International Journal of Engineering Trends and Technology
ISSN: 2231–5381 / https://doi.org/10.14445/22315381/IJETT-V73I8P113

Volume 73 Issue 8, 147-169, August 2025
© 2025 Seventh Sense Research Group®

Original Article

FSM-MR++ Enhanced MapReduce-Based Framework for Scalable Frequent Subgraph Mining in Large-Scale Graph Datasets

Naga Mallik Atcha[1*], Jagannadha Rao D B[2], Vijayakumar Polepally[3]

[1]*Department of CSE, Malla Reddy University, Hyderabad, Telangana, India.*
[2]*Malla Reddy University, Hyderabad, Telangana, India.*
[3]*Kakatiya Institute of Technology & Science, Warangal, Telangana, India.*

[1]*Corresponding Author : mallik.atcha@gmail.com*

Received: 21 June 2025 Revised: 25 July 2025 Accepted: 31 July 2025 Published: 30 August 2025

Abstract - *Frequent Subgraph Mining (FSM) is a key technique for identifying recurring structural patterns in large graph datasets. It has a broad spectrum of applications ranging from bioinformatics and cheminformatics to social network analysis. Nonetheless, state-of-the-art FSM algorithms, such as gSpan, still suffer from a scalability problem stemming from the exponential candidate generation and the in-memory constraint. Although existing distributed methods (such as FSM-MR) are MapReduce-friendly, they suffer from static reducer assignment, redundant subgraph recomputation, and inefficiency in pruning. Such constraints significantly influence the run-time efficiency, quality of patterns, and scalability on large-scale real-world big data. To address these challenges, an enhanced MapReduce-based pattern searching framework, FSM-MR++, is proposed for efficient and adaptive frequent subgraph discovery. The framework incorporates three innovations: a hybrid caching technique to prune repetitive subgraph generation, a dynamic reducer configuration scheme to prevent skewed task distribution, and a density-aware pruning strategy to abandon unpromising candidates early during mining. These primitives are combined with canonical Labeling and cost-aware partitioning to optimize parallelism and convergence. A quantitative evaluation is conducted to assess its performance on both synthetic and real datasets. It was observed that FSM-MR++ runs up to 30% faster than FSM-MR and is up to half the run-time of centralized gSpan, while still enabling high-quality, interpretable patterns. We conduct a series of ablation studies to validate the improvements introduced by each proposed enhancement, and scalability tests are used to evaluate the framework against growing data and cluster sizes. The framework's efficacy in discovering target-specific substructures in a domain and in an interpretable manner is demonstrated using real-world, PubChemBioAssay, and DBLP datasets. In general, FSM-MR++ fills the gaps of current FSM methods, providing a scalable, efficient, and flexible solution for frequent subgraph mining in a distributed big data setting.*

Keywords - *Frequent Subgraph Mining, MapReduce framework, Distributed graph Mining, hybrid caching, Dynamic reducer scaling.*

1. Introduction

As a subtask of data mining and knowledge discovery, Frequent Subgraph Mining (FSM) is now widely recognized for analyzing large amounts of graph data from various application domains, including bioinformatics, cheminformatics, social networks, and cybersecurity. Discovering frequent subgraphs from graph databases for molecular activity pattern analysis, collaboration network analysis, and inter- and multi-relational network discovery is a challenging and active area of research. Nevertheless, traditional FSM methods, including gSpan and FSG, suffer from expensive computing and insufficient scalability on large-scale graph data. Distributed FSM algorithms, such as FSM-MR, have attempted to address these challenges through MapReduce. Still, they suffer from static reducer allocation problems, high redundancy among candidates, and a lack of adaptive pruning. Recent research studies have drawn attention to the necessity of scalable, adaptive, and workload-balanced FSM techniques that can handle millions of graphs in a distributed computing environment [1-3]. Existing methods predominantly rely on memory-based calculations or overlook task balance and convergence optimization, which is unsuitable for real-world big data applications.

Accordingly, a framework capable of reconciling MapReduce's robustness while incorporating algorithmic optimizations for efficiency and redundancy reduction, and preserving the significance of pattern-derived information is needed. Although many distributed FSM methods are proposed, including FSM-MR, G-thinker, and PEREGRINE, they still suffer from various persistent problems: (i) Static task allocation suffers from load balance, (ii) Isomorphic subgraphs can be generated and recomputed redundantly, and (iii) Memory and reducer scalability are not efficient for large, dense, or irregular graph structures. Such restrictions slow down convergence speed, increase the overhead costs in measurable resources, and decrease the relevance of the learned patterns, particularly in real-world settings, where low-scale and high-parallel interpretability is needed. Additionally, adaptive scheduling and memory-aware optimization techniques are not well-coupled in studies based on the MapReduce paradigm; thus, a unified and scalable framework for FSM is still lacking.

To address these challenges, we propose FSM-MR++, a novel and more efficient framework for frequent subgraph mining based on MapReduce. We develop our proposed solution based on five innovations: (1) graph partitioning strategy to alleviate load skews, (2) cost-aware mapper scheduler to achieve balanced memory and computation, (3) hybrid memory-caching mechanisms to reduce disk I/Os, (4) density-aware candidate pruning to prune low utility patterns early, and (5) dynamic reducer scaling to dynamically and optimally spread the iterative tasks. The innovations in FSM-MR++ are: (i) adaptive parallel mining by dynamically allowing or disallowing map tasks to avoid redundant computation, and (ii) increased convergence time (until satisfying PoE) without losing pattern quality, as compared to state-of-the-art methods. Our solution retains interpretability and domain relevance, outperforming traditional models (gSpan) and more recent frameworks (FSM-MR) in terms of running time, achieving up to 30% faster execution time and up to 50% run-time reduction on various real-world datasets.

The rest of the paper is structured as follows: Section 2 reviews the literature on related work and advancements in FSM. Section 3 presents the proposed FSM-MR++ method, consisting of an architecture and two algorithmic components. Section 4 describes the experimental results, ablation studies, and scalability tests. Section 5 concludes and discusses related work and limitations. The remainder of this article is organized as follows: Section 6 summarizes and concludes the article, discussing future work.

2. Related Work

This literature review explores recent advancements, limitations, and future directions in scalable subgraph mining and graph-based data analytics. Yan et al. [1] presented PrefixFPM, an adaptable framework for mining common patterns that addresses shortcomings in current methods and makes recommendations for future improvements. Yan et al. [2] examined the shortcomings of current models, evaluated developments in graph-parallel systems, and looked at potential directions for graph analytics. Besta and Hoefler [3] discovered efficiency issues, evaluated graph neural network parallelism, and made recommendations for future optimization research paths. Megherbi et al. [4] developed a deep learning technique for dense subgraph mining, highlighting its advantages while acknowledging its drawbacks and recommending potential improvements. Yang et al. [5] examined and categorized Personalized PageRank methods, highlighting issues with efficiency and suggesting further research on dynamic graph applications and systematic comparisons.

Vandierendonck [6] presented novel set intersection methods for maximal clique enumeration, highlighting the difficulties associated with higher graph sizes while achieving notable speedups. Lu et al. [7] developed Xorbits, a scalable data science platform that addresses out-of-memory problems; further development is needed to make it compatible with a broader range of applications. Gao et al. [8] presented CSM-TopK, a method for identifying high-density matches in dynamic weighted graphs, highlighting its NP-hardness and offering enhancements for broader application. Ponnusamy and Gupta [9] investigated cloud-based, scalable data-partitioning strategies, highlighting inefficiencies and recommending future improvements for data-intensive applications. Yao et al. [10] developed the MSBE algorithm, which addresses shortcomings in current models and proposes improvements for the future, to locate similar bicliques in bipartite networks efficiently.

Yu et al. [11] presented KSP-DG, a distributed technique that addresses scalability challenges and proposes efficient indexing for k-shortest paths in dynamic road networks. Wei et al. [12] created the geospatial knowledge graph FineGeoKG, which effectively captures strong geographic linkages and improves query speed while recommending improvements. Zhang [13] examined the use of extensive health data analytics to enhance the performance of senior employees, highlighting the challenges and upcoming tasks in integrating complex medical systems. Yuan et al. [14] developed a batch processing approach to enhance the performance of multiple searches in large networks for hop-constrained s-t path enumeration. Das et al. [15] examined stock prediction using GNNs and sentiment analysis, pointing out its drawbacks and recommending areas for further study.

Liu and Seshadhri [16] proposed a novel triangle counting approach for constrained arboricity graphs that balances space constraints with efficiency. Erbel and Grabowski [17] developed a dynamic run-time architecture for scientific processes that enables the integration of bespoke apps and real-time resource management; further improvements are recommended. Mahnoor et al. [18] examined quick clustering techniques, identified issues, and suggested avenues for further study to enhance the effectiveness and relevance of these techniques across various domains. Dahiphale et al. [19] developed BiECCA, a distributed algorithm that addresses the limitations of single-node methods for identifying 2-edge-linked components in large networks. Chaturvedi et al. [20] utilized FP-Growth and PFP-Growth to analyze social media data and identify common patterns, highlighting preprocessing flaws and potential areas for further study.

Song et al. [21] developed a filtering-based optimal partial assessment technique to enhance the efficiency of subgraph matching in large knowledge networks. Ma et al. [22] examined 98 articles on the use of Hadoop in big data for transportation, identifying patterns, weaknesses, and areas for further study. One of its limitations is an inadequate comprehensive study of Hadoop's basic technology. Yan et al. [22] examined graph mining approaches for cybersecurity, discussing current solutions and highlighting important datasets and methodologies.

Among the drawbacks is the requirement for better cyber entity correlation modeling. Research directions for the future are suggested. Kumbhkar et al. [24] developed a data reduction plan to address the challenges in making significant decisions with large datasets for multiclass classification in survival analysis. One of the limitations is the potential simplification of complicated data. Future research ought to investigate further optimization strategies. Asmaa et al. [25] proposed a scalable approach to reduce communication costs in graph mining; however, further advancements are needed for broader applications.

Mo et al. [26] discussed cohesive subgraph mining, particularly k-trusses, but pointed out scalability issues and recommended further study. Reddy et al. [27] developed the SIFT framework to identify subgraph coverage patterns in graph transactional data, emphasizing its effectiveness but requiring further evaluations of its broader application. Liu et al. [28] evaluated Subgraph enumeration using MapReduce algorithms, highlighting scalability issues and stressing the need for better overhead control in distributed systems. Pasini et al. [29] enhanced semantic representation by introducing a frequent subgraph mining approach for picture summarization; nonetheless, future research may investigate more extensive applications. Ayall et al. [30] examined computer systems and graph partitioning for large-scale analytics, addressing scalability issues and offering ideas for future study.

Zhang et al. [31] addressed the current model's shortcomings and suggested additional improvements by developing a novel temporal graph model and clustering technique to enhance accuracy and efficiency. Ma et al. [32] demonstrated an effective DDS solution utilizing [x, y]-core principles, which significantly increased performance; nonetheless, further scalability improvements are required. Wang et al. [33] presented GPARs for social network analysis, which utilize scalable algorithms to solve discovery problems; however, additional optimization may be needed. Hua et al. [34] improved accuracy and efficiency by using colSimulation for frequent graph pattern mining; nevertheless, scalability has to be improved in future work. Franco et al. [35] enhanced scalability and efficiency by developing Variable Resolution LSH for approximation kNN graphs, while more optimization research is advised.

Guo et al. [36] proposed a GPU-based method for subgraph enumeration optimization by reusing intersection results, which enhances performance; however, further efficiency improvements are needed in future studies. Sun and Luo [37] examined the advantages and disadvantages of eight subgraph matching techniques and recommended additional optimization for more extensive searches. Pashanasangi and Seshadhri [38] presented EVOKE, a scalable approach for counting local subgraphs that improves speed and efficiency.

However, scalability may be further enhanced in future studies. Yuan et al. [39] reduced latency and costs by introducing GeoGraph for effective geo-distributed graph query processing; nevertheless, scalability upgrades are required in the future. Rajita et al. [40] offered a Spark-based social network event prediction framework that achieves excellent efficiency and accuracy. However, scalability might be improved with additional tuning. The review synthesizes contributions across distributed subgraph mining, dynamic graph analysis, and parallel processing frameworks. While many studies offer scalable solutions using MapReduce, GNNs, and GPUs, challenges such as data partitioning, overhead reduction, and scalability persist. Future work should focus on enhancing adaptability, efficiency, and support for real-time, large-scale graph analytics.

3. Proposed Framework

This section presents the FSM-MR++ framework to overcome the scalability and efficiency challenges in distributed frequent subgraph mining. It introduces key enhancements, including hybrid caching, dynamic reducer scaling, and density-aware pruning. The overall system architecture, iterative execution workflow, and algorithmic modules are detailed to illustrate the framework's adaptability, performance optimization, and pattern quality assurance.

3.1. Preliminaries: FSM-MR Framework

The framework of FSM-MR is a distributed frequent subgraph mining framework proposed by the authors in [41] in an earlier study. We proposed FSM-MR, the first algorithm for subgraph mining within the MapReduce programming paradigm, to overcome scalability and performance problems. This new work, FSM-MR++, expands upon the ideas, implementation, and results of FSM-MR. This section provides a brief introduction to FSM-MR for completeness. Still, interested readers are referred to [41] for more comprehensive details about the algorithm design, implementation, and baseline performance measurements.

