Journal of Engineering Science and Technology
Vol. 12, No. 2 (2017) 349 - 364

REGULAR PATTERN MINING (WITH JITTER) ON WEIGHTED-DIRECTED DYNAMIC GRAPHS

A. GUPTA, H. K. THAKUR*, T. GUPTA, S. YADAV

Department Of Computer engineering, Netaji Subhas Institute of Technology, University of Delhi, Dwarka, Delhi, India-110078
*Corresponding Author: hardeokumar@gmail.com

Abstract

Real world graphs are mostly dynamic in nature, exhibiting time-varying behaviour in structure of the graph, weight on the edges and direction of the edges. Mining regular patterns in the occurrence of edge parameters gives an insight into the consumer trends over time in ecommerce co-purchasing networks. But such patterns need not necessarily be precise as in the case when some product goes out of stock or a group of customers becomes unavailable for a short period of time. Ignoring them may lead to loss of useful information and thus taking jitter into account becomes vital. To the best of our knowledge, no work has been yet reported to extract regular patterns considering a jitter of length greater than unity. In this article, we propose a novel method to find quasi regular patterns on weight and direction sequences of such graphs. The method involves analysing the dynamic network considering the inconsistencies in the occurrence of edges. It utilizes the relation between the occurrence sequence and the corresponding weight and direction sequences to speed up this process. Further, these patterns are used to determine the most central nodes (such as the most profit yielding products). To accomplish this we introduce the concept of dynamic closeness centrality and dynamic betweenness centrality. Experiments on Enron e-mail dataset and a synthetic dynamic network show that the presented approach is efficient, so it can be used to find patterns in large scale networks consisting of many timestamps.

Keywords: Regular patterns, Jitter, Quasi-regular patterns, Centrality, Impact factor.

1. Introduction

Dynamic graphs, i.e., graphs that show time-varying temporal behaviour, find immense applications in social, biological, physical and information systems. In the past few years, many methods [1-19] have been proposed and studied

Nomenclatures

$B_c(i)$	Betweenness centrality of node i
$C_c(i)$	Closeness centrality of node i
$DB_c(i)$	Dynamic betweenness centrality of node i
$DC_c(i)$	Dynamic closeness centrality of node i
G	Subgraph
G_s	Summary graph
K	Maximum length jitter
L	Length of the rule
P	Regular Pattern
S	Minimum threshold for frequent
Th	Minimum consecutive repetition of a string for regular

Abbreviations

IF	Impact Factor
WDDN	Weighted-Directed Dynamic Network

extensively the dynamic graphs. Most of the methods [3-5, 9-12, 17-18] are in the sub-domain of frequent subgraphs. Frequent subgraphs are subgraphs that are present in more than a threshold number of timesteps in dynamic graphs. A graph represents a network; hence the article has used graph and network interchangeably.

However, mining only frequent subgraphs does not help in analysing several aspects of the network. To gain further insight, researchers mine repetitive patterns in frequently occurring subgraphs. Such patterns find immense applications in making future predictions such as stock prediction, weather forecasting, social network analysis, disease prediction, analysing consumer trends and many more. These patterns are further divided into two sub categories: i) Periodic patterns and ii) Regular patterns.

Periodic subgraph mining involves finding subgraphs that are found at fixed intervals. Lahiri et al. [9] make use of frequent subgraphs and pattern tree to mine periodic patterns. Han et al. [18] present several algorithms for efficient mining of partial periodic patterns by exploring some interesting properties related to partial periodicity such as the Apriori property and the max-subpattern hit set property, and by shared mining of multiple periods.

Mining regular patterns involves finding a repeated pattern in the sequence of some graph attributes (such as edges and nodes) representing certain characteristic or behaviour. Qin et al. [10] find regular evolution patterns in dynamic networks, taking into account one jitter. However, they consider only un-weighted and undirected graphs. Gupta et al. [11, 12] mine regular patterns on weighted and directed networks as well. Most of the above mentioned works mine only ideal patterns (that are seldom found in real-world applications) and do not consider the practical aspect of jitter/aberrations.

