I.J. Information Engineering and Electronic Business, 2016, 4, 1-9
Published Online July 2016 in MECS (http://www.mecs-press.org/)
DOI: 10.5815/ijieeb.2016.04.01

Top-k Closed Sequential Graph Pattern Mining

K. Vijay Bhaskar
GITAM University/CSE, Visakhapatnam, 530045, India
E-mail: vbreddy.vijay@gmail.com

Dr. R.B.V Subramanyam
NIT Warangal/CSE, Warangal, 506004, India
E-mail: rbvs66@nitw.ac.in

Dr. K. Thammi Reddy
GITAM University/CSE, Visakhapatnam, 530045, India
E-mail: thammireddy@gitam.edu

S. Sumalatha
NIT Warangal/CSE, Warangal, 506004, India
E-mail:katam.suma@gmail.com

Abstract—Graphs have become increasingly important in modeling structures with broad applications like Chemical informatics, Bioinformatics, Web page retrieval and World Wide Web. Frequent graph pattern mining plays an important role in many data mining tasks to find interesting patterns from graph databases. Among different graph patterns, frequent substructures are the very basic patterns that can be discovered in a collection of graphs. We extended the problem of mining frequent subgraph patterns to the problem of mining sequential patterns in a graph database. In this paper, we introduce the concept of Sequential Graph-Pattern Mining and proposed two novel algorithms SFG(Sequential Frequent Graph Pattern Mining) and TCSFG(Top-k Closed Sequential Frequent Graph Pattern Mining). SFG generates all the frequent sequences from the graph database, whereas TCSFG generates top-k frequent closed sequences. We have applied these algorithms on synthetic graph database and generated top-k frequent graph sequences.

Index Terms—Data mining, graph mining, frequent sequential patterns, closed sequential patterns.

I. Introduction

Frequent graph patterns are substructures that appear in a graph data set to a frequency not less than a user-specified threshold [9]. Structural forms such as subtrees, subgraphs, sublattices are referred as substructures. A frequent structural pattern is a substructure that appears frequently in a graph database. Sequential pattern mining, the mining of frequently occurring sub-sequences as patterns, was introduced in [13] and has become an important problem in data mining. Sequential pattern mining discovers frequent subsequences as patterns. According to the GSP algorithm [14], sequential pattern mining is to find all sequences whose support is greater than the user-specified minimum support, such sequence is called a frequent sequence. In SPADE [8], the set of all frequent sequences is discovered by reducing database scans and it minimizes I/O costs. The GSP and SPADE follow candidate generation and test approach.

The PrefixSpan algorithm [5] is a pattern-growth approach to sequential pattern mining and it follows divide-and-conquer strategy. BIDE [6] is an algorithm used for mining frequent closed sequences without candidate generation. These algorithms [5,6] grow patterns by constructing projected databases to reduce the search space for a pattern. In this paper, we investigated the problem of mining sequential patterns in graph databases. Our approach finds sequential patterns occurring in graph databases.

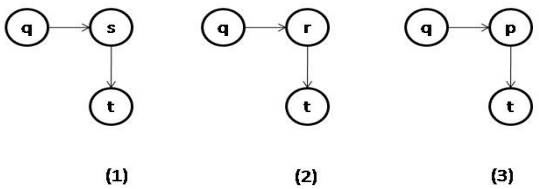

Fig.1. Three graphs where each vertex represents a web page

In "Fig. 1" each node represents a web page. Let us consider that the edges of the given graphs represent the order of visiting the web pages. For example, Graph1 represents the sequence q,s,t. Similarly Graph2 represents the sequence q,r,t and Graph3 represents the sequence q,p,t. Observing the three given graphs, it is clear that page t is accessed after page q(not immediately after q). The resulting pattern is <qt> which is not a subgraph. Given the above three graphs as input, any existing subgraph mining algorithm will not generate <qt> as a pattern. Many graph mining algorithms exist to find the frequent subgraphs, maximal frequent subgraphs, closed frequent subgraphs, and constraint-based closed frequent subgraphs. The sequence <qt> is not considered as an

output pattern by the existing graph mining algorithms. Though <qt> is not a subgraph, it represents a pattern showing a sequence q followed by t. In this paper, we proposed an algorithm to find sequential patterns from graph databases known as Sequential Graph-Pattern Mining.

We proposed SFG(Sequential Frequent Graph pattern mining) algorithm to mine frequent sequences in a graph database. As the number of edges in a graph increases, the number of sequential patterns also increases exponentially. This requires further analysis of frequent sequential graph patterns. To overcome this difficulty, we proposed TCSFG(Top-k Closed Sequential Frequent Graph pattern mining) algorithm to generate top-k closed sequential graph patterns.

The rest of the paper is organized as follows. Section II describes the related work in subgraph mining. Section III present problem definition and section IV describe SFG, TCSFG algorithms. Our experimental results and performance analysis are presented in section V and our work is concluded in section VI.

