Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 89 (2016) 378 – 385

Procedia Computer Science

Twelfth International Multi-Conference on Information Processing-2016 (IMCIP-2016)

Optimizing Frequent Subgraph Mining for Single Large Graph

Aarzoo Dhiman* and S. K. Jain

National Institute of Technology, Kurukshetra 136 119, India

Abstract

Frequent subgraph mining (FSM) is defined as finding all the subgraphs in a given graph that appear more number of times than a given value. It consists of two steps broadly, first is generating a candidate subgraph and second is calculating support of that subgraph. When the input to FSM algorithm is a single graph, calculating support of subgraph needs identifying its isomorphisms in the input graph. Identifying subgraph isomorphisms is NP-Complete problem. Evidently, fewer the number of candidates, fewer the support computations needed. In this paper we present a filtration technique that reduces the number of candidate subgraphs thereby reducing the overall time complexity by 7 to 18 % experimentally.

Peer-review under responsibility of organizing committee of the Organizing Committee of IMCIP-2016

Keywords: Frequent Subgraph Mining; Graph; Optimization; Single Graph; Subgraph Isomorphism.

1. Introduction

The huge size and connections among data items make it more challenging to store and query using relational models. Modeling data as *graph* generates an expressive and general-purpose structure. Graphs exploit the connectivity among data in form of relationships to fasten the processing. The increased work on graph databases is driven by the huge success of graph based business models in form of social graphs, web graphs etc.[1]

Analysis of data is of crucial importance when amount of data is huge. *Knowledge discovery* from data is the primary goal of any data mining tool. When the data is complex and has a lot of relationships, the need to discover *structural knowledge* from data also becomes important. *Frequent subgraph mining* (FSM) algorithms play an important role in further expanding the use of data mining techniques to graph based datasets. It is major research theme in data mining to generate recurrent structures, themes, ideas in a given graph. It has major applications in motif detection, social network monitoring, fraud detection etc.[2] Other applications of FSM include acting as the intermediate output of other graph mining tasks such as graph classification, graph indexing, graph clustering etc.[3]

FSM is defined as finding all the *subgraphs* in a given graph that appear more number of times than a given value. Briefly working of a FSM algorithm is as follows: The input to FSM algorithm is a graph dataset G and user defined *minimum support* (min_sup) and the output is the set of *frequent subgraphs*. The naïve FSM algorithm consists of two steps. First, to find out all the *candidate subgraphs* from G and second, to calculate their *support*. The search for candidate subgraphs starts with a single node or edge. New edges are added iteratively to the frequent subgraphs to

*Corresponding author. Tel.: +91 -7876145644
E-mail address: aarzoodhiman@gmail.com

Peer-review under responsibility of organizing committee of the Organizing Committee of IMCIP-2016
doi:10.1016/j.procs.2016.06.085

generate new candidates. Support is calculated for each and every candidate. Candidate subgraph is said to be frequent if its support is more than or equal to *min_sup* value.

Graph datasets are available in two types. First where the dataset comprises of a number of small graphs called the *transactional setting* e.g. biological and chemical datasets etc.[4] Second where dataset comprises of a *single massive graph* e.g. social networks, computer networks etc.[5] Single graphs are considered to be more general data model but expensive because of the repetitions and redundancy among data. In our work, we are considering only the single graph setting. The FSM algorithm is different for both the types due to different definitions of support as given below:

- *Transactional graph setting*: Frequent are all the subgraphs with support in a fraction of corresponding graphs.
- *Single graph setting (mono-graph setting)*: Frequent are all the subgraphs with frequency atleast a certain number of times in the large graph.

Motivation: An algorithm GraMi[3] using pattern growth approach for candidate generation and *CSP* to calculate support, which are known to be computationally most efficient take $O(2^{N^2} \cdot N^n)$ time where N and n are number of nodes in graph G and subgraph S. This time complexity is noticeably exponential to the problem size and that's why there is need of *optimizations* to improve the execution performance.

In this work, we propose a *filtration* method as optimization that works with every FSM algorithm. Our filtration technique exploits the property of *repeating edges* of subgraph to eliminate the infrequent subgraphs from FSM process without calculating the exact support value.

Contribution:

- An *upper-bound* measure is proposed here as a filtration method to optimize FSM.
- The filtration method is general purpose. That is, it can be applied with any FSM algorithm.
- The filtration method removes some infrequent subgraphs in the process without reaching the support computation phase.
- Experimental evaluation of the performance shows between 7 to 18% reduction in time of FSM.

