Frequent Subgraph Mining Algorithms for Single Large Graphs- A Brief Survey

Aarzoo Dhiman and S.K. Jain
Department of Computer Engineering
National Institute of Technology
Kurukshetra, India
aarzoodhiman@gmail.com, skj_nith@yahoo.com

Abstract— **Modeling data as a graph has proved an efficient approach considering huge amount of data and large number of relationships among them. The process of mining data sets, represented as graph structures, called graph mining is widely studied in bioinformatics, computer networks, chemical reactions, social networks, program flow structures, etc. Frequent Subgraph Mining is defined as finding all the subgraphs in a given graph that appear more number of times than a given value. Frequent subgraph mining process for single large graph consists of three phases, i.e., candidate generation, support computation and result generation and sets of techniques used in each phase. This paper provides a brief survey of the frequent subgraph mining algorithms focusing on the type of techniques they use in the algorithm in respective phases.**

Keywords— Graph, Semi-structured data, Frequent Subgraph Mining, Single Graph Setting

I. INTRODUCTION

The vast interconnections between real world objects can be best modeled as *graph* databases which try to mimic those relationships in consistent and intuitive way. As the size of data grow and data become less structured, graph databases become extremely helpful in understanding scenarios such as retails suggestion engines, protein-protein interaction networks, social network monitoring, fraud detection etc [1]. Graph mining, one of the novel approaches for mining the structured data represented by graph, has gained much attention in the last few decades. Most studied graph mining techniques are as follows: (1) Graph Classification, a type of supervised learning, (2) Graph Clustering, a type of unsupervised learning and (3) Subgraph Mining, which we will discuss here [2].

Frequent Subgraph Mining is defined as finding all the *subgraphs* in a given graph that appear more number of times than a given *support threshold*. It is a widely studied problem as it results in the recurrent structures, themes or ideas in the given graph database which can be used further for performing other graph mining applications such as graph classification, graph partitioning, graph clustering, graph correlations etc. There are two main categories in which algorithms for graph mining problem have been designed, as follows:

1) *Transactional graph setting*: In which a set of graphs is the input to the algorithm and a frequent subgraph is the one that appears in a pre-specified number of those transactional graphs.
2) *Single graph setting (mono-graph setting)*: In which a single large graph is the input to the algorithm and a frequent subgraph is the one whose *embeddings* occur for a pre-specified number of times in the graph.

Single large graph setting is considered to be a more general approach as it is capable of representing most semi-structured data very efficiently. This type has proven to be computationally more expensive, according to the literature, due to large space requirement and high time complexity. Working in this setting introduces other major complications. First is violation of *downward closure property* due to overlapping embeddings and second is less understandability of the user due to huge size of result set produced at lower support threshold. The main performance parameters used for the evaluation of a frequent subgraph mining algorithm in case of single large graph are:

- Amount of time required by the algorithm. Here time is the total time spent on CPU or the total of the time spent on data mining and support computation.
- Total number of candidate subgraphs generated.
- Size of the largest subgraph found.
- Number of the graph isomorphism computation.

In this survey a brief overview of the different types of algorithms, devised for frequent subgraph mining from single graph setting, on the basis of the different types of techniques used in the different phases of the subgraph mining process has been done. The phases are used as the ground of comparing the algorithms surveyed.

The rest of the paper is organized as follows. Section 2 briefly explains the different terminologies used in graph theory. Section 3 presents the different phases in frequent subgraph mining with the brief description of the approaches used per phase and the algorithms using the respective approach. Section 4 gives a brief discussion on the techniques used by the algorithms and some new points inferred from the literature. Section 5 concludes the work.

II. PRELIMINARIES

A graph is a set of *vertices* and *edges* representing different objects in data and relationships between objects respectively. The object definition may depend on the type of dataset being used. For example, in a twitter data set, the

978-1-5090-0673-1/16/$31.00 ©2016 IEEE

object is a user and the relationship among users is represented by edges. Similarly, in a real life XML movie database, where vertices represent the XML elements and edges represent the inheritance relations and references.