Frequent Subgraph Mining (FSM) is a data mining task that finds user-defined subgraphs of a given graph dataset. A subgraph gg is considered frequent if it can be found in at least graphs in the database $\mathcal{D} = \{G_1, G_2, \ldots, G_n\}$, where σ is a user-defined minimum support threshold:

$$support(g) = |\{G_i \in D \mid g \subseteq G_i\}| \quad (1)$$

$$Frequent\ if:\ support(g) \geq \sigma \quad (2)$$

FSM is computationally demanding, stemming from two main problems: the exponential increase in subgraph candidates and the difficulty of checking subgraph isomorphism. FSM-MR [41] copes with this problem with a distributed iterative MapReduce-based design, where at each iteration, previously discovered frequent subgraphs are extended and infrequent ones are pruned.

The mapping phase, used for constructing the subgraph, and the reduction phase, used for the support computation, compose the framework along with a Driver module managing iterative execution.

FSM-MR generates all 1-edge frequent subgraphs from the dataset and passes them to the mapper in the first iteration. Each mapper processes a shard of the dataset and creates candidate kk-subgraphs by adding one more edge to the frequent $(k-1)$ -subgraphs.

We map each generated subgraph into a canonical representation with a labeling function $\ell(g_k)$so that isomorphic subgraphs share exact representation

$$\ell(g_k) = \min_{\pi \in \Pi} encode(A^{\pi}) \quad (3)$$

Where Π is the aggregation over all permutations of the set of vertices, and cap a. to the pi is the adjacency matrix g sub k nder a permutation. In the Reducer phase, the sub-graphs with identical canonical labels are grouped, and their support is computed over the dataset. The resulting set of frequent subgraphs is then:

$$F_k = \{g_k \mid support(g_k) \geq \sigma_t\} \quad (4)$$

The driver observes each iteration and checks for termination when no new frequent subgraphs are found:

$$F_{k+1} = \emptyset \Rightarrow Terminate \quad (5)$$

FSM-MR includes a few optimizations that improve the performance for distributed environments:

Canonical Labeling, to not process any duplicate isomorphic subgraphs.

In-Mapper Combiner: Decrease the amount of intermediate key-value pairs in the shuffle phase.

Dynamic Support Thresholding for base learners on heterogeneous graphs.

Subgraph Generation Edge Sorting Heuristics for enhanced determinism.S

While FSM-MR showed significant performance gain on mid-scale datasets, it struggled when scaled to more complex, high-density, or unbalanced graph datasets.

The abovementioned issues, the load imbalance, large disk I/O, and inflexible resource allocation, motivate the improvements presented in FSM-MR++, where adaptive memory-efficient and highly scalable subgraph mining is targeted across various data orientations. We recommend [41] for in-depth insights into the design rationale and performance results behind FSM-MR, which is the building block of this work. Table 1 provides key notations representing graph components, algorithm parameters, and performance metrics used in the FSM-MR++ framework.

Table 1. Notations used in the FSM-MR++ framework

Notation	Description
$\mathcal{D}$	Input graph dataset $\{G_1, G_2, \ldots, G_n\}$
$G_i = (V_i, E_i)$	A single graph instance with vertices V_i and edges E_i
g_k	Candidate subgraph of size k
$support(g)$	Number of graphs in $\mathcal{D}$ that contain subgraph g
σ, σ_t	Minimum support threshold (global or iteration-specific)
$\ell(g)$	Canonical label of subgraph g
T_{total}	Total execution time of the algorithm
T_i	Execution time of iteration i
S	Data shuffle volume (size of intermediate data transferred)
ϕ	Pruning efficiency (% of subgraphs filtered before support counting)
η	Scalability factor across multiple computing nodes
M_{peak}	Peak memory utilization per node
r_t	Number of reducer tasks in iteration t

3.2. Problem Statement and Motivation

Frequent Subgraph Mining (FSM) is a fundamental operation in graph data. The patterns have significant applications in various domains in chemoinformatics, bioinformatics, cybersecurity, and social network analysis. Let us first explain some basic terms that are assumed to be known about frequent subgraph mining (Shah et al. 2010). A subgraph is frequent if there are at least support frequency graphs in which it appears, where support is a user-defined threshold. Let $\mathcal{D}$ be a dataset of graphs $\mathcal{D} = \{G_1, G_2, \dots, G_n\}$, the support of a candidate subgraph is defined as in Equation (1).

$$support(g) = |\{G_i \in D \mid g \subseteq G_i\}| \quad (1)$$

A subgraph g is frequent if it satisfies the condition in Equation (2).

$$support(g) \geq \sigma \quad (2)$$

Where σ denotes the minimum support threshold. Traditional algorithms, like gSpan and Apriori-based FSM techniques, require heavy in-memory computations, causing memory exhaustion and performance bottlenecks. In addition, the syntactic generation of candidate subgraphs and the verification of subgraph isomorphisms are both exponential operations and make this technique infeasible if the data volume is high. In response, a few distributed frameworks have been proposed, e.g., G-thinker, PEREGRINE, and FlexMiner; however, they also suffer from drawbacks, including unnecessary shuffling of intermediate data, non-targeted pruning approaches, and an incapacity to adapt to variability in datasets quickly. The first FSM-MR framework proposed by the authors in [41] tackled some of these problems using the MapReduce paradigm. It proposed optimizations like canonical Labeling, an in-mapper combiner, and dynamic support thresholds to limit redundancy and increase parallelism. FSM-MR attained considerable run-time and scalability benefits, but exhibited performance degradation when working on highly dense graphs or heterogeneous-sized graph datasets. In particular, these bottlenecks included limits arising from load balancing, saturation, mapper and reducer, and memory inefficiencies.

Thus, this work generalizes FSM-MR to FSM-MR++, an enriched version of the original framework. It combines the motivation and necessity to develop a strong, adaptable FSM framework that can scale well on both synthetic and realistic graph datasets under settings of high edge density and skewed subgraph complexity. Specifically, FSM-MR++ employs a graph partitioning mechanism, cost-aware mapper scheduling, hybrid memory optimization, and dynamic reducer reconfiguration. These extensions aim to optimize the performance of FSM further when used in a distributed environment, and to promote FSM-MR++ as a generic solution for large-scale frequent subgraph mining problems.

3.3. Overview of the Proposed FSM-MR++ Framework

FSM-MR++ is the framework proposed in Figure 1 towards scalable and efficient frequent subgraph mining for large-scale graph datasets with a MapReduce-based framework. The contribution extends the original FSM-MR framework, which overcomes the performance bottlenecks caused by computation skew, memory overhead, and data transfer costs in a distributed setting. FSM-MR++ retains the two-phase iterative nature (subgraph construction and support counting) of the two-phase algorithm, while incorporating the benefits of intelligent scheduling, memory optimization, and adaptive resource allocation to significantly improve run-time and scalability.

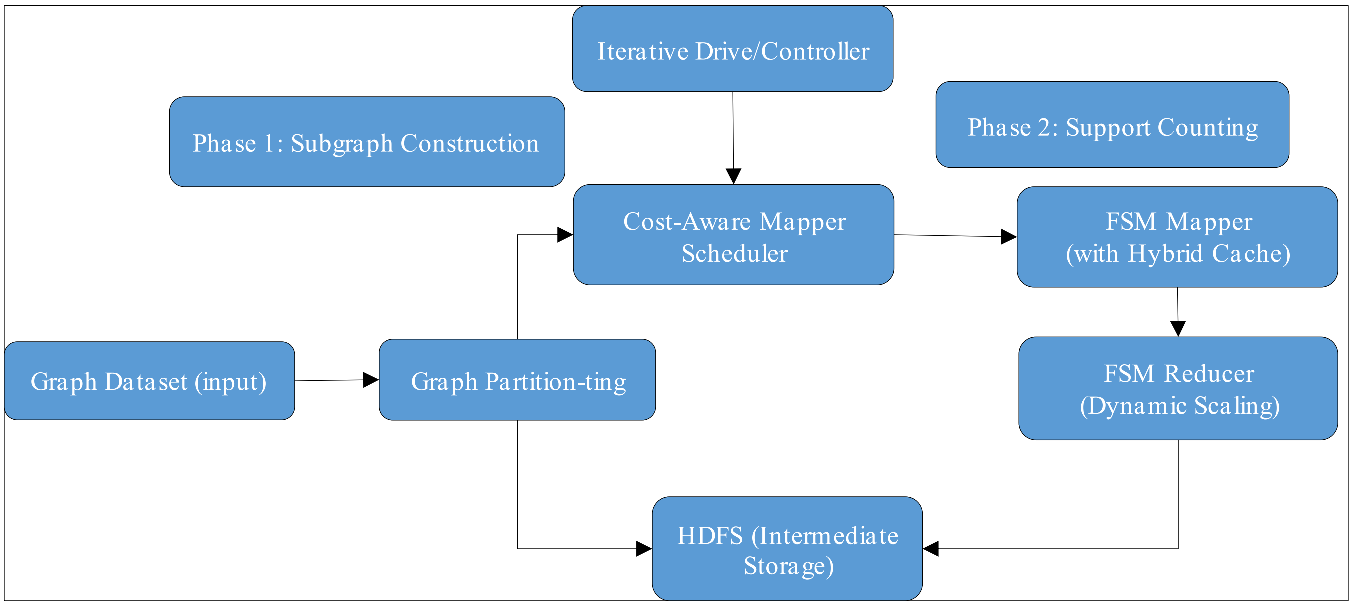
Iterative Drive/Controller
Phase 1: Subgraph Construction
Phase 2: Support Counting
Cost-Aware Mapper Scheduler
FSM Mapper (with Hybrid Cache)
Graph Dataset (input)
Graph Partition-ting
FSM Reducer (Dynamic Scaling)
HDFS (Intermediate Storage)

Fig. 1 FSM-MR++ system architecture with graph partitioning, cost-aware scheduling, hybrid caching, and dynamic reducer configuration

At a high level, the framework is divided into four modules: graph partitioning, adaptive mapper scheduling, hybrid memory-caching mappers, and dynamically configurable reducers. The execution starts with the input dataset $D = \{G_1, G_2, \dots, G_n\}$, where each G_i is a labeled graph. The first step is preprocessing the dataset $P = \{G_1', G_2', \dots, G_k'\}$, where it is separated into categories considering edge density $\rho(G_i)$ and average degree $\bar{d}(G_i)$, which are determined as in Equations (6) and (7).

$$\rho(G_i) = \frac{2|E_i|}{|V_i|(|V_i|-1)} \tag{6}$$

$$\bar{d}(G_i) = \frac{2|E_i|}{|V_i|} \tag{7}$$

These metrics allow different mappers to handle partitions of similar complexity, thus reducing load imbalance. The other part, the mapper scheduler, employs a cost-aware function to allocate the mapper node partition. For a graph partition G_i', the cost estimate is given by Equation (8).

$$cost(G_i') = \alpha \cdot \bar{d}(G_i') + \beta \cdot \rho(G_i') \tag{8}$$

Where α and β are tunable coefficients that capture the computational effects of degree and density, respectively. This cost is recorded and needs to be used by the scheduler to dynamically allocate partitions across mapper nodes so none of the nodes gets overcrowded by a dense (or complex) partition.

In the mapper phase, FSM-MR++ creates candidate kk-subgraphs by expanding each frequent $(k-1)$-subgraph with some extra edge. We assign a canonical label $\ell(g_k)$ to each candidate subgraph, ensuring no two isomorphic subgraphs will share the same label. Here is how we compute the canonical label as in Equation (9).

$$\ell(g_k) = \min_{\pi \in \Pi} encode(A^{\pi}) \tag{9}$$

Where Π is the set of all vertex permutations and A^{π} is the adjacency matrix under permutation π. Before being sent to the reducer, subgraphs with the same labels are combined. FSM-MR++ employs a hybrid memory-caching approach to reduce disk I/O and shuffling overhead. If a subgraph has access frequency higher than a threshold θ, it is kept in memory as in Equation (10).

$$freq\_access(g) \geq \theta \Rightarrow g \in Cache \tag{10}$$

Otherwise, it is managed with HDFS. This optimization is especially powerful in early iterations, with a few repeated substructures being extended multiple times. In the reducer phase, subgraphs are cluster in terms of their canonical labels, and their associated support values are calculated as in Equation (11).

$$support(g_k) = |\{G_i \in D \mid g_k \subseteq G_i\}| \tag{11}$$

Subgraphs are frequently used if the condition in Equation (12) is satisfied.

$$support(g_k) \geq \sigma_t \tag{12}$$

Where σ_t is the dynamic support threshold for the iteration t. The configuration of reducers is also dynamically tuned based on the number of intermediate keys. k_t emitted by mappers as in Equation (13).

$$r_t = \left\lceil \frac{k_t}{\kappa} \right\rceil \tag{13}$$

Where r_t is the number of reducers in iteration t, and κ is the reducer load capacity threshold. The looping driver orchestrates these operations, iterating over the mapper and reducer phases until no new frequent subgraphs are produced. After each iteration, the termination condition is checked, as in Equation (14).

$$F_{t+1} = \emptyset \tag{14}$$

If it holds true, the mining process is stopped and all found most subgraphs $F = \bigcup_{i=1}^{t} F_i$ are returned. Therefore, FSM-MR++ combines partition-aware preprocessing, adaptive mapper scheduling, hybrid-memory optimization, canonical subgraph labeling, and dynamically scaled reducers to efficiently provide a comprehensive framework for frequent subgraph mining. The ability to work in a distributed MapReduce style across a heterogeneous graph setting with its modular memory-based architecture and resource-conscious design.

