applied sciences

MDPI

Article

FSM-BC-BSP: Frequent Subgraph Mining Algorithm Based on BC-BSP

Fangling Leng *, Fan Li, Yubin Bao, Tiancheng Zhang and Ge Yu

School of Computer Science and Engineering, Northeastern University, Shenyang 110169, China; 2301892@stu.neu.edu.cn (F.L.); baoyb@mail.neu.edu.cn (Y.B.); tczhang@mail.neu.edu.cn (T.Z.); yuge@mail.neu.edu.cn (G.Y.)
* Correspondence: lengfangling@mail.neu.edu.cn

Abstract: As graph models become increasingly prevalent in the processing of scientific data, the exploration of effective methods for the mining of meaningful patterns from large-scale graphs has garnered significant research attention. This paper delves into the complexity of frequent subgraph mining and proposes a frequent subgraph mining (FSM) algorithm. This FSM algorithm is developed within a distributed graph iterative system, designed for the Big Cloud (BC) environment of the China Mobile Corp., and is based on the bulk synchronous parallel (BSP) model, named FSM-BC-BSP. Its aim is to address the challenge of mining frequent subgraphs within a single, large graph. This study advocates for the incorporation of a message sending and receiving mechanism to facilitate data sharing across various stages of the frequent subgraph mining algorithm. Additionally, it suggests employing a standard coded subgraph and sending it to the same node for global support calculation on the large graph. The adoption of the rightmost path expansion strategy in generating candidate subgraphs helps to mitigate the occurrence of redundant subgraphs. The use of standard coding ensures the unique identification of subgraphs, thus eliminating the need for isomorphism calculations. Support calculation is executed using the Minimum Image (MNI) measurement method, aligning with the downward closure attribute. The experimental results demonstrate the robust performance of the FSM-BC-BSP algorithm across diverse input datasets and parameter configurations. Notably, the algorithm exhibits exceptional efficacy, particularly in scenarios with low support requirements, showcasing its superior performance under such conditions.

Keywords: BC-BSP; frequent subgraph; graph division; parallel algorithm

Citation: Leng, F.; Li, F.; Bao, Y.; Zhang, T.; Yu, G. FSM-BC-BSP: Frequent Subgraph Mining Algorithm Based on BC-BSP. *Appl. Sci.* **2024**, *14*, 3154. https://doi.org/10.3390/app14083154

Academic Editor: Chilukuri K. Mohan

Received: 9 March 2024
Revised: 4 April 2024
Accepted: 4 April 2024
Published: 9 April 2024

1. Introduction

In recent years, the pervasive influence of "big data", propelled by the advancement of information technology, has permeated diverse fields, including but not limited to data mining [1–3], computer biology [4], environmental science [5], e-commerce [6], and social network analysis [7]. Within these domains, the analysis and extraction of concealed information from extensive datasets have become standard practices, often approached from innovative perspectives [8–10]. As a general data structure, the graph has found widespread application across the aforementioned fields. Simultaneously, the quest to design efficient graph mining algorithms and extract valuable subgraph patterns from graph data has become a focal point of extensive attention and research.

The challenges associated with frequent subgraph mining in this paper primarily revolve around two key aspects [11,12]. Firstly, the algorithm's complexity [13] poses a significant hurdle. The algorithm's complexity is contingent on two pivotal phases: subgraph isomorphism and candidate pruning. Subgraph isomorphism comparison stands out as the foremost challenge, whereas the effective implementation of candidate pruning holds the potential to substantially reduce the computational load in identifying frequent subgraphs in subsequent stages. Another challenge in frequent subgraph mining stems

from the escalating volume of data within a single graph, exceeding the memory capacity of a single machine [14]. Consequently, the adoption of a distributed graph processing framework becomes imperative for algorithm implementation. Within this implementation process, the integration of the algorithm into the distributed graph processing framework becomes a critical consideration in the realm of frequent subgraph mining.

2. Related Work

Frequent subgraph mining can be broadly categorized into two directions: mining frequent subgraphs within a graph sets and mining frequent subgraphs within a large graph [15–17]. Noteworthy contributions in this domain include DyFSM [18], which introduces the concept of frequent subgraph mining in dynamic databases, and MaNIACS [19], a sampling-based randomized algorithm designed to compute high-quality approximations of subgraph patterns that exhibit frequency within a single, large, vertex-labeled graph. The focus of this paper lies predominantly in the realm of implementing frequent subgraph mining algorithms within a large graph.

2.1. Single Frequent Subgraph Mining Algorithm

The frequent subgraph mining algorithms designed for single-machine environments include SUBDUE [20], VSGRAM [21], gSpan [22], and GRAMI [16]. SUBDUE, an early representative algorithm, addresses the challenge of mining frequent subgraphs within a single, large graph. It achieves graph compression by substituting the vertices of the input graph with frequent patterns. These algorithms employ heuristic search strategies and approximate processing methods. However, it is important to note that these methods may not discover all frequent subgraphs, leading to the loss of many frequent patterns during the mining process. Additionally, their computational load is substantial, making it challenging to scale up to large graph pattern mining. The VSGRAM algorithm introduces a novel pruning strategy designed for processing sparse graphs, but it exhibits limitations in terms of scalability. gSpan employs a minimal Depth First Search (DFS) code for subgraph identification, utilizes the rightmost path for path expansion, and effectively prunes candidate subgraphs. On the other hand, the GRAMI algorithm stands out as the most proficient single-machine frequent subgraph mining algorithm for a single graph. It transforms the frequent subgraph mining problem into a constraint satisfaction problem model, enabling the efficient mining of single graphs with millions of vertices, surpassing the performance of other single-machine frequent subgraph mining algorithms.

2.2. Distributed Frequent Subgraph Mining Algorithm

Distributed frequent subgraph mining algorithms typically fall into two categories [23,24]: those based on MapReduce [25] and those utilizing distributed memory frameworks like Spark [26,27]. Each category presents its own set of advantages and disadvantages. In the context of MapReduce-based algorithms for frequent subgraph mining, one challenge lies in the inability to share data between different jobs due to the nature of Hadoop. Consequently, data sharing between jobs necessitates reading from the Hadoop Distributed File System (HDFS), leading to frequent I/O access during iterative calculations, which can adversely impact the algorithm performance. On the other hand, frequent subgraph mining algorithms based on Spark offer advantages in terms of iterative computation. However, these algorithms may face challenges related to scalability. Notably, FSM-H [28] and MRFSM [29] represent distributed mining algorithms that leverage the iterative MapReduce, effectively combining Hadoop and FSM algorithms. T-FSM [30], employing a unique task-based execution engine design, ensures high concurrency, bounded memory consumption, and effective load balancing.