Definition 1: A labeled or attributed graph is represented as $G = (V, E, L, l)$, where V is set of vertices and E is the set of edges i.e. $E \subseteq V \times V$ and l is the function that assigns label from the set L to the vertex from the set V i.e. l: $V \cup E \rightarrow L$.

Defintion 2: A graph $S = (V_S, E_S, L_S, l)$ is subgraph of graph $G = (V, E, L, l)$ iff $V_S \subseteq V$, $E_S \subseteq E$ and $L_S(u) = L(u)$ $\forall u \in V_S \cup E_S$. A subgraph is also called substructure, subpattern in some literature.

The naïve approach to mine frequent subgraph in a single graph is as follows:

1) Find the frequent subgraph of size one. Here, size may mean the number of vertices or edges in the subgraph.
2) The subgraph is extended to form new subgraph, by adding a new vertex or edge, and then frequency of that subgraph called support is calculated and if it comes out to be more than given support threshold, the subgraph is added to frequent subgraph set.
3) Repeat step 2 until no more frequent subgraphs exist.

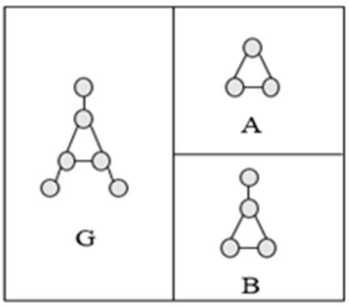
The step 2 consists of two main steps. One is candidate generation and the other is support computation, where support computation is considered to be most expensive step. The most straightforward method to calculate the support of a subgraph is to look for the isomorphism of the subgraph in the given graph. But *subgraph isomorphism* is a NP- complete problem.

Definition 3: Let $S = (V_S, E_S, L_S, l)$ be a subgraph of a graph $G = (V, E, L, l)$. A subgraph S is isomorphic to G under a bijective function $f : V_S \rightarrow V$ satisfying (a) $L_S(v) = L(f(v))$ $\forall v \in V_S$, and (b) $(f(u), f(v)) \in E$ and $L_S(u, v) = L(f(u), f(v)) \forall (u, v) \in E_S$.

The subgraph isomorphism of a subgraph S to input graph G is also called an instance, occurrence or embedding in the literature. Here, we will use the terms interchangeably. The typical way of evaluating the support of a subgraph is by counting the number of isomorphs in the given graph.

Definition 4: Downward Closure property states that for every graph G, A and B, where A is a subgraph of B and G is the input graph, the support of subgraph B in G should never be greater than support of subgraph A in G. Maintaining downward closure property is of crucial importance because it helps in pruning out the search space and without it exhaustive search is unavoidable [3].

Here as seen in the figure 1, graph A is subset of graph B, which are subgraphs of the given graph G. The graph G contains the embeddings of subgraphs A and B. According to the naïve definition support of subgraph A is 1 in G and support of subgraph B is 3 (but not independently) in G. Considering auto-morphism further doubles up the supports.



This has violated the downward closure property due to *overlapping* of embeddings.

Fig. 1. The input graph G and the graphs A and B such that A⊆B and A and B are subgraphs of graph G.

Definition 5: An overlap of two or more embeddings e.g. e_1 and e_2 of a subgraph S occurs when $e_1(V_S) \cap e_2(V_S) \neq \emptyset$.

To solve this problem only one out of all the overlapping embeddings should be counted. Therefore, the *overlap graph* (or instance graph) of the subgraph is generated and in different literatures different type of support computation measures are used, which will be discussed in further sections.

III. FREQUENT SUBGRAPH MINING

FSM process is defined as the process of finding all the frequent subgraphs from the input database that appear more than the given support threshold. With reference to literature related to FSM algorithms it was identified that this process mainly consists of three phases. These phases are Candidate Generation, Support Computation and Result Generation. Further subsections give details of different techniques being used per phase with brief description of algorithms using the respective techniques.

