Review Article **Open Access**

Büşra Güvenoğlu and Belgin Ergenç Bostanoğlu*

A qualitative survey on frequent subgraph mining

https://doi.org/10.1515/comp-2018-0018
Received July 9, 2018; accepted December 5, 2018

Abstract: Data mining is a popular research area that has been studied by many researchers and focuses on finding unforeseen and important information in large databases. One of the popular data structures used to represent large heterogeneous data in the field of data mining is graphs. So, graph mining is one of the most popular subdivisions of data mining. Subgraphs that are more frequently encountered than the user-defined threshold in a database are called frequent subgraphs. Frequent subgraphs in a database can give important information about this database. Using this information, data can be classified, clustered and indexed. The purpose of this survey is to examine frequent subgraph mining algorithms (i) in terms of frequent subgraph discovery process phases such as candidate generation and frequency calculation, (ii) categorize the algorithms according to their general attributes such as input type, dynamicity of graphs, result type, algorithmic approach they are based on, algorithmic design and graph representation as well as (iii) to discuss the performance of algorithms in comparison to each other and the challenges faced by the algorithms recently.

Keywords: frequent subgraph mining, graph mining, data mining

1 Introduction

Data mining is the process of automatically extracting previously unknown, meaningful and useful knowledge from large databases [1]. In today's world, the data grows day by day and it is necessary to find accurate and interesting information from this large volume of data. For this very reason, data mining has become an important area for researchers.

The data processed in the data mining can be obtained from many sources where different data types can be used. Examples of these different data types are text data, sound data, image data, graph data, etc. Since graphs better represent the complex and arbitrary relations among data attributes, they are used to represent data in a wide spectrum of areas, such as users (nodes) and their relationship (edges) in social networks, atoms (nodes) and bonds (edges) between them in chemical structures, proteins (nodes) and protein interactions (edges) in biological network, computer (nodes) and links between them in computer networks [2]. Graph mining is a data mining subdivision where the data is represented by a graph [3].

One of the important data mining tasks is the problem of finding frequent subgraphs in a graph database. The aim of frequent subgraph mining (FSM) is to find all subgraphs whose number of occurrences is at least equal to the number of user-defined threshold [4]. In many domains it is necessary to find these common structures, because these repetitive structures can provide a better understanding of the data or give a different perspective about data. These frequent patterns are used in determining the similarities between the graphs [5], clustering [6], graph indexing [7] and classification [8].

Most FSM algorithms consist of two important phases: candidate generation and frequency calculation. Candidates are generated using breadth first strategy or depth first strategy. One of the most important factors affecting the performance of the algorithm when generating candidates is the generation of the same candidate more than once. Since the data grows, number of candidates generated grow. Duplicated and redundant candidates should be avoided during candidate generation for an efficient algorithm. The next phase in the FSM algorithm is to calculate the frequency of the candidates generated and to determine which are the most frequent among them. To calculate the frequency of a subgraph, it is necessary to find the number of graphs that are isomorphic to this subgraph in a database. The subgraph isomorphism testing is a fundamental problem of these algorithms since this problem is NP-complete [9]. The cost of finding isomorphic graphs

Büşra Güvenoğlu: Izmir Institute of Technology, Department of Computer Engineering, Izmir/Turkey;
E-mail: busraguvenoglu@iyte.edu.tr
***Corresponding Author: Belgin Ergenç Bostanoğlu:** Izmir Institute of Technology, Department of Computer Engineering, Izmir/Turkey; E-mail: belginergenc@iyte.edu.tr

increases exponentially as the size of the graph database increases.

The solutions proposed by different FSM algorithms can be divided into different categories according to input type, dynamicity of graphs, result type, algorithmic approach, algorithmic design and graph representation.

In FSM algorithms, the first issue to be considered is the type of input used. There are two different problem statements according to the input type. The dataset used in the algorithm may also be a single large graph or small graphs (set of graphs) called transactions. Frequency calculation of a subgraph in the single large graph dataset is different from the transactional dataset. While calculating the frequencies of candidates in a transactional dataset, the number of transactions that contain this subgraph is calculated. However, since there is no transaction in a single large graph dataset, different methods were proposed [10–12]. Another issue to be considered in FSM algorithms is the properties of the graphs. Graphs used in FSM algorithms may have different properties from each other, for example, graphs may be directed or undirected, multiple edges between nodes may or may not be allowed. Most FSM algorithms work with connected graphs. Based upon the properties exhibited by the graphs, the solutions suggested for FSM algorithms are adapted.

Apart from these, graphs can be static (time invariant) or dynamic (time varying). Static graphs do not change over time. Most existing FSM algorithms are suitable for static graphs and small datasets. These algorithms generally scan entire dataset for each candidate to calculate its frequency. As dynamic graphs change over time, the size of these graphs also changes. It is not an efficient and practical method to scan the entire dataset to calculate the frequency of each candidate after change occurs. Besides, with time information, dynamic graphs can represent the evolution of frequent subgraphs in a period. Frequent subgraphs can provide better insight about the local and structural changes of data over time. FSM algorithms must be adapted for dynamic graphs.

The design of algorithms obviously depends on what the users expect from the results of these algorithms. While most FSM algorithms find all frequent subgraphs in a dataset, some algorithms find a more meaningful subset (such as closed frequent subgraphs [13]) of these frequent subgraphs to reduce search space and facilitate working on big data.

The algorithms examined in this area generally use two different algorithmic approaches: apriori based and pattern growth-based approaches. Apriori based algorithms [14–16] generate candidates based on breadth first strategy and apply subgraph isomorphism testing to calculate frequencies of candidates. Especially, when the dataset is large, these algorithms suffer from generating too many candidates. Pattern growth-based algorithms [13, 14, 17, 18] generate candidates based on depth first strategy. The pattern growth approach generally avoids the cost of generating candidates and subgraph isomorphism testing. These candidates are generated by extending frequent subgraphs starting from minimal frequent subgraphs by adding one edge at every step until they are still frequent. However, these algorithms might suffer from the generation of the same candidates.

Many algorithms that solve the frequent pattern mining problem give good results on small datasets but are not suitable for working on big data. However, the real datasets contain very large data and therefore the algorithms should be able to work effectively on big data. As the data size grows, the data cannot fit in the memory on a single machine, or it may not be an efficient and practical method to work on a single machine with this large amount of data. For this reason, parallel algorithms have been developed to find frequent subgraphs in big data [19–21].

The most popular methods used to represent graphs are the adjacency matrix and adjacency list. Graphs should be represented uniquely to facilitate subgraph isomorphism testing. Since there may be more than one adjacency matrix representing the same graph, new methods are proposed such as canonical adjacency matrix (CAM) [14] and min DFS code [17].