II. Related Work

Many algorithms exist in the literature for mining frequent graph patterns [1,7,9,16,18]. AGM [1] algorithm efficiently mines frequently appearing induced subgraphs from a graph data database. Experimental results in [1] show that AGM finds all frequent induced subgraphs containing 300 chemical compounds in 40 minutes to 8 days, as the minimum support threshold varied from 20% to 10%. FSG[9] algorithm uses a sparse graph representation which minimizes both storage and computation. The performance of FSG was worse where the number of vertex and edge labels was small. AGM [1] and FSG [9] are Apriori-based algorithms for mining frequent substructures from graph data. Apriori-based algorithms generate subgraph candidates from frequent subgraphs and prunes false positives. These algorithms [1,9] suffer from the problem of subgraph isomorphism test and candidate generation. Candidate generation and pruning false positives are costly.

The gSpan [18] was the first algorithm to discover all the frequent subgraphs without candidate generation and it prunes false positives. gSpan discovers frequent substructures without candidate generation. A new lexicographic ordering among graphs was built in [18] and it maps each with a unique minimum DFS code as its canonical label. gSpan finds both frequent subgraphs and frequent induced subgraphs. It [18] uses rightmost extension technique to minimize the generation of the same subgraphs and explores the depth-first search in frequent subgraph mining.

Frequent subgraph mining is a challenging issue due to the generation of an exponential number of frequent subgraphs. Closed graph pattern mining was introduced in [16] to mine only closed frequent graph patterns. A graph g is closed if there do not exist any proper super graph of g with the same support as g. CloseGraph [16] mines closed frequent graph patterns using the concepts of equivalent occurrence and early termination to prune the pattern search space. It [16] reduces unnecessary subgraphs and also increases the efficiency of the mining process in the presence of large graph patterns. TSP [12], TFP [4] and CloSpan [17] are the recent work for closed sequential pattern mining. CloSpan is the first algorithm to solve closed sequential pattern mining problem. It [17] mines frequent closed sequential patterns and produces significantly less number of discovered sequences. The CloSpan mines long frequent sequences with low minimum support and without any information loss.

Mining top-k frequent closed patterns without minimum support was introduced in [4]. Top-down and bottom-up combined Fp-tree mining strategy was developed in [4] to raise the support and to discover closed frequent patterns. TSP algorithm mines top-k frequent closed sequential patterns of length no less than min_len, the minimum length of each pattern. It [12] doesn't require the minimum support threshold, it makes use of the length constraint and outperforms the closed sequential pattern mining algorithm that make use of minimum support threshold.

Several algorithms were proposed previously ranging from mining graph patterns with constraints [2,11,19] and without constraints [6,18] for mining closed graph patterns [16]. Frequent graph-pattern mining may result in a large number of patterns. To make the mining more effective, constraints are often used to confine the pattern search. In gPrune algorithm[2] the frequent graph patterns are generated based on the constraints such as density and diameter. The Density of a graph is defined as the ratio of the edge set to vertex set. The diameter of a graph is the maximum length of the shortest path over all pairs of vertices. The problem of incorporating structural constraints in mining frequent graph patterns was solved in gPrune.

Finding closed frequent graphs with connectivity constraints in relational graphs was introduced in [19]. Two approaches, namely CLOSECUT and SPLAT were proposed to speed up the mining process. The former was pattern growth approach and it has better performance on patterns with high support and low connectivity, later approach was pattern reduction approach and it achieves better performance for the high connectivity constraints. These [19] algorithms mines closed frequent graphs with connectivity constraints in relational graphs. DS-search algorithm [11] mines frequent subgraphs of limited diameter and symmetry. It [11] employs the tree decomposition structure of database graphs during the mining process and generates more structurally interesting patterns in the database.

Recently two sequential techniques[3,10] were developed to solve the problem of graph-coloring. Graph-coloring concept [10] was used in processor allocation to represent the busy processor and the busy processors are mapped in the graph using different colors. The authors in [3] investigated the problem of Star coloring and they used DNA sequence to construct a solution space for the star coloring problem

Two algorithms RP-FP and RP-GD were proposed in

[7]. These algorithms mine a representative set that summarizes frequent subgraphs. In these algorithms [7] a representative set RS is mined such that any frequent subgraph is covered by a representative in RS, and the value of |RS| is minimized. RP-FP and RP-GD algorithms have been proposed to summarize graph patterns efficiently using the concept of δ-cover and generates δ-jump patterns for a user specified δ. Even though there exist many subgraph mining algorithms, the problem of sequential pattern mining of graphs has not been investigated in the literature.

In this paper, we proposed a new algorithm SFG to mine sequential patterns from a graph database using pattern growth approach. We also extended our algorithm to mine top-k closed sequential patterns. We demonstrate that generating top-k closed frequent sequences reduce the number of output graphs without losing much information.

III. Problem Definition

In this section, we define sequential graph pattern and frequent sequential graph pattern mining problem.

A Labeled directed Multigraph can be represented as a tuple, G= (V,E,L,F) where V denotes a set of vertices, E denotes a set of edges, L denotes a set of labels and F is a labeling function that assigns labels to the vertices and edges. Every edge is a 4 tuple <s, d, Gid, lb>, where s is the source vertex, d is destination vertex, Gid is Graph id and lb is an edge label. A sample Graph data set is shown in "Fig. 2".