2.3. BC-BSP Calculation Model

The single-machine algorithm proves to be inefficient for large-scale graphs, primarily due to its difficulty in supporting the mining of frequent subgraphs with low support. Ex-

isting algorithms for the mining of frequent subgraphs from single graphs in a distributed environment fall short in facilitating subgraph pattern growth. Moreover, the frameworks that they rely on, such as Hadoop and the MapReduce computing framework, are poorly suited for the execution of iterative algorithms. MapReduce jobs inherently incur a default startup overhead. In the context of iterative calculations, the inability of different jobs to share data through memory necessitates the initiation of at least one job per iteration, assigning a fixed startup overhead, T, to each. Consequently, for tasks requiring n iterations, the total running time increases by at least $n*T$, a duration that is untenable for time-intensive tasks like frequent subgraph mining. This analysis does not even account for the fact that some tasks cannot be resolved within a single iteration. Additionally, MapReduce lacks the capability for resident tasks and fails to optimize static data efficiently. The Shuffle phase and the need to write data to a disk for inter-job data sharing contribute significantly to the I/O overhead. Furthermore, computational tasks cannot be stored in memory, complicating the optimization of static data across different iterations. When Hadoop executes a task, most map and reduce tasks are performed on separate physical nodes. Often, executing a reduce operation necessitates the retrieval of the results of map tasks from other nodes, leading to repeated read/write operations to a remote HDFS and, consequently, substantial disk I/O and network communication overheads. Moreover, the lack of indexing support in the HDFS exacerbates the challenge of optimizing the data retrieval efficiency. Although some frequent subgraph mining algorithms have been developed on Spark, limitations in Spark's scalability introduce additional bottlenecks in algorithm optimization. To address these issues, this paper introduces a novel frequent subgraph mining algorithm based on the BSP model, effectively overcoming the aforementioned challenges.

The frequent subgraph mining algorithm in this paper is implemented on the BC-BSP system [31]. BC-BSP is an open-source BSP [32] framework that leverages disk-cached data, exhibits robust fault tolerance, and supports multiple data input sources. The utilization of BC-BSP contributes to the efficiency and reliability of the frequent subgraph mining process outlined in this study. Figure 1 illustrates the architecture of BC-BSP. The components of BC-BSP include BSP JobClient, BSP Controller, WorkerManager, and ZooKeeper. The system operates based on the BSP Computing Model. In the BSP framework, computation is divided into a series of phases, each separated by a global synchronization period, denoted as L. These phases are known as super steps. During a super step, each processor is tasked with completing specific local computational operations. Additionally, the system uses a router to facilitate the acceptance and transmission of messages, ensuring that these messages reach the correct working node for the next phase. Subsequently, the system conducts a global check to verify that all processors have completed the current super step. The BSP model promotes a structured and synchronized approach to parallel computing, significantly improving the coordination and communication among processors. Within a super step, each process undergoes three principal operations: local computing, process communication, and barrier synchronization. A super step in the BSP model is constrained to a maximum duration of L time steps. The transition to the next super step is dependent on all processes completing the current one and assembling at the barrier. This global synchronization must be achieved before any process can advance to the next super step, ensuring continued and coordinated execution.

The primary objective of this paper is to address subgraph isomorphism and candidate subgraph growth within the context of frequent subgraph mining.

Contributions. The main contributions of this study are outlined as follows.

Firstly, the paper conducts a comprehensive analysis of the shortcomings associated with frequent subgraph mining algorithms. Subsequently, it proposes the utilization of the BSP model to implement a frequent subgraph mining algorithm tailored to large graphs. This choice aims to enhance the efficiency and scalability of the mining process.

Secondly, the paper introduces a novel frequent subgraph mining algorithm grounded in pattern growth and prior rules. The algorithm is structured into two main stages: the data preparation stage and the mining stage. The mining stage, in turn, is further

divided into the candidate subgraph generation stage and the support calculation stage. This subdivision facilitates the clear delineation of the iterative workflow for each stage, aligning with the super steps in the BC-BSP framework. By leveraging the information from the super steps, the algorithm's different iterative steps can be appropriately calculated in their corresponding super steps, enhancing the overall efficiency of the mining process.

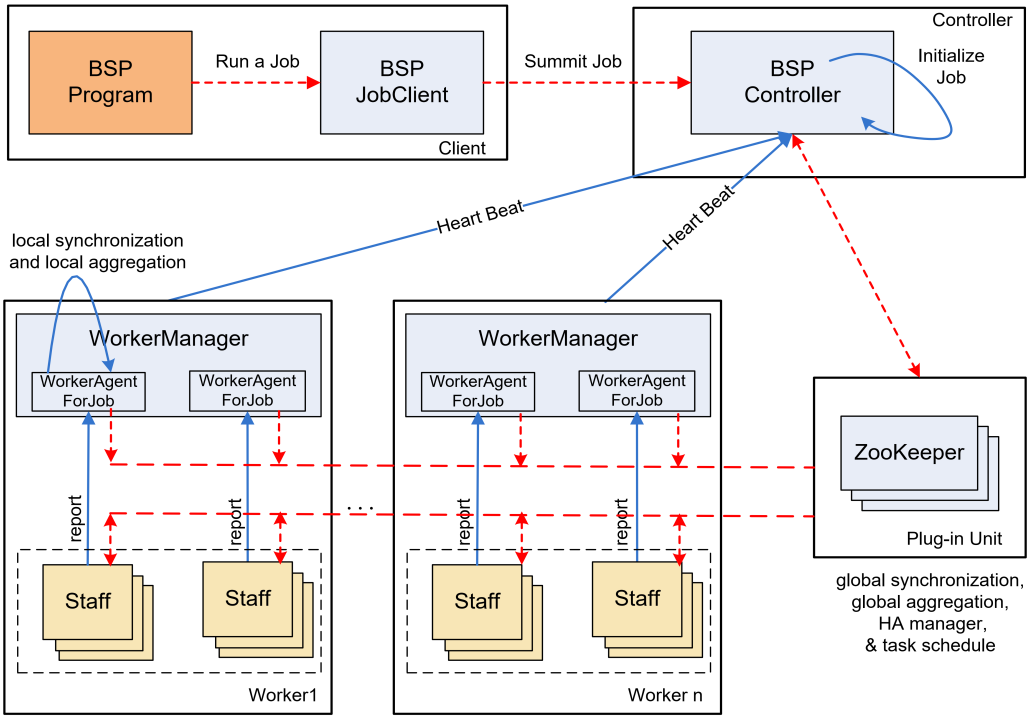

Figure 1. The architecture of BC-BSP.

Thirdly, the paper proposes a novel BC-BSP message sending and receiving mechanism designed to facilitate data sharing across different stages of the frequent subgraph mining algorithm. This mechanism enables the realization of global support calculation on large graphs by sending the same standard coded subgraph to the corresponding node. This approach enhances the efficiency of communication and coordination within the BC-BSP framework.

Additionally, the candidate subgraphs are generated through the implementation of the rightmost path expansion strategy, strategically minimizing the occurrence of redundant subgraphs. The use of standard coding ensures the unique identification of subgraphs, obviating the need for extensive isomorphism calculations. This design choice contributes to the computational efficiency and reduced complexity in the mining process. Furthermore, the MNI [33] measurement method is employed for support calculation. This method aligns with the attribute of downward closure, ensuring the accurate and efficient assessment of support levels in the mining algorithm. The existing literature has defined several anti-monotonic support calculation methods, such as Minimum Image (MNI), Harmful Overlap (HO), and the Maximum Independent Set (MIS), among others. The distinction between these methods lies in the extent of isomorphic graph coverage that they allow. This paper adopts the MNI measurement method, primarily because MNI is the only metric that can be effectively calculated, whereas the calculations for HO and MIS are NP-hard problems. Moreover, MNI offers an effective pruning strategy. The combination of these techniques enhances the overall performance and reliability of the frequent subgraph mining algorithm within the BC-BSP framework.

3. Proposed Method

3.1. Overall Design

This paper introduces an innovative frequent subgraph mining algorithm grounded in the BC-BSP framework, offering solutions to the previously identified challenges. The following elucidates the overarching concept of this paper based on the BC-BSP frequent subgraph mining algorithm.