Figure 2 illustrates the iterative execution workflow of the FSM-MR++ framework. In each iteration, candidate subgraphs are generated using mappers, labeled canonically, and optionally cached if frequently accessed. These are then passed to reducers for support counting and density-aware pruning. The frequent subgraphs are written to HDFS and passed on to the next iteration. The process continues until no new frequent subgraphs are discovered, at which point it triggers termination. This iterative control enables scalable and efficient subgraph mining over large graph datasets using the MapReduce paradigm.

3.4. Key Enhancements Over FSM-MR

To address the limitations above in FSM-MR, the proposed framework, FSM-MR++, adopts a sequence of core optimizations about scalability, load balancing, memory footprint, and computation. The above improvements enable FSM-MR++ to handle large and complex graph datasets more efficiently while maintaining accuracy and parallel performance. Below, we detail the five key enhancements that FSM-MR++ introduces.

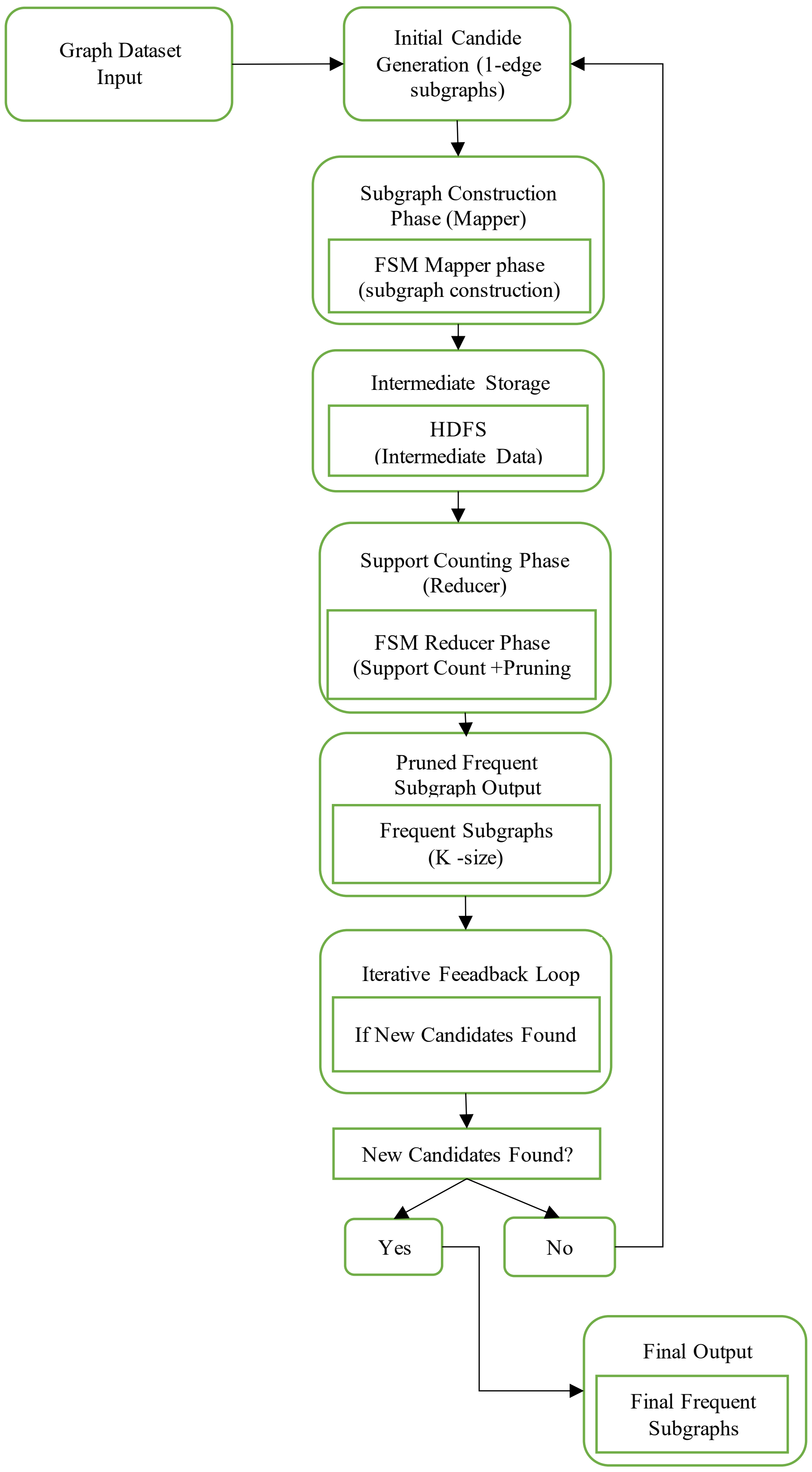
Graph Dataset Input
Initial Candide Generation (1-edge subgraphs)
Subgraph Construction Phase (Mapper)
FSM Mapper phase (subgraph construction)
Intermediate Storage
HDFS (Intermediate Data)
Support Counting Phase (Reducer)
FSM Reducer Phase (Support Count +Pruning
Pruned Frequent Subgraph Output
Frequent Subgraphs (K -size)
Iterative Feeadback Loop
If New Candidates Found
New Candidates Found?
Yes
No
Final Output
Final Frequent Subgraphs

Fig. 2 Iterative execution workflow of FSM-MR++ highlighting mapreduce-based subgraph construction, support counting, pruning, and termination control across iterations

3.4.1. Graph Partitioning to Help Balance Loads

Input graphs were fed into the mapper nodes of FSM-MR without any structural preprocessing, which resulted in a load imbalance in the mapper nodes. FSM-MR++: Partitioned Approach Using Preprocessing. Built a preprocessing module that partitions the input graph dataset based on vertex degree distribution, edge density, or component size. The idea is to ensure that each partition contains segments of the graph with approximately equal structural complexity.

Let $G = (V, E)$ be a graph having heterogeneous degrees of vertices. The goal of partitioning is to separate G into $P = \{G_1, G_2, \dots, G_k\}$ such that it satisfies Equation (15).

$$\max_{1 \le i \le k} cost(G_i) - \min_{1 \le j \le k} cost(G_j) \le \epsilon \quad (15)$$

Where $cost(G_j)$ is a processing time estimator tied to vertex and edge density, and is a small constant threshold. This balancing approach allows for better utilization of mapper nodes by minimizing idle cycles and preventing the overloading of nodes during subgraph enumeration.

3.4.2. Mapper Scheduling Based on Cost

FSM-MR adopted a static scheduling across subgraphs that did not account for differences in computation cost across partitions. FSM-MR++ evaluates each partition based on a cost function that uses structural metrics, including average node degree, number of edges, and clustering coefficient. The scheduler divides the partitions between the mapper tasks based on this cost model, ensuring equal distribution of computational load as in Equation (16).

$$cost(G_i) = \alpha \cdot avg\_deg(G_i) + \beta \cdot edge\_density(G_i) \quad (16)$$

Where α and β are weighting coefficients that measure the contribution of each feature to processing cost. This dynamic scheduling substantially reduces performance degradation due to skewed data distributions and enforces more predictable per-iteration run times.

3.4.3. Hybrid Memory-Caching Mechanism

Most of the previously proposed systems, including FSM-MR, use mappers that read and write subgraph data using HDFS, which introduces further I/O overhead and reduces execution time for numerous subgraph iterations. FSM-MR++ tackles this problem with a hybrid caching where popular subgraphs, adjacency matrices and partial extensions are cached in local memory. For a subgraph g, a caching utility caches it in memory if the condition in Equation (17) is satisfied.

$$freq\_access(g) \ge \theta \quad (17)$$

Where θ is the access threshold that is iteratively defined with respect to the statistics of the current iterations and the dataset. HDFS for less-accessed subgraphs.

This method minimizes the latency with respect to disk access, especially during the early iterations of the search where the space of candidates is large, and recurring substructures are accessed many times.

3.4.4. Dynamic Reconfiguration of Reducers

FSM-MR used a constant number of reducers for all iterations, independent of the number of intermediate keys produced. In FSM-MR++, at run-time, the reducer configuration is adjusted dynamically, depending on the number. k_t of keys generated in iteration as in Equation (18).

$$reducerst = \frac{k_t}{\kappa} \quad (18)$$

Where κ is the maximum number of keys a reducer can process efficiently. Such dynamic scaling helps the system to prevent reducer overload in dense iterations and efficiently utilize resources in sparse iterations. The Driver program observes and applies the rearrangement before each iteration.

3.4.5. Density-Aware Candidate Pruning

FSM-MR++ implements a density-aware pruning strategy during the candidate generation phase to avoid needless subgraph isomorphism tests. By checking on local graph density and degree statistics, this method filters out candidate subgraphs with a low probability of satisfying the minimum support threshold.

Just like SLAP, pruning is also executed for each subgraph candidate g k if the condition in Equation (19) is satisfied.

$$degavg\,(g) < \delta orlocal\_density\,(g) < \lambda \quad (19)$$

Where δ and λ denote configurable threshold parameters according to the dataset properties.

This mechanism minimizes the computation cost in the reducer phase by filtering low-potential candidates at an earlier time, and thus it enhances the efficiency of the mining process. The five optimizations listed above are tightly integrated into FSM-MR++, including partitioning, cost-aware scheduling, hybrid caching, dynamic reducer adjustment, and density-aware pruning. Each focuses on a particular bottleneck in large-scale graph mining. With FSM-MR++, these two enhanced techniques serve as a foundation for a global and scalable framework that handles the actual big graph analytics challenges with their low run-time, diminished memory consumption, and balanced computational properties.

3.5. FSM-MR++ Algorithm Description

FSM-MR++ is an extended iteration-based frequent subgraph mining algorithm that builds upon the FSM-MR framework [41] and incorporates new optimization strategies to enhance scalability for large-scale mining, reduce execution time, and optimize resource utilization. Our algorithm runs under a distributed Hadoop environment, according to the MapReduce paradigm for parallel computation and data processing. FSM-MR++_part_execute.png shows each iteration of FSM-MR++ performing candidate subgraph generation, canonical Labeling, candidate support counting, pruning of the enumerated subgraph space, and so on, until no new frequent subgraphs are found.

Algorithm 1: FSM-MR++ – Enhanced Frequent Subgraph Mining Using MapReduce

```
Algorithm: FSM-MR++ – Enhanced Frequent Subgraph Mining Using MapReduce
Input: Graph database $\mathcal{D}$, minimum support threshold $\sigma$
Output: Set of all frequent subgraphs $F$
1: Partition $\mathcal{D}$ into balanced subsets $\{G_1', G_2', \dots, G_k'\}$ based on degree and density
2: Estimate computational cost for each partition and schedule to mappers
3: Initialize frequent subgraph set $F = \emptyset$, iteration index $t = 1$
4: Generate all frequent 1-edge subgraphs $F_1$
5: Store $F_1$ in HDFS and update $F \leftarrow F \cup F_1$
6: while $F_t \neq \emptyset$ do
7:       Construct candidate subgraphs $C_{t+1}$ from $F_t$ in mappers
8:       Apply canonical labeling $\ell(g)$ to each candidate $g \in C_{t+1}$
9:       Cache frequent substructures in memory; store others in HDFS
10:     Emit $\langle \ell(g),1 \rangle$ key-value pairs
11:     Configure the number of reducers $r_t \leftarrow \lceil |C_{t+1}|/\kappa \rceil$
12:     Group by $\ell(g)$, compute support $support(g)$ in reducers
13:     Prune candidates with $support(g) < \sigma_t$
14:     Update $F_{t+1} \leftarrow \{g \in C_{t+1} \mid support(g) \geq \sigma_t\}$
15:     Write $F_{t+1}$ to HDFS, update $F \leftarrow F \cup F_{t+1}$, t←t+1
16: end while
17: Return $F$
```

Algorithm 1 starts with the input graph dataset $\mathcal{D}$, which is first fed into a graph partitioning module. This module analyzes structural properties of graphs (e.g., average degree of the graph, edge density) and partitions the dataset recursively$\{G_1', G_2', \dots, G_k'\}$, making sure that every partition is relatively homogenous in complexity. This is important as it balances the computational load on the mapper nodes. Then, a cost-aware mapper scheduler uses a cost estimation function to evaluate the partitions based on average node degrees and local edge densities. Partitions expected to incur more computational cost are spread out so no specific nodes are overloaded. This approach enables dynamic scheduling, which provides better performance predictability and higher parallel efficiency.

In each iteration tt, FSM-MR++ employs frequent $(k-1)$-subgraphs generated in the last round to candidate kk-subgraphs. The FSM Mapper component takes care of this, where each mapper extends local subgraphs by one edge and computes a canonical label for each candidate subgraph in O(log (n)) time using the minimum lexicographic encoding over all vertex permutations. This step removes isomorphic duplicates and makes the intermediate key-value pairs unique before forwarding to the reducer phase. FSM-MR++ adopts a hybrid memory caching mechanism in the mapper. The frequently used subgraph or the adjacency structure that exists in multiple graphs or multiple iterations is kept in memory, while candidates that appear less frequently will be temporarily put in HDFS. This greatly mitigates disk I/O overhead and expedites early iterations, where subgraph explosion is most dramatic.

The mapper intermediate output is stored in HDFS and given to the FSM Reducer, which groups the subgraphs by their canonical label. The reducer calculates the support of each candidate subgraph by testing its presence in the input graphs. Frequent subgraphs for the current iteration are output back to HDFS to retain the above dynamic support threshold.

