});
9             if t_k < Ĉ[i, j] then
10                Ĉ[k + 1, j] ← t_k;
11                T[i, j] ← k;
12 compute the order Ψ through T ;
13 return Ψ;
```

in $\widehat{\mathbf{C}}$ while the corresponding dividing index is recorded in **T** (lines 10-11). Finally, the order $\mathbf{\Psi}$ is derived from **T** (line 12).

Time complexity. Algorithm 2 takes $O(\sum_i ((p-i) \cdot nnz(\mathbf{A_i})$ $+n_{i-1}))$ time to compute the sparsity of the output matrix of each sub-chain. Algorithm 3 completes in $O(p^3)$ time, since it uses three nested for-loop in the dynamic programming process.

5.2 Parallel sparse matrix-matrix multiplication

We first discuss the data structure for sparse matrices and then present a parallel algorithm for sparse matrix-matrix multiplication.

• **Data structures for the sparse matrix.** The Compressed Sparse Row (CSR) format is widely used employed in both sequential [26] and parallel [5, 9, 15, 35, 67] sparse matrix multiplication to reduce the space cost. CSR comprises a list *el* storing non-zero elements as (col, val) pairs row by row and an array *row* indicating the index in *el* of the first non-zero element of each row. Figures 6(a) and (b) show a sparse matrix and its CSR representation, respectively.

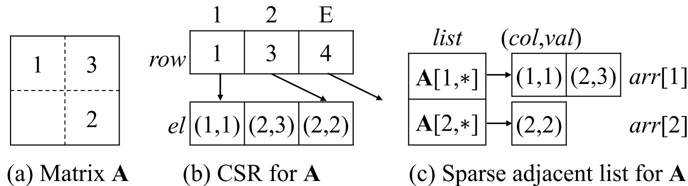


Fig. 6. CSR and sparse adjacent list for a matrix.

While the CSR format saves much space, it is not friendly for parallel multiplication, because during the parallel computation, when a thread is writing the i-th row of the output matrix to memory, it has to wait until all the other threads that are writing rows with indices less than i are

done [26, 40]. This issue is even prominent in the SMCM since it generates many intermediate matrices. To address synchronization issues, we utilize the *sparse adjacent list* [49]. For an $m \times n$ sparse matrix $\mathbf{A}$, the sparse adjacent list, denoted as $list$, consists of m entries. Each $list[i]$ points to an array $arr[i]$ storing non-zero elements in the i-th row of $\mathbf{A}$, where each element is represented as a (col, val) pair. Figure 6(c) depicts the sparse adjacent list for the matrix in Figure 6(a). This efficient representation allows parallel processing on multiple CPU cores, with each thread independently handling a row without synchronization.

• **A parallel algorithm for sparse matrix-matrix multiplication.** Algorithm 4 presents our proposed algorithm, where the input two sparse matrices are represented by sparse adjacent lists. Specifically, to gather all the rows of the output matrix $\mathbf{O} = \mathbf{A} \times \mathbf{B}$, we use three arrays: F, C, and V, each of size l (line 3). Particularly, F tracks whether non-zero columns have been visited or not, where the initial values are *false* denoting not visited, C records the indices of columns with non-zero elements, and V stores the result values of multiplying non-zero elements (line 4). We also use a variable *count* to keep track of the number of non-zero elements (line 5). We then complete the multiplication by two nested for-loops and parallize the outer for-loop using multiple threads (lines 6-13). During the multiplication process, if the j-th column has not been visited, we update $F[j]$ to be $true$, insert j into $C[count]$, increase the value of *count* by one, and compute $\mathbf{A}[i,k] \cdot \mathbf{B}[k,j]$, whose result is kept in $V[j]$ (lines 8-11). Otherwise, we just add $\mathbf{A}[i,k] \cdot \mathbf{B}[k,j]$ to $V[j]$ (lines 12-13).

In the last step within each thread, we aggregate the non-zero elements from C into $\mathbf{O}[i,*]$ (lines 14-15). That is, for each column index j, if the value stored in $V[C[j]]$ is non-zero, we insert a pair consisting of the column index and value into $\mathbf{O}[i,*]$.

Algorithm 4: Parallel sparse matrix-matrix multiplication

```
    Input: two sparse matrices A and B, whose sizes are m × n and n × l respectively
    Output: the output matrix O
 1  initialize matrix O ← ∅;
 2  foreach i = 1 to m in parallel do
 3      initialize three arrays, F, C, and V, whose sizes are l ;
 4      initialize all elements in F to be false;
 5      initialize count ← 1 ;
 6      foreach k ∈ {k|A[i,k] ≠ 0} do
 7          foreach j ∈ {j|B[k,j] ≠ 0} do
 8              if F[j] = false then
 9                  F[j] ← true, C[count] ← j;
10                  count ← count + 1;
11                  V[j] ← A[i,k] · B[k,j];
12              else
13                  V[j] ← V[j] + A[i,k] · B[k,j];
14      for j = 1 to count − 1 do
15          if V[C[j]] ≠ 0 then append (C[j], V[C[j]]) to O[i,*] ;
16  return O;
```

Example 6. *In Figure 7, to compute* $\mathbf{O} = \mathbf{A} \times \mathbf{B}$*, we run Algorithm 4 with two threads. Taking the second thread as an example, we use three arrays* F_2*,* V_2*, and* C_2 *to compute the 2-nd row of* $\mathbf{O}$*. All elements in* F_2 *are initialized to false. First, to compute* $\mathbf{A}[2,1] \cdot \mathbf{B}[1,*] = (0,12)$*, we run the following three steps: (1) update* $F_2[2]$ *to true, (2) append the column index 2 to* C_2*, and (3) set* $V_2[2] = 3 \cdot 4 = 12$*. To calculate* $\mathbf{A}[2,2] \cdot \mathbf{B}[2,*] = (0,5)$*, we just add* $1 \cdot 5 = 5$ *to* $V_2[2]$*, as* $F_2[2] = true$*. Finally, we build a sparse adjacent list* $\mathbf{O}[2,*]$ *with an element* $(2,17)$ *using* C_2 *and* V_2*. The same process is applied to the first thread.*

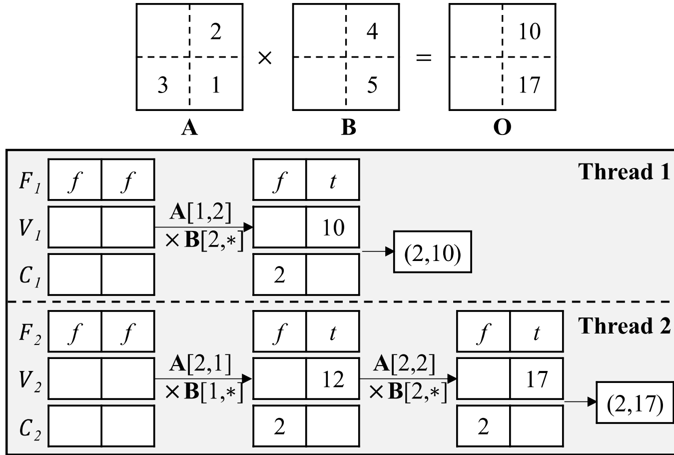


Fig. 7. An example of computing $\mathbf{A} \times \mathbf{B}$ using Algorithm 4.