In the context of this study, first the basic two steps of the FSM process are analyzed. The methods used in these two steps and the algorithms using these methods are examined. Then a categorization of FSM algorithms according to the aforementioned properties is presented. There are detailed surveys that compare FSM algorithms according to their different attributes [22–24]. However, they do not capture recent algorithms focusing on new requirements as dynamicity or volume of data. In real systems, data can change continuously over time. For example, in a social or telecommunication network, data only goes through once and it is very difficult to keep such data in the dataset. Such data is called stream data and existing FSM algorithms are not suitable for working on stream data. Since most FSM algorithms are also not suitable for big data, the algorithms proposed in this area recently are parallel algorithms. Generally, the existing FSM algorithms have been modified to work in parallel. In this survey, dynamic and parallel algorithms are also examined. Another novelty of this survey that it focuses primarily on FSM for the sake of simplicity rather than analysing both subtree and subgraph mining. If any two vertices of the graph are connected by exactly one simple path, this graph is called a tree. The subtree of a tree consists of a node in this tree and all its descendants within it. The subgraph of a graph consists of subset of the nodes and edges of this graph. On the other hand, a discussion on the performance comparison of the algorithms together with the challenges they face currently is also presented in this survey.

This paper is organized as follows: in the second section, the basic concepts that should be known about the problem of FSM are explained. In the third section, the process of FSM is addressed. In the fourth section, categorization of algorithms that solve frequent subgraph mining problems from different perspectives according to the properties of graphs are presented. In the fifth section we present a discussion on popular FSM algorithms and challenges of the problem and finally we conclude this paper in sixth section.

2 Background

This section introduces the definitions of key terms that are related to the Frequent Subgraph Mining (FSM) problems.

2.1 Basic graph terminology

A ***graph*** consists of a finite vertex (or node) set V and edge set E that connects the vertices to each other. Vertices and edges in a graph can have their own labels, and these labels need not to be unique. Such a graph is called a ***labeled graph.*** Figure 1 is an example of a labeled graph.

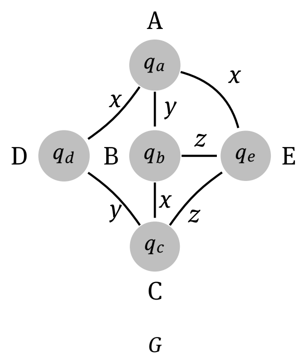


Figure 1: A labeled graph (G) example.

If the edges of a graph have direction, that is, the edges of a graph cannot be traversed in either direction, such graph is called a ***directed graph***, otherwise it is called an ***undirected graph.*** Figure 2 is an example of a directed graph. Figure 1 is an example of an undirected graph.

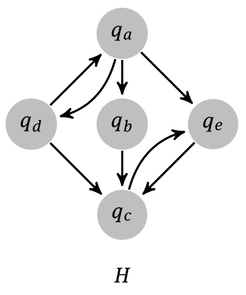


Figure 2: A directed graph (H) example.

In a graph, edges may have a weight, for instance, this weight may represent cost of traversing. In a graph, if any node connects to itself with an edge, it is called a loop. If a graph is undirected and unweighted and there is no more than one edge between any distinct two vertices of a graph and there is no loop, then this graph is called a ***simple graph***. Figure 3 is an example of a simple graph.

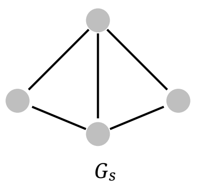


Figure 3: A simple graph (G_s) example.

If there is a path between every pair of vertices of an undirected graph, this is called a ***connected graph***. If this graph is directed, it is called a ***strongly connected graph***. FSM algorithms generally work with labeled, connected, simple graphs.

Graphs that change with the addition or deletion of nodes or edges over time are called ***dynamic graphs***.

If there is no edge between two nodes in a graph, these nodes are ***independent***. The set, which contains all the nodes that are independent from each other in the graph, is called the ***maximum independent set***.

2.2 Support measure

A support for a graph is equal to the number of occurrences of this graph in a graph dataset. A graph dataset can consist of a single large graph or multiple small graphs called transactions. The support calculation varies according to the graph dataset type.

In transactional dataset, the *support* σ ($0 < \sigma < 1$) of a graph is the ratio of the number of transactions to which this graph occurs to the total number of transactions.

In a single large graph, the support of a graph is the number of its occurrences in this dataset. However, there are some variants and subtleties, as explained in frequency calculation section 3.2.

2.3 Frequent subgraph

Given a graph $G = (V_g, E_g)$, a graph $H = (V_h, E_h)$ will be a ***subgraph*** of G if and only if the vertices and edges of graph H are a subset of the vertices ($V_h \subseteq V_g$) and edges ($E_h \subseteq E_g$) of graph G.

If the support of a subgraph is equal or greater than the user-defined minimum support threshold, then this subgraph is a ***frequent subgraph***. If a graph is frequent, all its subset must be frequent. This is called ***downward closure property***.

Let graph H be a subgraph of G, so the vertices of subgraph H are a subset of the vertices of graph G. If all edges between these subset vertices in the graph G are also found in the subgraph H, this subgraph H is also ***induced subgraph***. If a graph is ***closed graph***, none of the proper supersets of that graph can have the same support value with this graph.

2.4 Subgraph isomorphism

Suppose G and H are two graphs. There is an isomorphism between G and H, if there is a bijection f between the vertices ($f : V(G) \to V(H)$), that is, any two vertices (u and v) are adjacent in graph G, if and only if $f(u)$ and $f(v)$ are adjacent in H. These two graphs are called ***isomorphic***. They are topologically identical. An isomorphism from a graph to itself is called ***automorphism***. The problem of ***subgraph isomorphism*** is trying to find out if a graph is isomophic to some subgraph of the other graph. There are many subgraph isomorphism detection techniques [25–28]. Figure 4 is an example of isomorphic graphs.

To find a frequent subgraph in a single large graph, the embeddings of this graph is examined. Isomorphic graphs of a subgraph in a single large graph are called ***embeddings***. If two embeddings have the same set of edges, these two embeddings are identical to each other. If these two embeddings have no common edge, these embeddings are ***edge-disjoint embeddings***. An overlap graph

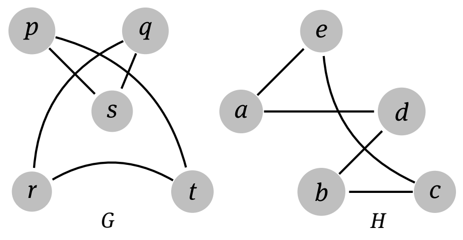


Figure 4: An example of isomorphic graphs.

is constructed based on these edge-disjoint embeddings and non-identical embeddings. To construct an ***overlap graph***, a vertex is created for each non-identical embedding of a subgraph and edges that represent pairs of non-edge-disjoint embeddings are created [10]. This overlap graph can be used to calculate the support of a subgraph. The support calculation using overlap graph is presented in the frequency calculation section 3.2.

2.5 Graph representation

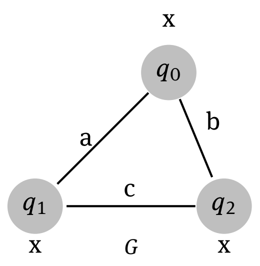
FSM algorithms generally use 2 different methods to represent graphs: adjacency matrix and adjacency list. In addition, some canonical labeling methods have been proposed to facilitate the subgraph isomorphism. The purpose of these methods is to represent each graph with a unique code.

2.5.1 Adjacency matrix

The rows and columns of the adjacency matrix represent the graph nodes. If there is an edge between two nodes (v_i and v_j), these two nodes are adjacent and (i, j) position of adjacency matrix is represented by 1 or the edge label between these two nodes, otherwise it is represented by 0. Depending on the position of the vertices in the row and column, there can be more than one adjacency matrix representing the same graph. AGM [15] and Subdue [12] algorithms use adjacency matrix to represent graphs. Figure 5 shows a graph G and its adjacency matrix.