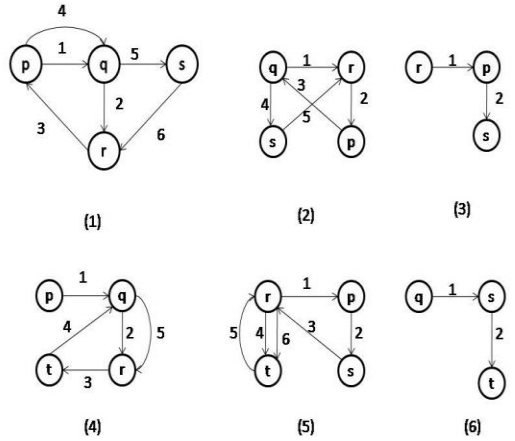

Fig.2. A sample graph data set

Definition 1 (Graph Sequence)

An ordered collection of edges in a graph is called a graph sequence. Every edge is assigned a time constant (time of the creation of the relationship between the vertices) as a label. A Graph Sequence can be represented as,

$GS=<(v_a,v_b),(v_b,v_c),\ldots.(v_l,v_m),(v_m,v_n)>$ where v_a, v_n are the source and destination of the Graph sequence and each (v_i,v_j) represents an edge from v_i to v_j. Compact representation of Graph sequence is given as $GS=<v_a,v_b,v_c,\ldots.v_n>$.

Definition 2 (Length of the Graph Sequence).

Given a Graph Sequence $GS =<v_a,v_b,v_c,\ldots.v_n>$ Where each v_i represents a vertex. Length of the Graph sequence is the number of edges in the sequence.

Definition 3 (Support of Graph sequence)

Support of Graph Sequence GS is the total number of graphs containing the sequence GS as a subsequence.

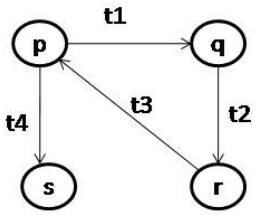

Fig.3. A sample graph

Graph sequence for the graph in "Fig. 3" is <(p-q),(q-r),(r-p),(p-s)> where t1,t2,t3,t4 are edge labels such that $t_1<t_2<t_3<t_4$. Its compact representation is <pqrps>. Length of the sequence is 4.

Definition 4 (Projected database of a sequence S)

Let S be a sequential pattern in a Graph database GD. The S projected database is the collection of subgraph sequences in GD with S as the first occurring sequence.

Definition 5 (Projected database of a vertex)

Given a Graph Database GD = {G_1, G_2,… Gn}, The projected database of a vertex V is the set of subgraphs in which the sequence of the subgraph starts with V.

PD_v = { SG_i / i ε 1,2,..n and v is the starting vertex in each SG_i}.For example, in the given graph database, the sequence that starts with vertex p as the source forms projected database of p. In "Fig. 2" edge 1 of Graph 1, edge 3 of Graph 2, edge 2 of Graph 3, edge 1 of Graph 4 and edge 2 of Graph 5 forms the sequence starting with p. Instead of storing the projected database separately for every vertex, considering the space constraints, we store only the <Graph-Id, Edge-Id> pairs. Hence p-projected database is <(1,1),(2,3),(3,2),(4,1), (5,2)>

Definition 6 (Support of a vertex)

Support of a vertex V is the number of sequences that start with V. A graph may contain multiple subsequences starting with V, but the count is only 1 for every graph.

Definition 7 (Canonical form)

Given a multigraph G_i with n vertices, m edges and a graph sequence $GS <v_1,v_2,\ldots.v_n>$, Canonical form of G_i is denoted as $(v_1,v_2,i,1),(v_2,v_3,i,2),\ldots.(v_{n-1},v_n,i,m)$

Definition 8 (Sequential Graph Pattern)

Given a graph sequence, a Sequential Graph Pattern is all possible subsequences.

For the Graph given in "Fig. 3" Graph Sequence is

<pqrps>, the possible sequential graph patterns are
<pq>,<pr>,<ps><pqr>,<pqp>,<pqs>,<pqps>,<pqrp>,<pqrs>,<pqrps>,<prp>,<prs>,<prps>,<qr>,<qp>,<qs>,<qrp>,<qrs>,<qrps>,<rp>,<rs>,<rps>

Definition 9 (Frequent Sequential Graph Pattern Mining)

Given a Graph database and minimum support, Frequent Sequential Graph-Pattern Mining is to find all the frequent sequences from the graph database.