Organization of Paper: The rest of the paper is organized as follows. Section 2 and 3 gives a brief background to FSM terminologies and literature. Our work is explained in Section 4 and Section 5 shows the performance improvements. Section 6 concludes the work described.

2. Preliminaries

Most of the concepts in graph modelling are directly taken from graph theory. 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.

Figure 1 gives an example representing a clipping of a *labelled* movie database, where each node represent a *director* entity. Here A, C, H, T (*action, comedy, horror and thriller*) are *node labels* representing the type of movie directed and each *edge label* represent the number of movies directed in collaboration. Such a graph is useful in mining the collaborations among the different kind of movie directors. For this example, frequent subgraphs are the collaborations which are most evident from the given dataset.

Definition 1: A *labelled* 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.

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 $(a) L_S(v) = L(f(v)) \forall v \in V_S$, $(b)(f(u), f(v)) \in E$ and $L_S(u, v) = L(f(u), f(v)) \forall (u, v) \in E_S$.

From Fig. 1, subgraph S has isomorphs in G as (d_3, d_4, d_7), (d_3, d_7, d_8), (d_4, d_5, d_6), (d_4, d_6, d_7). The *subgraph isomorphism problem* is that the nodes in a set of graphs may match in variety of ways. These matches can be

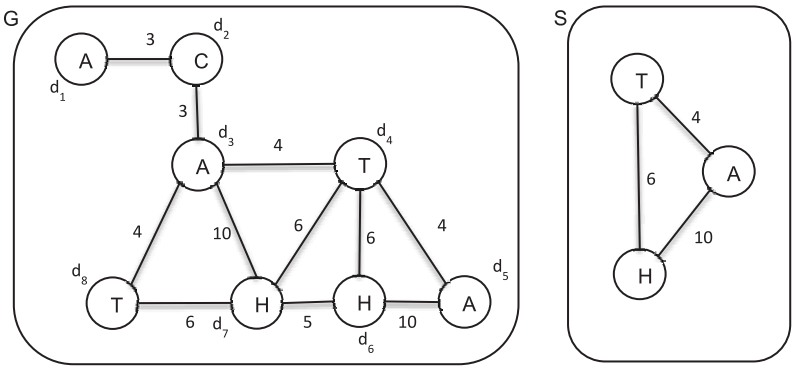

Fig. 1. The Input Graph G and Subgraphs S of G.

exponential in number in terms of the number of nodes. An isomorph is also called an *instance, occurrence* or *embedding* in the literature.

The typical way of evaluating the support of a subgraph is by counting the number of isomorphs in the given graph. In case of single graph setting the above definition doesn't always hold true. Here as seen in the Fig. 1, considering single node as a subgraph, T appears 2 times and $T - H$ appears 3 times. *Anti-monotonicity* states that if size of the subgraph is increasing then the corresponding support value must decrease or remain same. Maintaining anti-monotone property is of crucial importance because it helps in pruning out the search space and without it exhaustive search is unavoidable. This violation is due to *overlapping* among the isomorphisms of subgraph. That's why *anti-monotone support metrics* are used to calculate support of a subgraph[6]. An anti-monotone support metric is the one that follows anti-monotone property.

Definition 4: Anti-monotone 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.

Definition 5: An overlap of two or more isomorphs 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 isomorphs should be counted. Therefore, the *overlap graph* (or *instance graph*) of the subgraph is generated and in different literatures different type of anti-monotone support metrics are used as given in next section.

3. Related Work

There are many studies on mining frequent subgraphs from graph datasets in the literature. The existing research on finding frequent subgraphs in graph datasets fall under two categories: transactional graph setting and single graph setting. In the rest of the section the algorithms developed for candidate generation and support computation are given separately.

The candidate subgraphs are generated using the bottom-up approach. It starts with finding all the subgraphs with single edge. Candidate generation techniques can be broadly classified into two main categories, one that follows *Apriori based approach* (uses Breadth First Search) and other that follows *pattern-growth approach* (uses Depth First Search). In Apriori based approach, to generate a candidate subgraph of size k, frequent subgraphs of size $k - 1$ are merged. Ghazizadeh and Chawathe introduced a *summary based approach* called SEuS to discover frequent patterns in a single graph[7]. SEuS only gives efficient results when the frequent subgraphs generated are less in number. 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. 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. In 2002, Yan & Han discussed the challenges that occur in Apriori based approach. An algorithm gSpan[10] was proposed then, that uses pattern-growth

based approach. In this approach new subgraphs are generated by adding one edge at a time and mapping each graph to a unique minimal DFS code as its canonical label. In 2014, 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[3].

