EAI Endorsed Transactions
on Scalable Information Systems
Research Article EAI.EU

Finding Frequent Subgraphs and Subpaths through Static and Dynamic Window Filtering Techniques

Bhargavi B.* and K. Swarupa Rani

School of Computer and Information Sciences, University of Hyderabad, Hyderabad, Telangana, India

Abstract

Big data era has large volumes of data generated at high velocity from different data sources. Finding frequent subgraphs from the graph streams can be a challenging task as streams are non-uniformly distributed and continuously processed. Its applications include finding strongly interacting groups in social networks and sensor networks. To find frequent subgraphs, we proposed static single-window technique and dynamic sliding window techniques. We also proposed enhancements by extending proposed static approach with its variations and extending dynamic approach in variations of incremental strategy to find frequent subgraphs. We also solved the sub problem to extract frequent subpaths from sequence of paths. Its applications include finding congested sections in traffic analysis. We applied our proposed static and dynamic techniques to extract the frequent subpaths from sequence of paths. We experimented the proposed dynamic and static approaches with real and benchmark datasets.

Received on 17 February 2020; accepted on 09 April 2020; published on 16 April 2020

Keywords: graph stream, frequent subgraphs, subpath.

Copyright © 2020 Bhargavi B. *et al.*, licensed to EAI. This is an open access article distributed under the terms of the Creative Commons Attribution license (http://creativecommons.org/licenses/by/3.0/), which permits unlimited use, distribution and reproduction in any medium so long as the original work is properly cited.

doi:10.4108/eai.13-7-2018.163986

1. Introduction

In Big data era, we find large amounts of data generated from different data sources very fast. For instance, there are 22.2 million Twitter users in India and 152 million daily active Twitter users all over the world [14]. Data stream model deals with such Big data and data stream algorithms [6] make very few passes and takes lesser space. Data streams constitute of structured, semi-structured and unstructured data which are time consuming as streams are continuous and are also unbounded. Massive graphs are rendered as streams of graphs to analyze and extract useful and unknown information. Graph streams as dynamic stream model has been studied in the literature [1, 3, 8] which are the sequence of 'm' edges between 'n' nodes with the edges being updated sequentially. Processing graph streams are challenging as they have large volume and are highly dynamic in nature. In this paper, we review the problem of finding frequent subgraphs from the graph streams. A frequent subgraph is a connected subgraph that occurs above the given threshold in the sequence of graph streams. The problem of finding frequent subgraphs is defined as follows: Given a sequence of graph streams and a minimum support threshold, the problem is to find the frequent subgraphs having useful information from the graph streams efficiently. One of the applications of finding frequent subgraphs can be in social networks [9]. For instance, we can derive the groups of users who are frequently communicating in the social network. In bio-informatics, based on the frequent interactions between molecules, we can predict protein functions and identify types of diseases.

We also solve another sub-problem of extracting frequent subpaths from sequence of paths. The applications for finding frequent subpaths can be in IP routing in which we can find the frequent paths of data flows across multiple networks. In a traffic network, we can find the paths/sub paths that are frequently traversed by commuters.

Alfredo Cuzzocrea et al. [10] proposed two algorithms to discover collections of frequent subgraphs, one of which is the direct 1-step algorithm based vertical mining approach using Data Stream Matrix (DSMatrix). Besides, this approach [10] used sliding window technique to process the graph streams in finding the

*Corresponding author. Email: bhargavibbv@uohyd.ac.in

frequent subgraphs. This sliding window technique has the limitation of repeated calculations. To overcome this limitation, Kyoungsoo Bok et. al. [11] proposed an incremental frequent subgraph detection technique. The limitation of Kyoungsoo Bok et al. [11] approach is that their approach did not completely resolve the duplicate calculations.

We observed that the above solutions have certain limitations which include the partially resolved duplicate calculations. Another possible limitation, frequent subgraphs in the past can be infrequent due to incomplete storage of edges in the sliding window. To overcome these limitations, we proposed static and dynamic approaches to find the frequent subgraphs. The key contributions of this work include

- Proposed static and dynamic techniques to extract frequent subgraphs
- Compared the proposed techniques with the conventional approach thus evaluating the efficiency

This paper is a revised and extended version of our BigDML 2019 paper [16]. Our additional key contributions of this paper include the following:

- We improved and proposed static approach by computing actual minimum support.
- We also proposed partition based static approach with actual minimum support for sequential and parallel environments.
- We improved and extended the dynamic sliding window filtering technique with variable batch size described in Section 5.2.
- We solved the sub-problem to find frequent sub-paths from sequence of paths described in section 5.3 by applying our proposed static and dynamic techniques.
- We analysed our proposed static and dynamic techniques for efficiency on real and synthetic datasets in section 6.

The above key contributions of proposed approaches and its variations are also given in Table 1.

Section 2 deals with the preliminaries and problem definitions. Section 3 deals with literature survey of frequent subgraph algorithms and extracting frequent sub-paths from sequence of paths. Section 4 describes our proposed static and dynamic techniques to extract frequent subgraphs. Section 5 discusses the enhancements to our proposed techniques by solving the problem of finding frequent sub-paths from paths data for the directed graph. Section 6 describes the experiments and evaluation of results. Finally, in section 7, we conclude with future research directions.

Table 1. Proposed Approaches and its variations

Static Approach	Dynamic Approach
1. Single window with minimum support 2. Single window approach with actual minimum support 3. Partition based approach with actual minimum support in sequential and parallel environments	1. Incremental approach with fixed batch size of graph data with relative support 2. Incremental approach with variable batch size of graph data with relative support

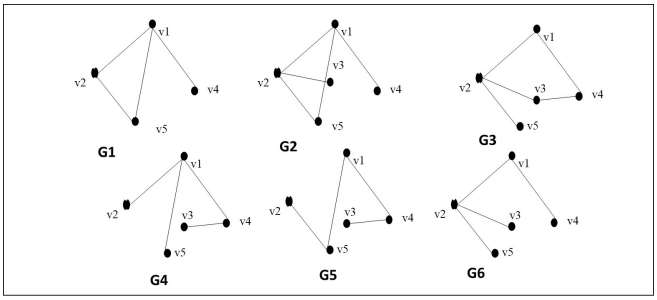

Figure 1. Sequence of graph streams G1, G2, G3, G4, G5, G6

2. Preliminaries

Definition 1. (Graph stream) *"Graph stream is defined as the sequence G_1, G_2, ... G_i . . .G_s, where each graph G_i is a set of edges. We assume that the edge set G_i contains only a small fraction of the underlying nodes"* [2].