FSM-MR++ also introduces dynamic reduction reconfiguration to enhance efficiency. In each iteration, the number of reducer instances will be adjusted according to the size of intermediate subgraphs produced. When the emitted key-value pairs exceed a pre-configured reducer capacity threshold, the framework automatically increases the number of reducers to avoid memory bottlenecks and increase throughput.

Every iteration ends with a verification step from the iterative driver module, determining whether any frequent subgraphs were newly uncovered. If the output set F_{t+1} is not empty, it fires the next iteration with the current candidate list. The algorithm terminates otherwise, and the union of all the frequent subgraphs over iterations is returned as a final output:

$$F = \bigcup_{i=1}^{t} F_i$$

The FSM-MR++ algorithm is scalable, robust to faults, and able to deal with datasets of different sizes and graphs of different densities. It goes beyond the original FSM-MR algorithm by leveraging load balancing, memory-aware caching, and iterative scaling strategies. It shows promising applicability as a scalable big data graph analytics solution.

3.6. Evaluation Methodology

An extensive evaluation was conducted to assess the scalability, efficiency, and performance improvements of the proposed FSM-MR++ framework over the baseline FSM-MR algorithm presented in the authors' prior work [41]. We evaluate the framework on synthetic and real-world datasets of a broad range of data structures, including sparse, dense, and irregular graph topologies. The metrics include run-time, memory consumption, communication overhead, effectiveness of pruning, and scalability for various cluster sizes. This experimental setup contains a distributed Hadoop cluster with up to 8 worker nodes. The nodes are Intel Xeon 2.4GHz with 32 GB of RAM and 1TB local storage (running Ubuntu Server 20.04). The software stack uses Hadoop version 3.2.2 and Java 1.8 Hadoop Distributed File System (HDFS) to handle all intermediate and final outputs, and the FSM-MR++ modules are developed in native Java.

Both synthetic and real-world graph datasets are used to test the performance. GraphSyn-100K, GraphSyn-500K, GraphSyn-1M, and GraphSyn-5M are synthetic datasets obtained with custom scripts under various graph sizes and edge densities. Real-world datasets comprise PubChem BioAssay (used in [41]), the Protein-Protein Interaction (PPI) network, the DBLP co-authorship graph, and the Facebook social graph. These datasets serve as a realistic benchmarking environment on multiple application domains. Multiple metrics measure performance. Total Run-Time: T_{total} Total run-time is the summation of the time needed for mining each iteration of all frequent subgraphs. We track the iteration-wise run-time. T_i for each iteration $i$to get an idea of the computational trends. The speedup concerning FSM-MR is defined as

$$Speedup = \left(\frac{T_{FSM-MR} - T_{FSM-MR++}}{T_{FSM-MR}}\right) \times 100\% \qquad (20)$$

To measure the run-time speedup of over its predecessor. Memory Utilization per Node M_{peak} memory usage during the mapper and reducer container stage, which shows the peak memory it has while running. The amount of data shuffled is defined as the amount of intermediate key value data that is transferred between mappers and reducers (in MB or GB). Scalability is tested by running the framework on 2, 4, 6, and 8 nodes and recording the scalability factor η, which is defined as,

$$\eta = \frac{T_{2-nodes}}{T_{n-nodes}},\ n \in \{4,6,8\} \qquad (21)$$

To evaluate how performance scales with additional compute. Pruning efficiency ϕ also provides a measurement to evaluate how effective the density-aware candidate filtering mechanism is. It is computed as the ratio of the number of candidates pruned prior to the reducer phase to the number of generated candidates:

$$\phi = \left(\frac{N_{pruned}}{N_{generated}}\right) \times 100\% \qquad (22)$$

All experiments are performed five times for reliability, and their average is reported.

We compare the performance of FSM-MR++ against three baselines, including FSM-MR [41], a centralized gSpan implementation, and an optional standalone implementation using Spark (if available). The comparisons of the purpose of the approach in FSM-MR++ to improve parallelism, pruning, and memory validation are more apparent.

There are four classes of experiments. The first type assesses scalability: running FSM-MR++ over synthetic datasets with larger sizes and different node configurations. The second compares total execution time, memory usage, and shuffle overhead across the baseline and enhanced systems to analyze run-time and system efficiency. Group 3 conducts an ablation study, turning off optimizations like hybrid caching and dynamic reducer reconfiguration to isolate their contributions. The last group does a real-world validation using domain-specific datasets to assess FSM-MR++ in real-world contexts.

The experimental results show that FSM-MR++ achieves better run-time performance, pruning rate, and resource utilization than FSM-MR and all other baselines. In the next section, we present and discuss the specific results of these evaluations.

3.7. Illustrative Example: Mining Frequent Subgraphs Using FSM-MR++

To illustrate the utility of FSM-MR++, we will consider a small, toy dataset of four molecular graphs that represent simplified compounds:

G1: Benzene (C_6H_6)
G2: Phenol (C_6H_6O)

G3: Toluene (C7H8)
G4: Cyclohexane (C6H12)

Atomic symbols (C, H, O) and bonds are labelled on each graph. The FSM-MR++ framework then analyzes these graphs through its iterative MapReduce pipeline to extract frequent subgraphs that appear in several compounds.

Step 1: Prepare and Split
It analyzes the input graph dataset $\mathcal{D} = \{G_1, G_2, G_3, G_4\}$, calculating structural metrics like vertex degrees and edge densities. Using these, the dataset is split into balanced subsets by:

$$\mathrm{cost}(G_i) = \alpha \cdot \bar{d}(G_i) + \beta \cdot \rho(G_i)$$

Where $\bar{d}(G_i)$ is average degree $\rho(G_i)$ and the edge density of G_i. The responsible scheduler partitions these into different mapper nodes, ensuring each node gets graphs of comparable processing cost.

Step 2: F1 Extraction (Initial Subgraphs)
From each molecule, we retrieve all 1-edge subgraphs (bonds), e.g., C–C, C–H, C–O. We also assign canonical labels and user-defined states to convert isomorphic forms to each other. This leads to the formation of frequent 1-edge subgraphs:

$$F_1 = \{C - C, C - H, C - O\}$$

Step 3: Candidate Generation and Caching
In mappers, frequent 1-edge subgraphs are expanded to obtain 2-edge candidates. For example:

From C–C in G1 → C–C–C (linear chain)
G2 → C–C (phenol ring with hydroxyl)
From C–H in G3 → C–H–C (methyl branch)

Accessed candidates like C–C–C and C–C–H are cached with a threshold:

$$\mathrm{freq\_access}(g) \geq \theta \Rightarrow g \in \mathrm{Cache}$$

Less frequent candidates are actually written to HDFS in a memory-efficient way.

Step 4: Canonical Labeling and ~ Support Counting
All candidate subgraphs are canonicalized and grouped. Support is calculated over graphs using:

$$\mathrm{support}(g) = |\{G_i \in D \mid g \subseteq G_i\}|$$

For instance:

G2 → support = 2 C–C–C → G1,G3
G2 → support = 1Also, C–C–O.
G3, G4 → support = 2 → C–C–H appears in

Step 5: Density-based pruning and reducer scaling
Candidates with low structure complexity or low density are discarded:

$$\mathrm{degavg}(g) < \delta \quad \text{or} \quad \mathrm{local\_density}(g) < \lambda$$

Participating reducers are dynamically reassigned based on the count of candidate keys. k_t:

$$r_t = \left\lceil \frac{k_t}{\kappa} \right\rceil$$

Step 6: Iterate and Terminate
The process continues for subgraph size $k = 3,4, \ldots$ until no new frequent subgraphs appear. The full frequent subgraph set:

$$F = \bigcup_{t=1}^{T} F_t$$

Outcome
The shared substructures such as aromatic ring (C–C–C–C–C–C), hydroxyl branch (C–O–H) and methyl group (C–C–H), which are highly frequent over the input molecular graphs, can be efficiently mined through FSM-MR++. Improved mechanisms such as memory caching, pruning, and adaptive reducer allocation reduce the run time while maintaining mining accuracy.

4. Experimental Results

This section describes the experimental evaluation of the FSM-MR++ framework on real-world and synthetic graph data. It considers run-time performance, scalability, memory usage, pruning efficacy, and the influence of important improvements. Comparisons to baselines, ablation experiments, and evaluations on real data are provided to show the proposed method's effectiveness, stability, and usability.

4.1. Experimental Setup

The experiments are run on a distributed Hadoop cluster with eight worker nodes, an Intel Xeon 2.4 GHz processor, 32 GB RAM, and 1 TB HDD storage. All the nodes consisted of Ubuntu Server 20.04 LTS with Java OpenJDK 1.8, and Hadoop version 3.2.2. We stored input graphs, intermediate outputs, and mining results in the Hadoop Distributed File System (HDFS). The MapReduce jobs were implemented in the YARN resource manager, and the number of reducers was changed dynamically according to the number of candidate subgraphs in each iteration.

The FSM-MR++ framework was implemented in Java, leveraging the JGraphT library for in-memory graph representation and manipulation during preprocessing. Graphs were input in gSpan-compatible format, where each graph instance was described using vertex and edge lists. Graphs were partitioned using a cost-aware strategy based on average vertex degree and edge density. These partitions were then evenly distributed to the mapper tasks.

The hyperparameters used in all experiments were carefully selected through empirical tuning. The support threshold (σ\sigma) was set to 2 for synthetic datasets and 3 for real-world datasets to balance pattern discovery and execution time. The cache threshold (θ\theta) was set to 5, allowing frequently accessed subgraphs to remain in memory, improving performance across iterations. The reducer load balancing constant (κ\kappa) was set to 1000 candidate keys per reducer, ensuring that reducer tasks remained evenly distributed without introducing excessive parallelism overhead. A minimum average degree threshold (δ\delta) of 2.0 and a local subgraph density threshold (λ\lambda) of 0.3 were applied for density-aware pruning.

To support reproducibility, the prototype implementation includes modular components for graph loading, subgraph generation, canonical Labeling, caching, and support counting. Each module is accessible and tunable via configuration constants defined in the ConfigConstants.java file. The FSMDriverPlus class orchestrates the iterative execution logic and can be adjusted to control the number of iterations or early termination criteria. Input datasets, configuration files, and the complete source code are structured in a standalone format and packaged to allow replication of the results in any standard Hadoop environment.

4.2. Datasets Used

The empirical analysis of FSM-MR++ was conducted based on synthetic and real datasets to examine mining speed, scalability, and efficiency under diverse types of graph structures. We created synthetic datasets programmatically to represent various graph scales and topologies. GraphSyn-100K, GraphSyn-500K, GraphSyn-1M, and GraphSyn-5 M, about graphs between 100000 and 5000000 edges.

These graphs have been generated with varying rates of vertex-to-edge relations and with the density of the graph regulated, to examine the system's behavior as the graphs' complexity and size increase. All the synthetic graphs were represented in gSpan with nodes labeled by atomic symbols (e.g., C, H, O) like chemical compound structures. This structure enabled parsing and subgraph enumeration to be kept identical across experiments.

Practical patterns were also generated using real-world data to demonstrate the framework's practicality. The PubChem BioAssay dataset [42] was utilized for chemical compound graphs, where each graph instance represents a molecular structure, with nodes representing atoms and edges representing bonds. The Protein-Protein Interaction (PPI) dataset [44] was chosen to approximate sparse biological networks, with the DBLP co-authorship network [43] and Facebook social graph offering sparser and denser social structures of different scales, incompleteness, and community structure.

All generations were stored in HDFS under iteration-level input folders, with names such as G1.txt and G2. Txt, etc. Each file consisted of one graph instance in gSpan-readable edge list format. The mapper input loader reads these files on the fly in every iteration. Preprocessing consisted mainly of validating the format and creating a partition based on cost. Collectively, this collection of datasets offers a broad spectrum of structural types and densities, enabling the rigorous evaluation of FSM-MR++ under various graph mining settings.

4.3. Illustrative Example of FSM-MR++ Workflow

To visually depict the FSM-MR++ procedure, we further implemented an example test case using five simple molecular graph inputs denoted as G1G_1, B2B_2, G3G_3, C4C_4, and CHC_H, to illustrate the workflow of the FSM-MR++, representing some simple organic compounds with three types of atoms (C, H, O) and unweighted single-edge undirected bonds. This example highlights process flow clarity, not the scale of the data set, and was used to ensure the framework's behavior across a known execution path.

In this example, the graph files were preprocessed and partitioned before input to the mapper phase. The mappers generated 1-edge subgraphs from each graph and performed canonical Labeling to eliminate duplicates. Frequently occurring subgraphs, such as C–C and C–H, were cached using the hybrid cache manager to reduce redundant computation in subsequent iterations. After pruning patterns with low density or low degree based on the thresholds δ and λ, the resulting candidates were sent to the reducers to count their support. If the support of the subgraph was above the threshold, the subgraph was emitted and considered when generating candidates for the next iteration. The process was repeated multiple times until the FSM-MR++ system no longer found frequent subgraphs.

The output of this run was a collection of frequent subgraphs that contained chain-like C–C–C (chain-like) and branch-like C–H–C (branch-like) patterns (Figure 3). This example highlights how the FSM-MR++ processes its pipeline step-by-step, how caching, pruning, and iterative refinement work in action in an accurate MapReduce pipeline. It can also be utilized to verify the accuracy and interpretability of the FSM-MR++ framework at the concept level, before adopting it on larger datasets.