A. CANDIDATE GENERATION

Candidate generation is the first step in FSM process where different subgraphs are generated who are candidate for becoming the frequent subgraphs. On the basis of study of different literature, the techniques used by different algorithms for candidate generation are detailed below as follows:

1) Apriori-Based Approach

Apriori based approach was first described in [4], but it can't be used directly in single large graph setting. Apriori based approach starts from a size one graph where size may mean number of edges or vertices. Next for generating the candidate subgraph of size k, frequent subgraphs of size (k-1) are merged. Here, frequent subgraphs are those subgraphs whose support is greater than the given support threshold. Different algorithms using Aprori, distinguish themselves on the basis of different building blocks they use such as vertices, edges and disjoint paths. Apriori is mostly considered to be a BFS way of traversing the graph because of its level by level approach.

Ghazizadeh and Chawathe introduced a summary based approach to discover frequent patterns in a single graph in [5].

They presented an algorithm which was user interactive and need not access the whole data set to reveal the candidates and their count. However, SEuS faced many limitations such as the algorithm could only give efficient results if the input graph contains relatively small number of frequent graphs. Later, this limitation was addressed by Kuramochi & Karypis and they proposed two algorithms for frequent pattern mining namely VSiGram and HSiGram [6]which only differ in the type of search strategy being used i.e. DFS and BFS respectively. HSiGram joins only two frequent subgraphs of size k with minimum canonical labeling iff they don't have any subgraph in common to generate a size k+1 subgraph. On the other hand, VSiGram has removed the limitation of producing duplicate and redundant graphs occurred due to the use of BFS. VSiGram keeps track of the parent of size k while generating the candidate graphs of size k+1. The algorithm keeps on eliminating subgraphs which are not unique or which don't belong to the given parent from the set of all possible k+1 subgraphs. [7] provides implementation improvements in VSiGram by adding parallelism to the algorithm by using OpenMP. Another algorithm proposed by Vanetik et al. [8] [9] modeled the semi-structured data as a graph and introduced a new building block of disjoint paths to generate candidate graphs. They used the concept of path covers, which are edge disjoint set of paths, to generate the candidates and tried to reduce the candidates generated by using the minimal path numbers thereby reducing computation cost of support. Other variation by these authors was provided for mining maximal partially labeled frequent patterns in [10] .

2) Pattern Growth Approach

In [11], Yan & Han discussed the challenges that occur in Apriori based approach. First, the candidate generation of size k from size (k-1) subgraphs is a very complicated and costly process and second, the pruning of the false positives is costly as subgraph isomorphism is NP- Complete. To remove these limitations of apriori based approach, pattern growth approach was first introduced in [11] by Yan and Han. In this approach new subgraphs are generated by adding one edge or vertex at a time and mapping each graph to a unique minimal DFS code as its canonical label. Yan & Han announced gSpan to be the first algorithm that used DFS coding in frequent subgraph mining. A DFS code tree is constructed to represent relationships between graphs where each vertex of the tree represents a graph represented by the minimal DFS code. However, pattern growth approach may generate same subgraph many times that's why to eliminate redundancy rightmost extension code is used. gSpan was introduced for transactional setting, yet it works as the base for frequent subgraph mining in single large graph setting also.