To overcome the limitation, the paper presents a novel method to mine **quasi** - regular patterns in weighted and directed dynamic graphs. It involves finding a repeated pattern (that is repeating at least a threshold number of times) in the

sequence representing structures, weights and directions, allowing a jitter of suitable length. But finding such patterns independently on the entire dynamic network consumes a considerable amount of time. To get over this drawback, patterns in weight and direction sequences are found only on those subgraphs, which demonstrate some regularity in structure. This leads to considerable decrease in the size of the problem and thus reduces the run time substantially. It is based on the concept that subgraphs showing regularity in weights or directions must show regularity in structure. For example, let us use '1' and '0' to represent existence and absence of an edge at a particular timestep, and 'f' and 'b' to represent forward and backward edges respectively. Let direction sequence of an edge be 'bf0bf0bf0', and then the edge repeats with pattern 'bf0'. It is evident that the structure sequence of the edge must be '110110110', thus showing regularity in structure with pattern '110'.

Another aspect in analysing dynamic networks is to find the most central node in the network. Lerman et al. [16] elaborate some such centrality measures in temporal graphs. But these measures do not take into account the regularity observed in these graphs. To account for this, the paper introduces the concept of dynamic closeness centrality and dynamic betweenness centrality based on impact factor. The impact factors are computed on the mined regular patterns.

The contribution of this paper is twofold. First, we propose a method to mine quasi regular patterns with k consecutive jitter in weighted directed dynamic network. Second, a notion for dynamic closeness centrality and dynamic betweenness centrality are introduced using impact factor.

The article is organized as follows: Section 2 presents definitions used in the article. The proposed method has been discussed in detail in Section 3. Experimental results and concept of dynamic closeness centrality and dynamic betweenness centrality have been discussed in Section 4, followed by the conclusions in Section 5.

2. Definitions

In this section, Dynamic Network and the types of patterns to be mined are defined.

Let $G = (V,E,W)$ denote a graph, where V,E,W are set of vertices, set of edges and set of weights on corresponding edges respectively. An edge is represented as an ordered pair (u,v), denoting a directed edge from u to v, where $u,v \in V$. Thus $E = \{(u,v) \mid u,v \in V\}$. Likewise, $W = \{w_{u,v} \mid (u,v) \in E\}$, where $w_{u,v}$ represents weight of the directed edge from u to v.

Definition 1 - Weighted-Directed Dynamic Network [12]**:** Given series of snapshots $G_{ss} = \langle G_1, G_2, G_3, G_4 \ldots G_T \rangle$ of a dynamic network *WDDN* at various timesteps, such that $1 \leq t \leq T$. Then *WDDN* represents a Weighted-Directed Dynamic Network where $G_t = (V_t, E_t, W_t)$.

Defined next, are the types of occurrence sequence constructed from the Weighted-Directed Dynamic Network *WDDN*.

Definition 2 - Occurrence Sequence [10, 12]**:** Occurrence sequence is defined with respect to structure, weight and direction as follows-

Occurrence Sequence on Structure: The occurrence sequence of an edge *e* is a string of 0s and 1s of length *T*, where 0 denotes absence and 1 presence of an edge *e* at timestep *t*. For example, the sequence "11001111" for an edge *e* shows the presence of *e* at timesteps 1, 2,5,6,7 and 8 and absence at timesteps 3 and 4. In this article we have represented occurrence sequence on structure as occurrence sequence.

Occurrence Sequence on Weight: The occurrence sequence of an edge *e* on weight is a string of length *T* of weight labels at timestep *t* of the edge *e*. For example, the sequence "pq00rslm" depicts weight labels of the edge *e* at various timesteps. Here, 0 denotes absence of the edge at particular time *t*.

Occurrence Sequence on Direction: The occurrence sequence of an edge *e* on direction is a string of length *T* of 'b's, 'f's and 0s of length *T* depicting the direction of edge *e* at each timestep, where 'b', 'f' and '0' respectively represent backward, forward and absence of the edge *e* at the particular time *t*.

Now, we arrange all edges of Weighted-Directed Dynamic Network *WDDN* in summary graph.

Definition 3 - Summary Graph [10, 12]**:** The summary graph G_s of a Weighted-Directed Dynamic Network $WDDN = <G_1,G_2,\ldots,G_T>$ maps each edge *e* to an occurrence sequence, weight sequence and direction sequence of length *T*. For convenience, we represent a summary graph as a matrix of size $|E|$ *X 5*, where $|E|$ is the total number of edges in the dynamic network. The first two columns of G_s store the two vertices of an edge and 3rd, 4th and 5th columns store respectively the occurrence sequences on structure, weight and direction.

Defined next, are the types of patterns needed to be mined from the Weighted-Directed Dynamic Network *WDDN*.