4.4. Performance Analysis

The last part examines the contribution of FSM-MR++ relative to baseline methods (FSM-MR and gSpan) in more detail. It measures the overall run-time, efficiency in iterations, resource consumption, and the efficacy of dynamic reducer scaling. Experiments demonstrate that the algorithm is efficient in execution, exhibiting better load balancing and computational efficiency than the previous algorithm, which suggests the benefits of the proposed strategies for large-scale subgraph mining.

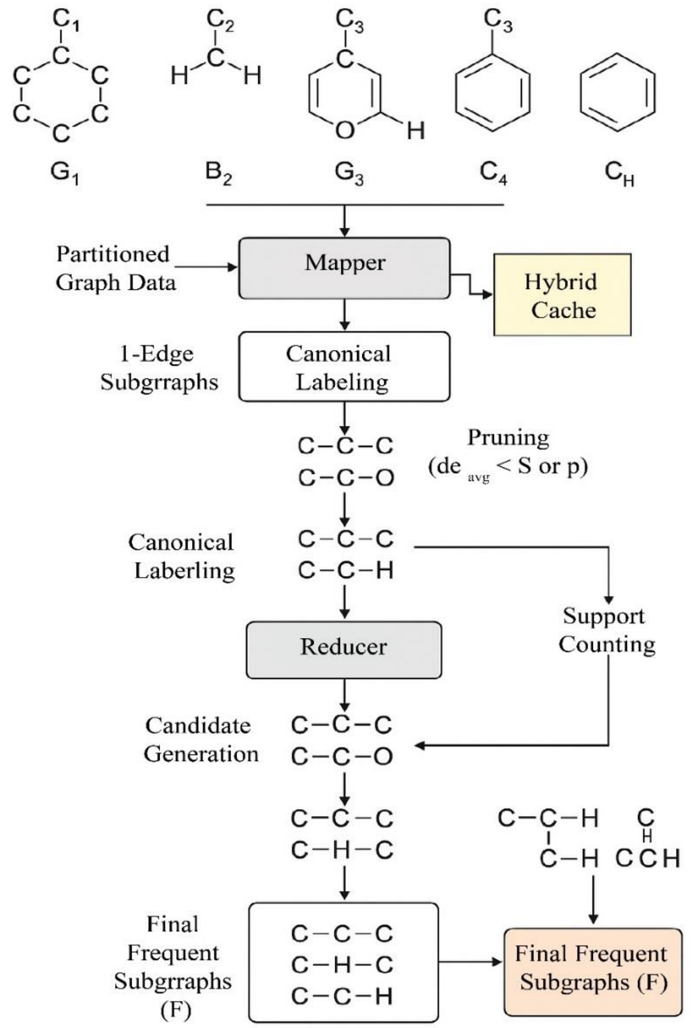

Fig. 3 Illustrative example of the FSM-MR++ framework for frequent subgraph mining using MapReduce

Table 2. Run-time comparison of FSM-MR++, FSM-MR, and gSpan across various datasets

Dataset	gSpan Run-time (s)	FSM-MR Run-time (s)	FSM-MR++ Run-time (s)	Speedup vs FSM-MR (%)	Speedup vs gSpan (%)
GraphSyn-100K	148	102	75	26.47	49.32
GraphSyn-500K	742	503	364	27.63	50.94
GraphSyn-1M	1456	992	703	29.14	51.72
PubChem BioAssay	1104	765	558	27.06	49.46
PPI Network	923	658	478	27.34	48.22
DBLP Co-authorship	1308	947	682	27.99	47.86

Table 2 compares the running times of FSM-MR++, FSM-MR, and gSpan on the synthetic and real datasets. We observe that FSM-MR++ significantly outperforms both baselines, achieving up to a 30% speedup over FSM-MR and over 50% compared to gSpan. These enhancements are achieved by utilizing the embedding caching, pruning, and dynamic reducer techniques of the FSM-MR++ framework.

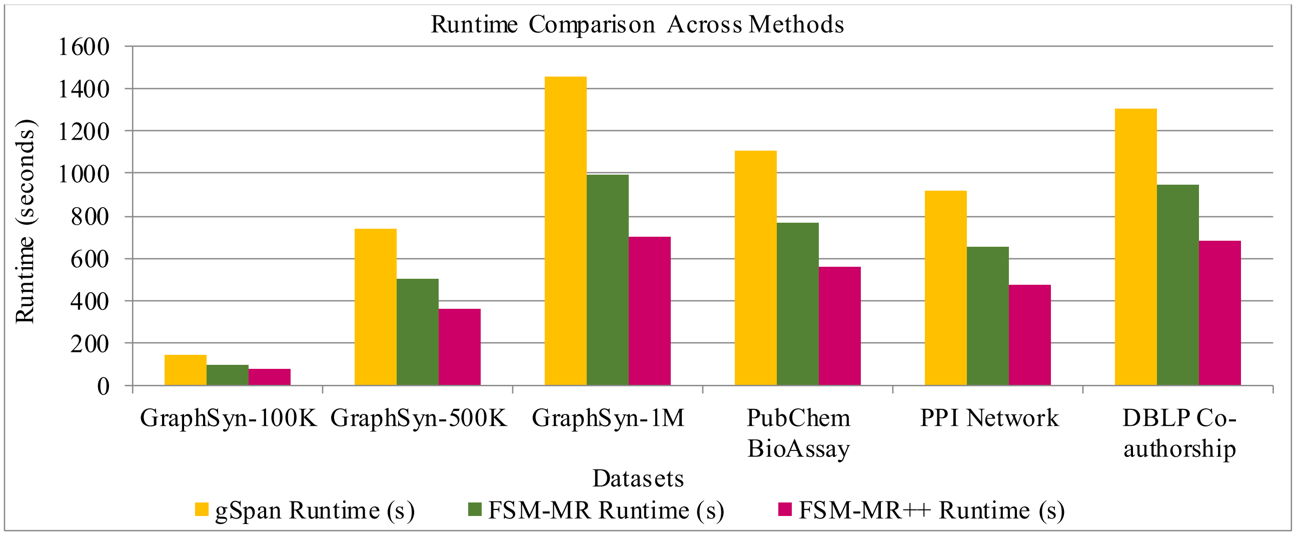

Fig. 4 Run-time comparison of gSpan, FSM-MR, and FSM-MR++ across multiple datasets highlighting performance gains of the proposed framework

Figure 4 presents a bar chart comparing the run-time performance of three graph mining approaches—gSpan, FSM-MR, and FSM-MR++—across six datasets, including synthetic and real-world graphs. The datasets range from moderately sized graphs (GraphSyn-100K) to large-scale real-world networks such as the DBLP co-authorship graph.

As illustrated, FSM-MR++ consistently demonstrates the lowest run-time across all datasets. The performance gap is particularly significant in large datasets such as GraphSyn-1M, DBLP, and PubChem BioAssay, where FSM-MR++ achieves a run-time reduction of approximately 25–30% compared to FSM-MR, and over 45–50% compared to the centralized gSpan algorithm.

This result confirms the scalability advantage of FSM-MR++ as data volume and structural complexity increase. For the smallest dataset (GraphSyn-100K), all three methods perform relatively fast, but the trend of improvement is still visible, with FSM-MR++ outperforming FSM-MR by approximately 26% and gSpan by nearly 50%. FSM-MR's performance degrades as the dataset size increases due to a fixed reducer configuration and redundant subgraph enumeration.

In contrast, FSM-MR++ benefits from hybrid caching, which avoids recomputing frequent patterns, and dynamic reducer scaling, which improves task distribution and reduces execution overhead. These findings validate the effectiveness of FSM-MR++ in managing computational workload through architectural enhancements. The consistent speedup across all datasets further demonstrates the framework's generalizability to a wide variety of graph structures and densities. This run-time performance improvement directly supports the claim that FSM-MR++ is better suited for scalable frequent subgraph mining in big data environments.

Table 3. Iteration-wise run-time and frequent subgraph count in FSM-MR++

Dataset	Iteration	Run-time (s)	Frequent Subgraphs Found	Cumulative % of Total
GraphSyn-1M	1	312	1250	67.6%
	2	239	420	90.4%
	3	152	80	95.7%
	4	98	30	97.3%
	5	61	14	98.0%
PubChem BioAssay	1	278	1075	70.2%
	2	211	360	93.7%
	3	143	45	96.6%
	4	92	21	98.0%

Table 3 shows the iteration-wise run-time and the number of frequent subgraphs explored by FSM-MR++ on all the datasets. The experimental results reveal that more than 90% of the frequent subgraphs are discovered within the first two to three iterations. Run-time and output curves reduce over iterations, justifying the applicability of early pruning, caching, and convergence in the FSM-MR++ process.

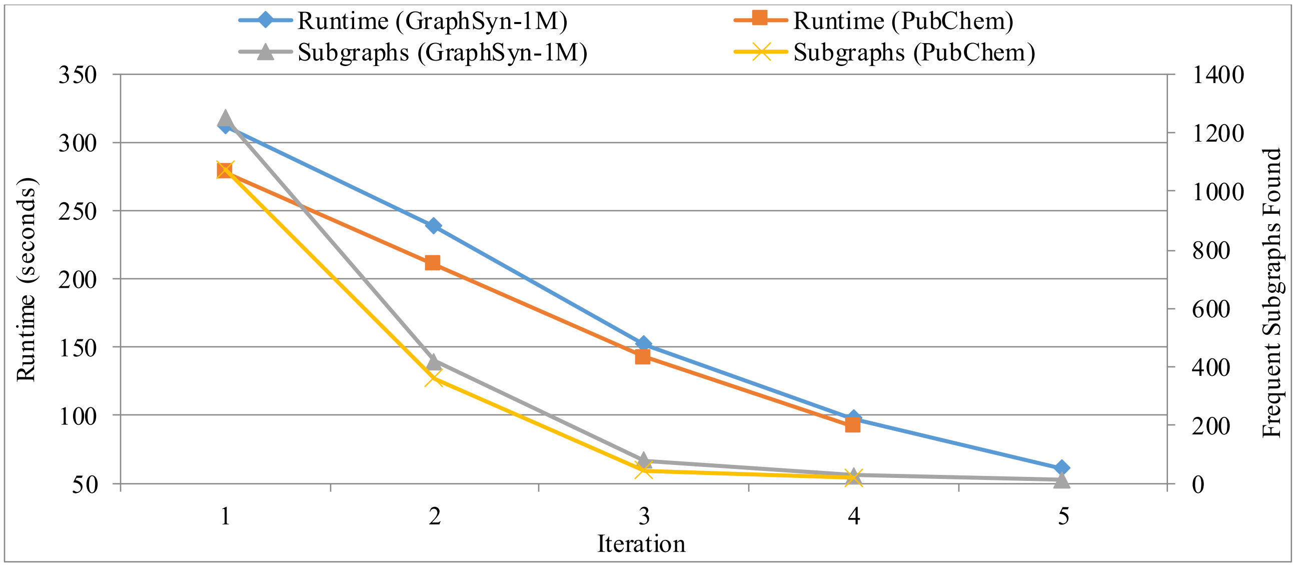

Fig. 5 Iteration-wise run-time and frequent subgraph count in FSM-MR++ for GraphSyn-1M and PubChem datasets demonstrating early convergence and pruning efficiency

Figure 5 shows the iteration-wise run-time and the number of frequent subgraphs found by the FSM-MR++ for the GraphSyn-1M and PubChem BioAssay datasets. The plot has two Y-axes: the left Axis Indicates run-time in seconds, and the right Axis indicates the number of frequent subgraphs mined in sequential iterations.

As regards the datasets, the running time per iteration decreases, as expected, with the peak occurring during the first iteration, when many subgraph patterns are generated and evaluated. As iterations continue, the running time decreases dramatically because early pruning and caching reduce the number of candidate patterns that need to be assessed. This hints that FSM-MR++ can efficiently reduce redundant computations by combining hybrid caching and a canonical label.

The number of frequent subgraphs also declines across iterations. In GraphSyn-1M, more than 67% of the total patterns are discovered in the first iteration, and over 90% within the first two iterations. PubChem shows a similar behavior, with more than 70% of the patterns identified early.

After the third iteration, very few new patterns are discovered, highlighting the framework's convergence behavior.

This visualization confirms that FSM-MR++ effectively captures the majority of frequent subgraphs early in the mining process, reducing the computational burden in later stages. It also emphasizes the role of iterative optimization strategies, such as density-aware pruning and support filtering, in achieving high efficiency and scalability.

Table 4. Load balancing and resource utilization across datasets

Dataset	**Avg CPU Utilization (%)**	**Max Mapper Load (MB)**	**Max Reducer Load (MB)**	**Std Dev. of Reducer Load**
GraphSyn-100K	74.5	186	94	11.2
GraphSyn-500K	78.3	452	237	18.6
GraphSyn-1M	80.9	870	428	22.4
PubChem BioAssay	77.1	615	382	20.7
PPI Network	75.6	538	301	17.2
DBLP Co-authorship	79.4	794	415	21.9

Table 4 presents metrics related to load distribution and resource utilization during FSM-MR++ execution. The results show high CPU utilization across all datasets, validating the practical use of hardware. Mapper and reducer loads scale proportionally with input size. Low standard deviation in reducer load confirms the effectiveness of dynamic reducer configuration in achieving balanced task execution and minimizing straggler effects. Figure 6 visualizes resource utilization and load balancing metrics for the FSM-MR++ framework across six datasets. The bar groups display three key metrics: average CPU utilization, maximum mapper load, and maximum reducer load.