Next step is to compute the support of the candidate subgraph. Naive definitions of support have the problem that they are not anti-monotonic; thus they cannot be used effectively in subgraph mining, as anti-monotonicity is required to prune the search space. An intuitive support measure – size of *maximum independent set (MIS)* of the overlap graph – is proposed and shown to be admissible by Vanetik, Shimony and Gudes[11]. Overlap graph is the graph in which each vertex represents the isomorph of subgraph in the given data graph. Edge between two vertices of overlap graph exists if the isomorphs of the subgraph overlap i.e. they share same vertex. The use of MIS as a support measure was first suggested in Vanetik *et al.*[8], and was shown to be useful as a major component of an apriori-based algorithm for graph mining. It was later used by Kuramochi and Karypis[6]. Fiedler and Borgelt[12] defined the concept of *harmful overlapping* such that when the isomorphs are created from the same ancestor then only it destroys anti-monotonicity. Bringmann and Nijssen[13] gave a support measure *Minimum Image based (MNI)* that is computationally less expensive but provides a superset of the results of the alternative metrics.

There are other algorithms in the literature also which have used overlap graphs but not in the same way as discussed above. Jiang *et al.*[14] gave a new measure called G-Measure to calculate the support of *globally distributed subgraph* in a single graph with introduction of new conditions to satisfy anti-monotonicity under different operations on overlap graph[11] which are forming clique, vertex removal and edge addition. Hellal and Romdhane[15] 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 isomorphs of the subgraph and the edge in an overlap graph exists if there is no overlapping between different isomorphs of the same subgraph.

Optimizations that eliminate the infrequent candidate in the initial phases and speed up the support computation are given in the algorithm GraMi[3]. These optimizations are push-down pruning, lazy search, unique labels, decomposition pruning and auto-morphism. Push down pruning exploits the property of extension by eliminating the domains of child subgraph which have been eliminated in parent subgraph already. Unique labels optimization is applied on the subgraphs that consist of labels that are not repeating. Auto-morphism is an isomorphism of a graph to itself. This optimization decreases the search space by pruning out equivalent branches whenever an auto-morphism exists. Lazy search optimization considers the fact that if the computation takes longer then some specified time bound (specified in the algorithm) then it gives an intuition that the subgraph may result out to be infrequent. If search is timed out, the algorithm stores the search state in the timed-out-search set of nodes with incomplete check. These searches will only be resumed when the non-timed out cases are not sufficient to show that a subgraph is frequent. In decomposition pruning to reduce the problem size, the algorithm decomposes the input subgraph S into a set of distinct subgraphs *Set*. In contrast to subgraph extension, a set called *Set* is constructed by removing one edge at a time from S and adding to *Set* the connected component that includes that edge.

4. Optimized Frequent Subgraph Mining

4.1 Frequent subgraph mining algorithm

Broadly, FSM consists of two steps: 1. Candidate Generation and 2. Support Computation. The input to FSM is a labelled graph G and a user defined minimum support min_sup and output is frequent subgraph set F. The naïve approach of FSM is as follows:

1. Find and store all the frequent subgraphs with one edge from G.
2. The subgraph is extended to form new subgraph, by adding a new edge. Calculate its support and if it comes out to be more than min_sup, the subgraph is added to F.
3. Repeat step 2 until no more frequent subgraphs exist.

The space and time complexity of FSM algorithm is determined by complexity of candidate generation and support computation. A graph G with N nodes requires $O(2^{N^2})$ time to find candidate subgraphs[3], which is exponential in problem size. Candidate Generation computes all such subgraphs and support computation calculates the support of

each subgraph. The subgraph with $k+1$ edges is generated by extending frequent subgraphs with k edges. The extension is performed recursively for subgraph g until all the frequent subgraphs with g embedded are discovered[10]. Number of subgraphs generated vary inversely with *min_sup*. Lower the *min_sup*, higher is the number of subgraphs generated.