Definition 2. (Frequent Subgraph) A subgraph $G_s(V_s, E_s)$ is a part of a graph G(V,E) such that $V_s \subset V$ and $E_s \subset E$. A frequent subgraph is a connected subgraph that occurs above the given threshold (***th***) in the sequence of graph streams.

Definition 3. (Path) "*Given a graph G(V, E), a path p of length k from a vertex u to u′ is a sequence (v_0, v_1, . . . , v_k) of vertices such that $v_i \in V$, $v_0 = u$ and $v_k = u'$ and (v_{i-1}, v_i) $\in E$ for i = 1, 2, . . . , k*" [4].

Definition 4. (Subpath) *"A path Q in G is said to be a subpath of P if Q = (w_0,w_1, . . .,$w_{k'}$), where (w_0, w_1, . . . ,$w_{k'}$) is a contiguous sub-sequence of path P(v_0, v_1, . . . , v_k), i.e., if, for some i such that $0 \le i \le i+k' \le k$, we have $w_0 = v_i$, $w_1 = v_{i+1}$, . . . , $w_{k'} = v_{i+k'}$"* [4].

Definition 5. (Minimum Support) Minimum support is defined as the threshold specified by the user.

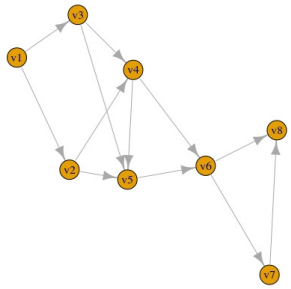

Figure 2. Directed Graph, G

Definition 6. (Actual Minimum Support) Actual minimum support is defined as the minimum support based on user's mining requirements which is appropriate to the database to be mined [17].

Definition 7. (Relative Support) We define the relative support as the partial minimum support assigned to subset of data. In this paper, we adopt the filtering threshold [19] to find the relative support.

2.1. Problem Definitions

In this paper, we propose techniques to find frequent subgraphs from graph stream data and to find frequent subpaths from sequence of paths for a directed graph efficiently.

Finding frequent subgraphs from graph stream data. Given a sequence of graph streams, for a minimum support threshold (***th***), the problem is to find the frequent subgraphs from the graph streams efficiently.

For instance, for the sequence of graph streams, shown in Fig. 1, with ***th***=3, the set of frequent subgraphs include { <(v1, v2), (v1, v4)>, <(v1, v2), (v1, v5)>, <(v1, v2), (v3, v4)>, < (v1, v4), (v3, v4) >, <(v1, v4), (v1, v5) > , <(v1, v2), (v1, v4), (v1, v5) >, < (v1, v2), (v1, v4), (v2, v3)>}

Finding frequent sub-paths from paths data. Given a sequence of paths of a graph, for a minimum support (***th***), the problem is to extract frequent sub-paths from paths data.

For instance, for the sequence of paths in table 2 of the graph in Fig. 2., the set of frequent sub paths with ***th***=3 are {(v5, v6, v7), (v3, v5, v6)}.

Table 2. Data of Paths

S.No.	Paths
1	(v1, v2, v5, v6)
2	(v1, v3, v5, v6, v7)
3	(v1, v2, v4)
4	(v1, v2, v4, v5)
5	(v2, v4, v5)
6	(v2, v5, v6)
7	(v2, v4, v6, v7)
8	(v4, v5, v6, v7)
9	(v4, v6, v8)
10	(v3, v4, v5, v6)
11	(v3, v5, v6)
12	(v3, v5, v6, v7)

3. Literature Survey

Massive graphs are considered as streams of data to analyze and extract useful information. Henzinger et. al. [7] were the first to introduce graph streams and they also considered graph problems of paths and connectivity. Andrew McGregor [1] presented a detailed survey of graph streams. Due to the dynamic nature [12], [3] and the large volume of graph stream data, Nan Tang et. al. [8] proposed graph summarization sketch that can store frequent counts and paths of graph streams.

Alfredo Cuzzocrea et al. [10] studied various methodologies of mining dense patterns in graph streams and proposed probabilistic algorithms for determining such structural patterns effectively and efficiently. Alfredo Cuzzocrea et al. [10] presented two algorithms to extract frequent subgraphs - (i) Indirect 2-step algorithm (ii) Direct 1-step algorithm. Experimental results by Alfredo Cuzzocrea et al. [10] stated that mining with DSMatrix consumes lesser memory due to the information stored in a secondary storage device as they store the existence of edges in bit vectors. Kyoungsoo Bok et. al. [11] observed that the algorithm proposed by Alfredo Cuzzocrea et. al. [10] has a limitation of duplicate calculations. They introduced *slidenum* variable [11] to store the frequency of edges incrementally for batches of graph streams to resolve duplicate calculations.

From the literature, we observe that while finding frequent subgraphs, although sliding window based techniques execute fast, they may lead to loss of useful historical information. We also observe that we need to reduce duplicate calculations further.

We observed that finding frequent sub paths from paths is another problem in the literature that can be related to the problem of finding frequent subgraphs.We identified and formulated ways to apply our proposed and extended techniques to find frequent subpaths from sequence of paths. Sumanta Guha [4]

Table 3. DSMatrix for graph streams of Fig. 1

Edge	G1	G2	G3	G4	G5	G6
(v1, v2)	1	1	1	1	0	1
(v1, v4)	1	1	1	1	1	1
(v1, v5)	1	1	0	1	1	0
(v2, v5)	1	1	1	0	1	1
(v2, v3)	0	1	1	0	0	1
(v3, v4)	0	0	1	1	1	0

developed Apriori based technique to extract frequent sub paths from paths in an undirected graph. Schwartz et al. [5] studied demand of frequent sub paths in a transportation network traversed by several users. Hence, there is need to find techniques that discover frequent subgraphs and frequent sub paths efficiently by storing useful historical information.