Table 1. Canonical representation of Graph database

Graph-Id	Graph Sequence (Ordered Edges)	Canonical form
1	p→q, q→r,r→p,p→q,q→s,s→r	(p,q,1,1),(q,r,1,2),(r,p,1,3), (p,q,1,4),(q,s,1,5),(s,r,1,6)
2	q→r, r→p, p→q, q→s,s→r	(q,r,2,1),(r,p,2,2),(p,q,2,3), (q,s,2,4),(s,r,2,5)
3	r→p, p→s	(r,p,3,1),(p,s,3,2)
4	p→q, q→r, r→t, t→q, q→r	(p,q,4,1),(q,r,4,2),(r,t,4,3), (t,q,4,4),(q,r,4,5)
5	r→p, p→s, s→r, r→t, t→r, r→t	(r,p,5,1),(p,s,5,2),(s,r,5,3), (r,t,5,4),(t,r,5,5), (r,t,5,6)
6	q→s, s→t	(q,s,6,1),(s,t,6,2)

Table 2. Projected database of vertices

Vertex v	Projected Database	Support (v)	Nodes visited from v	Frequent Sequential Length-1 patterns
P	{(1,1),(2,3),(3,2),(4,1),(5,2)}	5	<r:7,q:5,s:4,t:3>	<pr>,<pq>,.<ps>,<pt>
Q	{(1,2),(2,1),(4,2),(6,1)}	4	<r:6,s:3,p:2,t:2>	<qr>,<qs>,<qp>,<qt>
R	{(1,3),(2,2),(3,1),(4,3),(5,1)}	5	<p:4,s:4,q:3,t:3>	<rp>,<rs>,<rq>,<rt>
S	{(1,6),(2,5),(5,3),(6,2)}	4	<r:4,t:3>	<sr>,<st>
T	{(4,4),(5,5)}	2	<r:2,q:1>	<tr>

Definition 10 (Top-k Closed Graph Sequence)

A Graph sequence S is said to be a closed graph sequence if S is frequent and there is no proper super sequence with the same support. A closed graph sequence is said to be a top-k closed sequence of minimum length *l* if there exists no more than (k-1) closed graph sequences whose length is at least *l* whose support is higher than that of S.

IV. ALGORITHMS

In this section, we present SFG and TCSFG algorithms to generate frequent sequential graph patterns and Top-*k* closed frequent sequential graph patterns.

A. Algorithms Description

A Graph database is given as input to each of these algorithms. The concept of SFG algorithm generates set of sequential frequent graph patterns. To reduce redundant patterns, we followed an order based on the frequency of sequences in the database. Sequences with higher frequency are grown first, followed by sequences of lower frequency.

B. SFG

SFG algorithm generates set of sequential frequent graph patterns. The main steps include the generation of the set of all frequent length-1 sequences S', pruning infrequent edges and vertices from Graph database D, sorting S' in descending order of its frequency, and growing each edge in the sequence.

Algorithm 1. SFG

Input: A Graph database GD, a minimum support threshold min_sup.
Output: Set of sequential frequent graph patterns SP.
1: Scan the Graph database once,
   a. Find the projected database of each vertex v in the input vertex set V and let it be PD_v. Find the support count of each vertex.
   b. Find the set of vertices V_u that can be visited from each vertex v and also find the support count of the sequence<v,u> where u ε V_u .
2: Sort the vertex set V in descending order of their support count and insert them in a priority queue Q. Do not insert the vertex whose support count is less than the min_sup.
3: For every vertex v in the priority queue Q
4: For every vertex u ε V_u
5: a. If the support count of the sequence <vu> is greater than or equal to minsup threshold then add the sequence <vu> to S.
   b. SP ← SP U S.
6: Sort S in descending order of their supports.
7: SequenceMining (PD_v, S)

In Line 1 of the algorithm, single scan of the graph database is done. The projected database of each vertex and their support counts are calculated. For every sequence that grows with a vertex, it is ensured that only its projected database is scanned. This reduces the search space and scan time. Find the set of vertices that can be visited from each vertex v and the support count of length-1 sequences. Line 2 of the algorithm removes the projected database of an infrequent vertex. The vertex whose support count is less than the min_sup threshold is an infrequent vertex. The remaining vertices are sorted in descending order of their support count. Lines (3-5) finds length-1 frequent sequences and calls the subprocedure SequenceMining to find the frequent sequences of length

more than one. SFG algorithm iterates until there are no vertices that can grow the sequence.

Subprocedure1. SequenceMining(PD,S)

Input: Projected database PD, Sequential Pattern S
Output: Set of sequential frequent graph patterns SP.
1: if S is empty then return.
2: For every sequence S′ in S
PD_s←Find_projected_database(PD,S′)
3: Scan the projected database PD_s and find the vertices visited from the sequence S′ and their support counts.
4: For every vertex v visited from S′,
a. Add the sequence <S′v> to S″ only if the support count of the sequence <S′v> satisfies the min_sup threshold.
b. SP← S″.
5: SequenceMining(PD_s , S″,min_sup)

SequenceMining is a recursive procedure that grows every sequence in S. Lines 2-3 of the procedure finds the projected database of the sequences in S and scans the projected database of each sequence to find the vertices visited from the sequence and the support count of the new sequence. Line 4 of the procedure grows the sequence and returns if there is no sequence to be grown. The recursive procedure finds length-2 frequent sequences in the first run, length-3 frequent sequences in the second run and so on. It finds the set of all frequent sequences of length more than one and performs a one edge growth of the sequence in each recursive call.