In response to the shortcomings observed in existing frequent subgraph mining algorithms, our approach advocates for the utilization of the BSP model, exemplified by the BC-BSP framework. The BSP model is chosen for its ability to structure computations into synchronized super steps, thereby facilitating parallel processing and global synchronization. A pivotal contribution of this paper lies in the introduction of a BC-BSP message sending and receiving mechanism. This mechanism is designed to foster efficient data sharing across various stages of the frequent subgraph mining algorithm. Notably, it plays a crucial role in achieving global support calculation on large graphs by dispatching the same standard coded subgraph to the corresponding nodes. By employing this mechanism, the paper enhances the communication efficiency and coordination within the BC-BSP framework. The proposed frequent subgraph mining algorithm incorporates several strategic components to address the challenges associated with large graph data. The generation of candidate subgraphs employs the rightmost path expansion strategy, effectively mitigating redundancy. Furthermore, standard coding is implemented for unique subgraph identification, eliminating the need for extensive isomorphism calculations and thereby enhancing the overall computational efficiency. The adoption of the MNI measurement method for support calculation aligns with the downward closure attribute, ensuring both accuracy and efficiency in the mining process. The algorithm is structured into distinct stages, including data preparation and mining, with the mining stage further divided into candidate subgraph generation and support calculation. This granularity in design facilitates the clear and efficient mapping of each iterative step of the algorithm to the corresponding super step within the BC-BSP framework.

In conclusion, this paper articulates a comprehensive approach to frequent subgraph mining, leveraging the capabilities of the BC-BSP framework. Through thoughtful algorithmic design and the strategic utilization of the BSP model, our proposed methodology addresses the challenges associated with large graph data, thereby offering improved performance and scalability in the domain of frequent subgraph mining.

In a general context, common frequent subgraph mining algorithms can be dissected into three primary phases within the mining workflow.

(1) Candidate Subgraph Generation: The initial step involves the creation of candidate subgraphs. Candidate subgraphs are generated, transitioning from F^k frequent subgraphs to F^{k+1} frequent candidate subgraph extensions. This step provides essential input for subsequent frequent pattern mining. The prevalent strategies for subgraph extension include depth-first and breadth-first approaches. While centralized algorithms typically favor depth-first extension due to its superiority over breadth-first extension, distributed algorithms demonstrate increased efficiency when expanding subgraphs in a breadth-first manner. This paper employs a rightmost path extension method, which extends the frequent subgraphs from the preceding pattern by adding an edge to the rightmost path of the graph. This extension strategy facilitates parallelization and minimizes redundant candidate subgraph generation, thereby contributing to effective pruning.

(2) Subgraph Isomorphism: The second phase involves subgraph isomorphism, necessitating an assessment of isomorphism following the generation of candidate subgraphs. Commonly, standard encoding schemes are employed for isomorphism judgment. This involves generating a sequence of edges according to a prescribed sequence, with the isomorphic graphs generating the same standard coding sequence to complete the subgraph isomorphism process. The standard code utilized in this paper is the minimal DFS code.

(3) Support Degree Calculation: The final phase revolves around support degree calculation. Different measures yield varying results for support degrees, contingent upon the specific practical application scenario. In this paper, the MNI support algorithm is employed to determine support degrees. The selection of this algorithm is informed by its practical applicability and effectiveness in the given context.

After a detailed examination of the specific solutions for various facets of frequent subgraph mining tasks, let us delve into how BC-BSP can be employed to implement each aspect of the work.

BC-BSP stands as a large-scale graph processing system with disk temporary storage, grounded in the overarching synchronous parallel computing model. Throughout its operation, BC-BSP is essentially organized into the following distinct processes.

(1) Data Reading and Partitioning: The initial phase involves reading data and partitioning them. Graph data are sourced from an external graph database, and the data are segmented based on a defined split function. This division is crucial in distributing the workload efficiently among the compute nodes.
(2) Local Iteration Calculation: In this phase, each compute node undertakes local iteration calculations. Every node processes the segment of the graph data assigned to it, performing local computations relevant to the frequent subgraph mining algorithm.
(3) Message Sending and Global Synchronization: Subsequently, there is a phase dedicated to message sending and global synchronization. The computed results are transmitted to the specified node, and the system awaits the completion of global synchronization for the ongoing super step. This synchronization ensures that all nodes have completed their local computations before progressing to the next stage.
(4) Result Storage: The final step involves saving the computed results to an external storage device, such as a HDFS. This ensures the persistence of the valuable information gleaned from the frequent subgraph mining algorithm.

In summary, the BC-BSP framework is effectively leveraged across these four key processes, demonstrating its capability to orchestrate large-scale graph processing and facilitate the implementation of each facet of the frequent subgraph mining workflow. The systematic approach of data reading, local computation, message passing, global synchronization, and result storage contributes to the overall efficiency and reliability of the frequent subgraph mining algorithm within the BC-BSP framework.

The frequent subgraph mining process can be broadly categorized into two main phases: the preparatory phase and the mining phase. In the preparatory phase, tasks such as graph data reading and preprocessing are completed. The mining phase encompasses candidate subgraph growth, subgraph isomorphism, and support degree calculation. Each mining stage corresponds to frequent subgraphs of a specific size, building upon the output of the previous stage. BC-BSP organizes these tasks into super steps, wherein each super step is dedicated to accomplishing specific work. Communication between different super steps is facilitated through messages, creating a structured chain of message sending, reception, and processing to formulate the iterative job. The computational process on each vertex in BC-BSP is abstracted out and uniformly defined, ensuring consistency in the tasks performed during each scale mining stage. A practical implementation involves executing tasks such as generating candidate subgraphs, conducting subgraph isomorphism checks, and performing support calculations within individual super steps. The intuitive approach is to allocate specific super steps for designated tasks. For instance, during super step 1, graph data reading and preprocessing are completed. Subsequent super steps, such as steps 2 and 3, are dedicated to 1_frequent subgraph mining, steps 4 and 5 to 2_frequent subgraph mining, and so on, until the desired number of super steps or until no more frequent subgraphs are generated.

The synergy between frequent subgraph mining and BC-BSP computation is depicted in Figure 2, highlighting the systematic and structured approach adopted to efficiently handle each stage of the mining process within the BC-BSP framework.

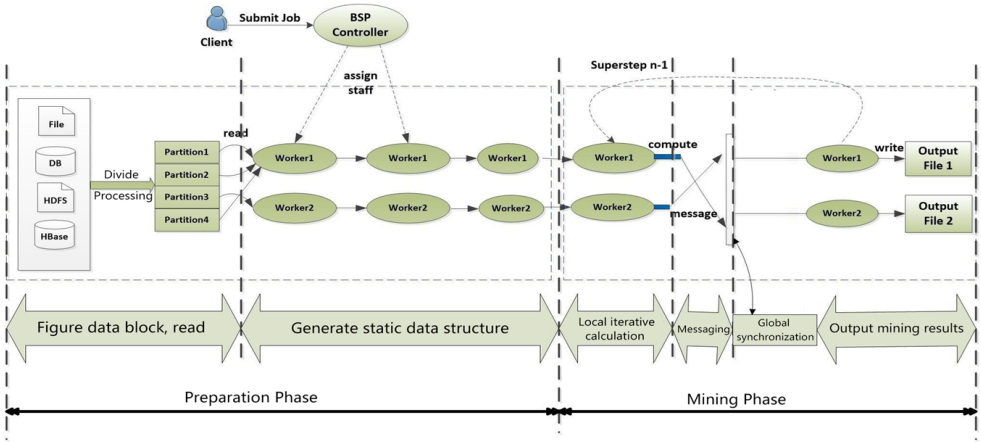

Figure 2. Calculation process of subgraph mining-based BC-BSP.