4. Static and Dynamic Techniques for Finding frequent Subgraphs

We have extended the direct 1-step algorithm of Alfredo Cuzzocrea et al. [10] by modifying the parameters of sliding window size and by using relative support. We have proposed static single window approach and dynamic approach of sliding window to find frequent subgraphs from graph streams using DSMatrix.

4.1. DSMatrix

Data Stream Matrix or DSMatrix constitutes assigning the presence (or absence) of each edge by a bit 1 (or 0) for each graph of graph stream data [10]. For instance, consider the sequence of graph streams in figure 1. Table 3 shows the contents of DSMatrix. We find the rowsum from DSMatrix to compute the frequency of every edge in graph stream data.

4.2. Static Single Window Technique

In the static single-window, we consider the entire data set of graph streams as a single window. We describe the proposed static single window approach in StaticFreqSubgraph algorithm. In this algorithm, for the entire graph streams, it creates a DSMatrix. The rowsum of the DSMatrix is computed and compared to the minimum support threshold *minsup*. If rowsum is greater than or equal to *minsup*, the resultant edges are the frequent singleton edges. Then, combination of edge pairs are found based on the neighbouring information (in list, *N*). The AND operation is performed between the k-frequent singleton edge combinations' bit vectors by checking whether the edges are present for each graph to form a bit vector for the combination.The sum of non-zero bits is computed and compared to the given minimum support to check if it is frequent or not. If the resultant sum is greater than or equal to *minsup*, the k+1-subgraph is frequent. Thus, the algorithm generates all the possible frequent subgraphs.

Algorithm 1 describes the static single window approach. Step 2 to step 6 is used to create DSMatrix (Mat_A). The rowsum is computed in step 7 and frequent singleton edges are generated through step 8 to step 11. Finally, frequent subgraphs are generated using step 12-16.

For example, for graph streams of Fig. 1 with *minsup*=3, the resultant frequent singleton edges are {(v1, v2), (v1, v4), (v2, v5), (v1, v5),(v2, v3),(v3, v4)}. The resultant frequent subgraphs for the graph streams of Fig. 1 generated using StaticFreqSubgraph algorithm are { <(v1, v2), (v1, v4)>, <(v1, v4), (v1, v5)>, <(v2, v3), (v2, v5)>, <(v1, v2), (v2, v3)>, <(v1, v2), (v1, v4), (v1, v5)>}.

Algorithm 1: StaticFreqSubgraph

```
Input : Edges in graph streams, total number of graphs in the whole data and minsup = minimum
        support threshold
Output: Set of frequent subgraphs
// m= number of graph streams, n= number of edges in the entire graph streams
   sequence
// Mat_A[m][n] = 2-D DSMatrix with m rows and n columns
// rowsum = count of 1s in a row of DSMatrix
1  Create neighbouring list, N for the graph streams by finding common vertex for the stream of edges
2  for each graph g_j do
3      if edge i ∈ g_j then
4          Mat_A[i][j]=1
5      else
6          Mat_A[i][j]=0
7  Calculate rowsum for each row of Mat_A.
8  if (rowsum(i) ≥minsup) then
9      Edge i is frequent
10 else
11     Edge i is not frequent
12 Join 'k' frequent connected edges with common vertices using neighboring list N to get 'k+1'
     frequent subgraphs , f_l for all k≤m.
13 Compute freq(f_l) by AND of the bit vectors of 'k' recurrent edges for the graph streams in the
     DSMatrix
14 if freq(f_l)≥minsup then
15     f_l is frequent.
16 else
17     f_l is infrequent.
```

4.3. Dynamic Approach of Sliding Window Technique

In this approach, we proposed a dynamic sliding window technique which we describe in DynamicFreqSubgraph algorithm. DSMatrix for each batch is created and a relative support is applied. For example, if batch 1 contains 3 graph instances, then we calculate 40% of frequent edges of the 3 graph instances and store it in the map. This technique is incrementally applied for the next batches. The frequency threshold for the edges is based on the number of graph instances. Thus, this approach preserves the previous history information. After the singleton frequent edges are calculated, the combination is formed based on the neighbouring information. DynamicFreqSubgraph algorithm computes AND operation on the edges and

Algorithm 2: DynamicFreqSubgraph

```
   Input : Edges in the graph streams, percentminsup
   Output: Set of Frequent subgraphs
   // n=Total number of graph streams in the entire sequence
   // DSMatrix[m][n]= 2D matrix of m rows and n columns
   // batchsize = number of graphs in a batch
1  totalbatch= n / batchsize
2  batchpointer=0
3  k=0
4  repeat
5      Compute DSMatrix for each batch of graph streams
6      Calculate edge_count for each batch
7      minsup=percentminsup*(batchpointer*batchsize+1)
8      k=k+1
9      if (edge_count[i] ≥ minsup) then
10         Store < edge i, edge_count[i] > in a map M
11     batchpointer= batchpointer + batchsize
12 until k < totalbatches;
13 relativesup=⌈(percentminsup*n)⌉
14 for i=0 to m do
15     if (edge_count[i]≥ relativesup) then
16         Edge i is frequent
17     else
18         Edge i is not frequent
19 Join 'k' recurrent connected edges with common edge based on neighbouring information to 'k+1'
     recurrent connected edges by intersecting their bit vectors from DSMatrix, f_I for all k≤m.
20 if freq(f_I)≥minsup then
21     f_I is frequent.
22 else
23     f_I is infrequent.
```

Table 4. Frequent Edges with count for Batch 1

edge	count
(v1,v2)	2
(v1, v4)	2
(v1,v5)	2
(v2,v5)	2
(v2, v3)	1

Table 5. Frequent Edges with count for Batch 1 and Batch 2

edge	count
(v1,v2)	4
(v1, v4)	4
(v1,v5)	3
(v2,v5)	3
(v2, v3)	2
(v3, v4)	2

calculates the final row sum. If the final row sum satisfies the minimum support threshold, then it is marked as frequent. Algorithm 2 describes the dynamic approach to compute the frequent subgraphs from the sequence of graph streams.