Time complexity. Algorithm 4 has a time complexity of $O(l + \frac{\delta}{t})$. Here, δ denotes the number of numerical multiplications of $\mathbf{A} \times \mathbf{B}$, and t is the number of threads. Initialization of F, C and V takes $O(l)$ time, while parallel multiplication takes $O(\frac{\delta}{t})$ time.

5.3 The overall SMCM algorithm

To solve the SMCM problem, we follow the SMCM order derived by our ordering algorithm and each time we multiply two sparse matrices. Algorithm 5 presents the overall algorithm RS-estimator based sparse Matrix chain Multiplication (RoseMM). We first estimate the sparsity of sub-chains $\widehat{\mathrm{S}}$ by Algorithm 2 (line 1). Then, we derive a good order using Algorithm 3 (line 2). Finally, by following the order, we apply our parallel sparse matrix-matrix multiplication algorithm to get the final output matrix $\mathbf{O}$ (line 3).

Algorithm 5: `RoseMM`

Input: a matrix chain $\mathbf{A_1}, \cdots, \mathbf{A_p}$
Output: the output matrix $\mathbf{O} = \mathbf{A_{1,p}} = \mathbf{A_1} \times \mathbf{A_2} \times \cdots \times \mathbf{A_p}$

1 compute $\widehat{\mathrm{S}}$ using Algorithm 2;
2 compute the order Ψ of SMCM using Algorithm 3 with $\widehat{\mathrm{S}}$;
3 compute $\mathbf{A_{1,p}}$ in parallel using Algorithm 4 and the order Ψ;
4 **return** $\mathbf{O}$;

Table 5. Datasets used in our experiments.

Dataset	Vertex count	Edge count	Vertex types	Number of SMCMs (3, 4, 5, 6, 7)
FourSquare	43.2K	405.5K	5	(32, 80, 100, 100, 100)
IMDB	471.8K	521.0K	4	(18, 36, 54, 100, 100)
DBLP	490.7K	1.1M	4	(18, 36, 54, 100, 100)
DBpedia	9.0M	31.2M	414	(100, 100, 100, 100, 100)
FreeBase	29.1M	105.6M	984	(100, 100, 100, 100, 100)

Time complexity. By considering the time cost of Algorithms 2, 3, and 4, we can conclude that Algorithm 5 completes in $O\Big(\frac{1}{t}\cdot\Big(\sum_i\left((p-i)\cdot nnz(\mathbf{A_i})+n_{i-1}\right)+\Delta\Big)+p^3+d\Big)$ time, where t is the number of threads used, $d=\max\{n_2,\cdots,n_p\}$, and Δ denotes the total number of numerical multiplication calculations.

6 EXPERIMENTS

We now present the experimental results. We describe the setup in Section 6.1. We assess the accuracy of sparsity estimators in Section 6.2, and evaluate the efficiency of SMCM algorithms in Section 6.3. Section 6.4 discusses the results of some application studies.

6.1 Setup

Datasets. To simulate the SMCM process, we extract various chains of sparse matrices from five real-world heterogeneous information networks (HINs): FourSquare[1], IMDB[2], DBLP[3], DBpedia[4], and FreeBase[5]. Table 5 shows the statistics of each HIN. The choice of using HINs is motivated by the inherent diversity of HINs, encompassing various types of vertices and relationships, which contribute to the richness of matrices and, in turn, enhance the comprehensive evaluation of SMCM algorithm generalization capabilities.

Naturally, the link relationships between two groups of vertices with different types in the HIN form a bipartite graph or a sparse matrix. For example, DBLP includes publication records in computer science areas, and the vertex types are authors (A), papers (P), venues (V), and topics (T). The link relationships between authors and papers form a bipartite graph, which can be represented by an adjacent matrix $\mathbf{M}(AP)$ or its transposed matrix $\mathbf{M}(PA)$. As a result, by considering a chain of such link relationships, which corresponds to a meta-path, we can obtain the semantic relationships between vertices by multiplying a chain of matrices. For instance, the meta-path "$A\rightarrow P\rightarrow T\rightarrow P\rightarrow A$" means two authors having papers sharing the same topics, and we can find all such author pairs by $\mathbf{M}(AP)\times\mathbf{M}(PT)\times\mathbf{M}(TP)\times\mathbf{M}(PA)$, where $\mathbf{M}(PT)$ and $\mathbf{M}(TP)$ are the adjacency matrices between paper and topic vertices.

Sparse matrix chains. To assess the efficiency of SMCM algorithms, we generate five sets of meta-paths, whose lengths are 3, 4, 5, 6, and 7 respectively, and each meta-path corresponds to a matrix chain. Note that if the HIN has over 100 meta-paths with a specific length, we select the 100 meta-paths whose result matrices have the highest sparsity. Table 5 shows the numbers of meta-paths (chains of matrices) in these five sets.

Sparsity estimators. We assess the following estimators:

[1]https://sites.google.com/site/yangdingqi/home/foursquare-dataset
[2]https://www.imdb.com/interfaces/
[3]http://dblp.uni-trier.de/xml/
[4]https://wiki.dbpedia.org/Datasets
[5]http://freebase-easy.cs.uni-freiburg.de/dump/

- MNC [53]: the state-of-the-art matrix sparsity estimator;
- MetaAC [32]: a statistically unbiased sparsity estimator;
- DMap [32]: a statistically unbiased sparsity estimator with block cells;
- LGraph [12]: a graph-based sparsity estimator with r=32 [53];
- RS-estimator: our proposed sparsity estimator.

All estimators utilize the same DP algorithm as depicted in Algorithm 3 and runtime kernels [32].