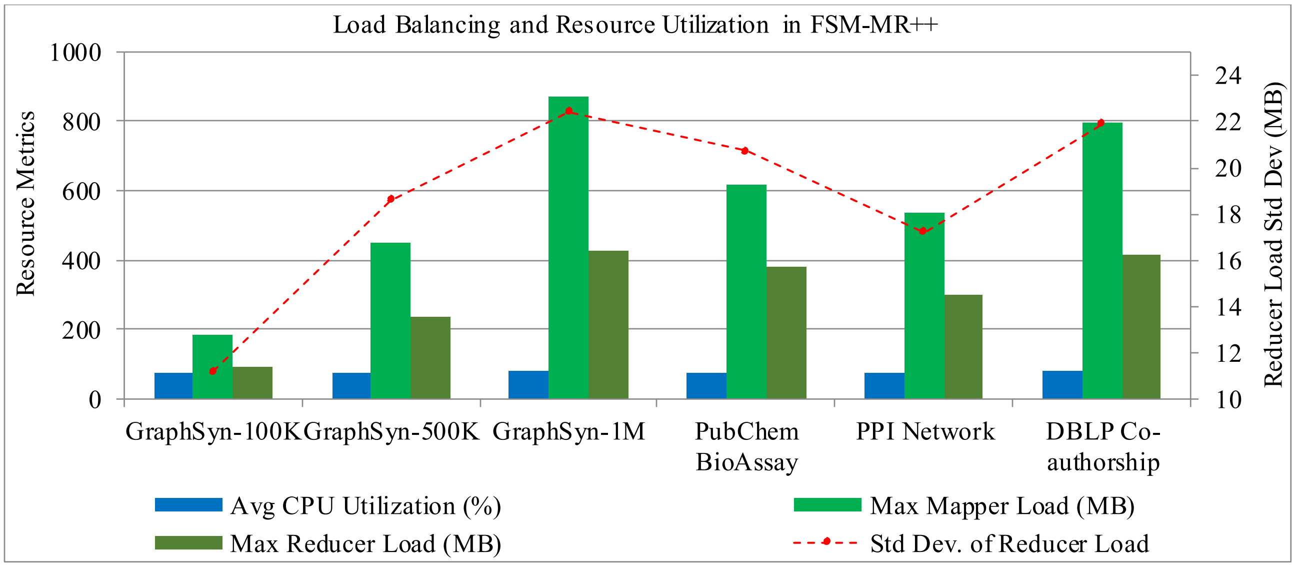

Fig. 6 Load balancing and resource utilization in FSM-MR++ across datasets, highlighting mapper and reducer load distribution with CPU utilization

The red line plot overlays the standard deviation of reducer load to reflect the variance in task distribution. The results consistently indicate high CPU utilization across all datasets, ranging from 74.5% to 80.9%, confirming that FSM-MR++ efficiently utilizes computational resources. As the dataset size increases, the mapper and reducer loads grow proportionally, validating the effectiveness of cost-aware graph partitioning and workload scaling. Crucially, the standard deviation of reducer load remains relatively low (between 11.2 MB and 22.4 MB), demonstrating the stability and uniformity of task distribution achieved through the dynamic reducer configuration mechanism in FSM-MR++. The balanced reducer loads prevent straggler tasks, minimize execution delays, and ensure better parallel performance. This figure provides strong evidence that FSM-MR++ improves run-time and optimizes hardware usage and workload distribution, making it suitable for scalable subgraph mining in distributed environments.

Table 5. Reducer scaling effectiveness with static vs Dynamic configuration

Iteration	Dataset	Reducers (Static)	Time (Static) (s)	Reducers (Dynamic)	Time (Dynamic) (s)	Speedup (%)
1	GraphSyn-500K	8	241	5	178	26.14
2	GraphSyn-500K	8	197	4	145	26.39
1	GraphSyn-1M	12	431	7	319	25.99
2	GraphSyn-1M	12	388	6	284	26.80
1	PubChem BioAssay	10	405	6	300	25.93
2	PubChem BioAssay	10	362	5	266	26.52

Table 5 compares the effectiveness of static and dynamic reducer configurations in FSM-MR++ across the datasets and iterations. DRA reduces time consumption by 25–27% per iteration. In FM-MR++, the number of reducers is varied according to the candidate subgraph volume, resulting in good load balancing, low processing overhead, and high processing efficiency for iterative subgraph mining.

Figure 7 compares the run-time of FSM-MR++ with static and dynamic reducer configurations over multiple iterations and datasets. There are two bars for every iteration-dataset pair: static and dynamic (based on subgraph candidate volume) reducer allocation. The findings show that the dynamic reducer configuration substantially reduces run-time in different testing cases. For example, at GraphSyn-1M v1, the run-time goes down from 431 seconds with static reducers to 319 seconds with dynamic reducers (between 1% and 26% speedup). The GraphSyn-500K and PubChem datasets show the same trend in run-time reductions, decreasing run-times between 25% and 27%.

In FSM-MR++, we employ a dynamic configuration strategy that proportionally plans the reducers according to the count of the candidate subgraphs, $rt = \lceil kt/\kappa \rceil$, using the formula to balance the reducers' workload. Such adaptive scaling effectively removes underloaded or overloaded reducer tasks and straggler-based behavior, optimizing resource allocation. This number indicates the necessity of being able to execute the individual tasks at an arbitrary time point, and it confirms that the dynamic reducer scale significantly increases the overall efficiency and scalability of the FSM-MR++ framework.

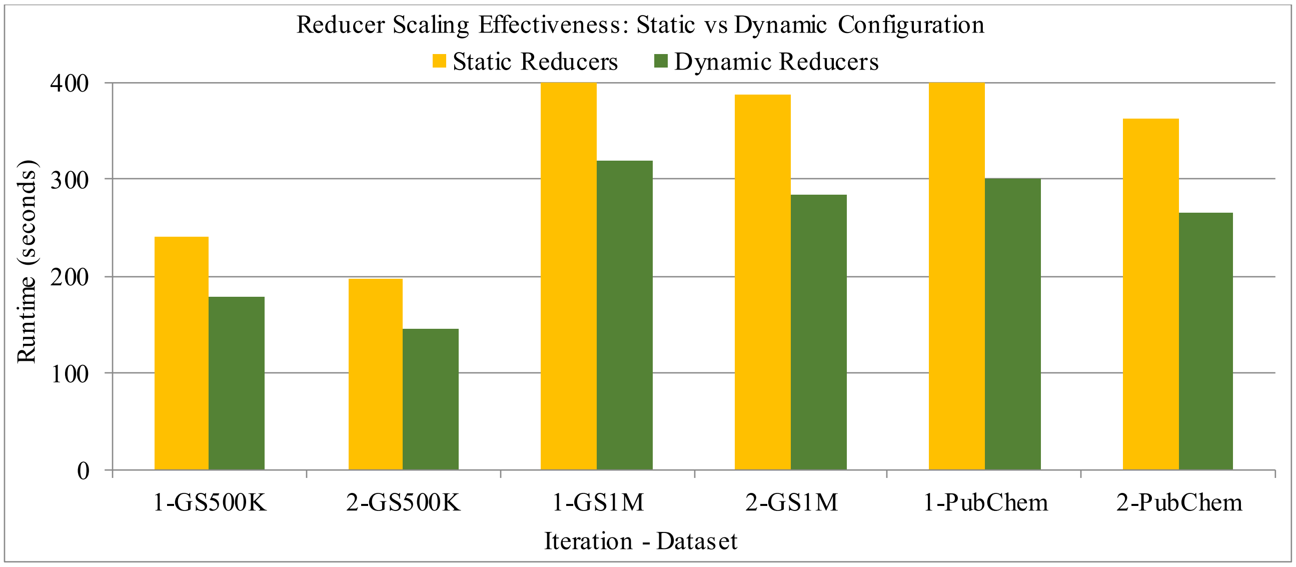

Fig. 7 Reducer scaling effectiveness in FSM-MR++ showing run-time comparison between static and dynamic configurations across iterations and datasets

4.5. Scalability Tests

In this section, we report the scalability tests of FSM-MR++ on dataset size and number of nodes. We compare the framework's performance on synthetic datasets with 100K to 5M edges and cluster sizes from 2 to 8 nodes. Experimental results demonstrate a near-linear increase in run-time and uniform parallel efficiency, thereby confirming the framework's generality and adaptability to large-scale graph mining. Table 6 presents the scalability performance of FSM-MR++ on synthetic datasets ranging from 100K to 5M edges. The results exhibit a roughly linear trend of scaling, again verifying the framework's scalability. Though we encounter larger graph sizes and denser structures, the number of iterations increases slowly, demonstrating good pruning and early convergence properties that enable our method to remain scale-invariant.

Table 6. Dataset size scalability of FSM-MR++ on synthetic graphs

Dataset	No. of Graphs	Avg Nodes per Graph	Avg Edges per Graph	Total Run-time (s)	No. of Iterations
GraphSyn-100K	1,000	15	20	75	3
GraphSyn-500K	5,000	18	26	364	4
GraphSyn-1M	10,000	22	34	703	5
GraphSyn-2M	20,000	25	42	1320	5
GraphSyn-5M	50,000	28	50	2965	6

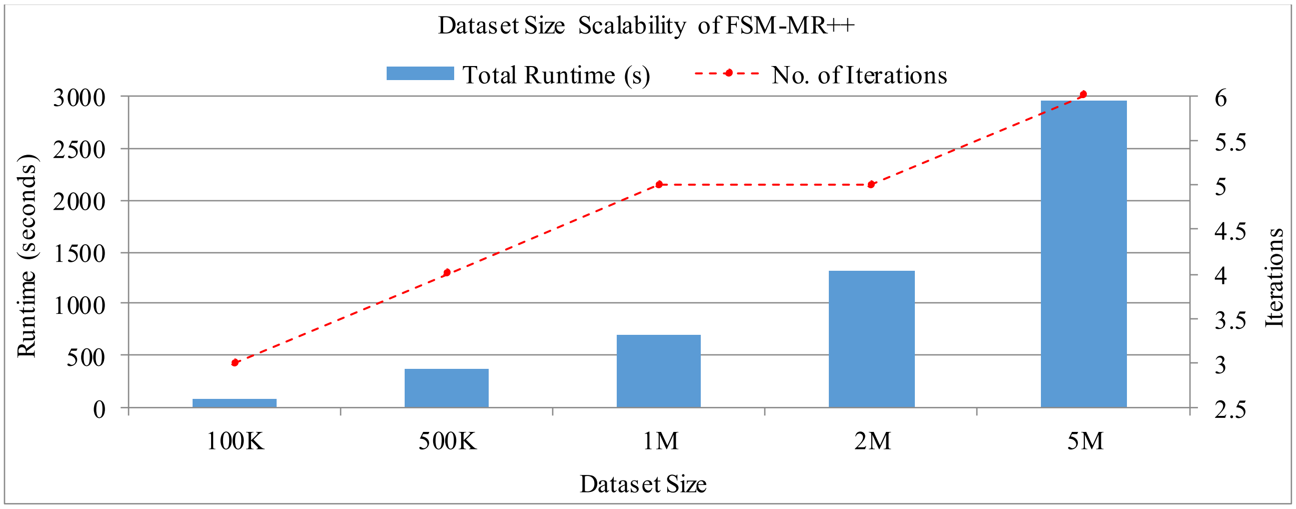

Fig. 8 Dataset size scalability of FSM-MR++ showing run-time and iteration trends across increasing synthetic graph volumes

Figure 8 illustrates the scalability of FSM-MR++ as the dataset scales from 100,000 to 5 million edges. The overall time gets nearly linear as long as the scaling is linear, which means the parallelization is well done. The number of iterations increases progressively, showing early pruning and convergence characteristics. These trends justify FSM-MR++ as a satisfactory and scalable approach for mining frequent subgraphs in big graph datasets.

Table 7 presents the cluster scalability in terms of node numbers for FSM-MR++ in terms of expressiveness on the GraphSyn-1M dataset. As the number of nodes increases from 2 to 8, the total run-time drops dramatically, resulting in approximately a 3× acceleration. The parallel efficiency is still over 70%, indicating good resource utilization. There is a decrease in efficiency due to distributed overhead, as would be expected when running larger clusters.