Illustration for Dynamic Sliding Window Technique. For instance, in the graph streams of Figure 1, Let batch size=2. Then, batch 1 constitutes graph streams G1, G2 of Fig. 1. For batch 1, the frequent singleton edges along with edge count are stored in map ***M*** (as shown in step 10 of Algorithm 2) for each edge satisfying 40% of batch size, i.e., ⌈0.4*2⌉=1. Thus, the initial map ***M*** for Fig. 1 with relative support 1 includes all the edges with count greater than or equal to 1 as shown in Table 4.

Table 6. Frequent Edges with count for Batch 1, Batch2 and Batch 3

edge	count
(v1,v2)	5
(v1,v4)	6
(v1,v5)	4
(v2,v5)	5

This map is incrementally maintained as shown in Table 5 for the next batch with relative support 2. When the batch 3 is encountered, the edge counts are incremented as shown in Table 6 with relative support of 4. Thus, the resultant frequent singleton edges by applying Algorithm 2 with the given minimum support of 4 are {(v1, v2), (v1, v4), (v1, v5), (v2, v5)}.

5. Enhancements to the Proposed Static and Dynamic Sliding Window Filtering Technique and Finding Frequent Subpaths

In this section, we describe the improved and extended dynamic sliding window filtering technique and address the problem of finding frequent sub paths from paths data by using our proposed static and dynamic techniques. We adopt the polynomial strategy [17] and modify it to compute the actual minimum support based on dataset information for graph stream data. We use this minimum support in the proposed static approach to compute frequent singleton edges. Besides, we propose partition based sequential and parallel static approach with actual minimum support. In addition, we observed that our earlier proposed dynamic approach may miss some of the frequent singleton edges for large graph streams with the iterative increase in relative threshold. To overcome this limitation, we modified the dynamic approach by using incremental relative support with variable batch size which is described in section 5.2.

5.1. Enhancement #1: Computing Actual Minimum Support for Proposed Static Approach

We consider the minimum support given by the user as input to compute the actual minimum support based on distribution of frequent singleton edges in the graph stream database in the interval [*amin*, *bmax*]. *amin* denotes minimum frequency of the edge and *bmax* denotes the maximum frequency of the edge in the graph stream data. We adopt the approximate polynomial function of degree *i* [17] and compute the actual minimum support (*Actualminsup*) based on the user given minimum support (*minsup*) through the equation below:

$$Actualminsup^i = minsup * (bmax^i - amin^i) + amin^i \quad (1)$$

We then apply the proposed static approach (described in 4.1) with the actual minimum support to compute the resultant frequent subgraphs. For instance, consider the graph stream data in figure 1, with the user given minimum support, *minsup*=0.5, the minimum frequency *amin*=1/6 and maximum frequency *bmax*=5/6. The actual minimum support computed using equation (1) with linear strategy (*i*=1) is 0.49 and with cubic strategy (*i*=3) is 0.83. We use actual minimum support when the user is not an expert about the minimum support using which the significant frequent subgraphs are to be retrieved.

Proposed Partition based Static Approach. We modify our proposed static approach with single window by partitioning the data into windows of fixed size for faster execution. We then compute the actual minimum support for each window used to compute frequent singleton edges. We can run each partition in parallel to extract the frequent singleton edges. Algorithm 3 describes our proposed partition based static approach with actual minimum support.

Algorithm 3: PartitionStaticFreqSubgraph

Input : Edges in graph streams, total number of graphs in the whole data, *minsup* = minimum support threshold, partition_no be number of partitions
Output: Set of frequent subgraphs
// m= number of graph streams, n= number of edges in the entire graph streams sequence
amin= $\frac{1}{m}$ // Mat_A[m][n] = 2-D DSMatrix with m rows and n columns
// rowsum = count of 1s in a row of DSMatrix
Create neighbouring list, N for the graph streams by finding common vertex for the stream of edges
for *each partition p* ∈ *[1, partition_no]* **do**
  for *each graph* g_j **do**
    if *edge i* ∈ g_j **then**
      Mat_A[i][j]:=1
    else
      Mat_A[i][j]:=0
  Calculate *rowsum* for each row of Mat_A.
  Compute *bmax*, maximum frequency of any edge by considering maximum *rowsum*
  bmax= $\frac{bmax}{m}$
  Compute *Actualminsup* using (1)
  for *each edge i* **do**
    if *(rowsum(i)* ≥*Actualminsup)* **then**
      Add edge i to set *F*
Join 'k' recurrent connected edges from *F* with common edge using neighboring list N and to get 'k+1' recurrent connected edges , f_l for all k≤m.
Compute freq(f_l) by intersecting the bit vectors of 'k' recurrent edges from DSMatrix and adding all the resultant non-zero bits
if *freq(f_l)*≥*Actualminsup* **then**
  f_l is frequent
else
  f_l is infrequent

Illustration of Proposed Partition based Static Approach. For graph streams, with actual minimum support using linear strategy of 2.774~3 (given user *minsup*=2), with partition size=2, the frequent singleton edges for the first partition are {(v1, v2), (v1, v4), (v2, v5)}. The frequent singleton edges for the second partition are {(v1, v4)}. The resultant frequent subgraphs include { <(v1, v2), (v1, v4)>, <(v1, v4), (v2, v5)>, <(v1, v2), (v1, v4), (v2, v5)>}.

5.2. Enhancement #2: Dynamic Sliding Window Filtering Technique

Algorithm 4: DynamicVarFreqSubgraph

Input : Edges in graph streams, Let *rel* be percentage for computing relative support
Output: Set of Frequent subgraphs
// n=Total number of graph streams in the entire sequence
// DSMatrix[m][n]= 2D matrix of m rows and n columns
// batchsize = variable number of graphs in a batch
// Let *b* be the number of batches in the graph
totalbatch= n / batchsize
k1=1
Compute DSMatrix for each batch of graph streams
repeat
  Calculate edge_count for each batch
  batchpointer=0
  minsup=⌈*rel**(batchsize)⌉
  if *existsEdge(i)* **then**
    ec=edge_count[i]+ Edge count of i in the current batch
    minsup=⌈*rel**(k1-batchno[i]+1)*batchsize⌉
    if *(ec* ≥ *minsup)* **then**
      // Edge i is frequent
      Update edge count of i to ec in map M
  if *(!existsEdge(i)* & *edge_count[i]* ≥ *minsup)* **then**
    Store < edge i, batchno[i], edge_count[i] > in a map M
  batchpointer= batchpointer + batchsize
  k1=k1+1
until *k1 leq b*;
Join 'k' frequent connected edges with common edge to 'k+1' frequent connected edges by intersecting their bit vectors from DSMatrix, f_l for all k≤m.
minsup=⌈*rel**n⌉
if *freq(f_l)*≥*minsup* **then**
  f_l is frequent
else
  f_l is infrequent.