SMCM algorithms. We compare the following algorithms:

- L2R: it adopts the left-to-right chain order and uses Algorithm 4 for sparse matrix multiplication;
- Naive: it regards all the input matrices as dense matrices, then uses the classic dynamic programming to determine an optimal order, and finishes SMCM using Algorithm 4;
- MNC-SAL [53]: it adopts MNC estimator in chain ordering and uses Algorithm 4 for sparse matrix multiplication;
- DMap-SAL [32]: it employs DMap estimator in chain ordering and uses Algorithm 4 for sparse matrix multiplication;
- LGraph-SAL [12]: it uses LGraph estimator in chain ordering and uses Algorithm 4 for sparse matrix multiplication;
- MetaAC-SAL [32]: it adopts MetaAC estimator in chain ordering and uses Algorithm 4 for sparse matrix multiplication;
- RSE-CSR: it adopts RS-estimator in chain ordering and the state-of-the-art parallel matrix chain multiplication algorithm using CSR format [40];
- RoseMM: Our proposed parallel SMCM algorithm.

We have also tried to use existing sparse matrix-matrix multiplication algorithms, including GPU implementations such as cuSPARSE [45], bhSPARSE [40], and TileSpGEMM [44], in SMCM, but all of them encountered the out-of-memory issues, so we omit their results.

All the algorithms mentioned above are implemented in C++ and compiled with the gcc 9.4.0 compiler using the -O3 optimization level. The experiments are run on a Linux machine running Ubuntu Linux 20.04.5 LTS. This machine is equipped with dual Intel Xeon(R) Gold 6338 2.0GHz processors (64 cores) and 496GB of RAM. The number of threads t varies from 8 to 64, and we set t=64 by default. For the coefficients of the cost model in Equation (2), we set $\alpha = -130$, $\beta = -16$, and $\gamma = 83$ through multi-linear regression with the least squares fitting method [8, 32], based on a random selection of ten thousand matrix products from the five datasets.

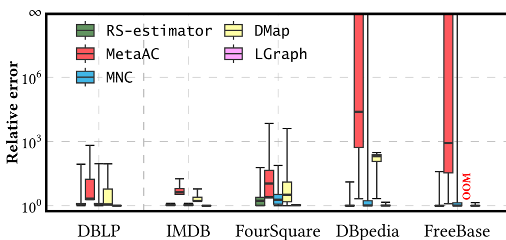


Fig. 8. Relative error of all estimators for sparse matrix-matrix multiplication on all datasets.

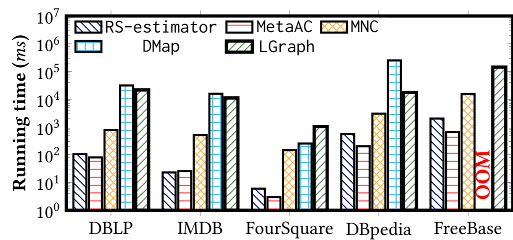


Fig. 9. Runtime of all sparsity estimators for SMCM on all datasets.

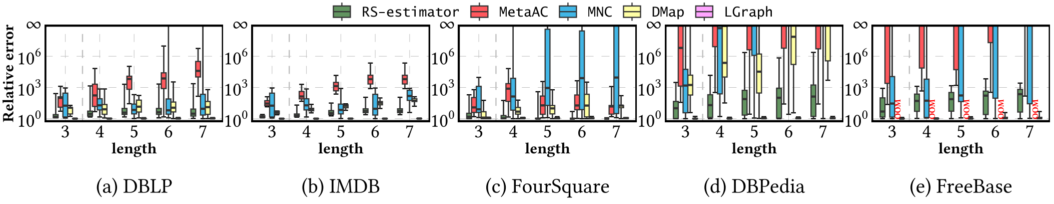


Fig. 10. Relative error of all estimators for SMCM on all datasets.

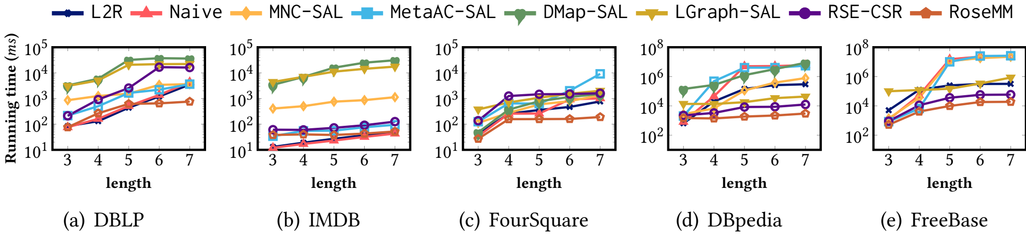


Fig. 11. Effect of the length of matrix chain.

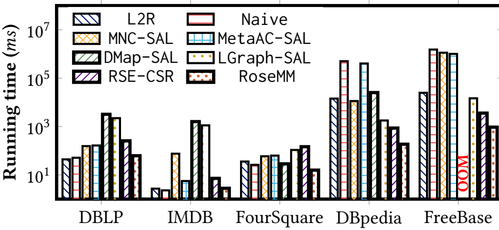


Fig. 12. Efficiency of all SMCM algorithms on all datasets.

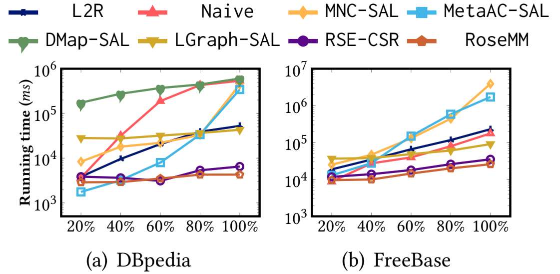


Fig. 13. Scalability test.

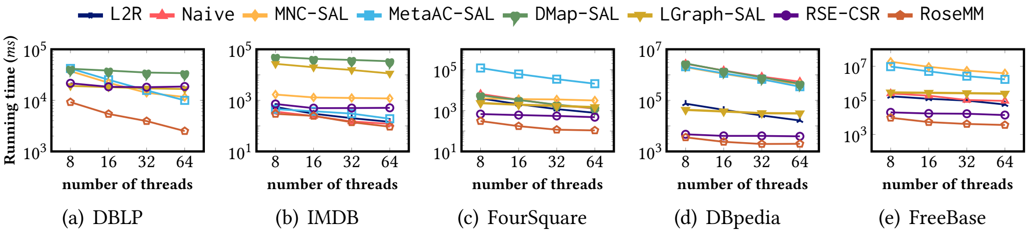
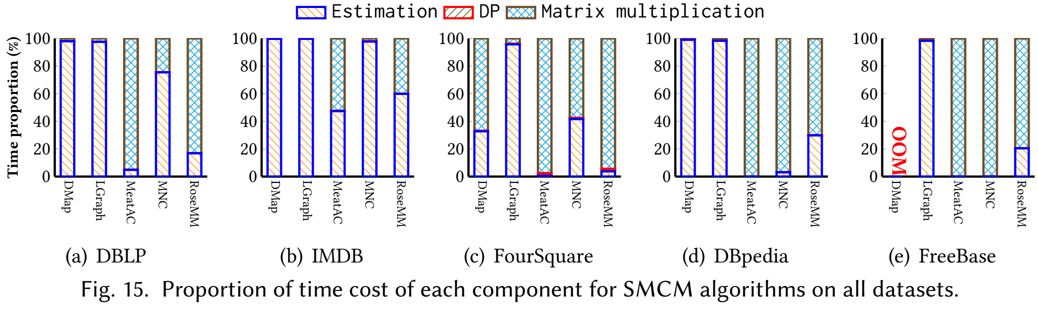


Fig. 14. Effect of the number of threads.