After a meticulous examination of the intricate flow of the frequent subgraph mining algorithm and the computational workflow of BC-BSP, the integration of the frequent subgraph mining algorithm with the BC-BSP computation framework is elucidated. The procedure for the frequent subgraph mining algorithm based on BC-BSP unfolds as follows.

(1) After the user submits the job, the client initiates communication with the HDFS. The primary objective is to partition the original graph data into logical fragments based on the specified method of graph data division. The client then adjusts the partitioned data according to the user-specified number of partitions and the graph data division approach. Subsequently, the modified data, now structured into logical fragments, are submitted to the cluster for further processing. This initial phase ensures proper data organization and distribution, laying the foundation for subsequent parallelized computation within the cluster.

(2) During the first iteration step, specifically in the phase of generating frequent 1_subgraphs, the initial input is the graph in the form of an adjacency list. The process commences with the reading of this adjacency list, essentially constituting the first job. The objective at this stage is to obtain the 1_subgraphs with specified encoding. Subsequently, identical specification encodings are aggregated together, seeking the count to identify frequent occurrences. The results are then recorded. The key information for the frequent edge consists of the vertex ID, while the associated value includes both edge information and specification encoding. Essentially, this step involves acquiring frequent 1_subgraphs, marking the completion of the first iteration. Alternatively, the process of generating frequent 1_subgraphs can be divided into two super steps. The rationale for this division is that calculating the frequency by seeking an edge graph might be relatively inefficient in a single super step. In the first super step, an edge graph with specified encoding is obtained. Following this, the specification encoding is hashed, resulting in a hash value representing a vertex ID number. Subsequently, the specification encoding and the information about an edge graph are packaged into a message for transmission. In the next super step, the system receives the message and performs a statistical count of the occurrences of the edge graph. This division enhances the efficiency by optimizing the calculation of the edge graph frequency, leading to an effective and streamlined process in generating frequent 1_subgraphs.

(3) In the second iteration, during the stage of generating candidate subgraphs, the focus is on creating candidate frequent 2_subgraphs. Each vertex corresponds to a set of key–value pairs obtained from the previous stage, with the vertex ID serving as the key. The goal is to obtain the specification encoding of the candidate 2_subgraphs. Following this, hash encoding is applied based on the specification encoding, and the count of vertices is determined. This information is then transmitted via a message. To iteratively generate candidate frequent multi-subgraphs, it is crucial to address scenarios where each vertex may handle multiple different keys. To manage this complexity,

partitioning is done based on the key. Utilizing the hash, vertices with the same key are grouped together, and the value within each group undergoes an extension of graph information. By extending the k_subgraphs, candidate (k + 1)_subgraphs are derived. To obtain the specification encoding of the candidate (k + 1)_subgraphs, hashing is again performed according to the specification encoding. The remainder of the vertex is identified, and this information is sent as part of the message. This iterative process ensures the systematic generation of candidate frequent multi-subgraphs, effectively building upon the results of the previous stages.

(4) In the third iteration, the focus shifts to frequent counting, culminating in the identification of frequent subgraphs. When a vertex receives messages from the previous super step, these messages contain the same specification encoding intended for that vertex. However, due to potential hash conflicts, a vertex may receive messages with multiple different specification encodings. Consequently, each vertex must process these received messages by regrouping them based on the differences in the specification encoding. To address this, messages can be regrouped through rehashing, and, subsequently, frequent counting is performed within each group. This entails applying the MNI support calculation method for frequent counting. If the frequency counts surpass a predefined threshold, the pertinent information is stored in the HDFS. Following the frequent counting phase, hash coding is applied to the specifications of the identified frequent subgraphs. The resulting hash values correspond to the IDs of specific vertices, and this information is transmitted through messages. This iterative process ensures the systematic identification and storage of frequent subgraphs meeting the specified support threshold.

The iterative process involves $2n$ super steps, aligning with the second iterative step for the generation of candidate subgraphs. The subsequent $2n + 1$ super step process aligns with the third iterative step, involving the calculation of support for candidate subgraphs. This iterative cycle continues until the specified number of iterations is reached or no more frequent subgraphs are generated. The systematic and consistent execution of these super steps ensures the effective generation and evaluation of candidate subgraphs, leading to the identification and storage of frequent subgraphs that meet the specified support threshold. This iterative approach allows for flexible and scalable adaptation to different graph mining scenarios and ensures the comprehensive exploration of the graph data.

3.2. Detailed Design

After a comprehensive analysis of the algorithm's overall flow, the detailed design of the algorithm is presented below. The distributed frequent subgraph mining algorithm, FSM-BC-BSP, based on BC-BSP, is primarily divided into four major phases: data preparation, local computing, messaging, and synchronization.

The data preparation phase involves reading the graph database, data division, obtaining edge frequent subgraphs, and generating static data. The local calculation phase focuses on generating candidate subgraphs and performing isomorphism judgment. The messaging phase facilitates data sharing. In addition to fulfilling the super step synchronization tasks mandated by the BC-BSP architecture, the synchronization phase is primarily dedicated to frequent counting, support calculation, and tasks related to writing frequent subgraphs to the HDFS.

The different stages are interconnected through the BC-BSP system's super step mechanism until frequent subgraphs are no longer generated or until the predefined number of super steps is reached.

Subsequently, this paper will elucidate the specific implementation details of each stage.

3.2.1. Data Division

During the data preparation phase, FSM-BC-BSP employs a strategy to divide the graph data G into multiple partitions. A more intuitive partitioning strategy involves segmenting the large graph into different partitions based on the number of edges. This strat-

egy is particularly effective on datasets where most vertices have a greater number of edges. However, it is important to note that such a partitioning strategy might face challenges with datasets exhibiting extremely unevenly distributed edges. In cases where the edges are distributed unevenly, this strategy could result in an unbalanced load distribution, potentially leading to nodes experiencing an overflow or saturation due to excessive computational loads.

In the context of the FSM-BC-BSP algorithm, the number of partitions is a crucial parameter. Experimental evaluations are conducted to assess the algorithm's performance under different numbers of partitions. The objective is to strike a balance between load balancing and efficient parallel processing, considering the characteristics of the specific dataset being analyzed. Experimentation with varying numbers of partitions allows for an empirical understanding of how the algorithm performs under different configurations, guiding the selection of an optimal number of partitions for a given dataset.

3.2.2. Obtaining Edge Frequent Subgraphs

In the data preparation phase, which corresponds to the frequent 1_subgraph mining stage, the algorithm takes the following steps.

(1) Infrequent Edge Removal: While scanning the entire graph database, infrequent edges are removed, retaining only the frequent subgraphs.
(2) Vertex Renumbering: After removing infrequent edges, the vertices on the entire large graph are renumbered. It is essential to ensure that vertices and edges maintain consistent labeling.
(3) Initialization for Frequent 1_Subgraph Mining: BC-BSP determines the current number of super steps. When it is identified as super step 0, the process for frequent 1_subgraph mining is initiated.
(4) Static Data Generation: The static data needed for each iteration are generated after the 0th iteration. Specifically, an Edge Extend Map (EEM) is created. The EEM stores information about which vertices each vertex can extend to during the mining process.
(5) Message Transmission: During message transmission between super steps, in addition to the graph information of the current mode, the EEM needs to be sent. This facilitates efficient communication and information sharing between different super steps.
(6) Support Calculation Optimization: In the first super step, when receiving the message sent by the 0th super step, support calculation is optimized. Since non-frequent edges have been removed while reading the graph data, support calculation is no longer necessary. Instead, the algorithm focuses on expanding the current edge based on the data structure of the EEM.