In the proposed dynamic sliding window technique (described in Section 4.2), we observed that with increase in relative support, we may miss some of the significant frequent subgraphs. Hence, we modified the proposed dynamic approach by storing the batch number in addition to frequency for the edges in a map. For the first batch, we start with relative support and find the frequent singleton edges. The relative support is iteratively increased with next batch if the edge obtained in the current batch is frequent in the previous batch. If the edge is not frequent in the previous batch, but is frequent in the current batch, then we compare the frequency of the edge in the current batch with the relative support. For the next batch, the relative support is iteratively incremented. Thus, this approach preserves the previous history information. In addition, our proposed approach described in Algorithm 4 considers batches with variable sizes. After the frequent singleton edges are calculated, the combination is formed based on the neighbouring information. DynamicVarFreqSubgraph algorithm computes AND operation on the neighbouring frequent singleton edges and calculates the final row sum. If the final row sum satisfies the relative frequency threshold, then it is frequent. Algorithm 4 describes the dynamic sliding window filtering technique for graph stream data to compute the frequent subgraphs from the sequence of graph streams.

Illustration of Dynamic Sliding Window Filtering Technique. For instance, in the graph streams of Fig. 1, let batch size=3. Then, batch 1 constitutes graph streams G1, G2

Table 7. Frequent Edges with count for Batch 1

edge	Batch No.	count
⋄(v1, v2)	1	3
⋄(v1, v4)	1	3
⋄(v1, v5)	1	2
⋄(v2, v3)	1	2
⋄(v2,v5)	1	3
(v3, v4)	1	1

Table 8. Frequent Edges with count for Batch 1 and Batch 2

edge	Batch No.	count
⋄(v1, v2)	1	5
⋄(v1, v4)	1	6
⋄(v1, v5)	1	4
⋄(v2, v3)	1	3
⋄(v2, v5)	1	5
⋄(v3, v4)	2	2

and G3 of Fig. 1. For batch 1, the frequent singleton edges along with edge count are stored in map ***M*** for each edge with its relative frequency in the batch satisfying 40% of batch size. Thus, the initial map ***M*** for Fig. 1 with relative support 2 includes all the edges with count greater than or equal to 2 denoted by ⋄ symbol in Table 7.

This map is incrementally maintained as shown in Table 8 for the next batch, with relative support 3, if the edge is frequent in the previous batch. If the edge is not frequent in the previous batch, then the relative support for such edge is 2. For instance, in table 8, the edge (v3, v4) is not frequent in batch 1, but is frequent in the current batch as the relative support for that edge is set to 2. Thus, the resultant frequent singleton edges by applying Algorithm 4 with the relative frequency threshold are {(v1, v2), (v1, v4), (v1, v5), (v2, v5), (v2, v3), (v3,v4)}. The resultant frequent subgraphs with relative support 3 are { <(v1, v2), (v1, v4)>, <(v1, v2), (v1, v5)>, <(v2, v5), (v1, v5)>,<(v2, v3), (v2, v5)>, < (v1, v2), (v1, v4), (v1, v5) > }.

5.3. Enhancement #3: Finding frequent sub paths from sequence of paths

For a directed graph, given a sequence of reachable paths [15], we can extract the frequent sub paths by using our proposed static and dynamic techniques. For this application, algorithm 1 and algorithm 2 described in Section 4.1 and section 4.2 can be modified by considering each path as a graph stream input and the resultant output are frequent subpaths. In addition, while extracting frequent sub paths, we consider the sequence of neighbouring vertices that form a subpath. For instance, for the directed graph in Fig 2, let us assume that the edges (v1, v2) and (v1, v3) are frequent. Then we cannot join the edges (v1, v2) and (v1, v3) as they do not form a subpath. Application areas of frequent sub paths extraction include analysis of traffic sub-routes based on the routes taken by vehicles (stored as paths database) to extract congested sections. In IP routing, we can analyse the routes taken by messages to extract the hot-spots. For instance, for graph G of figure 2, Table 2 shows some of the paths extracted from the graph G. Algorithm 3 and Algorithm 4 can also be similarly applied to find the frequent subpaths from sequence of paths for the given directed graph. The following subsections illustrate the extraction of frequent sub paths from paths data using the proposed static and dynamic techniques.

Illustration of static single window technique to find frequent sub paths. By considering each path as input for static single window technique, we retrieve the frequent sub paths for the set of paths. To extract frequent sub paths, we construct DSMatrix for the unique edges present in the paths set. Then, we extract frequent singleton edges from DSMatrix as described in Algorithm 1. From the frequent edges and neighbouring information, we extract the frequent sub-paths. For instance, for the set of paths of Table 2, with *minsup*=3, the frequent singleton edges extracted using static single window technique are {(v1, v2), (v2, v4), (v2, v5), (v3, v5), (v4, v5), (v5, v6), (v6, v7)} and the resultant frequent sub-paths are { (v3, v5, v6), (v5, v6, v7)}

Illustration of dynamic sliding window filtering technique to find frequent sub-paths. By considering each path as input for dynamic sliding window filtering technique, we retrieve the frequent sub paths for the set of paths. Initially, we construct the DSMatrix for the data of paths for each batch. Then, we extract the frequent singleton edges by using dynamic variable sliding window filtering technique described in Algorithm 4. For instance, let batch size=3 for paths data of Table 2. By using dynamic sliding window filtering technique, the resultant frequent singleton edges for the first batch with relative support=3 for path 1 to path 6 are { (v1, v2), (v5, v6)}. The resultant frequent singleton edges for the second batch for the remaining paths include { (v1, v2), (v5, v6), (v6, v7)}. Finally, the resultant frequent edges are { (v1, v2), (v5, v6), (v6, v7) } and the resultant frequent sub paths are { (v5, v6, v7) }.