6.2 Efficiency and accuracy evaluation of sparsity estimators

In this section, we evaluate the sparsity estimation accuracy of MetaAC, MNC, DMap, LGraph, and RS-estimator on all datasets. To establish a ground truth for sparsity, we first perform exact matrix chain multiplications for all datasets, allowing us to derive the exact sparsity of the output matrices. Subsequently, we employ the concept of relative error, introduced by Sommer et al. [53], as a benchmark metric for sparsity estimation accuracy evaluation. Assuming that $\mathbf{O}$ is the output



(a) DBLP (b) IMDB (c) FourSquare (d) DBpedia (e) FreeBase

Fig. 15. Proportion of time cost of each component for SMCM algorithms on all datasets.

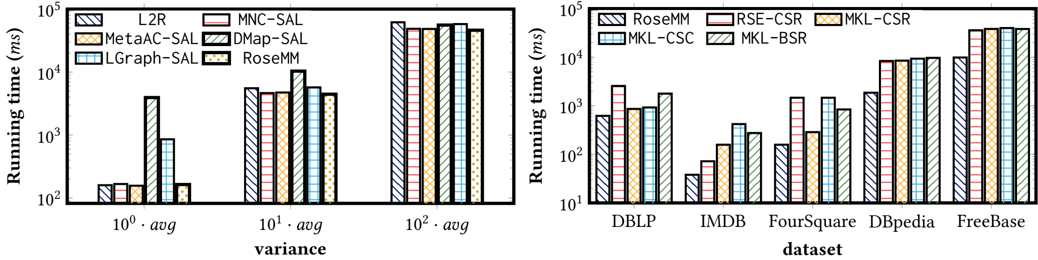
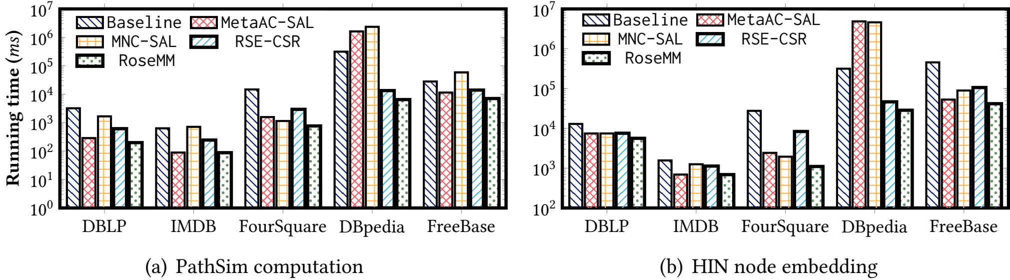


Fig. 16. Runtime of all SMCM algorithms with the mix of dense and sparse operations (length = 5).

Fig. 17. Runtime of SMCM with different sparse matrix multiplication on all datasets (length = 5).



(a) PathSim computation (b) HIN node embedding

Fig. 18. Efficiency in application studies (length = 4).

matrix, the relative error is defined as:

$$\varepsilon(\mathbf{O}, \widehat{\mathbf{O}}) = \frac{\max\left\{\rho(\mathbf{O}), \widehat{\rho}(\mathbf{O})\right\}}{\min\left\{\rho(\mathbf{O}), \widehat{\rho}(\mathbf{O})\right\}}. \tag{19}$$

Here, the relative error value falls within the range of $[1, \infty)$, with values closer to 1 signifying a more precise estimation, indicating proximity to the exact sparsity. For the figures reporting the relative errors below, we use box plots to depict the relative errors of MNC, MetaAC, and RS-estimator, where each box plot displays the corresponding median, lower quartile, upper quartile, minimum, and maximum values.

• **Sparse matrix-matrix multiplication.** In this experiment, we randomly select 100 matrix pairs for each HIN, then measure the relative error of multiplying them, and finally report the relative error of all the estimators in Figure 8. Clearly, RS-estimator achieves the lowest relative error compared to DMap, MetaAC, and MNC. Particularly, it exhibits a relative error close to 1 on

the DBLP and IMDB datasets, indicating that the estimated value closely aligns with the exact value. It's important to note that both MNC and MetaAC can yield zero estimated values, leading to significant relative errors, as observed in datasets like DBpedia and FreeBase. This is because MetaAC's estimated value decreases exponentially as the number of columns in the first matrix increases. For MNC, its estimated value relies on the product of the number of non-zero elements in the i-th column of the first matrix and the number of non-zero elements in the i-th row of the second matrix, so its estimation can reach zero when both input matrices are sufficiently large and sparse. As the dataset size increases, the accuracy of DMap decreases, and it fails to produce results on Freebase due to out-of-memory (OOM) issues. While LGraph achieves superior accuracy among all estimators, it does so at the expense of efficiency, a trade-off to be discussed later.

• **SMCM.** Figure 10 depicts the relative error of all the estimators for SMCM. We also evaluate the runtime of all estimators for SMCM on all datasets, as depicted in Figure 9. Again, RS-estimator achieves the lowest relative error compared to DMap, MetaAC, and MNC. The main reason is that RS-estimator focuses on estimating the row-wise sparsity, which offers a more fine-grained sparsity, enhancing its robustness throughout the SMCM process. On smaller datasets such as DBLP, the accuracy of DMap outperforms MetaAC and MNC, however, the relative error significantly increases as the dataset size increases, reaching its lowest accuracy on DBpedia. Furthermore, DMap exhibits the longest runtime among all estimators, surpassing other methods' runtime by up to four orders of magnitude. For the largest dataset, Freebase, DMap fails to produce results due to OOM. As for LGraph, it achieves the highest estimation accuracy, and its accuracy is less influenced by the length of the SMCM. However, this comes at the expense of a trade-off, as the runtime of our proposed RS-estimator is up to two orders of magnitude faster than LGraph.