Subprocedure2. Find_projected_database(PD,S′)

Input: Projected database PD, Sequential Pattern S′, where S′=<v_a,v_b,……v_n>
Output: PDS, where PDS is a Projected database of S′.
1: PDS←Φ
2. for every graphid, edgeid pair (i,j) in PD
3: Scan graph i from j to find k, where k is the first occurrence of v_n.
4: PDS←PDS ∪ (i,k)
5: return PDS

Example: Given the graph sequence database as shown in Table 1 with min_sup=2, frequent sequential patterns are mined in the following manner.

1. Scan the graph database once and find the projected database PD_v of each vertex and their support counts. The database is divided into 5 partitions, the first partition starts with p as the source vertex, the second partition starts with q as the source and so on.
2. Find the vertices visited from each vertex and their support count. Find length 1 frequent graph sequences. Projected database and frequent length 1 graph sequences are shown in Table 2.
3. Vertices visited from vertex p and their support count is <r:7, q:5, s:4, t:3>. Length 1 frequent graph sequences are <pr:7,pq:5,ps:4,pt:3>.The sequence <pr:7> represents that vertex r is visited 7 times after vertex p. Similarly <pq: 5> represents that vertex q is visited 5 times after p and so on.
4. Descending order of vertices based on their support count is <p:5>,<r:5>,<q:4>,<s:4>,<t:2>. Call the sub procedure SequenceMining with the Projected database of the vertex p and the sequences grown from p, <pr>,<pq>,<ps>,<pt> as input.
5. Grow the sequence with the highest support first. In our example, grow the sequence <pr> first. Scan the projected database of p to construct the projected database of <pr>. Projected database of p is { (1,1), (2,3), (3,2), (4,1), (5,2)}. Subprocedure2 is called to find the projected database of the sequence <pr> as shown below.

   a. Scan (1,1)as follows, the first occurrence of p as a source vertex in graph 1 from the edge 1 is 1. Now check the first occurrence of r as a source after the first occurrence of p, it is found at 3. Note the graph-id, edge-id pair (1,3).
   b. Similarly, scan (2,3) as follows, the first occurrence of p as a source vertex in graph 2 from edge 3 is 3. Now check the first occurrence of r as source after the first occurrence of p, it is not found. This indicates that the sequence <pr> cannot be grown in graph 2.
   c. Similarly, scan (3,2) and the sequence <pr> cannot be grown in graph 3.
   d. Scan (4,1) and note the pair (4,3), the first occurrence of r as a source in graph 4 after first occurrence of p is 3.
   e. Scan (5,2) and note the pair (5,4), the first occurrence of r as a source in graph 5 after first occurrence of p is 4.

Now the list of pairs noted in the steps a to e form the projected database of the sequence <pr>, {(1,3),(4,3),(5,4)}.

6. Scan the projected database of <pr> and find the vertices visited from <pr>.
   pr <t:3, q:2 ,p:1, s:1> , vertex t is visited 3 times after <pr>, vertex q is visited 2 times , vertex p is visited 1 time, and vertex s is visited 1 time. Remove the infrequent sequences <prp>, <prs>. Length-2 frequent sequences obtained so far in descending order of their supports are <prt:3> <prq:2>.
7. Grow the sequences <prt>, <prq> in the similar manner.

All the frequent sequential graph patterns generated from the vertex p are shown in "Fig 4". The Time complexity of SFG depends on the time taken for scanning a graph database and a number of subsequences generated. It is given by t_g+ O (st_p), where t_g is the time taken to scan graph database, s is the number of subsequences and t_p is the time taken to scan the

projected database of each subsequence.

C. TCSFG

The number of sequential patterns generated for a given *min_sup* increase exponentially with an increase in the average number of edges in the graph database. To reduce the output search space, we proposed TCSFG algorithm which generates top-k closed sequential graph patterns.

Given a Graph database, minimum support, and k number of patterns, Top-k Closed Sequential Graph Pattern Mining is to find only top-k closed frequent sequences from the graph database.

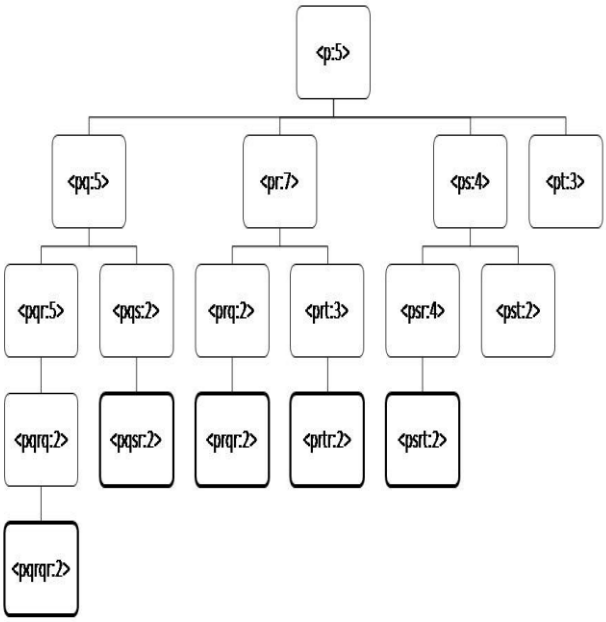

Fig.4. Frequent sequential graph patterns generated from vertex p.