2.5.2 Adjacency list

In an adjacency list, all nodes are kept in an array. Also, each element(node) of this array points to a list where all other nodes that are adjacent to this node are kept. FSG [16], MOFA [18] and p-MOFA [20] algorithms use adjacency



	q_0	q_1	q_2
q_0	0	1	1
q_1	1	0	1
q_2	1	1	0

Figure 5: Adjacency matrix of a graph G.

list to represent graphs. Figure 6 shows a graph H and its adjacency list.

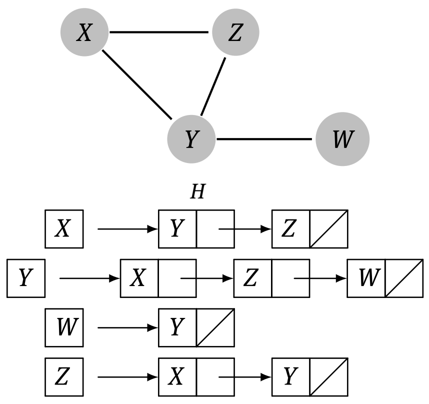


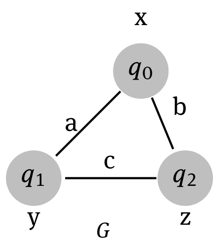
Figure 6: Adjacency list of a graph H.

2.5.3 Canonical labeling

The adjacency matrix and the adjacency list can not uniquely identify a graph. Because a graph can be represented by more than one adjacency matrix and adjacency list depending on the order in which the nodes are enumerated. A canonical labeling strategy is proposed, which can uniquely identify a graph. A label of a graph is obtained by concatenating rows and columns of its adjacency matrix. As there are multiple adjacency matrices to represent the graph, there are also multiple possible labelings of a graph. To reduce the number of possible labelings, the invariant properties of the nodes, such as the label and neighbors of a node, are used to divide to adjacency matrix into partitions [16]. Graphs are represented with minimum or maximum labels according to the lexicographic order to avoid this problem. This label is called canonical label of graph. To solve the problem of subgraph isomorphism, graphs must be represented uniquely. If canonical labels of two subgraphs are identical, these subgraphs are isomorphic to each other [16]. To identify isomorphic graphs, each graph must be represented only by one canonical label. Several different canonical labeling methods have been proposed.

2.5.3.1 CAM-Canonical adjacency matrix

In an adjacency matrix, if there are labels of nodes on diagonal entries and label of edges on off diagonal entries, this matrix is called *canonical adjacency matrix* [14]. The canonical code of a subgraph is obtained by concatenating the upper or lower triangular entries of the adjacency matrix. Since there are multiple adjacency matrices of any subgraph, there are multiple canonical codes. The minimum or maximum canonical code with respect to lexicographic order is used to represent the subgraph. FFSM [14], HSIGRAM and VSIGRAM [10] algorithms use canonical adjacency matrix to represent graphs. Figure 7 shows canonical matrix and canonical code for a graph G.



x		
a	y	
b	c	z

code = *xaybcz*

Figure 7: Canonical adjacency matrix and code of a graph G.

2.5.3.2 Minimum DFS code

The minimum DFS code method has been proposed to represent the graphs uniquely in the gSpan algorithm [17]. In this algorithm, the graph is traversed using depth first search method. While traversing a graph, each edge is assigned a unique subscript called edge code according to the discovery time of nodes.

Assume that q_a and q_b are identifiers and a and b are labels of two nodes, l_{ab} is label of edge between the nodes. If the node q_a is discovered before the node q_b according to the depth first strategy, this edge is represented by 5-tuple [17]: $\langle\, q_a, q_b, a, l_{ab}, b \,\rangle$.

The DFS code of the graph consists of all the edge codes that represent its edges. Then a single DFS code tree is constructed containing all graphs in the graph dataset. Each node of this tree represents a graph's DFS code. Since a graph can be represented by more than one DFS code, the graph is assigned to the first DFS code found by preorder search in the DFS code tree. This code is called the minimum DFS code and is used as a canonical label of this graph. CloseGraph [13], GERM [29] and p-gSpan [20] algorithms use minimum DFS code to represent graphs.

3 Process of frequent subgraph mining

The aim of Frequent Subgraph Mining (FSM) is to find all frequent subgraphs in a graph dataset. In general, the FSM consists of two phases. The first step is the generation of candidate subgraphs. The second step is the frequency calculation of the generated subgraphs to determine whether they are frequent or not.

The methods used by the algorithms examined in this study to generate candidates and calculate the frequencies are given in Table 1.

3.1 Candidate generation

The most important point to note during candidate generation is that each candidate should be generated only once. FSM is an extension of frequent itemset mining from itemsets to graph data [30]. The downward closure property used in frequent itemset mining is also used in this field to narrow the search space and avoid duplicates while generating candidates. In the frequent itemset mining, if an itemset is frequent all its subset must be frequent, but in the frequent subgraph mining if a graph is frequent all its subgraph must be frequent. The commonly used strategies for candidate generation are as follows: *level-wise join strategy*, *extension strategy* and *join and extension strategy*.

The aim of the Stream FSM algorithm [31] is to transform the single large dynamic graphs with streaming updates into streaming graph transactions. These graph transactions are mined by other FSM algorithms that use graph transactions as input to find frequent subgraphs. For this reason, the candidate generation and frequency calculation method of the Stream FSM algorithm is based on the FSM algorithm used. In this study, it is assumed that the Stream FSM algorithm applies gSpan algorithm to extract frequent subgraphs [31].

3.1.1 Level-wise join strategy

In level-wise join, two k-size subgraphs are combined and a new subgraph of $(k + 1)$-size is generated. To be able to join two k-size subgraphs, both subgraphs must have a common $(k - 1)$-size subgraph [16].

The level-wise join has some disadvantages. First, more than one candidate can be obtained by a single join operation. Second, the same candidates, can be generated because of different join operations. And the generated candidates may not provide the downward closure property.

In AGM algorithm [15], the graphs are represented by an adjacency matrix and two k-size adjacency matrices are joined to generate a new candidate. To be able to combine two matrices, the elements of these two matrices must be the same except for the last row and last column.

One of the algorithms that generate candidates using the level-wise strategy is the FSG algorithm [16]. A k-size subgraph has $k-1$ different subgraphs of size $k-1$. Instead of considering all these subgraphs when doing the join, two subgraphs with the smallest first and second canonical labels according to lexicographic order among these $(k - 1)$ different subgraphs are joined.

SUBDUE [12], HSIGRAM [10], and SEuS algorithms [11] generate candidates with level-wise join strategy.

3.1.2 Extension strategy

In the extension strategy, new connected $(k + 1)$-size subgraph is obtained by adding an edge to all possible k-size embeddings.

VSIGRAM [10] and MOFA [18] algorithms use the extension strategy.

In the MOFA algorithm, the nodes and edges of embeddings are marked. The marked nodes are extended by adding possible edges that have not been marked before.

Both algorithms preserve the connectivity of embeddings while performing the extension operation.