As the chain length increases, the accuracy of all methods decreases because SMCM tends to increase the number of non-zero elements. For instance, on the first three datasets, MetaAC achieves initially excellent accuracy with only marginal increases in errors, but it exhibits an upward trend in errors as the chain length grows. On the last two datasets, the performance of MNC, MetaAC and MDap is notably poor. In particular, the relative error of MNC, MetaAC and MDap reaches up to ∞, signifying a failure of the estimator due to most predicted elements being zero.

6.3 Efficiency evaluation of SMCM

• **Overall efficiency results.** We evaluate the runtime of all SMCM algorithms on all HINs, and report the results in Figure 12. Clearly, our RoseMM algorithm achieves the best efficiency on most datasets, especially on DBpedia and FreeBase datasets. It is up to three orders of magnitude faster than the state-of-the-art algorithms MNC-SAL and MetaAC-SAL, and up to two orders of magnitude faster than LGraph-SAL. The runtime of L2R and Naive is comparable to MNC-SAL and MetaAC-SAL due to the subpar accuracy of the two estimators in these datasets. Note that on IMDB, our algorithm is slightly slower than L2R and Naive. This is primarily due to the dataset's relatively low number of non-zero elements, which results in fast matrix multiplication and reduces sensitivity to the execution order. Note that the runtime of SMCM does not purely rely on the sparsity; instead, factors such as matrix size and structure should also be considered. These factors significantly influence overall efficiency. Moreover, both L2R and Naive bypass the need for sparsity estimation, saving extra time. Besides, RoseMM is up to 10× faster than RSE-CSR. The performance difference can be attributed to the improved parallelism offered by the utilization of the sparse adjacency list, which effectively minimizes thread synchronization overhead.

• **Effect of the length of matrix chain.** Figure 11 shows the effect of matrix chain length $l \in \{3, 4, 5, 6, 7\}$ on processing time. It's noteworthy that the runtime of L2R, Naive, DMap-SAL, MNC-SAL, and MetaAC-SAL exhibits a notable increase as l becomes larger. This phenomenon primarily stems from a reduction in estimation precision as l increases. In contrast, our method

shows a relatively modest increase in time consumption. This is because RS-estimator can achieve higher accuracy and reduce calculation time, particularly for lengthy matrix chain multiplications. While LGraph achieves high accuracy, its inefficient estimation significantly prolongs the entire SMCM process.

• **Effect of the number of threads.** Figure 14 presents the efficiency by varying the number of threads t from 8 to 64 across all datasets. As t increases, the runtime of L2R, Naive, and RoseMM shows linear reduction, indicating strong parallel scalability. Besides, RoseMM outperforms RSE-CSR by up to 20×. This performance discrepancy is primarily attributed to the additional synchronization overhead introduced by CSR. In addition, DMap-SAL, MetaAC-SAL, and MNC-SAL incur significant time costs, mainly attributed to the substantial overhead resulting from the order obtained by their estimators. Although LGraph-SAL demonstrates high accuracy, the estimator's time consumption renders it inefficient.

• **Scalability test.** To test the scalability, we randomly select 20%, 40%, 60%, 80%, and 100% of edges from each HIN, and then obtain five sub-HINs induced by these edges respectively. For lack of space, we only show the results on DBpedia and FreeBase in Figure 13 since the trends are similar on other datasets. It's evident that as the dataset size grows, the time cost of all algorithms increases, but on all datasets, the curves of our algorithm have lower slopes, so it achieves better scalability than L2R, Naive, LGraph-SAL, DMap-SAL, MetaAC-SAL, and MNC-SAL.

• **Proportion of time cost of each component for SMCM algorithms.** In this experiment, we evaluate the proportion of time cost for each component of the SMCM algorithms on all datasets, including the sparsity estimation phase, dynamic programming phase, and execution of matrix multiplication phase. Note that the runtime of the sparsity estimation phase consists of the time cost of sketch construction and sparsity estimation. The results are depicted in Figure 15. It's essential to highlight that, in our RoseMM approach, sparsity estimation contributes minimally to the overall time cost, especially in datasets like FourSquare, where matrix multiplication dominates the execution time. The dynamic programming phase proves highly efficient across all datasets, benefiting from obtaining non-zero elements during the estimation phase, resulting in a mere few milliseconds of processing time. Conversely, DMap and LGraph face challenges due to the time cost of estimation exceeding that of matrix multiplication, making them less suitable for large-scale datasets. DMap, in particular, encounters out-of-memory issues. While MetaAC and MNC demonstrate high efficiency, their effectiveness is compromised by lower estimation accuracy, notably affecting the efficiency of multiplication execution. This is particularly evident in datasets such as DBpedia and Freebase, where matrix multiplication time prevails.

• **Efficiency on the mix of dense and sparse operations.** In this experiment, we evaluate the mix of dense and sparse operations following the experiment setting of MetaAC [32]. Specifically, we randomly generate five matrix chains, each consisting of five $100k \times 100k$ input matrices, to perform mixed operations. The sparsity of the matrices follows a normal distribution. The bounding $<min, max, avg>$ distributions for data skew are $<10^{-10}, 10^{0}, 10^{-5}>$ with variances $\{avg, 10 \cdot avg, 100 \cdot avg\}$. The results are presented in Figure 16, at a variance of $10^{0} \cdot avg$, our method demonstrates optimal efficiency, surpassing DMap by up to 24×. As the variance increases, the execution times for each method become comparable. This is attributed to the diminishing impact of the order on the overall runtime of SMCM with the growing sparsity.

• **Effect of different matrix multiplication algorithms.** In this experiment, we evaluate a variety of algorithms with different storage formats for SMCM. Specifically, we have compared the sparse matrix multiplication algorithms within the Intel oneAPI Math Kernel Library (MKL) [59], and the algorithms employing various matrix storage formats, including CSR (MKL-CSR), CSC (MKL-CSC), and BSR (MKL-BSR). All methods are implemented with our proposed RS-estimator for

SMCM. The experimental results are presented in Figure 17. RoseMM outperforms RSE-CSR by up to 2.24× and is 4.6×, 9.27×, and 7.23× faster than MKL-CSR, MKL-CSC, and MKL-BSR, respectively.

6.4 Application studies

In this section, we apply our proposed SMCM algorithm to two real applications and analyze its empirical results.