Further in [12], Miyoshi et al. used gSpan to solve the problem of frequent itemset discovery from single graph maintaining the structure of graph. Here, the vertices of the input graph contain various quantitative itemsets. They solved the problem by using algorithm named FAG-gSpan that uses two techniques (1) gSpan [11] to enumerate frequent subgraph and (2) QFIMiner to discover frequent quantitative itemsets from the vertices of the graph pattern. In [13], Jiang et al. considered the problem of finding frequent subgraphs (patterns) which are also distributed graphically on the input graph, termed as G-Patterns. They introduced the algorithm termed G-Miner which uses DFS canonical labeling and calls a subgraph globally distributed if its instances don't share any common edge in the graph. G-Measure gives the measure of distributiveness of a subgraph in the large graph. In [14], Hellal and Romdhane used the minimal DFS code for enumerating the subgraphs but they used SMNOES as their support computation measure. In this algorithm named NODAR, they relabeled the frequent vertices with the labels in descending support and eliminated the infrequent vertices and edges in every iteration. After every iteration subgraph is extended by using rightmost path to eliminate the redundant graph generation. In [15], Elseidy et al. mapped the problem of subgraph isomorphism to a CSP problem and solved the CSP problem to enumerate the subgraphs in the algorithm named GraMi. GraMi also introduced heuristic search algorithm and optimizations of many types such as, push down pruning, unique labels, automorphism, lazy search and decomposition pruning which helps to prune out the search space and decrease the time complexity by 2 order of magnitude. Extensions of GraMi are also introduced by authors which takes into account the semantic and structural constraints and approximation of results. AGraP, introduced in [16], focuses on the inexact matching which allows structural differences in both vertices and edges. The algorithm considers the noise in the data to be the reason of some small structural differences in different patterns due to which inexact matching should be allowed. The algorithm uses similarity measure, to check the inexactness between patterns, by using the edit distance between the vertices and dissimilarity threshold.

B. SUPPORT COMPUTATION

During the candidate generation phase, the subgraphs of the given graph are generated but the subgraphs are considered to be frequent only if they exceed the support threshold given by the user. To calculate the support of a subgraph, the approach used is to check for the isomorphs of the subgraph in the graph and count the number of isomorphs. Subgraph isomorphism is considered to be NP-Complete problem and that's why this step is considered to be most expensive step of FSM. These isomorphs of the subgraph are also called instances or embeddings of the subgraph but these embeddings may overlap and can violate the downward closure property. This property is of critical importance to deal which the large search space because without downward closure property it will become difficult to prune out the search space. Broadly there exists two main approaches for support computation are explained as follows:

1) Overlap Graph

The concept of using the overlap graph was first introduced in [8] by Vanetik et al. Overlap graph is the graph in which each vertex represents the embedding in the given graph. Edge between two vertices of overlap graph exists if the embeddings of the subgraph overlap i.e. they share a common vertex. The size of *Maximal Independent Set (MIS)*

in the overlap graph is used to calculate the support of the subgraph.

Definition 6: An *Independent Vertex Set* of a graph *G* is the set of vertices which do not have any edge between any vertices belonging to independent vertex set. An independent vertex set *I* is maximal if there exists no other independent vertex set *S* with size greater than that of *I*.

Algorithms in [6] and [9] have also used the concept of MIS to calculate the support of the subgraph. In [17], Vanetik et al. gave the necessary and sufficient conditions for satisfying downward closure property based on different operations performed on the overlap graph. These operations, as the names suggest, are clique contraction, vertex addition and edge removal. They provided the proof that MIS obey downward closure property under the specified operations. In [3] Fiedler and Borglet explained that not every overlap violates the downward closure property. So they introduced the concept of *Harmful Overlap* and defined any overlap to be harmful if the overlapping embeddings are either equivalent or their ancestors were equivalent.

There are other algorithms in the literature also which have used overlap graphs but not in the same way as discussed above. Jianga et al. in [13] gave a new measure called G-Measure to calculate the support of globally distributed subgraph in a single graph. They introduced new conditions to satisfy downward closure property using different operations on instance graph than [17] which are forming clique, vertex removal and edge addition. Hellal and Romdhane extended this no MIS approach by using the new support measure called SMNOES, they gave a new way of constructing an overlap graph where vertices of the graph are the embeddings of the subgraph and the edge in an overlap graph exists if there is no overlapping between different embeddings of the same subgraph.

2) *Without Overlap Graph*

Solving MIS and constructing Overlap graphs are computationally very expensive. SEuS given in [5] uses a measure for support computation to provide user with intermediate results in the earlier phases so as to give him an essence of frequent subgraphs generated. For this estimated support computation for subgraph *S* is done, which is the minimum value in the set of counter for the edges and nodes in the isomorph *S'* of the subgraph S in level-0 summary. For calculating the accurate value the pointer to the parent of the subgraph S is used. Also, Bringmann and Nijssen gave a new measure for support computation without needing to construct an overlap graph called *Minimum Image Based Suppport (MNI)* [18].