When a candidate subgraph has been generated, next step is calculating its support. To calculate the support all the isomorphisms of the subgraph in the input graph are identified and counted. Identifying subgraph isomorphism is a NP-Complete problem. In literature no better than $O(n^k)$ polynomial time bound is known[16]. That's why it is of crucial importance to optimize a FSM algorithm.

Evidently, for fewer candidates, fewer support computations are needed to be performed. That's why it is desirable to generate candidates as few as possible. This work proposes filtration as optimization to FSM. It eliminates some of the candidate subgraphs from the FSM process without letting them reach support computation phase. The parameter used for the filtration process is upper-bound to the support. Next section gives the detailed introduction of the filtration process and upper-bound measure.

4.2 Upper-bound calculation – the filtration method

Some subgraphs with two or more edges may contain same edge appearing multiple times. We exploit this property of subgraphs of edge repetition to perform the filtration. The upper-bound (u_s) to support of a subgraph is the maximum possible value of the support any subgraph can have. It is calculated for all the subgraphs that has repeating edges by using the support values of the repeating edge. If u_s comes out to be less than *min_sup*, the subgraph is considered to be infrequent hence can be eliminated from further processing.

It is being called the filtration method because it filters out some of the infrequent subgraphs without needing calculate their support. As stated earlier computation of support needs solving a NP-Complete problem i.e. subgraph isomorphism problem, filtering out some of the infrequent subgraphs prior to computing support prove beneficial.

Principle: The upper-bound works on the principle that only frequent edges are allowed to extend a frequent subgraph ensuring anti-monotone property. That edge of the subgraph which occurs minimum number of times in the graph decides the support of the subgraph. Only the frequent edges are allowed to appear in any candidate subgraph thus repeating edges are the deciding factor of the support of a subgraph. Due to the overlapping of the subgraphs the repeating edge cannot provide the exact value of support and hence gives the upper-bound to the support

$$\text{Mathematically, } u_s = \frac{\text{support of repeating edge in } G}{\text{number of times edge repeats itself in } S}, \tag{1}$$

where G and S are input graph and candidate subgraph respectively.

The upper-bound calculation takes input from the intermediate outputs of FSM algorithm. As illustrated in the example given in Fig. 2, edge (X, Z, a) is occurring twice in the subgraph S. Upper-bound for the subgraph $u_s = 1$. If *min_sup* is given to be 2, then S is infrequent and hence no support calculations is done for S. In case of two or more such repeating edges, minimum of all the u's is taken as upper-bound.

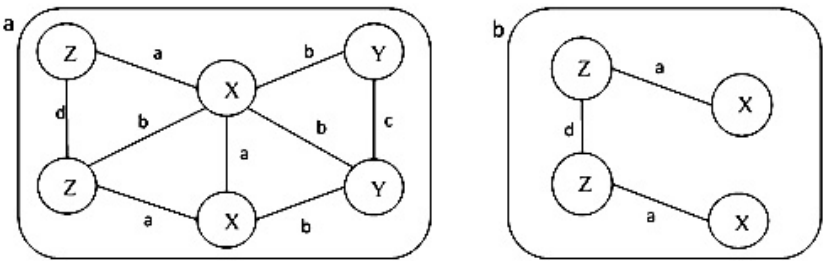

Fig. 2. (a) A Sample Graph Data G and (b) Subgraph S of Graph G.

Algorithm 1: FrequentSubgraphMining (G, min_sup)
Output: set of frequent subgraphs F
1. $F \leftarrow \emptyset$
2. Let F' be the set of frequent subgraphs with one edge.
3. **For each** $e \in F'$ **do**
4. $F \leftarrow F \cup$ SubgraphExtension(e, G, min_sup, F'F
5. Remove e from F' and G
6. **Return** F

Algorithm 2: SubgraphExtension (s, G, min_sup, F'F
Output: all the frequent subgraphs of G that extend s
1. **If** $s \neq \min(s)$
2. **Return**
3. Insert s to F
4. set $CS \leftarrow \emptyset$
5. **for each** $e \in F'$ **do**
6. Extend s using e. let it be S.
7. **If** S is not already generated do
8. $CS \leftarrow CS \cup S$
9. **For each** $c \in CS$ **do**
10. **If** c contains repeating edge **do**
11. Calculate $u_c = \frac{support\ of\ repeating\ edge\ in\ G}{number\ of\ times\ edge\ repeats\ itself\ in\ S}$
12. **If** $u_c < min\_sup$ **do**
13. Remove c from CS
14. **Else if** $sup(c) \geq min\_sup$ **do**
15. $F \leftarrow F \cup$ SubgraphExtension(c, G, min_sup, F'F
16. **Return** F

Fig. 3. The Pseudo-Code of Optimized FSM Algorithm.