Algorithm 2. TCSFG

Input: A Graph database GD, minimum support threshold min_sup, minimum length of the sequence min_len, number of closed patterns k .

Output: Top-k closed sequential patterns.

1: Scan the Graph database GD to find projected databases PD of all the vertices and their support counts.

2: Sort the vertices in descending order of their supports.

3: Find all frequent Length-1 sequences S and add all of the sequences to the priority queue, Q.

4: Top-kClosedSeqMining(PD,Q)

Line 3 of algorithm2 sorts all frequent length 1 sequences in descending order of their supports. In our example, out of 16 sequences, 15 are frequent as shown in Table 2. Line 4 of algorithm2 calls the sub-procedure Top-kClosedSeqMining to mine top-k sequences whose length is not less than min_len.

Subprocedure3.Top-kClosedSeqMining(PD,S)

Input: Projected database PD, Set of Sequential Patterns S

Output: Top-k Closed sequential frequent graph patterns CSP.

1: if S is empty then return

2: if a number of closed sequential graph patterns is equal to k then return.

3: For every sequence S′ in S

PD_s←find_projected_database(PD,S′, min_sup)

4: Scan the projected database PD_s and find the vertices visited from the sequence S′ and their support counts.

5: For every vertex v visited from S′,

Add the sequence <S′v> to S″ only if the support count of the sequence <S′v> satisfies the min_sup threshold.

6. If there is no sequence S″ with support equal to the support of S′ and if the length of the sequence S′ is not less than *l* then add S′ to set of closed patterns.

CSP←CSP U S′.

7: Top-kClosedSequenceMining(PD_s,S'')

Let s′ be the number of closed sequences obtained and s′<<s where s is the number of frequent sequences. The time complexity of TCSFG is given by t_g+O(s′t_p) , where t_g is the time taken to scan the graph database and t_p is time taken to scan the projected database of each subsequence.

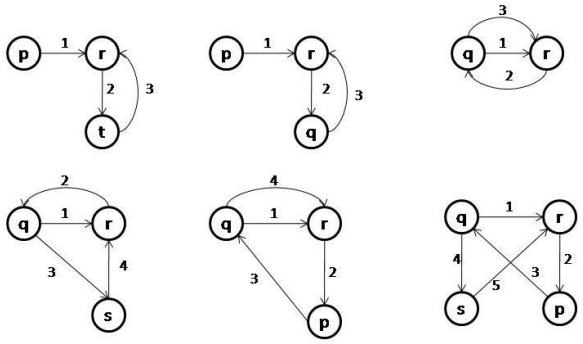

Fig.5. Top-k closed graphs

Example: Given the graph sequence database as shown in Table 1 with min_support=2, min_len =3 , k=6, top-k closed sequential patterns are mined as follows:

1. Descending order of frequent length 1 Sequences are, <pr:7>,<qr:6>,<pq:5>,<ps:4>,<rp:4>,<rs:4>,<sr:4>,<pt:3>,<qs:3>,<rq:3>,<rt:3>,<st:3>,<qp:2>,<qt:2>,<tr:2>.
2. Select the first sequence <pr:7> and grow. Length2, length3, sequences generated from <pr> are <prt:3>,<prq:2>,<prtr:2>, <prqr:2>. Among these 4 sequences only <prtr:2> and <prqr:2> are added to set of closed patterns. <prq:2> is not

closed as it is having a super sequence <prqr:2> with the same support. <prt:3> is closed, but the length of the sequence is less than *l*. The Closed patterns generated from the sequence <pr:7> are highlighted as shown in "Fig.4".

3. The next sequence to be grown is <qr:6>. Closed Sequences whose length greater than or equal to *l* are <qrqr:3>, <qrqsr:2>, <qrpqr:2>, <qrpqsr:2>. The algorithm stops as the number of sequences obtained till now are 6.

The Top-k closed output graphs are shown in "Fig. 5".

V. Results

We implemented the SFG and TCSFG algorithms and tested them on synthetic dataset produced by a graph database generator [15]. It is based on the IBM Quest Synthetic Data Generation Code for Association and sequential patterns.

The graph generator generates the data sets based on the four parameters: D be the total number of graphs in the database, V be the number of vertex labels and E be the number of edge labels, T be the average size of each graph based on the number of edges and M be the average density of each graph which is defined as the number of edges in the graph divided by the number of edges in a complete graph. "Fig. 6" shows the result where the size of the data set (D) is varied between 100 and 1000 graphs. Other values of the parameters used are: V = 20, E = 20 , T = 20 and M=0.3.

"Fig. 6" and "Fig. 7" shows the variations in the number of sequential graph patterns generated and the time taken by the SFG and TCSFG algorithms as the minimum support is varied. This shows that in the case of SFG algorithm the growth of frequent sequences increases exponentially with reduced minimum support and hence more analysis time. This might result in less scalability for large graphs because the number of subsequences increases exponentially.