Defintion 7: MNI based support of *p* in *G* is defined by taking the vertices of *p* which can be mapped uniquely to the minimum number of vertices in *G*.

The authors ensured that this support definition obeys the downward closure property (the proof provided in [18]). GraMi, [15] uses MNI for support counting which works efficiently and produces correct results with less complexity.

C. *RESULT GENERATION*

The FSM algorithms for single large graph can also be categorized based on the type of results generated as output. The output can be complete as well as approximate. The desirable characteristic of any algorithm is that it should be complete in the sense that not a single subgraph must be left during computation and all the candidates must exactly match its isomorphs but this requirement may pose some limitations also, such as:

- Mining of some specific kind of datasets is possible in a satisfactory amount of time e.g. SiGram [6] is given to work on sparse graphs only. Other algorithms also exist that consider input datasets with small number of connected components and degree of the vertices to be low and bounded etc.
- To serve the completeness of algorithm, numerous iterations are needed to be performed to find out subgraph isomorphism which is already known to be NP-complete hence increases the complexity of algorithm.
- The number of frequent patterns generated by complete algorithms can be huge which makes it difficult for the user to process it and mine knowledge from them [19].

So, some algorithms that give approximate results are also provided in the literature which may not result in complete set of frequent subgraphs or exact matches but the subgraphs generated as output are definitely frequent. This means these algorithms result in no false positives.

1) *Complete Results*

Very less literature has studied the problem of mining the frequent subgraphs from the single large graph that produces complete results. Algorithms such as graph mining using paths, SiGram, G-Miner, NODAR presented in [8], [6], [13], [14] respectively, are complete in the respect that they provide exact and complete results for a given data set. Literature review of the above stated algorithms has been given in the previous sections.

2) *Approximate Results*

Some algorithms exist in literature which provide approximate results or work on inexact or approximate matches. Here inexact graph match means that some differences in the candidate subgraph and its isomorph are allowed. The difference can be due to any noise or distortion which results in transformation such as addition, deletion or substitution of any node or edge [20]. AGraP, gApprox and SEuS presented in [16], [21] and [5] respectively are such algorithms defined in literature. AGrap and SEuS have already been explained above where they allow inexact matches and approximate results respectively. gApprox given in [21] allows the semantic differences (i.e. labels) and structural differences (i.e. edges), in the subgraph mining, linked in a way of inexactness. The algorithm uses two thresholds in mining i.e. one for approximation measure and other for support computation. DFS coding is used in subgraph

enumeration and mining. An extension of GraMi is provided in [15] called AGraMi that results in approximate set of frequent subgraphs ensuring the expensive performance requirements. The results given by AGraMi depend on the parameters α and β where α is inversely proportional to the approximation error.

TABLE I. FREQUENT SUBGRAPH MINING ALGORITHMS APPLICABLE ON SINGLE LARGE GRAPH WITH THEIR MERITS AND DEMERITS

Sr. no.	Name of Algorithm	Type of input graph	Phase 1	Phase 2	Phase 3	Merits	Demerits
			Candidate Generation technique used	*Support Computation technique used*	*Type of Result Generated*		
1.	SEuS(‘02)	Directed graph	Apriori	Using summary pointers	Approximate/ Complete	No need to access whole database for support computation	Creating summary becomes overhead at low support threshold.
2.	SiGram(‘05)	Undirected Sparse graph	Apriori	MIS	Approximate/ Complete	Frequent subgraph discovery methods are exact, approximate and upper bound	Memory exhausts and run-time exceeds boundary for dense graphs.
3.	Vanetik et.al.(‘06)	Directed and Undirected graph	Apriori	MIS	Complete	Less candidate subgraphs generated, hence less support computation required.	Time complexity is exponential for dense graphs.
4.	G-Miner(‘09)	Directed and Undirected graph	Pattern Growth	G-Measure	Complete	Takes global distributiveness as a measure of computation	For dense graphs algorithm run out of memory
5.	NODAR(‘12)	Undirected graph	Pattern Growth	SMNOES	Complete	Reduces subgraph isomorphisms	No major improvements for single graph setting
6.	GraMi(‘14)	Directed and Undirected graph	Pattern Growth	MNI	Approximate	2 orders of magnitude faster than other algorithms	--
7.	AGrap(‘14)	Undirected graph	Pattern Growth	MNI	Approximate	Approximately 5*(gApprox subgraphs) are generated at high similarity thresholds	Unable to handle large dense graphs (in order of thousands of vertices)