• **PathSim [55].** Given an HIN, PathSim measures the similarity between two vertices of the same type using a symmetric meta-path $\mathcal{P}$, which consists of a sequence of vertex types and edge types. The PathSim formula employing SMCM is expounded in Section 1. To solve the SMCM, for each HIN we first randomly select 10 meta-paths whose lengths are four, which is a widely adopted parameter in PathSim evaluation [55]. Then, for each meta-path, we compute the PathSim values between all the vertex pairs whose PathSim values are larger than zero. We report the average runtime of all SMCM algorithms in Figure 18(a), where Baseline is the PathSim computation algorithm proposed in [55]. Comparatively, RoseMM exhibits remarkable performance advantages: it is up to 48× faster than the baseline, up to 5.6× faster than RSE-CSR, up to 249× faster than MetaAC-SAL, and up to 361× faster than MNC-SAL.

• **Node embedding [27, 34].** HIN node embedding algorithms [17, 19, 48, 52, 66] often leverage meta-path-guided random walks to capture semantically meaningful information from the various entity types and their relationships. The formulation of meta-path-guided random walks is detailed in Section 1. To evaluate the efficiency of SMCM algorithms for node embedding, we first randomly select 20 meta-paths, each with a length of 4, and then run all the SMCM algorithms. We display their average runtime in Figure 18(b), where Baseline denotes the state-of-the-art meta-path guided random walk approach in [52, 65]. The results illustrate that our RoseMM algorithm achieves up to 11× faster performance than the baseline, and is as much as 7.6× faster than RSE-CSR. Besides, RoseMM outperforms MetaAC-SAL and MNC-SAL by a factor of up to 170× and 161× respectively.

7 CONCLUSIONS

In this paper, we study the problem of sparse matrix chain multiplication (SMCM) and develop an efficient algorithm. We first introduce a simple yet effective estimator, called row-wise sparsity estimator (RS-estimator), which exploits the structural properties of matrices for better estimation, and then we provide an extensive theoretical analysis of its accuracy. Based on RS-estimator, we further develop an algorithm to determine a good order of SMCM and propose an efficient parallel SMCM algorithm, called RoseMM, by employing a sparse adjacent list data structure to reduce the synchronization costs with multiple CPU threads. Extensive experiments with large sparse matrices extracted from five real-world graphs show that RS-estimator achieves higher accuracy than state-of-the-art sparsity estimators, and RoseMM is up to three orders of magnitude faster than state-of-the-art SMCM algorithms. In the future, we will implement our algorithm on a distributed platform and test its performance.

Acknowledgements. This work was supported in part by NSFC under Grants 62202412, 62102341, and 62302421, Guangdong Talent Program under Grant 2021QN02X826, and Basic and Applied Basic Research Fund in Guangdong Province under Grant 2023A1515011280. This paper was also supported by Shenzhen Stability Science Program and Guangdong Key Lab of Mathematical Foundations for Artificial Intelligence.

REFERENCES

[1] Grey Ballard, Aydin Buluc, James Demmel, Laura Grigori, Benjamin Lipshitz, Oded Schwartz, and Sivan Toledo. 2013. Communication optimal parallel multiplication of sparse random matrices. In *Proceedings of the twenty-fifth annual ACM symposium on Parallelism in algorithms and architectures*. 222–231.

[2] Henrik Barthels, Marcin Copik, and Paolo Bientinesi. 2018. The generalized matrix chain algorithm. In *Proceedings of the 2018 International Symposium on Code Generation and Optimization*. 138–148.

[3] Girish Biswas and Nandini Mukherjee. 2021. Memory Optimized Dynamic Matrix Chain Multiplication Using Shared Memory in GPU. In *International Conference on Distributed Computing and Internet Technology*. 160–172.

[4] Matthias Boehm, Douglas R Burdick, Alexandre V Evfimievski, Berthold Reinwald, Frederick R Reiss, Prithviraj Sen, Shirish Tatikonda, and Yuanyuan Tian. 2014. SystemML's Optimizer: Plan Generation for Large-Scale Machine Learning Programs. *IEEE Data Eng. Bull.* 37, 3 (2014), 52–62.

[5] Aydin Buluc and John R Gilbert. 2008. Challenges and advances in parallel sparse matrix-matrix multiplication. In *2008 37th International Conference on Parallel Processing*. IEEE, 503–510.

[6] Aydin Buluc and John R Gilbert. 2008. On the representation and multiplication of hypersparse matrices. In *2008 IEEE International Symposium on Parallel and Distributed Processing*. IEEE, 1–11.

[7] Timothy M Chan. 2007. More algorithms for all-pairs shortest paths in weighted graphs. In *Proceedings of the thirty-ninth annual ACM symposium on Theory of computing*. 590–598.

[8] Serafeim Chatzopoulos, Thanasis Vergoulis, Dimitrios Skoutas, Theodore Dalamagas, Christos Tryfonopoulos, and Panagiotis Karras. 2022. *arXiv preprint arXiv:2201.04058* (2022).

[9] Yuedan Chen, Kenli Li, Wangdong Yang, Guoqing Xiao, Xianghui Xie, and Tao Li. 2018. Performance-aware model for sparse matrix-matrix multiplication on the sunway taihulight supercomputer. *IEEE transactions on parallel and distributed systems* 30, 4 (2018), 923–938.

[10] Igor Chikalov, Shahid Hussain, and Mikhail Moshkov. 2011. Sequential optimization of matrix chain multiplication relative to different cost functions. In *SOFSEM 2011: Theory and Practice of Computer Science: 37th Conference on Current Trends in Theory and Practice of Computer Science, Novỳ Smokovec, Slovakia, January 22-28, 2011. Proceedings 37*. Springer, 157–165.

[11] Edith Cohen. 1994. Estimating the size of the transitive closure in linear time. In *Proceedings 35th Annual Symposium on Foundations of Computer Science*. IEEE, 190–200.

[12] Edith Cohen. 1998. Structure prediction and computation of sparse matrix products. *Journal of Combinatorial Optimization* 2 (1998), 307–332.

[13] Jeffrey Cohen, Brian Dolan, Mark Dunlap, Joseph M Hellerstein, and Caleb Welton. 2009. MAD skills: new analysis practices for big data. *Proceedings of the VLDB Endowment* 2, 2 (2009), 1481–1492.

[14] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. 2009. *Introduction to Algorithms*. MIT Press.

[15] Steven Dalton, Luke Olson, and Nathan Bell. 2015. Optimizing sparse matrix—matrix multiplication for the gpu. *ACM Transactions on Mathematical Software (TOMS)* 41, 4 (2015), 1–20.

[16] Gunduz Vehbi Demirci and Cevdet Aykanat. 2020. Scaling sparse matrix-matrix multiplication in the accumulo database. *Distributed and Parallel Databases* 38 (2020), 31–62.