As depicted in Figure 3, the algorithm, after reading the adjacency list of the original graph, performs the 0th iteration to generate an edge code and the EEM data structure. The unique minimal DFS code for each graph serves as the key during message sending. Subgraphs with the same specification encoding are sent to the same node along with the EEM data structure for all vertices included in the message. This structured approach ensures efficient processing and communication throughout the frequent subgraph mining process.

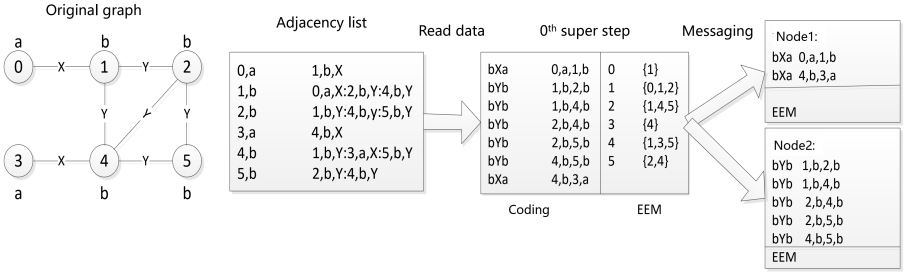

Figure 3. Example of data preparation.

3.2.3. Local Computing

The local computing phase of FSM-BC-BSP is responsible for generating candidate subgraphs and performing subgraph isomorphism tests. Candidate subgraphs are generated using the rightmost path extension strategy, and subgraph isomorphism is achieved through specification encoding.

For a given frequent pattern of size k, a candidate pattern of size $k + 1$ is obtained by adding a frequent edge belonging to the EEM during the candidate subgraph generation stage.

If a frequent edge introduces a new vertex, we say that this frequent edge is the outer edge; otherwise, for the inner edge, the latter simply connects the existing vertices in the frequent pattern. The newly added vertex in the outer edge is given an ID, which is the largest of the already existing vertices in the frequent pattern; the size of the vertex IDs represents the order in which they were added as they were expanded outside. In the topological structure of graph mining, the candidate pattern is called the frequent edge sub-node and the frequent edge is called the parent node of the candidate pattern. Based on this parent–child relationship, a candidate subgraph spanning tree can be constructed during the mining job. It should be noted that if the candidate pattern has $k + 1$ edges, the candidate pattern will have multiple paths in the candidate subgraph spanning tree according to the order of edges added. However, in FSM-BC-BSP, it is stated that only one build path will be considered valid and no other redundant candidate patterns will be generated. With such a constraint, it is guaranteed that such a candidate subgraph spanning tree is unique to an FSM-BC-BSP task. In this paper, the rightmost path generation strategy is used. In short, the rightmost path is the shortest path from the vertex with the smallest ID to the vertex with the largest ID, and the graph is extended on such a path.

In the process of building candidate subgraphs, multiple generation paths may produce the same candidate subgraph, necessitating subgraph isomorphism detection. FSM-BC-BSP uses the minimal DFS code specification encoding method. The same minimal DFS code within the same structure facilitates pruning and supports the calculation of candidate subgraphs on different vertices.

An example of the principle of the minimal DFS code is shown in Figure 4. The DFS of Figure 4a, depending on the order of the access vertices, may lead to different depth-first search trees, i.e., a graph may correspond to multiple depth-first search trees, Figure 4b–d are the depth-first search trees of Figure 4a. Converting each subscripted graph into an edge sequence leads to DFS encoding.

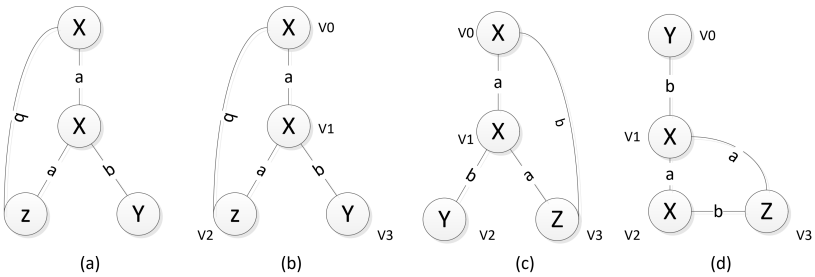

Figure 4. An example of DFS trees. (**a**) Original image; (**b**–**d**) Different depth-first search trees of (**a**).

DFS encoding establishes two types of order; in order to select the smallest encoding DFS encoding as the standard encoding, one is the order of the edge, which is also the access sequence, as shown in Figure 4b. For the forward edge order (0,1) (1,2) (1,3), backward edge order (2,0), one is the sequence order.

Table 1 presents the DFS encoding corresponding to the DFS tree in Figure 4.

For the edge sequences, we establish two edge sequences $e_1 = (i_1 , j_1)$ and $e_2 = (i_2 , j_2)$. If one of the following four conditions is true, then $e_1 \prec e_2$.

1. If both are forward edges, then $(j_1 < j_2)$ or $(i_1 < i_2$ and $j_1 = j_2)$;
2. If both are backward edges, then $(i_1 < i_2)$ or $(i_1 = i_2$ and $j_1 < j_2)$;

3. If e_1 is the backward edge, and e_2 is the forward edge, then $i_1 < i_2$;
4. If e_1 is the forward edge, and e_2 is the backward edge, then $j_1 \leq j_2$;

For the sequence order, it is necessary to define the order of different sequences in different DFS codes. Each DFS sequence is represented by a quintuplet $(i, j, l_i, l_{(i,j)}, l_j)$, as shown in Table 1, where i and j are the edge order and l_i and l_j are the vertex labels; $l_{(i,j)}$ is the connecting edge between them. The DFS dictionary sequence is defined as follows: the edge order has the first priority, the vertex l_i has the second priority, the edge label $l_{(i,j)}$ has the third priority, and l_j has the fourth priority, i.e., two DFS sequence sizes need to be compared according to the order of priority. For example, for the D_1 and D_2 sequences in Table 1, we take the compared first-priority order and find that the two edges are the same; then, we compare the vertex order (assuming that the label comparison is based on the natural order of the letters). We see that X < Y, so (0, 1, X, a, X) < (0, 1, Y, b, X) is obtained. Based on the above rules, $D_0 < D_1 < D_2$, the smallest of all DFS encodings for a given graph is the minimal DFS code.

Table 1. DFS codes.

D_0 (Figure 4b)	D_1 (Figure 4c)	D_2 (Figure 4d)
(0, 1, X, a, X)	(0, 1, X, a, X)	(0, 1, Y, b, X)
(1, 2, X, a, Z)	(1, 2, X, b, Y)	(1, 2, X, a, X)
(2, 0, Z, b, X)	(1, 3, X, a, Z)	(2, 3, X, b, Z)
(1, 3, X, b, Y)	(3, 0, Z, b, X)	(3, 1, Z, a, X)

3.2.4. Messaging and Synchronization

Before entering the synchronization phase, there are links for message sending and message receiving. The serialization and deserialization processes facilitate data sharing across different phases of the algorithm, although these processes are not reiterated.

The synchronization phase is responsible for counting the isomorphic graphs within a single graph. As previously mentioned, this paper utilizes the MNI support measure method.