5.4. Analysis of Proposed Static and Dynamic Approaches

The proposed static single window approach can be used for small datasets as this approach efficiently stores all the frequent subgraphs. Thus, the useful

Table 9. Characteristics of Proposed Static and Dynamic Approaches compared to Conventional Approach

Characteristic	Proposed approaches		Conventional
	Static	Dynamic	
Size	Entire graph stream data	incremental data	sliding window size
Minimum Support	user given minimum support and actual minimum support [17] can be used	relative support [19]	user given minimum support
Applicability	for existing data	present stream of data	present stream of data
Streams of data	single window	incremental in batches	sliding window
Loss of historical data	No loss of frequent subgraphs	No loss of significant frequent subgraphs	Loss of some significant frequent subgraphs
Time	faster technique in distributed and parallel environments	moderate technique without loss of significant information	faster technique with loss of information
Parallelism	can be applied by dividing the data and *minsup* into equal partitions	cannot be applied	cannot be applied
Distributed	can be applied by using MapReduce technique	cannot be applied	may not be applicable

history information is retained. In addition, our proposed static approach with actual minimum support can retrieve the required number of frequent subgraphs/subpaths based on user minimum support. However, for large streams of graph data, we can use the partition based static approach by running each partition of data in parallel to find frequent singleton edges. For large streams of data with variable batch size, the proposed dynamic sliding window approach efficiently finds the frequent subgraphs as relative support computation involves the number of graph streams. With the increase in the number of graph streams, the relative support is also proportionately increased thus storing the significant frequent subgraphs as well as minimizing the duplicate calculations. In addition, to extract frequent sub paths from large sequence of paths, our proposed dynamic sliding window filtering technique can be applied for large directed graphs. Table 9 shows different characteristics of proposed static and dynamic techniques compared to the conventional approach [10].

6. Experimental Evaluation

The direct 1-step algorithm (*Conventional approach*) of Alfredo Cuzzocrea et al. [10] and its modified versions, the proposed static single window approach , proposed static approach with actual minimum support using linear strategy and cubic strategy, proposed partition based static approach in sequential and parallel environments, dynamic sliding window technique (*DynFixed*) and dynamic variable sliding window filtering techniques (*DynVar*) are implemented in C++. The specifications of the system, in which these algorithms are implemented, is Linux Ubuntu Desktop with 8GB RAM and Core i5 Intel Xeon 2.8 Ghz processor.

The *minsup* is varied from 10% to 80% of number of graph streams/paths. For conventional approach, we set window size=5 and batch size=100. In dynamic approach and its variations, the batchsize is 100 for fixed batches and for variable batches, the batch size is randomly varied such that total number of batches are set to 10. In the experiments, the actual minimum support is computed based on the user given minimum support using equation 1 with i=1 for linear strategy

and i=3 for cubic strategy. The relative support for batches are varied from 10% to 80% for the proposed dynamic sliding window filtering technique.

6.1. Experiment #1: Finding frequent subgraphs from graph stream data with minimum support and actual minimum support

To extract frequent subgraphs from graph stream data, the proposed algorithms are experimented on real dataset, Connect-4 [13]. We have considered each record of Connect-4 dataset as graph stream and constructed 1000 graph instances. The Connect-4 dataset can be used to find the most frequent moves chosen by the winner of the game. In each record, there are 1s, -1s and 0s for each position of 6*7 matrix of Connect4 dataset. During pre-processing of the dataset, each grid position is rendered as vertex. Thus, there are 42 vertices. The adjacent 1s and -1s in those grids are taken as edges. The two adjacent 1s or -1s represent an edge denoted using their respective grid positions.

Table 10 shows the number of frequent singleton edges with varying *minsup* for proposed static approach and conventional approach. From table 10, we observe that with increase in *minsup*, the number of frequent singleton edges for conventional approach reduced to zero from 50% *minsup*, whereas our proposed static approaches retained frequent subgraphs. From table 10 and table 11 , we also observe that the actual minimum support computed using linear strategy ($Actualminsup_l$) is closer to the user given minimum support than that of cubic strategy ($Actualminsup_c$) for graph stream data. Fig. 3 shows the number of frequent singleton edges extracted using our proposed approaches, i.e. static approach (*Static*), static approach with actual minimum support computed using linear strategy (*StaticLinear*) and cubic strategy (*StaticCubic*) and conventional approach. We observe that more number of frequent singleton edges are retained using our proposed static approach with linear strategy than the conventional approach with increase in minimum support.

6.2. Experiment #2: Finding frequent sub graphs through sequential and parallel approaches

Table 12 shows the execution time of our proposed partition based static approach (with no. of partitions=4) with actual minimum support computed using linear strategy in sequential and parallel environment. Table 13 shows the execution time of our proposed partition based static approach in sequential and parallel environment using cubic strategy, with the proposed partition based static approach in parallel environment executing faster. We implemented the parallel environment based on multi-threading, with each thread executing each partition while using the computed actual minimum support for every partition.

Table 10. Number of frequent singleton edges (|FSE|) for graph stream data with varying *minsup* for proposed static approach compared to Conventional Approach

minsup	*|FSE| for Proposed Static approach*	*|FSE| for Conventional Approach*
0.1	121	104
0.2	101	72
0.3	87	47
0.4	71	23
0.5	59	0
0.6	45	0
0.7	34	0
0.8	23	0

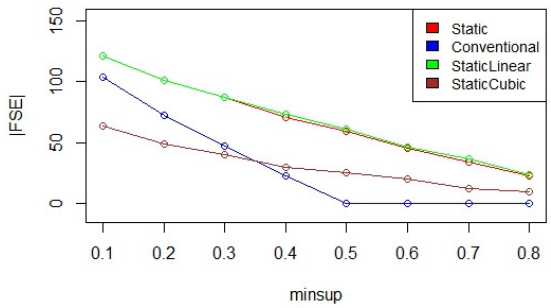

Figure 3. Number of frequent singleton edges (|FSE|) for proposed approaches with actual support using linear strategy and cubic strategy compared to conventional approach for graph stream data

6.3. Experiment #3: Finding frequent subgraphs through dynamic sliding window filtering approach

Table 14 shows the number of frequent singleton edges extracted with relative support varying from 10% to 80% of batch size using our proposed dynamic sliding window technique (*DynFixed*) and dynamic variable sliding window filtering technique (*DynVar*). We observe that with increase in relative support, the number of frequent singleton edges extracted decreased. We also observe that we can consider the relative support from 20% upto 60% to retrieve significant number of frequent singleton edges and thus the frequent subgraphs.