[17] Yuxiao Dong, Nitesh V Chawla, and Ananthram Swami. 2017. metapath2vec: Scalable representation learning for heterogeneous networks. In *Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining*. 135–144.

[18] Yixiang Fang, Yixing Yang, Wenjie Zhang, Xuemin Lin, and Xin Cao. 2020. Effective and efficient community search over large heterogeneous information networks. *Proceedings of the VLDB Endowment* 13, 6 (2020), 854–867.

[19] Tao-yang Fu, Wang-Chien Lee, and Zhen Lei. 2017. Hin2vec: Explore meta-paths in heterogeneous information networks for representation learning. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*. 1797–1806.

[20] Xinyu Fu, Jiani Zhang, Ziqiao Meng, and Irwin King. 2020. Magnn: Metapath aggregated graph neural network for heterogeneous graph embedding. In *Proceedings of The Web Conference 2020*. 2331–2341.

[21] Vijay Gadepally, Jake Bolewski, Dan Hook, Dylan Hutchison, Ben Miller, and Jeremy Kepner. 2015. Graphulo: Linear algebra graph kernels for nosql databases. In *2015 IEEE International Parallel and Distributed Processing Symposium Workshop*. IEEE, 822–830.

[22] Jianhua Gao, Weixing Ji, Fangli Chang, Shiyu Han, Bingxin Wei, Zeming Liu, and Yizhuo Wang. 2023. A systematic survey of general sparse matrix-matrix multiplication. *Comput. Surveys* 55, 12 (2023), 1–36.

[23] John R Gilbert, Cleve Moler, and Robert Schreiber. 1992. Sparse matrices in MATLAB: Design and implementation. *SIAM journal on matrix analysis and applications* 13, 1 (1992), 333–356.

[24] Sadashiva S Godbole. 1973. On efficient computation of matrix chain products. *IEEE Trans. Comput.* 100, 9 (1973), 864–866.

[25] Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*. 855–864.

[26] Fred G Gustavson. 1978. Two fast algorithms for sparse matrices: Multiplication and permuted transposition. *ACM Transactions on Mathematical Software (TOMS)* 4, 3 (1978), 250–269.

[27] Yu He, Yangqiu Song, Jianxin Li, Cheng Ji, Jian Peng, and Hao Peng. 2019. Hetespaceywalk: A heterogeneous spacey random walk for heterogeneous information network embedding. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*. 639–648.

[28] Kartik Hegde, Hadi Asghari-Moghaddam, Michael Pellauer, Neal Crago, Aamer Jaleel, Edgar Solomonik, Joel Emer, and Christopher W Fletcher. 2019. Extensor: An accelerator for sparse tensor algebra. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*. 319–333.

[29] Guyue Huang, Guohao Dai, Yu Wang, and Huazhong Yang. 2020. Ge-spmm: General-purpose sparse matrix-matrix multiplication on gpus for graph neural networks. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 1–12.

[30] Dylan Hutchison, Bill Howe, and Dan Suciu. 2017. LaraDB: A minimalist kernel for linear and relational algebra computation. In *Proceedings of the 4th ACM SIGMOD Workshop on Algorithms and Systems for MapReduce and Beyond*. 1–10.

[31] Moritz Kaufmann, Manuel Then, Alfons Kemper, and Thomas Neumann. 2017. Parallel Array-Based Single-and Multi-Source Breadth First Searches on Large Dense Graphs.. In *EDBT*. 1–12.

[32] David Kernert, Frank Köhler, and Wolfgang Lehner. 2015. SpMacho-Optimizing Sparse Linear Algebra Expressions with Probabilistic Density Estimation. In *EDBT*. 289–300.

[33] Bogyeong Kim, Kyoseung Koo, Undraa Enkhbat, Sohyun Kim, Juhun Kim, and Bongki Moon. 2022. M2Bench: A Database Benchmark for Multi-Model Analytic Workloads. *Proceedings of the VLDB Endowment* 16, 4 (2022), 747–759.

[34] Ni Lao and William W Cohen. 2010. Relational retrieval using a combination of path-constrained random walks. *Machine learning* 81 (2010), 53–67.

[35] Jeongmyung Lee, Seokwon Kang, Yongseung Yu, Yong-Yeon Jo, Sang-Wook Kim, and Yongjun Park. 2020. Optimization of GPU-based sparse matrix multiplication for large sparse networks. In *2020 IEEE 36th International Conference on Data Engineering (ICDE)*. IEEE, 925–936.

[36] Charles Eric Leiserson, Ronald L Rivest, Thomas H Cormen, and Clifford Stein. 1994. *Introduction to algorithms*. Vol. 3. MIT press Cambridge, MA, USA.

[37] Jiajun Li, Ahmed Louri, Avinash Karanth, and Razvan Bunescu. 2021. GCNAX: A flexible and energy-efficient accelerator for graph convolutional neural networks. In *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 775–788.

[38] Keqin Li. 2007. Analysis of parallel algorithms for matrix chain product and matrix powers on distributed memory systems. *IEEE Transactions on Parallel and Distributed Systems* 18, 7 (2007), 865–878.

[39] Colin Yu Lin, Ngai Wong, and Hayden Kwok-Hay So. 2013. Design space exploration for sparse matrix-matrix multiplication on FPGAs. *International Journal of Circuit Theory and Applications* 41, 2 (2013), 205–219.

[40] Weifeng Liu and Brian Vinter. 2014. An efficient GPU general sparse matrix-matrix multiplication for irregular data. In *2014 IEEE 28th International Parallel and Distributed Processing Symposium*. IEEE, 370–381.

[41] Weifeng Liu and Brian Vinter. 2015. A framework for general sparse matrix–matrix multiplication on GPUs and heterogeneous processors. *J. Parallel and Distrib. Comput.* 85 (2015), 47–61.

[42] Jaeseok Myung and Sang-goo Lee. 2012. Matrix chain multiplication via multi-way join algorithms in MapReduce. In *Proceedings of the 6th International Conference on Ubiquitous Information Management and Communication*. 1–5.

[43] Kazufumi Nishida, Yasuaki Ito, and Koji Nakano. 2011. Accelerating the dynamic programming for the matrix chain product on the GPU. In *2011 Second International Conference on Networking and Computing*. IEEE, 320–326.

[44] Yuyao Niu, Zhengyang Lu, Haonan Ji, Shuhui Song, Zhou Jin, and Weifeng Liu. 2022. TileSpGEMM: A tiled algorithm for parallel sparse general matrix-matrix multiplication on GPUs. In *Proceedings of the 27th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*. 90–106.