Rightmost path extension strategy is a special type of extension strategy. In the extension strategy, every possible edge can be added to every possible node, whereas in this method the new edge can only be added to nodes on the rightmost path. In this method, a new subtree with size $(k+1)$ is generated by adding an edge to the rightmost path of a k-subtree. A new edge can be inserted between the rightmost node and existing node on the rightmost path or between the rightmost node and a newly introduced node. In the extension strategy, new edge is added every possible way but, in this method, new edge is added only on the rightmost path. However, this method has the following disadvantage: there can be too many nodes to which the new edge can be added. This method greatly increases the complexity of this algorithm.

Table 1: Process of FSM algorithms.

Algorithms	*Candidate generation*			*Frequency calculation*						
	Level-wise join	*Extension*	*Join extension*	*Database scan*	*Transaction list*	*Embedding list*	*Occurrence list*	*MIS*	*MNI*	*MDL*
AGM [15]	√			√						
FSG [16]	√				√					
FFSM [14]			√			√				
MOFA [18]		√				√				
gSpan [17]		√			√					
CloseGraph [13]		√			√					
GERM [29]		√							√	
StreamFSM [31]		√			√					
Time evolving graph [32]		√							√	
Subdue [12]	√									√
SEuS [11]		√		√						
HSIGRAM [10]	√							√		
VSIGRAM [10]		√						√		
FSM-H [19]	√						√			
gSpan-H [21]	√						√			
p-MOFA [20]		√				√				
p-gSpan [20]		√			√					

CloseGraph [13], gSpan [17], GERM [29], Time evolving graph [32] algorithms generate candidates with rightmost extension strategy.

3.1.3 Join and extension strategy

The Fast Frequent Subgraph Mining (FFSM) [14] algorithm presents two new methods to overcome the deficiencies of level-wise and extensions strategies: FFSM-join and FFSM extension and it uses CAM to represent the graphs.

In the FFSM-join method, up to two candidates are generated instead of too many candidates. However, this method may not always enumerate all the subgraphs.

In the extension method, there are many nodes in a graph where an additional edge can be added. However, this operation is costly. In the FFSM-extension strategy, a single fixed node is determined in the CAM and a new edge is added always between this fixed node and additional node during the extension process.

3.2 Frequency calculation

To calculate the frequency of any subgraph, as we mentioned before, it is necessary to find the number of graphs that are isomorphic to this subgraph in the graph dataset. The subgraph isomorphism problem is NP complete [9] and the computational cost increases exponentially as the problem size grows. For this reason, the subgraph isomorphism testing can be performed in reasonable time only on small graphs. While some FSM algorithms perform subgraph isomorphism testing to calculate the frequency of a graph, others have suggested different methods to avoid this test, or some intuition to speed up this test.

In addition, the frequency calculation varies according to the input of the algorithm. If an input is a transactional dataset, the number of different transactions that are encountered with subgraph is used as support. There are different methods used to calculate frequency in such datasets. These methods are database scan, transaction list, embedding list and occurrence list. However, if the input is a single large graph, there are different support measures to calculate the frequencies. These measures are maximum independent set (MIS) [33], minimum image based support (MNI) [34] and minimum description length principle [35].

3.2.1 Database scan

To calculate the frequency of candidate subgraphs, for each candidate, database is scanned from the beginning to end to determine how many different transactions include this candidate. For every candidate, re-scanning the entire database is not a very efficient method. Especially scanning the large databases significantly affects the runtime of the algorithm. The AGM algorithm [15] uses the database scan method to calculate the frequency of candidate subgraphs.

In Seus algorithm [11], a k-size subgraph is obtained by extension from a $(k-1)$- or $(k-2)$-size subgraph that is called parent. Pointers to all subgraphs that are isomorphic to a subgraph is stored on the disk. When calculating the frequency of a k-size subgraph, all isomorphic graphs of the parent of this subgraph can have accessed using this pointer. Then, it is checked whether the k-size candidate subgraph can be reached by adding an edge to these graphs. The number of subgraphs that a k-size graph can obtain is the frequency of the k-size subgraph.

3.2.2 Transaction list

There is a transaction identifier list for each frequent subgraph. To calculate the frequency of a k-size graph, the intersection of the TID (transaction identifier) lists of all its $(k-1)$-size subgraphs is checked. If the intersection size is greater than the user-defined support value, the frequency is calculated by performing a subgraph isomorphism test on the transactions at this intersection. However, this method has a disadvantage. These TID lists require a lot of memory to keep them in memory and these lists may not fit in memory for big data.

The FSG algorithm [16] calculates the frequency of candidates using the TID list.

The gSpan algorithm [17] and p-gSpan [20] algorithms store the transaction list of every discovered subgraph, and instead of looking at the intersection of transaction lists while computing the frequency of a candidate, the subgraphs that are isomorphic to this candidate subgraph are searched in that transaction list.

CloseGraph [13] algorithm also uses transaction lists while calculating the frequency of subgraphs.

3.2.3 Embedding list

While calculating the frequency of candidate subgraphs, embedding lists of discovered subgraphs are stored to avoid subgraph isomorphism testing. The frequency of a candidate subgraph is determined from the number of different graphs in its embedding list. However, this method also is not suitable for big data.

MOFA [18], p-MOFA [20] and FFSM [14] algorithm use embedding lists when calculating the frequency of candidates. While MOFA stores both nodes and edges, FFSM algorithm stores only nodes.

3.2.4 Occurrence list

One of the methods used to calculate the frequency of candidate subgraphs is the occurrence list. An occurrence list contains all the embeddings of a subgraph and information about the subgraphs that correspond to these embeddings in the transactional graph dataset. While calculating the frequency of a k-size graph, instead of solving the subgraph isomorphism problem, the intersection of occurrence lists of $(k-1)$-size subgraphs of k-size graph is checked.

The differences of occurrence list from the transaction list and embedding list is that, in the transaction list, the ids of the transactions encountered with subgraphs are stored in the embedding list, all embeddings of a subgraph are stored, but in the occurrence list both embeddings and transaction ids are stored.

FSM-H [19] and gSpan-H [21] algorithms use occurrence list while calculating the frequency of a graph.

3.2.5 Maximum independent set (MIS)

One of the methods used to calculate the frequency of a single large graph candidates is the maximum independent set [33]. Since there are no transactions in a single large graph that can be scanned to find the frequency of a subgraph, firstly all the embeddings of this subgraph in the graph are found and their overlap graph is constructed. Then, on this overlap graph, an exact or some maximal independent set is found.

The HSIGRAM [10] algorithm proposes two different frequency calculations. The first frequency calculation is done as follows: the frequency of a subgraph is equal to the size of the maximum independent set in the overlap graph. The second frequency calculation is done as follows: the frequency of a k-size subgraph is equal to the frequency of the connected subgraph which has the lowest frequency among all $(k-1)$-size subgraphs of this k-size subgraph.

The minimum value of these two frequencies is determined as the frequency of the subgraph. All the embeddings of this subgraph are found in the single large graph.

In the VSIGRAM algorithm [10], while calculating the frequency of a $(k+1)$-size subgraph that was obtained from *k*-size subgraph by extension, the frequency of this *k*-size parent subgraph is used.

3.2.6 Minimum image based support (MNI)

One of the methods used to calculate the frequency of single large graph candidates is the minimum image based measure [34]. In this method, the number of unique nodes in the graph dataset that can be mapped to a node of candidate subgraph are found. This process is done for all the nodes of candidate subgraph. The minimum one is considered as the frequency of this candidate subgraph.