Table 11. Number of frequent singleton edges (|FSE|) for graph stream data with varying *minsup* for proposed static approach with actual minimum support using linear strategy with and cubic strategy

minsup	$Actualminsup_l$	*|FSE| for linear strategy*	$Actualminsup_c$	*|FSE| for cubic strategy*
0.1	0.099	121	0.450	64
0.2	0.198	101	0.576	49
0.3	0.296	87	0.661	40
0.4	0.390	73	0.726	30
0.5	0.490	61	0.783	25
0.6	0.590	46	0.832	20
0.7	0.690	37	0.875	12
0.8	0.789	24	0.915	10

Table 12. Execution time (in *milliseconds*) to find frequent singleton edges for graph stream data using proposed static approach with actual minimum support through linear strategy in sequential and parallel environment

minsup	$Actualminsup_l$	*Execution time in sequential envt.*	*Execution time in parallel envt.*
0.1	0.102	0.417	0.262
0.2	0.199	0.454	0.260
0.3	0.296	0.388	0.245
0.4	0.394	0.414	0.271
0.5	0.49	0.430	0.270
0.6	0.59	0.381	0.215
0.7	0.687	0.355	0.209
0.8	0.785	0.394	0.203

Table 13. Execution time (in *milliseconds*) to find frequent singleton edges for graph stream data using proposed static approach with actual minimum support through cubic strategy in sequential and parallel environment

minsup	$Actualminsup_c$	*Execution time in sequential envt.*	*Execution time in parallel envt.*
0.1	0.455	0.391	0.187
0.2	0.573	0.369	0.167
0.3	0.656	0.391	0.253
0.4	0.722	0.377	0.218
0.5	0.078	0.360	0.198
0.6	0.827	0.334	0.177
0.7	0.870	0.363	0.187
0.8	0.909	0.34	0.148

6.4. Experiment #4: Finding frequent sub paths and their analysis with varying parameters

For reachability path queries with constraints, the result of the query is sequence of paths. This database constitutes the reachable paths for reachability queries with constraints. These paths are stored in log file. We generate synthetic graph, i.e., ER-graph [18] with n=1000 and m=2000 edges. Next, we generate 800 reachability queries and retrieve 1000 possible paths satisfying the given constraints based on constrained BFS technique. We used the same query generation process for constraint reachable paths addressed in [15]. Each path is considered as graph instance. We apply the conventional approach and the proposed static and dynamic techniques to find the frequent sub paths.

Table 14. Number of frequent singleton edges for graph stream data with varying % of relative support(*relsup*) for proposed dynamic approach with fixed batch size (*DynFixed*) and variable batch size (*DynVar*)

relsup	|FSE| for *DynFixed*	|FSE| for *DynVar*
0.1	127	124
0.2	108	108
0.3	90	89
0.4	76	75
0.5	64	64
0.6	51	49
0.7	39	39
0.8	25	25

Table 15 shows the number of frequent singleton edges extracted from the sequence of paths using our proposed static approaches compared to conventional approach. We observe that as the paths dataset is sparse, the number of frequent singleton edges extracted for conventional approach is very less and is zero from 20% of minimum support. Our proposed static approach has zero frequent singleton edges with the user given minimum support from 30%. From Table 16 and Table 15, we observe that the proposed static approach with actual minimum support using linear strategy and cubic strategy retained the significant frequent singleton edges than the conventional approach and proposed static approach with minimum support. Fig. 4 shows that number of frequent singleton edges are

Table 15. Number of frequent singleton edges (|FSE|) for paths data for proposed static approach compared to Conventional approach

minsup	|FSE| for proposed Static Approach	|FSE| for Conventional Approach
0.1	31	7
0.2	4	0
0.3	0	0
0.4	0	0
0.5	0	0
0.6	0	0

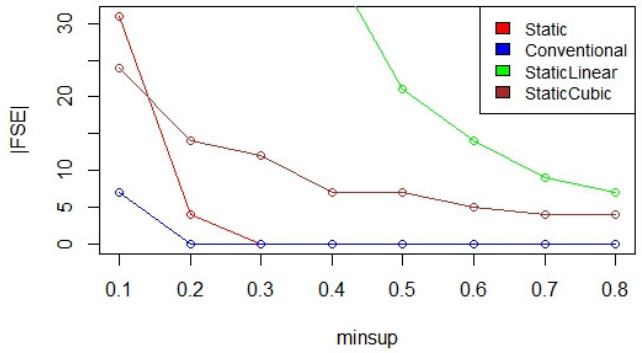

Figure 4. Number of frequent singleton edges (|FSE|) for proposed approaches with actual support using linear strategy and cubic strategy compared to conventional approach for sequence of paths

retained more using our proposed static approach using linear strategy with increase in minimum support for the generated paths data.

Table 17 and Table 18 shows the number of frequent singleton edges and frequent subpaths respectively that are extracted using our proposed dynamic approach with fixed batch size and variable batch size. Since the paths data is sparse, we observe that the relative support of 0.5% upto 1% retrieves significant number of frequent subpaths.

From the experiments and results, we observe that our proposed static approach with linear strategy extracted significant number of frequent singleton edges than that of cubic strategy. From these observations, we can conclude that our proposed techniques can efficiently extract frequent sub paths from sequence of paths.

7. Conclusion and Future Scope

To discover the collections of frequent edges, we proposed two approaches the static single window approach and dynamic approach of sliding window. The proposed static approach finds frequent subgraphs by considering the entire graph streams as a single window and applying the given minimum support for them. In addition, we adopt polynomial strategy to compute the actual minimum support from the user given minimum support and apply it to our proposed static approach that retained more number of frequent subgraphs. We also propose partition based static approach with actual minimum support that can be executed in parallel environment. In the dynamic approach, for each batch with variable size, we incrementally compute relative support and extract frequent edges. Finally, we join the frequent edges that share the common edge to extract the frequent subgraphs above the relative support computed for the entire graph stream data.