[45] NVIDIA. 2020. Nvidia cuSPARSE library. Retrieved from https://developer.nvidia.com/cusparse.

[46] Subhankar Pal, Jonathan Beaumont, Dong-Hyeon Park, Aporva Amarnath, Siying Feng, Chaitali Chakrabarti, Hun-Seok Kim, David Blaauw, Trevor Mudge, and Ronald Dreslinski. 2018. Outerspace: An outer product based sparse matrix multiplication accelerator. In *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 724–736.

[47] Md Mostofa Ali Patwary, Nadathur Rajagopalan Satish, Narayanan Sundaram, Jongsoo Park, Michael J Anderson, Satya Gautam Vadlamudi, Dipankar Das, Sergey G Pudov, Vadim O Pirogov, and Pradeep Dubey. 2015. Parallel efficient sparse matrix-matrix multiplication on multicore platforms. In *International Conference on High Performance Computing*. Springer, 48–57.

[48] Hao Peng, Ruitong Zhang, Yingtong Dou, Renyu Yang, Jingyi Zhang, and Philip S Yu. 2021. Reinforced neighborhood selection guided multi-relational graph neural networks. *ACM Transactions on Information Systems (TOIS)* 40, 4 (2021), 1–46.

[49] Berthold Reinwald, Shirish Tatikonda, and Yuanyuan Tian. 2016. Sparsity-driven matrix representation to optimize operational and storage efficiency. US Patent 9,396,164.

[50] Hongbo Rong, Jongsoo Park, Lingxiang Xiang, Todd A Anderson, and Mikhail Smelyanskiy. 2016. Sparso: Context-driven optimizations of sparse linear algebra. In *Proceedings of the 2016 International Conference on Parallel Architectures and Compilation*. 247–259.

[51] Oguz Selvitopi, Md Taufique Hussain, Ariful Azad, and Aydın Buluç. 2020. Optimizing high performance Markov clustering for pre-exascale architectures. In *2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 116–126.

[52] Chuan Shi, Binbin Hu, Wayne Xin Zhao, and S Yu Philip. 2018. Heterogeneous information network embedding for recommendation. *IEEE Transactions on Knowledge and Data Engineering* 31, 2 (2018), 357–370.

[53] Johanna Sommer, Matthias Boehm, Alexandre V Evfimievski, Berthold Reinwald, and Peter J Haas. 2019. Mnc: Structure-exploiting sparsity estimation for matrix expressions. In *Proceedings of the 2019 International Conference on Management of Data*. 1607–1623.

[54] Nitish Srivastava, Hanchen Jin, Jie Liu, David Albonesi, and Zhiru Zhang. 2020. Matraptor: A sparse-sparse matrix multiplication accelerator based on row-wise product. In *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 766–780.

[55] Yizhou Sun, Jiawei Han, Xifeng Yan, Philip S Yu, and Tianyi Wu. 2011. Pathsim: Meta path-based top-k similarity search in heterogeneous information networks. *Proceedings of the VLDB Endowment* 4, 11 (2011), 992–1003.

[56] Manuel Then, Moritz Kaufmann, Fernando Chirigati, Tuan-Anh Hoang-Vu, Kien Pham, Alfons Kemper, Thomas Neumann, and Huy T Vo. 2014. The more the merrier: Efficient multi-source graph traversal. *Proceedings of the VLDB Endowment* 8, 4 (2014), 449–460.

[57] Stijn Marinus Van Dongen. 2000. *Graph clustering by flow simulation.* Ph.D. Dissertation.

[58] Virginia Vassilevska, Ryan Williams, and Raphael Yuster. 2010. Finding heaviest H-subgraphs in real weighted graphs, with applications. *ACM Transactions on Algorithms (TALG)* 6, 3 (2010), 1–23.

[59] Endong Wang, Qing Zhang, Bo Shen, Guangyong Zhang, Xiaowei Lu, Qing Wu, and Yajuan Wang. 2014. *Intel Math Kernel Library.* Springer International Publishing, 167–188.

[60] Xiao Wang, Houye Ji, Chuan Shi, Bai Wang, Yanfang Ye, Peng Cui, and Philip S Yu. 2019. Heterogeneous graph attention network. In *The world wide web conference*. 2022–2032.

[61] Martin Winter, Daniel Mlakar, Rhaleb Zayer, Hans-Peter Seidel, and Markus Steinberger. 2019. Adaptive sparse matrix-matrix multiplication on the GPU. In *Proceedings of the 24th symposium on principles and practice of parallel programming*. 68–81.

[62] Michael M Wolf, Mehmet Deveci, Jonathan W Berry, Simon D Hammond, and Sivasankaran Rajamanickam. 2017. Fast linear algebra-based triangle counting with kokkoskernels. In *2017 IEEE High Performance Extreme Computing Conference (HPEC)*. IEEE, 1–7.

[63] Yixing Yang, Yixiang Fang, Xuemin Lin, and Wenjie Zhang. 2020. Effective and efficient truss computation over large heterogeneous information networks. In *2020 IEEE 36th international conference on data engineering (ICDE)*. IEEE, 901–912.

[64] Yongyang Yu, Mingjie Tang, Walid G Aref, Qutaibah M Malluhi, Mostafa M Abbas, and Mourad Ouzzani. 2017. In-memory distributed matrix computation processing and optimization. In *2017 IEEE 33rd International conference on data engineering (ICDE)*. IEEE, 1047–1058.

[65] Seongjun Yun, Minbyul Jeong, Raehyun Kim, Jaewoo Kang, and Hyunwoo J Kim. 2019. Graph transformer networks. *Advances in neural information processing systems* 32 (2019).

[66] Daokun Zhang, Jie Yin, Xingquan Zhu, and Chengqi Zhang. 2018. Metagraph2vec: Complex semantic path augmented heterogeneous network embedding. In *Advances in Knowledge Discovery and Data Mining: 22nd Pacific-Asia Conference, PAKDD 2018, Melbourne, VIC, Australia, June 3-6, 2018, Proceedings, Part II 22*. Springer, 196–208.

[67] Zhekai Zhang, Hanrui Wang, Song Han, and William J Dally. 2020. Sparch: Efficient architecture for sparse matrix multiplication. In *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 261–274.

[68] Yingli Zhou, Yixiang Fang, Wensheng Luo, and Yunming Ye. 2023. Influential community search over large heterogeneous information networks. *Proceedings of the VLDB Endowment* 16, 8 (2023), 2047–2060.

Received October 2023; revised January 2024; accepted March 2024