4. Experiments

4.1. Experimental Environment

This section details the experimental environment, employing a 48-node cluster server system. During the actual experiments, 28 nodes were utilized, including 1 master node and 27 slave nodes.

4.2. Experimental Data

The experiment utilizes four real datasets, As.txt, credit1, credit2, and DBLP, detailed in Table 2.

Table 2. The description of the datasets.

Dataset	Vertex	Edges	Vertices_Label Number	Edge_Label Number
As.txt	6474	12,572	62	61
DBLP	151,574	191,840	7	17
Credit1	14,700	14,000	59	20
Credit2	6300	98,576	59	51

4.3. Experimental Results Analysis

4.3.1. FSM-BC-BSP Algorithm Results

This section provides an in-depth analysis of the FSM-BC-BSP algorithm's performance under various input datasets and parameter configurations. The examined parameters primarily include minimum support, the number of steps in the super step, and the count of initial divisions. To gauge the impact of each parameter on the algorithm's performance, specific adjustments are made to one parameter at a time, keeping others constant. To ensure fairness, the output of the algorithm is consistently cleared from the HDFS with each run,

while maintaining consistent hardware conditions to eliminate additional factors affecting the performance.

(1) Different Datasets and Varied Minimum Support Corresponding to Subgraph Counts

Employing the FSM-BC-BSP algorithm on the credit1, credit2, and DBLP datasets, frequent subgraphs are mined under varying minimum support levels. The subgraph counts are then tabulated based on different support degrees. With an initial division count of nine and a super step count of four, the results indicate that, as expected, the number of subgraphs decreases exponentially with an increasing support degree. This trend aligns with common expectations. Notably, for the DBLP dataset, no corresponding frequent subgraph is identified when the support exceeds 16, a characteristic specific to the DBLP dataset. The statistical results are shown in Table 3.

Table 3. The number of frequent subgraphs with different support under different datasets.

Min_Sup	*n* = 2	*n* = 4	*n* = 8	*n* = 16	*n* = 32	*n* = 64	*n* = 128
Credit1	9166	1945	1493	1045	599	253	88
Credit2	2529	2250	1920	1522	1085	668	324
DBLP	243,473	114,099	63,245	853	0	0	0

(2) Different Datasets, Varied Minimum Support, and Algorithm Runtime

Applying the FSM-BC-BSP algorithm to the credit1, credit2, As, and DBLP datasets with varying support degrees, the runtime of the algorithm is statistically measured. With an initial fragment count of nine and a super step count of four, the algorithm demonstrates the efficient mining of frequent subgraphs across different support levels. Notably, the runtime is prolonged when the support is very small under the DBLP dataset. This extended runtime can be attributed to the uneven distribution of neighboring edges in DBLP vertices, following a power-law distribution. Such disparities lead to a large number of some frequent subgraphs and the minimal occurrence of others, resulting in inefficient resource utilization and an increased operating time, as depicted in Figure 5. The figure shows the trend of the algorithm based on the three datasets of credit1, credit2, and As.txt as the support running time changes. The running results based on DBLP are too different from the other three and cannot be displayed in the same graph.

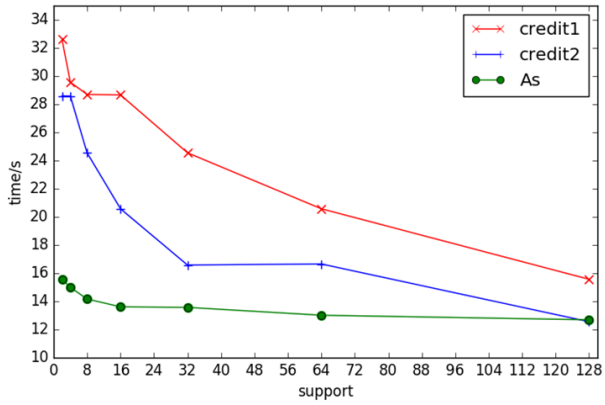

Figure 5. The times of frequent subgraphs with different support under different datasets.

(3) Different Datasets, Varied Initial Fragments, and Algorithm Runtime

Examining the credit1, credit2, and AS datasets under the FSM-BC-BSP algorithm with a consistent support level of four and a super step count of five, the algorithm's runtime is evaluated under different initial fragment counts. The results indicate a trend of initial downward and subsequent upward runtime variations with changes in the number of partitions. This behavior is attributed to the reduced parallelism when fewer partitions are

employed, resulting in an increased runtime. Conversely, when the fragment count exceeds a certain threshold, the benefits of parallelism are offset by network communication and load balancing issues, leading to an increased runtime, as shown in Figure 6.

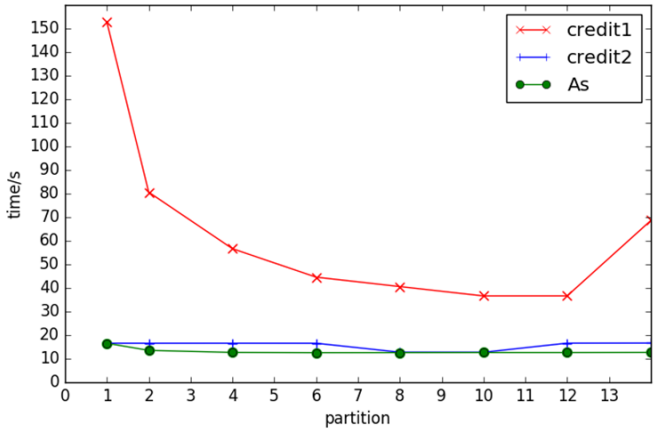

Figure 6. The times of frequent subgraphs with different partitions under different datasets.

(4) Same Dataset, Varied Initial Fragments, and Minimum Support Impacting Algorithm Runtime

Utilizing the credit1 dataset and aligning the initial fragment counts with the minimum support levels, the impact of these two parameters on the algorithm is explored. As illustrated in Figure 7, the FSM-BC-BSP algorithm achieves optimal performance with an initial fragment count of 10 and a support count of 60.

These comprehensive analyses provide valuable insights into the FSM-BC-BSP algorithm's behavior under diverse conditions, allowing for informed parameter tuning and optimization.

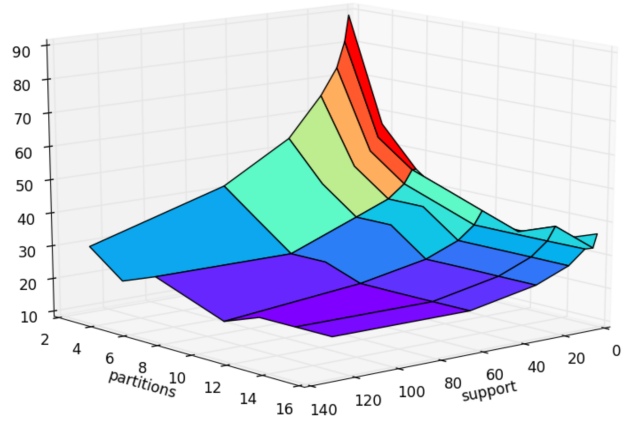

Figure 7. The times of frequent subgraphs with different partitions and different support

4.3.2. FSM-BC-BSP Algorithm Performance Evaluation

This section evaluates the performance of the FSM-BC-BSP algorithm through a comparative analysis with two other frequent subgraph mining algorithms, namely FSM-H based on MapReduce and GRAMI based on a single machine.