IV. DISCUSSION

Table 1 gives an organization of the frequent subgraph mining algorithms, discussed so far, along with the merits and demerits of the same. The improvements in the factors such as run-time and memory, hence the performance, is visible in the literature with the introduction of new algorithms over the years. New building blocks and efficient architectures used in different algorithms for the problem solving has major role in improvements related to performance. The performance evaluation criteria used mainly by the algorithms have been discussed in introduction section but on the basis of literature some other evaluation parameters also exist as given below.

- *Maximum size of dataset mined.* The size of the graph dataset is measured from the factors which are number of edges, nodes and distinct labels. GraMi [15] is capable of mining the twitter dataset that contains trillions of nodes and edges which could not be done by any other algorithm in present literature. Maximum size of datasets supported by other algorithms discussed here are comparatively much smaller.
- *Least threshold supported*. It has been observed from the literature that as the support threshold decreases, the performance decrements due to the increased computation because of large number of subgraphs generated. An algorithm is better if it is capable of carrying out the computations at low thresholds, GraMi [15] is one such algorithm.

In this paper, we have provided the techniques used in the different phases as ground of comparison for the algorithms. In candidate generation phase, pattern growth is a newer approach and gives better runtime to the algorithm because it uses the minimum DFS code for canonical labelling. It prevents the redundant candidate generation with help of rightmost path extension. On the other hand, Apriori needs two k-1 size frequent subgraphs to generate size k subgraph, which incurs high memory requirement. Support computation is already a very costly phase as it needs to find subgraph isomorphisms -a NP-complete problem. The cost further increases due to construction of overlap graph and then solving MIS which is NP-hard problem. So, by not using an overlap graph, algorithm can reduce the computational cost. Next, the type of results produced also influence the performance of the algorithm. There always exists a trade-off between the complete and approximate algorithms. The complete algorithms are desirable because they produce all the frequent subgraphs, but they need expensive subgraph isomorphism comparisons. Whereas, the approximate algorithms are computationally more efficient because of the inexactness used. Hence, the type of technique used is dependent on the problem for which the solution is being designed.

V. CONCLUSION

This paper provides a brief survey of the frequent subgraph mining algorithms designed for mining the frequent subgraphs from a single large graph. The state-of-the-art algorithms considered in this survey are the most frequently referred to in the literature which have made a great impact on the FSM process applicable on single large graph. It is divided into three main phases: candidate generation, support computation and result generation. The algorithms have been categorized on the basis of the different techniques they are using in different phases of the subgraph mining process and how the algorithms are using these techniques. These phases are considered to be most complex as well as costly steps in the subgraph mining.

REFERENCES

[1] D. Chakrabarti and C. Faloutsos, "Graph mining: Laws, generators, and algorithms," *ACM Computing Surveys (CSUR),* vol. 38, no. 1, p. article 2, 2006.

[2] S. U. Rehman, A. U. Khan and S. Fong, "Graph Mining: A Survey of Graph Mining," in *Seventh International Conference on Digital Information Management (ICDIM), 2012* , Macau, 2012.

[3] M. Fiedler and C. Borgelt, "Subgraph Support in a Single Large Graph," in *Seventh IEEE International Conference on Data Mining Workshops, 2007. ICDM Workshops 2007.*, Omaha, NE, 2007.

[4] R. Agrawal and R. Srikant, "Fast Algorithms for Mining Association Rules," in *20th VLDB Conference*, Santiago, Chile, 1994.