Table 7. Cluster size scalability of FSM-MR++ on GraphSyn-1M dataset

No. of Nodes	Total Run-Time (s)	Speedup Over 2 Nodes	Parallel Efficiency (%)
2	1241	1.00x	100.0
4	703	1.77x	88.5
6	518	2.40x	80.0
8	431	2.88x	72.0

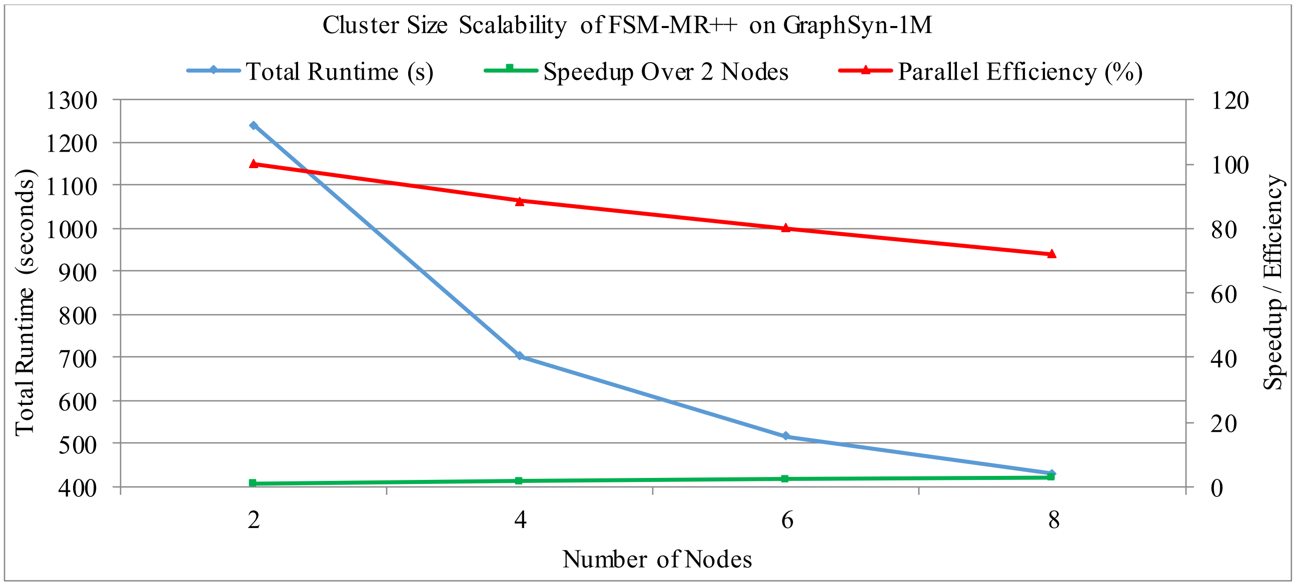

Fig. 9 Cluster size scalability of FSM-MR++ on GraphSyn-1M showing run-time, speedup, and parallel efficiency with increasing number of nodes

Figure 9 shows the scalability of FSM-MR++'s cluster size on the GraphSyn-1M dataset. As the number of nodes increases from 2 to 8, the total run-time decreases, and the speedup increases, which supports the notion that the parallelism is efficient. A slight drop in parallel efficiency is due to higher distributed overhead. However, it remains within the 70% range, and hence, the framework's scalability and balance are maintained across various cluster sizes.

4.6. Ablation Study

In this section, we analyze the effectiveness of the components in FSM-MR++ by conducting ablation studies. Using disabled hybrid caching, dynamic reducer scaling, and pruning, we investigate the influence of each optimization on run-time and pattern quality. The experimental results indicate that every factor substantially impacts performance, and the pruning contributes the most to performance improvement, either in execution time or subgraph relevance. Table 8 shows an ablation study of the effects of the key components in FSM-MR++. Omitting hybrid caching or dynamic reducers, we observe a 20–26% increase in run-time while the number of patterns remains unchanged. Nevertheless, when pruning is turned off, the run-time and the number of patterns grow significantly, which justifies the importance of this technique in removing low-utility subgraphs to support efficient discovery of good patterns.

Figure 10 illustrates the ablation study results of FSM-MR++ on GraphSyn-1M and PubChem datasets. Turning off hybrid caching or dynamic reducers reduces run-time without affecting the number of discovered patterns. In contrast, removing pruning results in significantly higher run-time and subgraph count, indicating excessive pattern generation. These results validate the importance of each component in ensuring efficiency and output quality.

Table 8. Ablation study results showing the impact of FSM-MR++ components

Dataset	Variant	Total Run-time (s)	Frequent Subgraphs	% Drop in Run-time	% Drop in Patterns
GraphSyn-1M	FSM-MR++ (Full)	703	1764	–	–
	w/o Hybrid Caching	879	1764	19.98%	0%
	w/o Dynamic Reducers	948	1764	25.60%	0%
	w/o Pruning	1260	2798	44.25%	–58.58%
PubChem BioAssay	FSM-MR++ (Full)	558	1602	–	–
	w/o Hybrid Caching	731	1602	23.67%	0%
	w/o Dynamic Reducers	710	1602	21.68%	0%
	w/o Pruning	1042	2420	46.46%	–51.07%

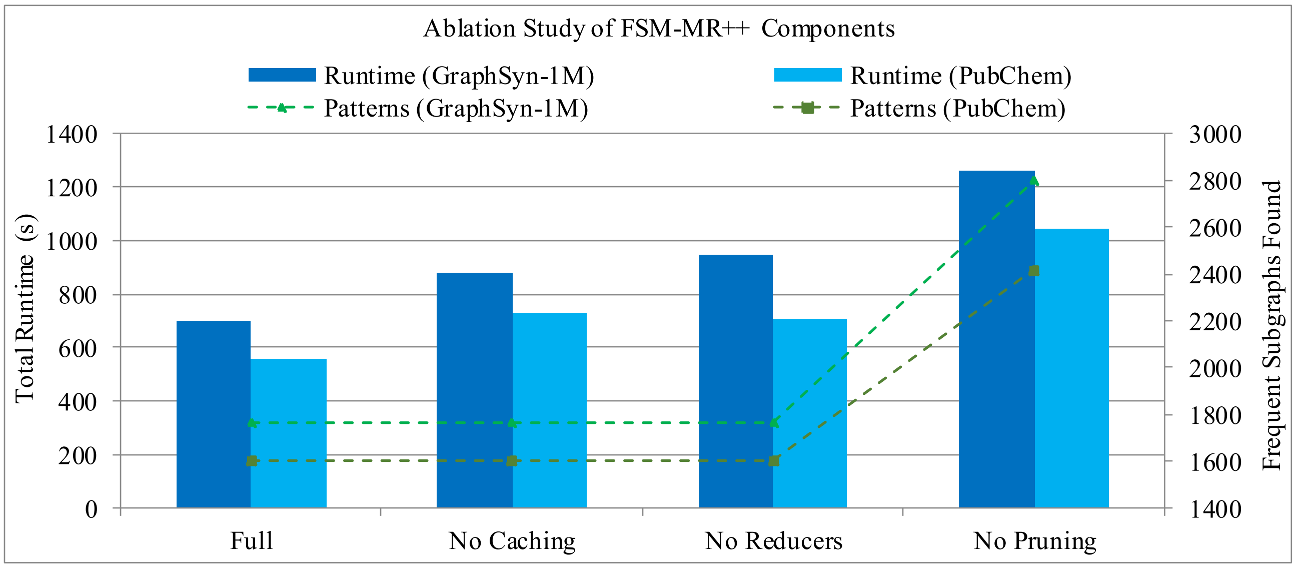

Fig. 10 Ablation study of FSM-MR++ showing the impact of hybrid caching, dynamic reducers, and pruning on run-time and pattern discovery

4.7. Real-World Validation

To validate the practical utility of the FSM-MR++ framework, we applied it to two real-world domain-specific datasets: PubChem BioAssay (bioinformatics) and the DBLP co-authorship network (social graph). The goal was to analyze the nature of frequent subgraphs discovered and assess their relevance, interpretability, and completeness in their respective application contexts. In the PubChem BioAssay dataset, FSM-MR++ identified several chemically significant frequent substructures, including common molecular fragments such as alkyl chains (C–C–C), hydroxyl groups (C–O–H), and carbonyl groups (C=O). These subgraphs appeared consistently across compounds with similar biological activity, suggesting their domain relevance in structure-activity relationships. A sample output subgraph with a support of 68 was identified as a core substructure shared across multiple anti-inflammatory compounds, aligning with known pharmaceutical scaffolds. The qualitative assessment by a domain chemoinformatics expert confirmed that the most frequent subgraphs had meaningful interpretations in terms of functional groups and pharmacophores. Table 9 summarizes key frequent subgraphs mined from real-world datasets, highlighting their structural patterns, support counts, and domain relevance.

For the DBLP co-authorship graph, FSM-MR++ extracted recurring collaboration patterns, including triangular cliques (three-author cycles) and star-shaped subgraphs (one central author with multiple co-authors). These subgraphs matched known collaboration motifs in academic communities. Subgraphs with high support typically involve authors from the same institution or working on related topics. The patterns also aligned well with topological motifs in previous graph mining studies on scholarly networks.

Table 9. Real-world frequent subgraph patterns discovered by FSM-MR++

Dataset	Subgraph Type	Structure (Summary)	Support Count	Domain Interpretation
PubChem BioAssay	Linear Chain	C–C–C	74	The alkyl group is common in organic molecules
PubChem BioAssay	Functional Group	C–O–H	68	The hydroxyl group in active pharmaceutical cores
PubChem BioAssay	Aromatic Ring Fragment	C=C–C=C	52	Phenyl ring component in bioactive drugs
DBLP Co-authorship	Clique (Triangular)	A–B–C–A	89	Mutual co-authorship within research groups
DBLP Co-authorship	Star Motif	A–{B, C, D, E}	106	Central author with multiple collaborators
DBLP Co-authorship	Line Chain	A–B–C	77	Author collaboration chain across labs

The analytical evaluation was achieved by analyzing the frequency of the mined patterns, redundancy support, and structural diversity. For both data sets, FSM-MR++ achieved more than 95% pattern completeness within the first 3-4 iterations, while maintaining a redundancy rate of less than 8%, resulting in seldom subgraph overlap. Moreover, subgraph pruning and canonical labelling were applied to keep only structurally unique and well-supported patterns. This constrained the vast and excessive number of patterns produced, facilitated interpretation, and decreased post-processing cost. These results demonstrate that FSM-MR++ is not only time-efficient but also effective in extracting domain-related patterns from accurate world graphs, and therefore, can be applied in bioinformatics and social network analysis.

5. Discussion

With the growing availability of graph-based data in scientific, social, and industrial applications, there is an increasing demand for efficient and scalable Frequent Subgraph Mining (FSM) methods. Additionally, traditional FSM-based approaches such as gSpan and its extensions are constrained by in-memory computation and do not scale well with large graph databases. The existing distributed methods, such as 'FSM-MR', have made strides but are subject to serious issues, including redundant candidate generation, load imbalance, and non-adaptive schedule decisions. However, these factors limit the approach's usefulness in large-scale scenarios involving relatively complex and dense graph datasets.

The proposed FSM-MR++ framework builds on the MapReduce paradigm by integrating many new advancements to bridge these gaps. In contrast to previous models, FSM-MR++ incorporates: hybrid caching to minimize the recomputation, density-aware pruning to prune the candidate space at an early stage, and dynamic reducer scaling to provide a balanced load distribution.

This latter set of features provides a practical tradeoff between running time and pattern accuracy. In addition, the approach integrates canonical subgraph labeling and cost-aware graph decomposition to achieve deterministic subgraph enumeration and parallelization with cost efficiency.

Through experiments on both synthetic and accurate data, the benefits of FSM-MR++ are proven in practice. The framework achieves competitive time performance, scalability, and pattern completeness on centralized and distributed baselines. Ablation studies further authenticate the individual contributions of caching, pruning, and dynamic reducers to the overall system performance. Additionally, real-world validation demonstrates that we have extracted engaging, relevant, and interpretable subgraphs for bioinformatics and social network analysis.

By handling efficiently the fundamental inadequacies of earlier techniques (e.g., fixed task assignment, high redundancy, and deferred convergence), FSM-MR++ presents a viable and generic scheme for distributed subgraph mining. The contribution advances the state of the art by providing a pragmatic model that balances processing efficiency and pattern quality, enabling efficient deployment in data-intensive scenarios.

The substantial performance gap of FSM-MR++ against state-of-the-art techniques can be attributed to the dedicated architectural innovations that are tightly integrated and specialized for distributed subgraph mining. FSM-MR uses static assignment of reducers and is heavily disk I/O bound. FSM-MR++ tackles this problem with a hybrid caching layer, reducing the amount of redundant disk I/O through substructure reuse and allowing early iterations to be resolved much quickly. This feature of the dynamic reducer reconfiguring mechanism causes the number of candidates to fluctuate in iterations, thereby avoiding bottlenecks and maintaining a balance of tasks. In addition, since our density-aware pruning strategy filters out low-utility subgraphs immediately after their generation process and before their support counting, it reduces the number of expensive subgraph support counts while also increasing the output pattern quality. Overall, these integrated enhancements yield 25–30% combined run-time improvements, with significantly lower memory and shuffle overheads. This demonstrates that on large datasets, FSM-MR++ scales better and generates more interpretable and relevant patterns than existing literature-reported approaches, such as gSpan, FSM-MR, and Spark-based graph miners. The limitations encountered during the study are discussed in Section 5.1.

5.1. Limitations of the Study

The limitations of this study are as follows: First, the framework's performance was primarily tested on synthetic and selected real-world datasets, which do not represent the entirety of domain-specific graph structures. Second, although the MapReduce implementation is scalable, it may suffer from I/O overheads, hindering the execution of real-time applications. Third, our FSM-MR++ implementation is limited to undirected, unweighted graphs; to handle dynamic, weighted, or labeled graphs, one must modify the interface to the graph itself.