The GERM [29] algorithm uses a minimum image based measurement. In the Time evolving graph [32], the frequency of a candidate subgraph is the ratio of the calculated minimum value to number of total nodes of the graph dataset.

3.2.7 Minimum description length principle (MDL)

The purpose of the MDL principle is to reduce the description length of the entire dataset [35].

Subdue algorithm [12] uses the MDL principle to find frequent subgraphs instead of the frequency. The aim of this algorithm is to compress the input graph to frequent subgraphs by using the MDL principle.

The Subdue starts with all the unique vertices and generates candidate subgraphs by extending each of these vertices with a new edge in all possible ways. The discovered graphs are replaced by a single node in the input graph. The total description length is used as the frequency of a candidate. The total description length is equal to the sum of the number of bits required to represent the candidate subgraph and the number of bits required to represent the input graph after changing all candidates with a single node. The candidate that minimizes this value is considered as frequent. Discovered frequent subgraph is replaced with a single vertex in the input graph and for next iteration this graph is used as input graph. This process repeats until all possible subgraphs are represented in the compressed data.

4 Categorization of FSM algorithms

This section provides an overview of the Frequent Subgraph Mining (FSM) algorithms, which vary according to input type, dynamicity of graphs, result type, algorithmic approach, algorithmic design and graph representation. In addition to the frequent subgraph finding phases, FSM algorithms have undergone various adaptations according to the approaches used in these phases and to the characteristics of the graphs.

In this section, the evolutionary process of FSM according to the requirements and the challenges is examined and a comparison of the FSM algorithms is given in Table 2.

4.1 Input type

There are two different types of graph datasets used in FSM: transactional dataset [13–18] and a single large dataset [10–12, 29, 31, 32]. When calculating the frequency of a graph in the transactional dataset, it is necessary to calculate the number of transactions that contain this graph.

Since there are no transactions in the single large graph, how the frequency of a subgraph is calculated is an important issue. While calculating the frequency of a subgraph in a single large graph, it is calculated by counting different embeddings of this subgraph in the single large graph. One of the most important problems with single large graphs is the overlap of embeddings of a subgraph. Because the overlap graphs can cause the failure of downward closure property. Another feature that separates single large graphs from transactional graphs is the need for more memory.

Graphs used in the problem of FSM can be undirected or directed graphs and multiple edges can be allowed between the graph nodes. Because of the direction between the nodes of a directed subgraph, there are more subgraphs than the same undirected graph. For this reason, the subgraphs obtained from a directed graph are less frequent and their computation time is shorter.

The AGM [15], Subdue [12] and CloseGraph [13] algorithms are suitable for both undirected and directed graphs. SEuS algorithm [11] works on directed graph. Time evolving graph algorithm [32] allows multiple edges.

4.2 Dynamicity of graphs

There are two types of graphs used in the problems of FSM: static graphs [10, 12–19] and dynamic graphs [29, 31, 32]. Static graphs do not change over time and can be stored in a dataset. Algorithms that provide solutions for FSM problems are usually suitable for static graphs.

Table 2: Categorization of FSM algorithms.

Algorithms	*Input type*	*Dynamicity of Graphs*	*Result type*	*Algorithmic approach*	*Algorithmic design*	*Graph representation*
AGM [15]	undirected /directed graph set	static	all induced frequent subgraphs	apriori	serial	adjacency matrix
FSG [16]	undirected graph set	static	all frequent subgraphs	apriori	serial	adjacency list
FFSM [14]	undirected graph set	static	all frequent subgraphs	pattern growth	serial	CAM
MOFA [18]	undirected /directed graph set	static	all frequent subgraphs	pattern growth	serial	adjacency list
gSpan [17]	undirected graph set	static	all frequent subgraphs	pattern growth	serial	min DFS code
CloseGraph [13]	undirected graph set	static	all closed frequent subgraphs	pattern growth	serial	min DFS code
GERM [29]	undirected single graph	dynamic	all frequent subgraphs	pattern growth	serial	min DFS code
StreamFSM [31]	undirected single graph	dynamic	all frequent subgraphs	pattern growth	serial	min DFS code
Time evolving graph [32]	undirected single graph	dynamic	all frequent subgraphs	pattern growth	serial	min DFS code
Subdue [12]	undirected single graph	static	approximate/ all	pattern growth	serial	adjacency matrix
SEuS [11]	directed single graph	static	all frequent subgraphs	pattern growth	serial	adjacency matrix
HSIGRAM [10]	undirected single graph	static	all frequent subgraphs	apriori	serial	CAM
VSIGRAM [10]	undirected single graph	static	all frequent subgraphs	pattern growth	serial	CAM
FSM-H [19]	undirected graph set	static	all frequent subgraphs	apriori	parallel	adjacency list
gSpan-H [21]	directed graph set	static	all frequent subgraphs	apriori	parallel	min DFS code
p-MOFA [20]	undirected graph set	static	all frequent subgraphs	pattern growth	parallel	adjacency list
p-gSpan [20]	undirected graph set	static	all frequent subgraphs	pattern growth	parallel	min DFS code

Dynamic graphs are constantly changing graphs. In these graphs, the change may be the addition of new nodes or edges, deleting or changing existing nodes or edges. In FSM algorithms, generally the database is scanned from beginning to end for each candidate to calculate the occurrences of candidates. As the dynamic graphs are constantly changing, the size of the data is constantly changing and the increasing volume of data leads to computing and mining challenges. It is no longer possible to efficiently mine data with more than one pass. After each stream, scanning entire database is not an efficient method. For this reason, frequent subgraph mining algorithms should be designed for dynamic graphs by scanning the database only once.

Dynamic graphics can evolve over a period rather than streams. In these graphs, frequent subgraph mining method can be used to find local and structural changes. For example, if we think about a chemical dataset, the proteins may bind to each other at certain times, and this bond may break at certain times. In such cases, it may be more interesting to analyze the evolution of graph. So, frequent subgraph mining algorithms need to be carefully designed to reveal the evolution of the underlying data.

Another issue that needs attention in dynamic graphs is that when the incoming changes are added to the graphs, each edge and node in this change must be considered separately. Each change should be made as soon as possible. One of the most important issues is that an infrequent subgraph may be frequent with updating later, or a frequent subgraph may not be frequent later. The obtained frequent subgraphs should be found with as few errors as possible.

4.3 Result type

FSM algorithms can be categorized according to the result set. Some FSM algorithms gSpan [17], FFSM [14] find all the subgraphs. However, in some cases it is not useful to have all the frequent subgraphs. Instead of finding all frequent subgraphs, smaller and more meaningful frequent sets like closed or approximate (subset of all frequent subgraph) are found.

gSpan [17] algorithm is not suitable for mining large patterns. Therefore, the CloseGraph [13] algorithm, a modification of the gSpan algorithm, has been proposed and this algorithm finds all closed subgraphs and well suited for big data. However, CloseGraph [13] and Subdue algorithm [12] can miss some subgraphs. The FSG algorithm [16] finds all frequent connected subgraphs instead of finding all frequent subgraphs. SEuS algorithm [11] provides approximate or complete set of frequent subgraphs. AGM [15] algorithm finds all induced frequent subgraphs.