As given in Fig. 3 algorithm 1 is the main graph mining loop and algorithm 2 performs the subgraph extension. Algorithm 1 starts with storing all the subgraphs with single edge in set F'. Step 3–5 ensures that all the edges are extended to generate candidate subgraphs. Extension procedure starts with a call to Algorithm 2. It uses the pattern-growth based approach to extend the subgraphs. The input subgraph S is extended only if it is the minimum DFS code (step 1–2). Subgraph S is extended using the edges from F' and all the extensions are saved in the candidate set CS (steps 5–8). All the subgraphs that satisfy the condition given in step 10 are removed from the candidate set (step 11). Support is calculated for the remaining subgraphs in CS and recursive call to algorithm 2 is made for frequent subgraphs (step 13). Step 10–13 removes some of the infrequent subgraphs from the candidate set CS and support is calculated only for the rest of subgraphs in CS.

Complexity: Total time complexity incurred due to introduction of filtration is dependent on two things:

1. *Cost to find the repeating edge in candidate subgraph s*. It would need traversing the subgraph. Complexity of traversing a graph using DFS is $O(n + e)$, where n and e are number of nodes and edges respectively.
2. *Cost to calculate value of* u_s. Value of u_s is calculated from the values which can be derived from other intermediate outputs. For example, the support value of repeating edge can be calculated by using the values saved during creation of F'.

Hence overall complexity incurred is $O(n+e)$ where the size of subgraph (i.e. n and e) is considered to be negligible as compared to size of input graph.

5. Performance Analysis

To evaluate the performance, comparison with a non-optimized FSM algorithm for undirected graphs given in GraMi[3] is made. The algorithm uses pattern growth[10] for candidate generation and MNI support measure[13] for support computation. The implementation is performed for subgraphs with two edges considering the fact that most subgraphs get eliminated early in the process. Completeness of the proposed optimization is ensured by matching the size of output i.e. the number of frequent subgraphs generated. The parameters used to evaluate the performance are 1) time required by algorithm and 2) number of subgraphs filtered out. All the experiments are performed using Java JRE v1.6.0 on Windows 7 machine with quad-core running at 2.30 GHz with 6 GB RAM and 1 TB disk.

Table 1. Characteristics of the datasets used.

Sr. No.	Dataset	# Nodes	Distinct Node Labels	# Edges	Node Labels Represent	Edge Labels Represent	Density
1.	Citeseer	3,312	6	4,732	Computer since area of the publication	Measure of similarity between corresponding pair of publications	Medium
2.	Citation	29,014	742	81,353	Document ID	Citation relation	Sparse

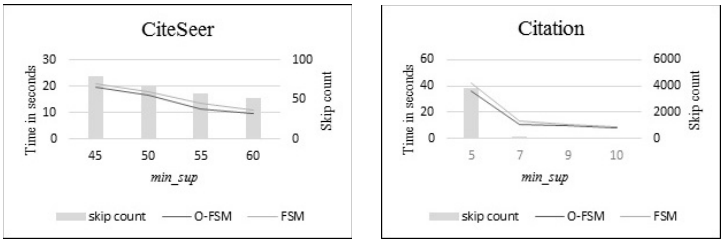

Fig. 4. The Line Graphs Representing the Variation of Time with *min_sup* Values on Two Datasets (a) CiteSeer and (b) Citations.

The experiments are performed with different *min_sup* values on two real graph datasets. The datasets used are citeseer (*http://cs.umd.edu/projects/linqs/projects/lbc*) and citation (*http://www.cs.cornell.edu/projects/kddcup/datasets.html*). Characteristics of the datasets are given in the Table 1.

Figure 4 gives the results of optimization on the datasets which depicts time taken (in seconds) by algorithm and the number of candidates eliminated due to optimization (skip count) against minimum support. In both the figures the dark line represents results from the optimized-FSM, light line represents the results from simple FSM algorithm and the bars represent the total number of candidates eliminated using O-FSM at different *min_sup* values. Different behaviors are shown by the algorithms on both the datasets because of their density. Citeseer is of moderate density as the time difference for $min\_sup = 45$ and $min\_sup = 60$ is approximately 10 seconds, given in Fig. 4(a). Citation is sparse dataset as the time difference for $min\_sup = 5$ and $min\_sup = 7$ is approximately 40 seconds, given in Fig. 4(b). It means that for sparse datasets with very less decrease in the *min_sup* the time taken and number of frequent subgraphs increase drastically.