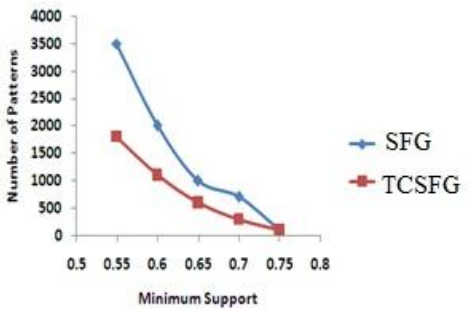

Fig.6. Number of patterns generated with respect to change in minimum support

These results point us to the problem of reducing these output sequences by generating top-k closed sequences as we discussed in TCSFG. This gives us an idea of how important it is to reduce the number of frequent sequences in the output using constraints, some of which are based on the support, graph structure and generating summarized patterns. These constraints are added in a TCSFG algorithm to generate top-k closed graph sequences. The running time and the number of patterns generated for TCSFG for the same input database is reduced by a factor when compared to SFG for smaller values of minimum support. "Fig. 8" shows the running time of SFG and TCSFG algorithms with varying number of graphs. During this experiment, the minimum support threshold is kept 20% of the size of the graph data set. These experimental results show that top-k closed patterns are generated in less time compared to all the sequential patterns with varying number of graphs as input. We also tested performance of the TCSFG algorithm by changing the value of k. We found that running time of TCSFG algorithm is varying linearly with the k value. "Fig. 9" shows the running time of the TCSFG algorithm for different values of k.

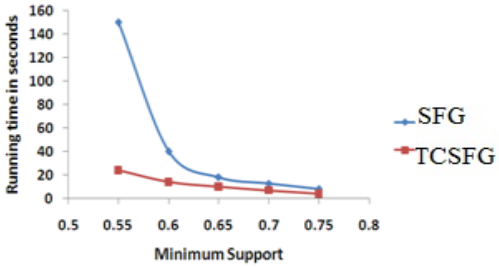

Fig.7. Performance of SFG and TCSFG

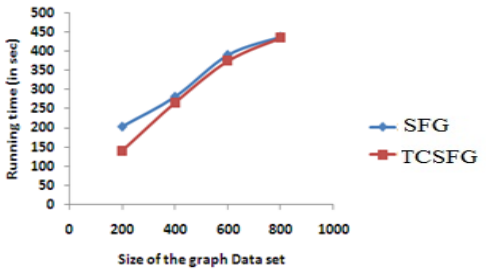

Fig.8. Running time with 20% minimum support

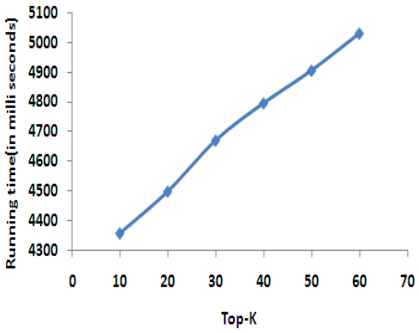

Fig.9. Performance of TCSFG on different k

As the size of the graph database increases, the number of frequent graph sequences increases much faster than the number of frequent closed graph sequences. The effectiveness of an algorithm depends on the number of useful patterns generated. As shown in the results, useful patterns without much loss of information can be obtained using TCSFG algorithm.

VI. Conclusion

In this paper, we studied the problem of mining sequential patterns and closed sequential patterns in large graph data sets. To the best of our knowledge, the problem of mining closed graph sequence is not dealt with much and this is the first piece of work to mine Top-k closed sequential graph patterns. We proposed two algorithms, SFG generates all the frequent sequences from the graph database, whereas TCSFG generates top-k frequent closed sequences. These algorithms use the concept of projected database for a graph to reduce the search space and an order based on the frequency of the patterns to generate the set of all patterns. These algorithms were verified on a synthetic database [15]. Based on this study, we conclude that mining top-k closed sequential graph patterns are preferable than the traditional closed graph mining. We further extend our study to push constraints in generating top-k closed sequential graph patterns.

References

[1] A. Inokuchi, T. Washio, and H. Motoda. "An apriori-based algorithm for mining frequent substructures from graph data," In Proc. PKDD, ser. LNCS, Springer, Vol.1910,pp. 13-23, July 2002. "doi: 10.1007/3-540-45372-5_2".

[2] F. Zhu, X.Yan, J. Han, and P. S.Yu, "GPrune: A constraint pushing framework for graph pattern mining," In Proc. PAKDD, ser. LNCS, Springer, vol. 4426, pp. 388–400, May 2007. "doi: 10.1007/978-3-540-71701-0_38".

[3] G. Sethuraman, and Kavitha Joseph. "Star Coloring Problem:The DNA Solution," International Journal of Information Technology and Computer Science, Vol. 4, No.3, pp.31-37, Apr.2012."doi: 10.5815/ijitcs.2012.03.05".

[4] J. Han, J. Wang, Y. Lu, and P. Tzvetkov. "Mining top-k frequent closed patterns without minimum support,". In Proc. ICDM 2002, Maebashi, Japan, pp. 211-218, Dec. 2002. "doi: 10.1109/ICDM.2002.1183905".