4.4 Algorithmic approach

FSM algorithms can be divided into two different categories according to their algorithmic approach: apriori based approach and pattern growth-based approach.

4.4.1 Apriori based algorithms

Apriori based algorithms [14–16] find all the connected frequent subgraphs. Apriori based algorithms generally consist of two steps: candidate generation and subgraph isomorphism test. Apriori based algorithms are an extension of the Apriori algorithm [30]. While Apriori algorithm works on itemsets, FSM algorithms work on graphs with the same logic.

In the first step, new candidates are generated from frequent subgraphs and checked whether they are frequent or not. These algorithms use the level-wise strategy for candidate generation. Apriori based algorithms suffer from too many candidate generations for big data. For this reason, these algorithms narrow the search space using the downward closure property. According to this property, if a subgraph is not frequent, an upper set containing it is not frequent. In the next step, it is no longer necessary to check whether any candidate set containing this subgraph is frequent or not. Especially when long patterns are present, the generation of the candidate set is still expensive. Apriori based algorithms reduce the number of candidates significantly but, these algorithms do not work efficiently in long patterns and when the minimum support threshold is small as too many candidates are generated, and this process requires a lot of database scanning. Apriori based algorithms also suffer from subgraph isomorphism testing.

4.4.2 Pattern growth-based algorithms

The second approach is pattern growth and this approach is the extension of FP-growth algorithm [36]. The aim of pattern growth-based algorithms [13, 14, 17, 18] is to find all the frequent patterns without the candidate generation and subgraph isomorphism test. This approach is based on the divide and conquer method. Instead of generating all the candidates, a new edge is added to every possible position of the existing frequent subgraph.

Apriori based algorithms suffer from too many database scans. For this reason, pattern growth-based algorithms use a more compact and smaller data structure instead of processing in database. The number of generated candidates in this approach are reduced considerably and the subgraph isomorphism test is better than the apriori based algorithms. However, this approach has a disadvantage. The same subgraph can be produced many times while adding a new edge to every possible position in the current frequent subgraph. This problem has been tried to be avoided by using the rightmost path extension strategy [13, 17].

4.5 Algorithmic design

Most of the algorithms that propose solutions to the FSM problem are suitable for working on small datasets. The existing algorithms assume that the generated candidates are small enough to fit in the main memory of the computer. When the data size gets larger, working on a single centralized machine is not an efficient method. Because the size of the input data may not be suitable for mining on a single machine, or generated candidates may not fit into the main memory of single machine. However, these algorithms are not scalable, so the main memory is bottleneck. For this reason, algorithms that work parallel [19–21] on more than one CPU have been proposed. In the parallel algorithm, the work to be done is assigned to the processes that will run parallel to each other.

In addition to FSM, there are three important issues to consider when developing a parallel algorithm. First, a parallel algorithm should be memory scalable. As the number of processes to run in parallel increases, the memory required to operate these processes should also be enough. The second issue to consider is that the tasks should be distributed equally in each process. A working time of a process may not always be predictable, so a task on a process may be completed before others. The last issue that needs to be taken into consideration is the remaining task in the other processes should be dynamically distributed to the idle processes. In this way, both the idle time of the processes is reduced, and the work is completed in a shorter time.

There are two different memory systems used when developing a parallel algorithm: shared memory systems and distributed memory systems.

4.5.1 Shared memory systems

Processes running on different machines share a common memory address space in shared memory systems. The most important advantage of these systems is that processes can communicate with each other through this memory address space. But there may be delays when the common address is on different machines with the running process.

In these methods, input data is partitioned. However, the important point here should be distributed to the workers in a balanced manner. For example, the size of data given to some workers may be small and work may be much earlier than other employees, so CPU time can be wasted. Another issue should be paid attention to the granularity of the parallelism, that is communication load of more than one processor. It should also be ensured that the shared data is consistent.

Parmol software [37] provides parallel implementation of the MOFA [18], gSpan [17], FFSM [14] and Gaston [38] algorithms. ParSeMiS tool [39] is a parallel implementation of the gSpan algorithm. p-MOFA and p-gSpan algorithms [20] are thread-based parallel versions of the MOFA [18] and gSpan [17] algorithms that use shared memory.

4.5.2 Distributed memory systems

In distributed memory systems, processes communicate with each other by network transmission or writing to a file or reading a file instead of sharing a memory space to communicate with each other. There are two programming models used in distributed memory systems.

The first is the message passing. With this method, the processes communicate with each other by sending messages over the network. For this reason, network bandwidth is one of the important factors affecting the system and network traffic should be reduced as much as possible.

The other method is MapReduce [40]. The map reduce model consists of two phases: map and reduce. The input of algorithm is assumed a (key, value) pair sets. Firstly, the map function is implemented to each (key, value) pair and emits the other (key, value) pairs then during the reduce phase the pairs that have the same key value are aggregated after the map function. Reduce function keeps these values in a sorted list and implements the reduce function to this list. In the message transfer model, the communication of process in the system is transparent to the user, but in the MapReduce method the user does not need to have detailed knowledge about the communication in an address field.

There are two parallel processing frameworks that allow big data to be distributed among computer clusters using simple programming models: Apache Hadoop [41] and Apache Spark [42] frameworks. Hadoop is an open source implementation of the MapReduce programming model and uses the Hadoop Distributed File System [43]. The Apache Spark Framework is an open-source cluster computing framework built on and extending Hadoop MapReduce. MIRAGE algorithm [44] is an iterative MapReduce-based frequent subgraph mining algorithm. FSM-H algorithm [19] is an apriori based algorithm that depends on MapReduce framework. gSpan-H algorithm [21] is a parallel implementation of gSpan algorithm [17] based on MapReduce framework.

5 Discussion of algorithms and challenges

Algorithms that work on the problem of Frequent Subgraph Mining (FSM) try to overcome some difficulties in this area. One of the most important problems encountered in solving the problem of FSM is costly candidate generation. If the size of the data is too large and the support threshold value is small, the number of candidates generated become enormous. Another problem is calculating the frequency of these candidates. To be able to calculate the frequency, it is necessary to find other subgraphs that are topologically identical (isomorphic) to the subgraph. The detection of isomorphic subgraphs is a NP-complete problem [9].

To find frequent subgraphs in a graph dataset, apriori based algorithms generate candidate subgraphs and apply subgraph isomorphism test to these candidates. Since these processes are computationally expensive, algorithms suffer from them.

To avoid the overhead of apriori based algorithms, pattern growth-based algorithms construct a DFS code tree instead of candidate generation and subgraph isomorphism testing. Each node of this tree represents a graph and these graphs are represented by minimum DFS codes. These minimum DFS codes are used to facilitate the subgraph isomorphism test. Because if the two subgraphs are isomorphic, these minimum DFS codes are the same. Pattern growth-based algorithms can efficiently generate candidates and facilitate subgraph isomorphism testing, but most pattern growth-based algorithms are not suitable for big data.

In this study, efficient support calculation methods and different support measures are examined according to the input type. Depending on the type of input, support measures vary. The aim is to provide effective and accurate frequency calculation. If an input is a transactional dataset, the number of different transactions that encountered with subgraph is used as support. In this case, one of the most inefficient methods is a database scan, because the database is scanned from beginning to end for each generated candidate. However, in other methods, the size of memory is an important challenge. Especially, as the data size increases, it requires a significant amount of memory to keep transaction, embedding or occurrence lists.