In addition, we also solve the problem of finding frequent subpaths from the sequence of paths by using our proposed techniques.From experiments, we observe that our proposed static approach with linear strategy retrieved significant number of frequent subpaths.

We can further extend this research by applying distributed techniques to partition the large dataset and then apply the proposed approaches to each partition and then group the resultant frequent subgraphs of each partition to get the final frequent subgraphs. We can also extend by extracting frequent subgraphs from graph streams arriving from different sources. We observe from the experiments that we can compute the actual minimum support for the relative support of our proposed dynamic sliding window filtering technique which is part of our future work. In addition, we can further extend by computing the actual minimum support to extract the frequent sub patterns based on the user minimum support using fuzzy membership based approach [17].

References

[1] Andrew Mcgregor. (2014) *Graph Stream Algorithms: a Survey* (SIGMOD).

[2] Aggarwal, Charu C. and Li, Yao and Yu, Philip S. and Jin, Ruoming (2010) *On Dense Pattern Mining in Graph Streams* Proceedings of the VLDB Endowment, 3(1-2), pp.975-984.

[3] Huang Z, Peng P. (2019) *Dynamic Graph Stream Algorithms in O(n) Space* Algorithmica. 2019 May 15;81(5):1965-87.

[4] Guha, Sumanta (2014) *Finding Frequent Subpaths in a Graph*, International Journal of Data Mining & Knowledge Management Process, Vol.4, No.5, pp. 35-46.

[5] Schwartz, S. and Balestrieri, L. and Borndörfer, R. (2017) *On Finding Subpaths With High Demand*, In

Table 16. Number of frequent singleton edges (|FSE|) for paths data using proposed static approach with actual minimum support using linear strategy and cubic strategy

minsup	$Actualminsup_l$	*|FSE| for linear strategy*	$Actualminsup_c$	*|FSE| for cubic strategy*
0.1	0.024	178	0.105	24
0.2	0.047	105	0.133	14
0.3	0.068	61	0.152	12
0.4	0.091	38	0.167	7
0.5	0.114	21	0.180	7
0.6	0.137	14	0.191	5
0.7	0.159	9	0.202	4
0.8	0.182	7	0.21	4

Table 17. Number of frequent singleton edges (|FSE|) for paths data with varying relative support(*relsup*) for proposed dynamic approach with fixed batch size (*DynFixed*) and variable batch size (*DynVar*)

relsup	|FSE| for *DynFixed*	|FSE| for *DynVar*
0.005	599	529
0.01	488	414
0.02	329	267
0.03	240	210
0.04	193	167
0.05	159	139
0.06	133	119
0.07	108	100
0.08	98	84
0.09	88	75
0.1	73	100
0.2	14	12
0.3	4	2

Table 18. Number of frequent subpaths (|FSP|) for paths data with varying relative support (*relsup*) for proposed dynamic approach with fixed batch size (*DynFixed*) and variable batch size (*DynVar*)

relsup	|FSP| for *DynFixed*	|FSP| for *DynVar*
0.005	75	75
0.01	35	35
0.02	9	9
0.03	9	7
0.04	5	5
0.05	3	3
0.06	2	2

Operations Research Proceedings, pp. 355-360, Springer, Cham.

[6] Bifet, A. and Holmes, G. and Pfahringer, B. and Gavaldà, R. (2017) *Mining frequent closed graphs on evolving data streams.* In Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining, pp. 591-599.

[7] Vitter and Jeffrey Scott (1998) *External Memory Algorithms* European Symposium on Algorithms pp. 1-25, Springer, Berlin, Heidelberg.

[8] Tang, N. and Chen, Q. and Mitra, P. (2016) *Graph Stream Summarization: From Big Bang to Big Crunch.* In Proceedings of the International Conference on Management of Data pp. 1481-1496.

[9] Leung, C.K.S. and Peng, P. (2016) *Parallel Social Network Mining for Interesting 'following' Patterns* Concurrency and computation: practice and experience, 28(15), pp.3994-4012.

[10] Cuzzocrea, A. and Han, Z. and Jiang, F. and Leung, C.K. and Zhang, H. (2015) *Edge-based Mining of Frequent Subgraphs from Graph Streams* Proceedia Computer Science, 60, pp.573-582.

[11] Bok, K. and and Choi, D. and Yoo, J. (2018) *Detecting Incremental Frequent Subgraph Patterns in IoT Environments* Sensors, 18(11), p.4020.

[12] Guha and Sudipto and Yoo, J. (2018) *Vertex and Hyperedge Connectivity in Dynamic Graph Streams* In Proceedings of the 34th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems pp. 241-247.

[13] Thomas Brewer (2019) *Connect4*, viewed 22 June 2019, <https://www.kaggle.com/tbrewer/connect-4/>.

[14] Craig Smith, (2020), DMR Publisher, *400 Interesting Twitter Statistics and Facts By the Numbers*, viewed 24 February 2020, <https://expandedramblings.com/index.php/twitter-stats-facts/>.

[15] Bhargavi B. and Swarupa Rani K. (2019) *Implicit Landmark Path Indexing for Bounded Label Constrained Reachable Paths*, International Journal of Recent Technology and Engineering (IJRTE), 8(4):p.10.

[16] Bhargavi, B. and Swarupa Rani, K. and Rohit Kumar and Sanmeet Kaur (2019) *Static and Dynamic Techniques to Extract Frequent Subgraphs from Graph Stream Data*, paper presented to International Conference on Big Data, Machine learning and Applications (BigDML 2019), NIT Silchar, Assam, India, 16-19 December, <http://bigdml.nits.ac.in/index.html#about/>.

[17] Zhang, S., Wu, X., Zhang, C. et al. (2008) *Computing the Minimum-Support for Mining Frequent Patterns* Knowl. Inf Syst 15, pp. 233–257.

[18] Jure Lescovec (2016) *SNAP: A general purpose network analysis and graph mining library in C++*, viewed June 2018, <http://snap.stanford.edu/snap>.

[19] Lee, Chang-Hung and Lin, Cheng-Ru and Chen, Ming-Syan (2005) *Sliding Window Filtering: An Efficient Method for Incremental Mining on a Time-Variant Database.*, Inf. Syst, Elsevier Science Ltd., Volume 30, number 3, pp. 227–244.