[5] J. Pei, J. Han, B. Mortazavi-Asl, J. Wang, H. Pinto, Q. Chen, U.Dayal, and M.C.Hsu. "Mining sequential patterns by pattern growth: Prefix span approach," IEEE Transactions on Knowledge and data engineering, Vol.16, No.11,pp.1424-1440,Nov2004, "doi:10.1109/TKDE.2004.77 ".

[6] Jianyong Wang and Jiawei Han, "Bide:Efficient mining of frequent closed sequences," In Proc. of 20th IEEE international conference on Data Engineering ,pp. 79-90, "doi:10.1109/ICDE.2004.1319986".

[7] Jianzhong Li, Yong Liu, and Hong Gao "Efficient Algorithm for Summarizing Graph Patterns," IEEE Transactions on Knowledge and Data Engineering, Vol.23, Issue:9,pp.1388-1405,2011,"doi:10.1109/TKDE. 2010.48".

[8] M. Zaki, "SPADE:An Efficient Algorithm for Mining Frequent Sequences," Machine Learning, Kluwer Academic Publishers, Vol.42, issue:1, pp.31-60, 2001, "doi: 10.1023/A:1007652502315".

[9] M. Kuramochi and G. Karypis, "Frequent Subgraph discovery," In Proc. 2001. Int. conf. Data Mining 2001, pp. 313-320, San Jose, CA, November 2001, "doi: 10.1109/ICDM.2001.989534".

[10] Mohammed Hasan Mahafzah. "An Efficient Graph-coloring Algorithm for processor Allocation," International Journal of Information Technology and Computer Science, Vol.5, No.7, pp. 43-48, June 2013, "doi: 10.5815/ijitcs. 2013.07.05".

[11] Natalia Vanetik. "Mining Graphs with Constraints on Symmetry and Diameter," In WAIM 2010 International Workshops, Vol. 6185,Springer, pp. 1-12, July 2010, "doi: 10.1007/978-3-642-16720-1_1".

[12] Petre Tzvetkov, Xifeng Yan, Jiawei Han. "TSP: Mining Top-K Closed Sequential Patterns," Third IEEE International Conference on Data mining, pp. 347-354, November 2003, "doi:10.1109/ICDM.2003.1250939".

[13] R. Agrawal and R. Srikant." Mining sequential patterns". In ICDE'95, Taipei, Taiwan, pp. 3-14, Mar. 1995.

[14] R. Srikant and R. Agrawal. "Mining sequential patterns: Generalizations and performance improvements". In EDBT'96 Proceedings of the 5th International Conference on Extending Database Technology: Advances in Database Technology,pp.3-17,"doi:10.1007/BFb0014140"

[15] Synthetic graph generated by IBM Quest Synthetic Data Generation Code for Associations and Sequential Patterns. [http://www.cse.ust.hk/graphgen/].

[16] X. Yan and J. Han. "CloseGraph: Mining closed frequent graph patterns". In KDD'03, Washington, D.C., August 2003, "doi:10.1145/956750.956784".

[17] X. Yan, J. Han, and R. Afshar. "CloSpan: Mining closed sequential patterns in large datasets," In SDM'03, San Fransisco, CA, pp. 166-177 May 2003.

[18] Xifeng Yan, Jiawei Han. "gSpan: Graph-Based Substructure Pattern Mining," In Proc ICDM 2002, pp. 721-724, "doi:10.1109/ICDM.2002.1184038".

[19] Yan X, Zhou XJ, Han J. "Mining closed relational graphs with connectivity constraints". In Proc of ACM SIGKDD international conference on knowledge discovery in databases (KDD'05), Chicago, IL, pp. 324–333, 2005, "doi:10.1145/1081870.1081908".

Authors' Profiles

K. Vijay Bhaskar is currently a PhD student in the department of Computer Science and Engineering, Gandhi Institute of Technology (GITAM). His current research areas of interest include Data Mining, Graph Databases, Network security, and Mobile Computing.

Dr. R.B.V.Subramanyam is an associate professor in the department of computer science and Engineering, National Institute of Technology, Warangal. He received his Master of Technology and Doctor of Philosophy from Indian Institute of Technology, Kharagpur. His areas of interest include Data mining, Distributed data mining, Graph databases, Fuzzy data mining, Big data analytics, Pattern recognition, High performance computing, Soft computing, Game theory, Outlier analysis.

Dr. K. Thammi Reddy is the Director of Internal Quality Control (IQC) and Professor of CSE at Gandhi Institute of Technology(GITAM).He is having Over 18 years of experience in Teaching, Research, Curriculam Design and consultancy. His research areas include Data warehousing and Mining, Distributed computing, Network Security etc.

S. Sumalatha is currently a PhD student in the department of Computer Science and Engineering, National Institute of Technology, Warangal. Her research areas of interest include Data mining, Big data analytics and Graph databases.

How to cite this paper: K. Vijay Bhaskar, R.B.V Subramanyam, K. Thammi Reddy, S. Sumalatha,"Top-k Closed Sequential Graph Pattern Mining", International Journal of Information Engineering and Electronic Business(IJIEEB), Vol.8, No.4, pp.1-9, 2016. DOI: 10.5815/ijieeb.2016.04.01