If the input is a single large graph, there are different support measures to calculate the frequencies. One of these methods is MIS support[33]. MIS support is computationally efficient but NP-hard [9] in number of graph vertices. The other method is MNI support[34]. MNI support is calculated in linear time but it may overestimate the frequency of a candidate. In terms of calculation time, MNI support is better than MIS support.

5.1 Performance comparison of FSM algorithms

In this section, the limitations of the algorithms and their performance comparison is summarized according to the information obtained from the experiments in related papers.

For each candidate, the AGM [15] algorithm scans the entire dataset from beginning to end to find out whether it is frequent or not. The FSG algorithm [16] uses canonical labels to facilitate the subgraph isomorphism problem. To create the canonical label of a graph, this algorithm suggests some heuristic methods. However, these heuristics require a large number of different edge labels to uniquely identify a graph using canonical labels. FSG algorithm outperforms the AGM algorithm [16].

The gSpan algorithm [17] contructs a DFS code tree instead of the candidate generation and uses minimum DFS code to facilitate the subgraph isomorphism. Therefore, gSpan algorithm requires less memory utilization than the FSG and outperforms it [17]. However, gSpan algorithm is not suitable for large patterns [13]. FFSM algorithm proposes two new methods for candidate generation: FFSM-join and FFSM-extension and uses CAM to facilitate subgraph isomorphism. FFSM is faster than the gSpan algorithm only on IC93 dataset [14].

MOFA algorithm [18] generates many duplicates. So, the generated frequent subgraphs may not be frequent actually. gSpan and FFSM algorithms outperform the MOFA algorithm [45].

CloseGraph algorithm [13] is based on gSpan algorithm. Since the gSpan algorithm is not suitable for large patterns, this algorithm finds only closed frequent subgraphs instead of finding all frequent subgraphs. It suggests two new concepts to narrow the search space: equivalent occurrence and early termination. In some cases, early termination may fail. Although this algorithm performs better than gSpan on large patterns, it may miss some important frequent subgraphs [13]. However, it can be ensured that the completeness of the algorithm results can be guaranteed by determining the cases where early termination failed.

The SIGRAM [10] algorithm aims to find frequent subgraphs in a single large graph. Based on the two different approaches, it proposes two new algorithms: HSIGRAM and VSIGRAM algorithms. While HSIGRAM algorithm uses apriori based strategy to generate candidates, VSIGRAM algorithm uses pattern growth-based strategy. According to the experimental results, VSIGRAM algorithm is faster than the HSIGRAM algorithm [10].

SUBDUE algorithm [12] is a heuristic algorithm and it compresses the input data by using minimum description length principle. Since this algorithm is suitable for finding small frequent subgraphs, it tends to miss large frequent subgraphs. Both VSIGRAM and HSIGRAM algorithms perform better than the SUBDUE algorithm [10].

SEuS algorithm [11] summarizes the graph dataset and the frequent subgraphs are searched in this summary. Due to the loss of data in the summarization phase, the output of algorithm is approximate frequent subgraphs which are the superset of complete frequent subgraphs. VSIGRAM algorithm outperforms the SEuS algorithm [10].

Stream FSM [31], GERM [29] and Time evolving graph algorithms [32] are FSM algorithms applied to dynamic graphics. Time-evolving graphs algorithm is the modification of GERM algorithm. The difference is that this algorithm allows the multiple edge between nodes and uses different confidence measure. Stream FSM algorithm outperforms GERM and SUBDUE algorithm on Twitter dataset [31]. The limitation of GERM and Stream FSM algorithms is the number of edges (degree of a node) that a node can have, as the developing graph grows. Because, as the degree of a node increases, the complexity of the algorithm will increase.

FSM-H [19] and gSpan-H [21] algorithms find all frequent subgraphs based on the Hadoop MapReduce software framework [41]. FSM-H algorithm generates candidates by using breadth first strategy and it applies subgraph isomorphism test. gSpan-H algorithm is a modification of gSpan algorithm. gSpan-H algorithm discovers frequent subgraphs without candidate generation and subgraph isomorphism testing. Since candidate generation and isomorphism checking is costly process, the performance of gSpan-H algorithm is better than the FSM-H algorithm [21].

p-MOFA and p-gSpan algorithms [20] are thread-based algorithms that run on a 12-processor shared memory system. p-MOFA is a modification of the MOFA algorithm and p-gSpan is a modification of the gSpan algorithm. The MOFA algorithm requires more memory usage than the gSpan algorithm because it must store all the embeddings for support evaluation. In addition, different load balancing techniques are used in these two different algorithms and these techniques significantly affect the performance of the algorithm. According to these factors, the p-gSpan algorithm performs better than the p-MOFA algorithm [20].

5.2 Current challenges of FSM

As mentioned before, an FSM algorithm has two major phases: candidate generation and frequency evaluation. Most of existing FSM algorithms are applied to small and static datasets. However, the data in today's world is changing and growing.

One of the most important disadvantages of most FSM algorithms is that they are not suitable for big data due to the costly candidate generation and subgraph isomorphism test. For this reason, existing algorithms need to be adapted to work on dynamic data. Few methods [29, 31, 32] that can work on dynamic data have been proposed.

Another problem is that as the data size increases, the number of generated candidates increases and the subgraph isomorphism testing becomes more difficult. The solution to both problems requires a lot of resources and time. Generated candidates may not fit in memory on a single machine or working with them on a single machine may not be an effective method. For this reason, methods that provide scalability and efficiency are necessary. Recent parallel algorithms have been proposed to work more efficiently and to make FSM algorithms work with big data and large patterns.

6 Conclusion

In this work, we focus on Frequent Subgraph Mining (FSM) and try to cover both early and recent literature on FSM from different perspectives. This area has many algorithms, research publications, development and application activities. Within the scope of this survey, popular FSM algorithms are investigated according to the different characteristics.

First, algorithms are examined according to the methods used in the main FSM processes, which are candidate generation and frequency calculation. Then, these algorithms are categorized by emphasizing various algorithmic features such as input type, dynamicity of graphs, result type algorithmic approach, algorithmic design and graph representation. In addition, the challenges of the FSM algorithms, the advantages and disadvantages of the methods used as well as the limitations of popular algorithms are discussed.

Nowadays, in many applications, data is either constantly changing or changing over a period. Therefore, in addition to computationally expensive candidate generation and subgraph isomorphism test, one of the problems facing FSM algorithms today is the change of data over time. To find frequent subgraphs of such data, it is necessary to adapt existing algorithms for dynamic data. Another problem of today is the size of the data. Since existing FSM algorithms are suitable for small data, there is a need for designs and methods that will enable these algorithms to work on the big data.