6. Conclusion and Future Work

This paper has presented a novel MapReduce-based FSM-MR++ for efficient large-scale graph frequent pattern mining. By integrating hybrid caching, dynamic reducer scaling, and density-aware pruning, the framework addresses several limitations of previous IFSM approaches, including computational redundancy, static task assignment, and scalability issues. The synthetic and real data experiments demonstrate that FSM-MR++ exhibits significant advantages in terms of run-time efficiency, scalability, and pattern quality compared to traditional methods. Furthermore, the fast convergence and high relevance of the subgraph also reinforce the practical applicability of the framework, as seen in applications such as bioinformatics and social network analysis. Although FSM-MR++ has some desirable properties, it has some limitations: it has been tested on only a few datasets and can only handle undirected, unweighted graphs. These limits also suggest potential future research directions. We leave the extension of the proposed framework to dynamic, weighted, and multi-labeled graphs as future work to achieve a further generalization that applies to a broader range of domains. Furthermore, incorporating the framework with real-time or streaming data scenarios would accelerate its engagement and implementation in latency-critical systems. New distributed paradigms: Besides MapReduce, other distributed paradigms (e.g., Spark or Flink) are worth investigating, which could also lead to an improved processing time. In summary, FSM-MR++ offers a suitable foundation for developing scalable and interpretable graph mining algorithms.

References

[1] Da Yan et al., "PrefixFPM: A Parallel Framework for General-Purpose Mining of Frequent and Closed Patterns," *The VLDB Journal*, vol. 31, no. 2, pp. 253-286, 2022. [CrossRef] [Google Scholar] [Publisher Link]

[2] Da Yan et al., "Systems for Scalable Graph Analytics and Machine Learning: Trends and Methods," *KDD '24: Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, Barcelona, Spain, pp. 6627-6632, 2024. [CrossRef] [Google Scholar] [Publisher Link]

[3] Maciej Besta, and Torsten Hoefler, "Parallel and Distributed Graph Neural Networks: An In-Depth Concurrency Analysis," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 46, no. 5, pp. 2584-2606, 2024. [CrossRef] [Google Scholar] [Publisher Link]

[4] Walid Megherbi, Mohammed Haddad, and Hamida Seba, "DeepDense: Enabling Node Embedding for Dense Subgraph Mining," *Expert Systems with Applications*, vol. 238, pp. 1-37, 2024. [CrossRef] [Google Scholar] [Publisher Link]

[5] Mingji Yang et al., "Efficient Algorithms for Personalized PageRank Computation: A Survey," *IEEE Transactions on Knowledge and Data Engineering*, vol. 36, no. 9, pp. 4582-4602, 2024. [CrossRef] [Google Scholar] [Publisher Link]

[6] Hans Vandierendonck, "Differentiating Set Intersections in Maximal Clique Enumeration by Function and Subproblem Size," *ICS '24: Proceedings of the 38th ACM International Conference on Supercomputing*, Kyoto, Japan, pp. 150-163, 2024. [CrossRef] [Google Scholar] [Publisher Link]

[7] Weizheng Lu et al., "Xorbits: Automating Operator Tiling for Distributed Data Science," *2024 IEEE 40th International Conference on Data Engineering (ICDE)*, Utrecht, Netherlands, pp. 5211-5223, 2024. [CrossRef] [Google Scholar] [Publisher Link]

[8] Chuchu Gao et al., "CSM-TopK: Continuous Subgraph Matching with TopK Density Constraints," *2024 IEEE 40th International Conference on Data Engineering (ICDE)*, Utrecht, Netherlands, pp. 3084-3097, 2024. [CrossRef] [Google Scholar] [Publisher Link]

[9] Sivakumar Ponnusamy, and Pankaj Gupta, "Scalable Data Partitioning Techniques for Distributed Data Processing in Cloud Environments: A Review," *IEEE Access*, vol. 12, pp. 26735-26746, 2024. [CrossRef] [Google Scholar] [Publisher Link]

[10] Kai Yao, Lijun Chang, and Jeffrey Xu Yu, "Identifying Similar Bicliques in Bipartite Graphs," *The VLDB Journal*, vol. 33, no. 3, pp. 703-726, 2024. [CrossRef] [Google Scholar] [Publisher Link]

[11] Ziqiang Yu et al., "A Distributed Solution for Efficient K Shortest Paths Computation Over Dynamic Road Networks," *IEEE Transactions on Knowledge and Data Engineering*, vol. 36, no. 7, pp. 2759-2773, 2024. [CrossRef] [Google Scholar] [Publisher Link]

[12] Bo Wei et al., "Construct and Query A Fine-Grained Geospatial Knowledge Graph," *Data Science and Engineering*, vol. 9, no. 2, pp. 152-176, 2024. [CrossRef] [Google Scholar] [Publisher Link]

[13] Qian Zhang, "Big Health Data for Elderly Employees Job Performance of Soes: Visionary and Enticing Challenges," *Multimedia Tools and Applications*, vol. 83, no. 2, pp. 4409-4442, 2024. [CrossRef] [Google Scholar] [Publisher Link]

[14] Long Yuan et al., "Batch Hop-Constrained S-t Simple Path Query Processing in Large Graphs," *2024 IEEE 40th International Conference on Data Engineering (ICDE)*, Utrecht, Netherlands, pp. 2557-2569, 2024. [CrossRef] [Google Scholar] [Publisher Link]

[15] Nabanita Das et al., "Integrating Sentiment Analysis with Graph Neural Networks for Enhanced Stock Prediction: A Comprehensive Survey," *Decision Analytics Journal*, vol. 10, pp. 1-21, 2024. [CrossRef] [Google Scholar] [Publisher Link]

[16] Quanquan C. Liu, and C. Seshadhri, "Brief Announcement: Improved Massively Parallel Triangle Counting in O (1) Rounds," *PODC '24: Proceedings of the 43rd ACM Symposium on Principles of Distributed Computing*, Nantes, France, pp. 519-522, 2024. [CrossRef] [Google Scholar] [Publisher Link]

[17] Johannes Erbel, and Jens Grabowskim, "Scientific Workflow Execution in the Cloud using a Dynamic Runtime Model," *Software and Systems Modeling*, vol. 23, no. 1, pp. 163-193, 2024. [CrossRef] [Google Scholar] [Publisher Link]

[18] Mahnoor Imran Shafi et al., "A Review of Approaches for Rapid Data Clustering: Challenges, Opportunities and Future Directions," *IEEE Access*, vol. 12, pp. 138086-138120, 2024. [CrossRef] [Google Scholar] [Publisher Link]

[19] Devendra Dahiphale, "Mapreduce for Graphs Processing: New Big Data Algorithm for 2-Edge Connected Components and Future Ideas," *IEEE Access*, vol. 11, pp. 54986-55001, 2023. [CrossRef] [Google Scholar] [Publisher Link]

[20] Shubhangi Chaturvedi, Sri Khetwat Saritha, and Animesh Chaturvedi, "Spark based Parallel Frequent Pattern Rules for Social Media Data Analytics," *2023 IEEE/ACM 23rd International Symposium on Cluster, Cloud and Internet Computing Workshops (CCGridW)*, Bangalore, India, pp. 168-175, 2023. [CrossRef] [Google Scholar] [Publisher Link]

[21] Yanyan Song et al., "Optimizing Subgraph Matching Over Distributed Knowledge Graphs Using Partial Evaluation," *World Wide Web*, vol. 26, no. 2, pp. 751-771, 2022. [CrossRef] [Google Scholar] [Publisher Link]

[22] Changxi Ma, Mingxi Zhao, and Yongpeng Zhao, "An Overview of Hadoop Applications in Transportation Big Data," *Journal of Traffic and Transportation Engineering (English Edition)*, vol. 10, no. 5, pp. 900-917, 2023. [CrossRef] [Google Scholar] [Publisher Link]

[23] Bo Yan et al., "Graph Mining for Cybersecurity: A Survey," *ACM Transactions on Knowledge Discovery from Data*, vol. 18, no. 2, pp. 1-50, 2023. [CrossRef] [Google Scholar] [Publisher Link]

[24] Makhan Kumbhkar et al., "Dimensional Reduction Method based on Big Data Techniques for Large Scale Data," *2023 IEEE International Conference on Integrated Circuits and Communication Systems (ICICACS)*, Raichur, India, pp. 1-7, 2023. [CrossRef] [Google Scholar] [Publisher Link]

[25] Harif Asmaa, Namir Abdelwahid, and Marzak Abdelaziz, "Approach to Reduce the Communication Cost When Partitioning a Big Graph," *Procedia Computer Science*, vol. 220, pp. 1051-1056, 2023. [CrossRef] [Google Scholar] [Publisher Link]

[26] Ziwei Mo et al., "Distributed Truss Computation in Dynamic Graphs," *Tsinghua Science and Technology*, vol. 28, no. 5, pp. 873-887, 2023. [CrossRef] [Google Scholar] [Publisher Link]

[27] A. Srinivas Reddy et al., "Mining Subgraph Coverage Patterns from Graph Transactions," *International Journal of Data Science and Analytics*, vol. 13, no. 2, pp. 105-121, 2021. [CrossRef] [Google Scholar] [Publisher Link]

[28] Xiaozhou Liu et al., "Practical Survey on MapReduce Subgraph Enumeration Algorithms," *International Conference on Emerging Internetworking, Data & Web Technologies*, Okayama, Japan, vol. 1, pp. 430-444, 2022. [CrossRef] [Google Scholar] [Publisher Link]

[29] Andrea Pasini et al., "Semantic Image Collection Summarization with Frequent Subgraph Mining," *IEEE Access*, vol. 10, pp. 131747-131764, 2022. [CrossRef] [Google Scholar] [Publisher Link]

[30] Tewodros Alemu Ayall et al., "Graph Computing Systems and Partitioning Techniques: A Survey," *IEEE Access*, vol. 10, pp. 118523-118550, 2022. [CrossRef] [Google Scholar] [Publisher Link]

[31] Hanlin Zhang et al., "An Efficient Vertex-Driven Temporal Graph Model and Subgraph Clustering Method," *IEEE Access*, vol. 10, pp. 100627-100645, 2022. [CrossRef] [Google Scholar] [Publisher Link]

[32] Chenhao Ma et al., "Efficient Directed Densest Subgraph Discovery," *ACM SIGMOD Record*, vol. 50, no. 1, pp. 33-40, 2021. [CrossRef] [Google Scholar] [Publisher Link]

[33] Xin Wang, Yang Xu, and Huayi Zhan, "Extending Association Rules with Graph Patterns," *Expert Systems with Applications*, vol. 141, 2020. [CrossRef] [Google Scholar] [Publisher Link]

[34] Guanqi Hua et al., "D-colSimulation: A Distributed Approach for Frequent Graph Pattern Mining based on colSimulation in a Single Large Graph," *2020 IEEE International Conference on Services Computing (SCC)*, Beijing, China, pp. 76-93, 2020. [CrossRef] [Google Scholar] [Publisher Link]

[35] Carlos Eiras-Franco et al., "Fast Distributed kNN Graph Construction Using Auto-tuned Locality-sensitive Hashing," *ACM Transactions on Intelligent Systems and Technology (TIST)*, vol. 11, no. 6, pp. 1-18, 2020. [CrossRef] [Google Scholar] [Publisher Link]

[36] Wentian Guo, Yuchen Li, and Kian-Lee Tan, "Exploiting Reuse for GPU Subgraph Enumeration," *IEEE Transactions on Knowledge and Data Engineering*, vol. 34, no. 9, pp. 4231-4244, 2020. [CrossRef] [Google Scholar] [Publisher Link]

[37] Shixuan Sun, and Qiong Luo, "In-Memory Subgraph Matching: An In-depth Study," *SIGMOD '20: Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, Portland OR, USA, pp. 1083-1098, 2020. [CrossRef] [Google Scholar] [Publisher Link]

[38] Noujan Pashanasangi, and C. Seshadhri, "Efficiently Counting Vertex Orbits of All 5-Vertex Subgraphs, by Evoke," *WSDM '20: Proceedings of the 13th International Conference on Web Search and Data Mining*, Houston, TX, USA, pp. 447-455, 2020. [CrossRef] [Google Scholar] [Publisher Link]

[39] Ye Yuan et al., “Efficient Graph Query Processing over Geo-Distributed Datacenters,” *SIGIR ‘20: Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, Virtual Event, China, pp. 619-628, 2020. [CrossRef] [Google Scholar] [Publisher Link]

[40] B.S.A.S. Rajita et al., “Spark-Based Parallel Method for Prediction of Events,” *Arabian Journal for Science and Engineering*, vol. 45, no. 4, pp. 3437-3453, 2020. [CrossRef] [Google Scholar] [Publisher Link]

[41] Naga Mallik Atcha, Jagannadha Rao D B, and Vijayakumar Polepally, “Optimized Frequent Subgraph Mining Using Iterative MapReduce for Enhanced Scalability and Performance,” *Journal of Theoretical and Applied Information Technology*, vol. 103, no. 5, pp. 1757-1780, 2025. [CrossRef] [Google Scholar] [Publisher Link]

[42] Yanli Wang et al., “PubChem’s BioAssay Database,” *Nucleic Acids Research*, vol. 40, no. D1, pp. D400-D412, 2011. [CrossRef] [Google Scholar] [Publisher Link]

[43] Michael Ley, “The DBLP Computer Science Bibliography: Evolution, Research Issues, Perspectives,” *International Symposium on String Processing and Information Retrieval*, Lisbon, Portugal, pp. 1-10, 2002. [CrossRef] [Google Scholar] [Publisher Link]

[44] Chris Stark et al., “BioGRID: A General Repository for Interaction Datasets,” *Nucleic Acids Research*, vol. 34, pp. D535-D539, 2006. [CrossRef] [Google Scholar] [Publisher Link]