(1) Different Datasets' Comparison at the Same Support Level

The initial part of the test involves comparing the runtime performance of FSM-BC-BSP, FSM-H, and GRAMI across different datasets, while maintaining the same support level of 4. Datasets such as As.txt, credit1, credit2, and DBLP are used. The results consistently indicate that the FSM-BC-BSP algorithm outperforms FSM-H and GRAMI on

all datasets. Figure 8 provides a visual representation of the runtime comparison on the diverse datasets.

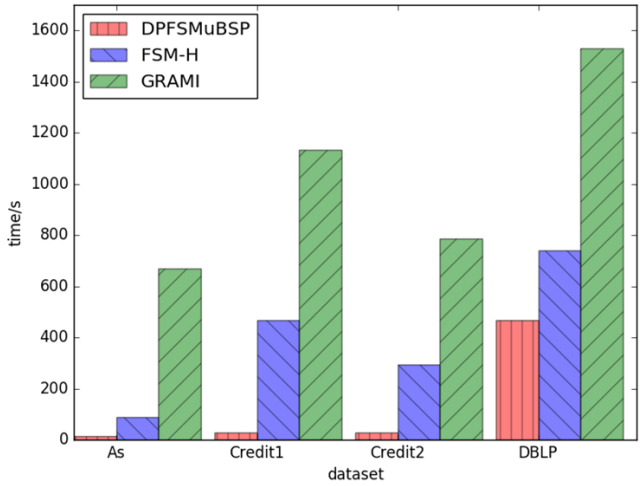

Figure 8. The times of frequent subgraphs with the same support under different datasets.

(2) Support-Dependent Performance within the Same Dataset

The second part of the test focuses on the runtime performance of FSM-BC-BSP, FSM-H, and GRAMI under varying support levels, utilizing the As.txt dataset with 6474 vertices. Figure 9 illustrates that, when the support is relatively large, the performance of the three algorithms is comparable. However, as the support increases, the FSM-BC-BSP algorithm demonstrates superior efficiency compared to FSM-H and GRAMI. Specifically, when the support level is 2, FSM-BC-BSP exhibits over eight times better performance than FSM-H and an impressive 1000 times improvement over GRAMI. The figure emphasizes the robust performance and stability of the proposed FSM-BC-BSP algorithm.

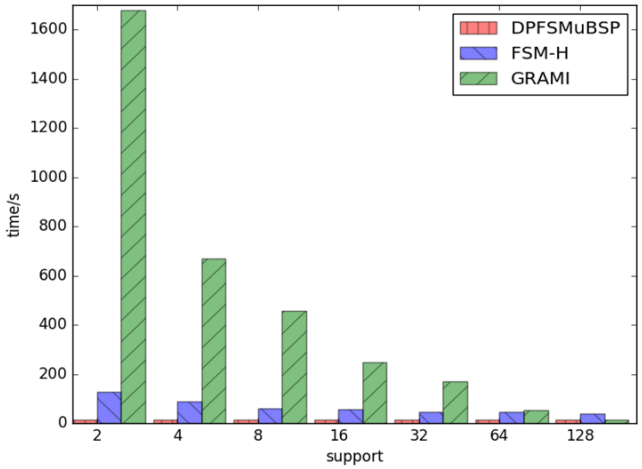

Figure 9. The times of frequent subgraphs with the same support under different datasets.

These comparative analyses provide compelling evidence of the superior performance of FSM-BC-BSP, especially when confronted with datasets of varying characteristics and support levels. The results underscore the efficiency, scalability, and stability of the FSM-BC-BSP algorithm in comparison to its counterparts, FSM-H and GRAMI.

5. Conclusions and Future Work

In recent years, the increased use of graph models to abstract scientific data has led to a surge in research on the mining of interesting patterns from large-scale graph data. Existing research primarily focuses on graph sets and single large graphs. However, the

current algorithms for the mining of frequent subgraphs based on single large graphs face challenges such as high complexity and poor scalability. Moreover, they struggle to handle the mining of frequent subgraphs on a large single graph.

Against this backdrop, this paper delves into the mining of frequent subgraphs on large graphs and introduces the FSM-BC-BSP algorithm to address the challenges associated with single-graph frequent subgraph mining. The algorithm utilizes the BC-BSP message-sending and message-receiving mechanism for data sharing across different stages of the frequent subgraph mining process. It employs standard subgraph coding, ensuring that identical subgraphs are sent to the same node for global support calculation on the large graph. Additionally, FSM-BC-BSP leverages the rightmost path expansion strategy to generate candidate subgraphs, minimizing redundancy. The use of standard coding helps to uniquely identify subgraphs, eliminating the need for subgraph isomorphism calculations. Furthermore, the algorithm adopts the MNI measurement method for support calculation, adhering to the downward closure attribute.

Performance testing of the FSM-BC-BSP algorithm across various datasets, initial partition counts, and support levels yielded promising results. In comparison to the FSM-H and GRAMI algorithms, FSM-BC-BSP demonstrated superior performance. Particularly in scenarios with low support, FSM-BC-BSP showcased outstanding efficiency. The FSM-BC-BSP algorithm represents a valuable contribution to the field of frequent subgraph mining, offering an effective solution for large single-graph scenarios. The proposed algorithm's scalability, efficiency, and ability to handle low-support situations make it a noteworthy advancement in the realm of graph data mining.

Certainly, this algorithm has inherent limitations—notably, the process of data transmission involves redundant static data that are required at each iteration step and remain unchanged. The algorithm proposed in this paper operates under the assumption that the input graph data are static. However, in practical applications, graph data often undergo frequent changes. For instance, within the DBLP user relationship graph, should a new author publish a paper, the graph structure would consequently experience some degree of alteration. This observation underscores the necessity for future research to focus on mining frequent subgraphs within dynamically changing graphs. Moreover, when it comes to accessing static data, the current algorithm can only sequentially match the static data stored on the hard disk. Indeed, as each frequent subgraph expands, the requisite static data are directly related to the current vertex data. Therefore, forthcoming research efforts should aim to establish a static data access model, designed to optimize the retrieval of such static data.

Author Contributions: Conceptualization, Y.B.; Data curation, Y.B.; Investigation, F.L. (Fan Li); Methodology, F.L. (Fangling Leng); Software, F.L. (Fan Li); Supervision, G.Y.; Validation, T.Z.; Visualization, F.L. (Fangling Leng); Writing—original draft, F.L. (Fangling Leng); Writing—review and editing, T.Z. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the National Natural Science Foundation of China (grant numbers 62137001 and 62272093).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Data is contained within the article.

Conflicts of Interest: The authors declare no conflicts of interest. The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript; or in the decision to publish the results.