References

[1] Han J., Pei J., Kamber M., Data mining: Concepts and Techniques, Elsevier, 2011
[2] Chakrabarti D., Faloutsos C., Graph mining: Laws, generators, and algorithms, ACM Computing Surveys (CSUR), 2006, 38(1), 2
[3] Rehman S. U., Khan A. U., Fong S., Graph mining: A survey of graph mining techniques, Seventh International Conference on Digital Information Management (ICDIM 2012), IEEE, 2012, 88-92
[4] Aggarwal C. C., Wang H., Graph data management and mining: A survey of algorithms and applications, Managing and Mining Graph Data, Springer, 2010, 13-68
[5] Raymond J. W., Gardiner E. J., Willett P., Heuristics for similarity searching of chemical graphs using a maximum common edge subgraph algorithm, Journal of Chemical Information and Computer Sciences, 2002, 42(2), 305-316
[6] Zimek A., Assent I., Vreeken J., Frequent pattern mining algorithms for data clustering, Frequent Pattern Mining, Springer, Cham, 2014, 403-423
[7] Yan X., Yu P. S., Han J., Graph indexing: a frequent structure-based approach, Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data, ACM, 2004, 335-346
[8] Acosta-Mendoza N., Gago-Alonso A., Medina-Pagola J. E., Frequent approximate subgraphs as features for graph-based image classification, Knowledge-Based Systems, 2012, 27, 381-392
[9] Garey M. R., Johnson D. S., Computers and Intractability, New York: wh freeman, 2002, 29
[10] Kuramochi M., Karypis G., Finding frequent patterns in a large sparse graph, Data Mining and Knowledge Discovery, 2005, 11(3), 243-271
[11] Ghazizadeh S., Chawathe S. S., Seus: Structure extraction using summaries, International Conference on Discovery Science, Springer, 2002, 71-85
[12] Holder L. B., Cook D. J., Djoko S., Substucture discovery in the SUBDUE system, KDD Workshop, 1994, 169-180
[13] Yan X., Han J., Closegraph: mining closed frequent graph patterns, Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, 2003, 286-295
[14] Huan J., Wang W., Prins J., Efficient mining of frequent subgraphs in the presence of isomorphism, Proceedings of the 2003 International Conference on Data Mining, IEEE, 2003, 549-552
[15] Inokuchi A., Washio T., Motoda H., An apriori-based algorithm for mining frequent substructures from graph data, European Conference on Principles of Data Mining and Knowledge Discovery, Springer, 2000, 13-23
[16] Kuramochi M., Karypis G., An efficient algorithm for discovering frequent subgraphs, IEEE Transactions on Knowledge and Data Engineering, 2004, 16(9), 1038-1051
[17] Yan X., Han J., gSpan: graph-based substructure pattern mining, Proceedings of International Conference on Data Mining, IEEE, 2002, 721-724
[18] Borgelt C., Berthold M. R., Mining molecular fragments: Finding relevant substructures of molecules, Proceedings of International Conference on Data Mining, IEEE, 2002, 211-218
[19] Bhuiyan M. A., Hasan M. Al, An iterative mapreduce based frequent subgraph mining algorithm, IEEE Transactions on Knowledge and Data Engineering, 2015, 27(3), 608-620
[20] Meinl T., Worlein M., Fischer I., Philippsen M., Mining molecular datasets on symmetric multiprocessor systems, Proceedings of the 2006 IEEE International Conference on Systems, Man and Cybernetics (SMC'06), IEEE, 2006, 2, 1269-1274
[21] Sangle M. M. H., Bhavsar P. S. A., gSpan-H: An iterative mapreduce based frequent subgraph mining algorithm, International Journal of Advance Research and Innovative Ideas in Education, 2016, 2(5), 169-177
[22] Jiang C., Coenen F., Zito M., A survey of frequent subgraph mining algorithms, The Knowledge Engineering Review, 2013, 28(1), 75-105
[23] Lakshmi K., Meyyappan T., A comparative study of frequent subgraph mining algorithms, International Journal of Information Technology Convergence and Services, 2012, 2(2), 23-29
[24] Muttipati A. S., Padmaja P., Analysis of large graph partitioning and frequent subgraph mining on graph data, International Journal of Advanced Research in Computer Science, 2015, 6(7), 29-40

[25] Schmidt D. C., Druffel L. E., A fast backtracking algorithm to test directed graphs for isomorphism using distance matrices, Journal of the ACM (JACM), 1976, 23(3), 433-445

[26] Ullmann J. R., An algorithm for subgraph isomorphism, Journal of the ACM (JACM), 1976, 23(1), 31-42

[27] McKay B. D., Practical graph isomorphism, 1981

[28] Cordella L. P., Foggia P., Sansone C., Tortorella F., Vento M., Graph matching: a fast algorithm and its evaluation, Proceedings of the Fourteenth International Conference on Pattern Recognition, IEEE, 1998, 2, 1582-1584

[29] Berlingerio M., Bonchi F., Bringmann B., Gionis A., Mining graph evolution rules, Joint European Conference on Machine Learning and Knowledge Discovery in Databases, Springer, 2009, 115-130

[30] Agrawal R., Srikant R., Fast algorithms for mining association rules, Proceedings of the 20th International Conference on Very Large Data Bases (VLDB), 1994, 1215, 487-499

[31] Ray A., Holder L., Choudhury S., Frequent subgraph discovery in large attributed streaming graphs, Proceedings of the 3rd International Conference on Big Data, Streams and Heterogeneous Source Mining: Algorithms, Systems, Programming Models and Applications, 2014, 36, 166-181

[32] Miyoshi Y., Ozaki T., Ohkawa T., Mining interesting patterns and rules in a time-evolving graph, Proceedings of the International MultiConference of Engineers and Computer Scientists 2011 (IMECS 2011), 2011, 1, 448-453

[33] Vanetik N., Gudes E., Shimony S. E., Computing frequent graph patterns from semistructured data, In Data Mining, 2002. ICDM 2003. Proceedings. 2002 IEEE International Conference on, IEEE, 2002, 458-465

[34] Bringmann B., Nijssen S., What is frequent in a single graph?, Pacific-Asia Conference on Knowledge Discovery and Data Mining, Springer, 2008, 858-863

[35] Rissanen J., Minimum description length principle, Wiley StatsRef: Statistics Reference Online, Wiley Online Library, 2014

[36] Han J., Pei J., Yin Y., Mining frequent patterns without candidate generation, ACM Sigmod Record, 2000, 29(2), 1-12

[37] Meinl T., Wörlein M., Urzova O., Fischer I., Philippsen M., The parmol package for frequent subgraph mining, Electronic Communications of the EASST, 2007, 1

[38] Nijssen S., Kok J. N., A quickstart in frequent structure mining can make a difference, Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, 2004, 647-652

[39] Philippsen M., Worlein M., Dreweke A., Werth T., ParSeMiS - the parallel and sequential mining suite, 2011, https://www2.informatik.uni-erlangen.de/EN/research/zold/ParSeMiS

[40] Dean J., Ghemawat S., Mapreduce: simplified data processing on large clusters, Communications of the ACM, 2008, 51(1), 107-113

[41] Hadoop A., Hadoop, 2009, http://hadoop.apache.org, 2009

[42] Spark A., Apache spark: Lightning-fast cluster computing, URL http://spark.apache.org, 2016

[43] Shvachko K., Kuang H., Radia S., Chansler R., The Hadoop distributed file system, 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST), IEEE, 2010

[44] Bhuiyan M. A., Hasan M. A., Mirage: An iterative mapreduce based frequent subgraph mining algorithm, arXiv preprint arXiv:1307.5894, 2013

[45] Wörlein M., Meinl T., Fischer I., Philippsen M., A quantitative comparison of the subgraph miners MoFa, gSpan, FFSM, and Gaston, European Conference on Principles of Data Mining and Knowledge Discovery, Springer, 2005, 392-403