[5] S. Ghazizadeh and S. S. Chawathe, "SEuS: Structure Extraction Using Summaries," in *Discovery Science*, Lübeck, Germany, Springer Berlin Heidelberg, 2002, pp. 71-85.

[6] M. Kuramochi and G. Karypis , "Finding Frequent Patterns in a Large Sparse Graph," *Data Mining and Knowledge Discovery,* vol. 11, no. 3, pp. 243-271, 11 2005.

[7] S. Reinhardt and G. Karypis, "A Multi-Level Parallel Implementation of a Program for Finding Frequent Patterns in a Large Sparse Graph," in *Parallel and Distributed Processing Symposium*, Long Beach, CA, 2007.

[8] N. Vanetik, Ben, Beer, Gudes, E. and Shimony, S.E., "Computing frequent graph patterns from semistructured data," in *IEEE International Conference on Data Mining, 2002. ICDM 2003 Proceedings.* , 2002.

[9] E. Gudes, Ben, Beer, S. Shimony and N. Vanetik, "Discovering Frequent Graph Patterns Using Disjoint Paths," *IEEE Transactions on Knowledge and Data Engineering,,* vol. 18, no. 11, pp. 1441 - 1456, 2006.

[10] N. Vanetik and E. Gudes, "Mining frequent labeled and partially labeled graph patterns," in *20th International Conference on Data Engineering, 2004. Proceedings.*, 2004.

[11] X. Yan and J. Han, "gSpan: graph-based substructure pattern mining," in *IEEE International Conference on Data Mining, 2002. ICDM 2003.*, 2002.

[12] Y. Miyoshi, T. Ozaki and . T. Ohkawa, "Frequent Pattern Discovery from a Single Graph with Quantitative Itemsets," in *IEEE International Conference on Data Mining Workshops, 2009. ICDMW '09.*, Miami, FL, 2009.

[13] X. Jianga, H. Xiong, C. Wang and A.-H. Tan, "Mining globally distributed frequent subgraphs in a single labeled graph," *Data & Knowledge Engineering,* vol. 68, no. 10, p. 1034–1058, 2009.

[14] A. Hellal and L. B. Romdhane, "NODAR: mining globally distributed substructures from a single labeled graph," *Journal of Intelligent Information Systems,* vol. 40, no. 1, pp. 1-15, 2013.

[15] M. Elseidy, E. Abdelhamid, S. Skiadopoulos and P. Kalnis, "GraMi: frequent subgraph and pattern mining in a single large graph," *Proceedings of the VLDB Endowment ,* vol. 7, no. 7, pp. 517-528 , 2014.

[16] M.-T. F. José, M. Flores-Garrido, Jesús-Ariel and Carrasco-Ochoa, "AGraP: an algorithm for mining frequent patterns in a single graph using inexact matching," *Knowledge and Information Systems,* vol. 44, no. 2, pp. 385-406, 6 5 2014.

[17] N. Vanetik , S. E. Shimony and E. Gudes, "Support measures for graph data," *Data Mining and Knowledge Discovery,* vol. 13, no. 2, pp. 243-260, 2006.

[18] B. Bringmann and S. Nijssen , "What Is Frequent in a Single Graph?," in *Advances in Knowledge Discovery and Data Mining*, vol. 5012, Osaka, Japan, Springer Berlin Heidelberg, 2008, pp. 858-863.

[19] M. Kuramochi and G. Karypis, "GREW- a scalable frequent subgraph discovery algorithm," in *Fourth IEEE International Conference on Data Mining, 2004. ICDM '04*, 2004.

[20] D. J. Cook and L. B. Holder, "Substructure Discovery using Minimum Description Length and Background Knowledge," *Journal of Artificial Intelligence Research,* vol. 1, no. 1, pp. 231-255, 1993.

[21] C. Chen, X. Yan, F. Zhu and J. Han, "gApprox: Mining Frequent Approximate Patterns from a Massive Network," in *Seventh IEEE International Conference on Data Mining, 2007. ICDM 2007.*, Omaha, NE, 2007.