References

1. Alessandro, L.; Luca, O.; Davide, A. Mining Big Data with Random Forests. *Cogn. Comput.* **2019**, *11*, 294–316.
2. Deng, Z.H. DiffNodesets: An efficient structure for fast mining frequent itemsets. *Appl. Soft Comput.* **2016**, *41*, 214–223. [CrossRef]

3. Aryabarzan, N.; Minaei-Bidgoli, B.; Teshnehlab, M. negFIN: An efficient algorithm for fast mining frequent itemsets. *Expert Syst. Appl.* **2018**, *105*, 129–143. [CrossRef]
4. Elsebakhi, E.; Lee, F.; Schendel, E.; Haque, A.; Kathireason, N.; Pathare, T.; Syed, N.; Al-Ali, R. Large-scale machine learning based on functional networks for biomedical big data with high performance computing platforms. *J. Comput. Sci.* **2015**, *11*, 69–81. [CrossRef]
5. Kayoko, Y. Visualization of GIS Analytic for Open Big Data in Environmental Science. In Proceedings of the International Conference on Cloud Computing and Big Data, Shanghai, China, 4–6 November 2015; pp. 201–208.
6. Liezhuan, L. Research on the Innovation of electronic commerce Business Model Based on Data Mining under the Background of Big Data. In Proceedings of the Information Systems and Computer Aided Education, Dalian, China, 24–26 September 2021; pp. 418–422.
7. Zhengwu, S.; Dan, C.L.; Yong, S. Big Data Analysis on Social Networking. In Proceedings of the Big Data, Los Angeles, CA, USA, 9–12 December 2019; pp. 6220–6222.
8. Demetrovics, J.; Quang, H.M.; Anh, N.V.; Thi, V.D. An optimization of closed frequent subgraph mining algorithm. *Cybern. Inf. Technol.* **2017**, *17*, 3–15. [CrossRef]
9. Nejad, S.J.; Ahmadi-Abkenari, F.; Bayat, P. A combination of frequent pattern mining and graph traversal approaches for aspect elicitation in customer reviews. *IEEE Access* **2020**, *8*, 151908–151925. [CrossRef]
10. Ansari, Z.A.; Abulaish, M. An efficient subgraph isomorphism solver for large graphs. *IEEE Access* **2021**, *9*, 61697–61709. [CrossRef]
11. Iqbal, R.; Doctor, F.; More, B.; Mahmud, S.; Yousuf, U. Big data analytics and computational intelligence for cyber–physical systems: Recent trends and state of the art applications. *Future Gener. Comput. Syst.* **2020**, *105*, 766–778. [CrossRef]
12. Ali, J.; Christopher, C.Y. Frequent Subgraph Mining Algorithms in Static and Temporal Graph-Transaction Settings: A Survey. *IEEE Trans. Big Data* **2022**, *8*, 1443–1462.
13. Mohammad, E.S.C.; Chowdhury, F.A.; Carson, K.L. A New Approach for Mining Correlated Frequent Subgraphs. *ACM Trans. Manag. Inf. Syst.* **2022**, *13*, 9:1–9:28.
14. Liu, M.; Li, P. SATMargin: Practical maximal frequent subgraph mining via margin space sampling. In Proceedings of the ACM Web Conference (WWW), Lyon, France, 25–29 April 2022; pp. 1495–1505.
15. Nguyen, L.B.; Vo, B.; Le, N.T.; Snasel, V.; Zelinka, I. Fast and scalable algorithms for mining subgraphs in a single large graph. *Eng. Appl. Artif. Intell.* **2020**, *90*, 103539. [CrossRef]
16. Elseidy, M.; Abdelhamid, E.; Skiadopoulos, S.; Kalnis, P. GraMi: Frequent subgraph and pattern mining in a single large graph. *Proc. VLDB Endowment* **2014**, *7*, 517–528. [CrossRef]
17. Le, N.T.; Vo, B.; Nguyen, L.B.; Fujita, H.; Le, B. Mining weighted subgraphs in a single large graph. *Inf. Sci.* **2020**, *514*, 149–165. [CrossRef]
18. Zhaoming, C.; Xinyang, C.; Guoting, C. Wensheng Gan: Frequent Subgraph Mining in Dynamic Databases. In Proceedings of the IEEE International Conference on Big Data, Sorrento, Italy, 15–18 December 2023; pp. 5733–5742.
19. Giulia, P.; Gianmarco, D.F.M.; Matteo, R. MaNIACS: Approximate Mining of Frequent Subgraph Patterns through Sampling. *ACM Trans. Intell. Syst. Technol.* **2023**, *14*, 54:1–54:29.
20. Chakravarthy, S.; Beera, R.; Balachandran, R. DB-Subdue: Database approach to graph mining. In Proceedings of the Pacific-Asia Conference on Knowledge Discovery and Data Mining, Sydney, NSW, Australia, 26–28 May 2004; pp. 341–350.
21. Kuramochi, M.; Karypis, G. Finding frequent patterns in a large sparse graph. *Data Min. Knowl. Discov.* **2005**, *11*, 243–271. [CrossRef]
22. Xifeng, Y.; Jiawei, H. gSpan: Graph-Based Substructure Pattern Mining. In Proceedings of the 2002 IEEE International Conference on Data Mining, Maebashi City, Japan, 9–12 December 2002; pp. 721–724.
23. Reinhardt, S.; Karypis, G. A multi-level parallel implementation of a program for finding frequent patterns in a large sparse graph. In Proceedings of the IEEE International Parallel and Distributed Processing Symposium, Long Beach, CA, USA, 26–30 March 2007; pp. 1–8.
24. Abdelhamid, E.; Abdelaziz, I.; Kalnis, P.; Khayyat, Z.; Jamour, F. ScaleMine: Scalable parallel frequent subgraph mining in a single large graph. In Proceedings of the Conference for High Performance Computing, Networking, Storage and Analysis, Salt Lake City, UT, USA, 13–18 November 2016; pp. 716–727.
25. Dean, J.; Ghemawat, S. MapReduce: Simplified data processing on large clusters. *Commun. ACM* **2008**, *51*, 107–113. [CrossRef]
26. Zaharia, M.; Chowdhury, M.; Das, T.; Dave, A.; Ma, J.; McCauly, M.; Franklin, M.J.; Shenker, S.; Stoica, I. Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing. In Proceedings of the 9th USENIX Symposium on Networked Systems Design and Implementation, San Jose, CA, USA, 25–27 April 2012; pp. 15–28.
27. Qiao, F.; Zhang, X.; Li, P.; Ding, Z.; Jia, S.; Wang, H. A parallel approach for frequent subgraph mining in a single large graph using spark. *Appl. Sci.* **2018**, *8*, 230. [CrossRef]
28. Mansurul, B.; Mohammad, A.H. An Iterative MapReduce Based Frequent Subgraph Mining Algorithm. *IEEE Trans. Knowl. Data Eng.* **2015**, *27*, 608–620.
29. Lu, W.; Chen, G.; Tung, A.K.; Zhao, F. Efficiently extracting frequent subgraphs using MapReduce. In Proceedings of the IEEE BigData, Santa Clara, CA, USA, 6–9 October 2013; pp. 639–647.

30. Yuan, L.; Yan, D.; Qu, W.; Adhikari, S.; Khalil, J.; Long, C.; Wang, X. T-FSM: A Task-Based System for Massively Parallel Frequent Subgraph Pattern Mining from a Big Graph. *ACM Manag. Data* **2023**, *1*, 74:1–74:26. [CrossRef]
31. Bao, Y.; Wang, Z.; Gu, Y.; Yu, G.; Leng, F.; Zhang, H.; Chen, B.; Deng, C.; Guo, L. BC-BSP: A BSP-Based Parallel Iterative Processing System for Big Data on Cloud Architecture. In Proceedings of the Database Systems for Advanced Applications, Wuhan, China, 22–25 April 2013; pp. 31–45.
32. Valiant, L.G. A bridging model for parallel computation. *Commun. ACM* **1990**, *33*, 103–111. [CrossRef]
33. Björn, B.; Siegfried, N. What Is Frequent in a Single Graph? In Proceedings of the Pacific-Asia Conference on Advances in Knowledge Discovery and Data Mining (PAKDD), Osaka, Japan, 20–23 May 2008; pp. 858–863.