The performance of the algorithm degrades with decreasing *min_sup* because of generation of a large number of candidate subgraphs. The performance of the algorithm also depends on the density of the dataset. The optimization introduced to FSM reports time reduction between 7–18%. The best performance of the optimization comes when the size of the subgraph is least and most of the infrequent subgraphs are filtered out in initial iterations only.

6. Conclusions

In this paper, we investigate the frequent subgraph mining algorithm and the factors behind its high time complexity. We discussed the need of optimization in FSM process. A novel graph based filtration method to optimize the mining process is introduced here. The implementation of the proposed work is performed for subgraphs with two edges on real datasets. The results of implementation show between 7 to 18% reduction in time complexity. For future work, the implementation can be extended to subgraphs with more than two edges and the upper-bound can be made tighter. Further work can be done to check the applicability of the optimization for multi-attribute graphs with parallel and self-edges.

References

[1] I. Robinson, J. Webber and E. Eifrem, Graph Databases, (2013).
[2] D. Chakrabart and C. Faloutsos, Graph Mining: Laws, Generators, and Algorithms, *ACM Computer Surveys*, vol. 38, p. 2, March (2006).

[3] M. Elseidy, E. Abdelhamid and S. Skiadopoulos, GRAMI: Frequent Subgraph and Pattern Mining in a Single Large Graph, *Proceedings VLDB Endowment*, vol. 7(7), pp. 517–528, (2014).
[4] M. Kuramochi and G. Karypis, Frequent Subgraph Discovery, *Proceedings 2001 IEEE International Conference on Data Mining, (ICDM)*, pp. 313–320, (2001).
[5] C. Aggarwal and H. Wang, Managing and Mining Graph Data, US-Springer, vol. 40, (2010).
[6] M. Kuramochi and G. Karypis, Finding Frequent Patterns in a Large Sparse Graph, *Data Mining and Knowledge Discovery*, vol. 11(3), pp. 243–271, (2005).
[7] S. Ghazizadeh and S. Chawathe, SEuS: Structure Extraction using Summaries, *Discovery Science*, pp. 71–85, (2002).
[8] E. Gudes, S. E. Shimony and N. Vanetiks, Discovering Frequent Graph Patterns using Disjoint Paths, *IEEE Transactions on Knowledge and Data Engineering*, vol. 18(11), pp. 1441–1456, (2006).
[9] N. Vanetik, E. Gudes and S. E. Shimony, Computing Frequent Graph Patterns from Semistructured Data 2002, *Proceedings of IEEE International Conference on Data Mining*, 2002, pp. 458–465, (2002).
[10] X. Yan and J. Han, gSpan: Graph-Based Substructure Pattern Mining, *Proceedings of Data Mining*, 2002, ICDM 2003, vol. 1(d), pp. 721–724, (2002).
[11] N. Vanetik, S. E. Shimony and E. Gudes, Support Measures for Graph Data, *Data Mining and Knowledge Discovery*, vol. 13(2), pp. 243–260, (2006).
[12] M. Fiedler and C. Borgelt, Subgraph Support in a Single Large Graph, In *Proceedings – IEEE International Conference on Data Mining, ICDM*, pp. 399–404, (2007).
[13] B. Bringmann and S. Nijssen, What is Frequent in a Single Graph?, In *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 5012, LNAI, pp. 858–863, (2008).
[14] X. Jiang, H. Xiong, C. Wang and A. H. Tan, Mining Globally Distributed Frequent Subgraphs in a Single Labeled Graph, *Data Knowledge Engineering*, vol. 68(10), pp. 1034–1058, (2009).
[15] A. Hellal and Romdhane L. Ben, NODAR: Mining Globally Distributed Substructures from a Single Labeled Graph, *Journal of Intelligent Information Systems*, vol. 40(1), pp. 1–15, (2013).
[16] V. Lipets, N. Vanetik and E. Gudes, Subsea: An Efficient Heuristic Algorithm for Subgraph Isomorphism, *Data Mining and Knowledge Discovery*, vol. 19(3), pp. 320–350, (2009).