Definition 4 - Regular Pattern: Given a sequence corresponding to structure, weight or direction for an edge *e*, the substrings in the corresponding sequence satisfying the condition that the substring must be repeated consecutively at least threshold *th* number of times, are defined as *Regular Patterns*. There can be zero, one or more than one regular patterns for an edge *e*. The edge e is called regular edge if it consists of regular pattern of non-zero length. We use a three-tuple (*l, p, start*) to represent a regular pattern, where *l* is the length of the rule *p* and *start* is the starting position of the rule *p* in the corresponding sequence.

So far, we have considered only precise patterns but real world patterns may not be rigidly regular. Therefore, the concept of jitter comes into picture.

Definition 5 - Jitter: Jitter is defined as length of the maximum successive irregularity *k*, acceptable in a sequence for it to be declared a regular pattern. For example, an edge e_1 with occurrence sequence on Structure "110111001101" is regular with the pattern "1101" where jitter length is one at the 8thtimestep. Similarly, an edge e_2 with occurrence sequence "110010101100" is regular with pattern "1100" with jitter of length 2 at the 6th and 7thtimesteps. We need to bind the limits of the value of *k* as per the required applications. Intuitively the value of *k* should be less than the length of the pattern. In this paper, the value of *k* is taken as half the length of the pattern.

Since we mine Regular Patterns with jitter, a different term, Quasi Regular Pattern is used.

Definition 6 - Quasi Regular Pattern: Quasi Regular patterns corresponding to the occurrence, weight or direction sequence on an edge *e* are the regular patterns found on the corresponding occurrence, weight or direction sequences by taking Jitter into account.

To extract useful information from the dynamic network, it is divided into special static subgraphs, called Pattern Subgraphs, based on the Quasi Regular Patterns found on the edges. These subgraphs are formally defined below.

Definition 7 - Pattern Subgraph [10]: A subgraph *g* of the dynamic graph *G* is called a pattern subgraph if it is connected and all its edges repeat with the same Quasi Regular Pattern . Such a subgraph is represented as a four tuple *(g,l,p,start*) where *g* is the subgraph, *l* and *start* represent respectively the length and starting instant of the repeating pattern *p*. The pattern *p* accounts for the dynamic nature of the subgraph, whereas the subgraph *g* itself is static in nature. These subgraphs are un-weighted and undirected as their weights and directions are already accounted for in the pattern *p*. For example, for the dynamic network shown in Fig. 1, the subgraph {(1,2),(1,4)} forms a pattern subgraph repeating with pattern "1101" in structure and "pq0q" in weight sequence.

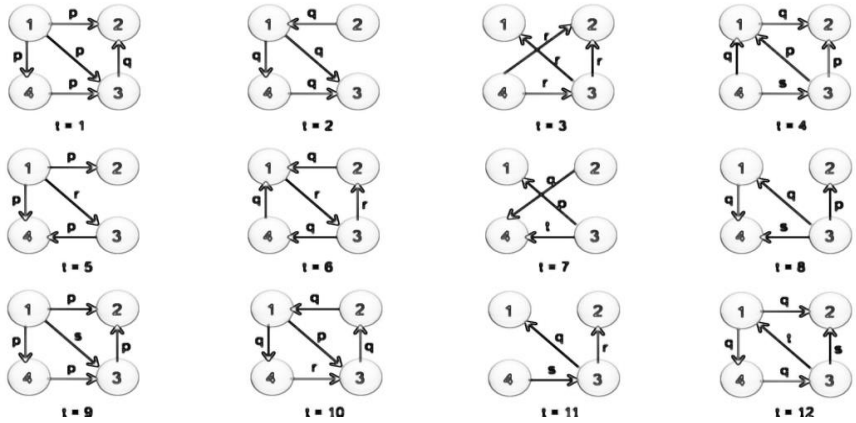


Fig. 1. Weighted directed dynamic network.

3. Detailed Method

Having formally defined the preliminaries, we now present the detailed method to find quasi-regular patterns in this Section.

3.1. Constructing a summary graph

The Summary Graph G_s of a weighted-directed dynamic network *WDDN* = $<G_1,G_2,....,G_T>$ is constructed corresponding to the occurrence, weight and direction sequences. The infrequent edges, i.e., the edges that occur less than a threshold *s* number of times are removed. Here *s* is taken as 3 and *T* as 12. A summary graph of the remaining edges is obtained and shown in Table 1.

Table 1. Summary Graph G_s corresponding to the network shown in Fig. 1.

V1	V2	Occurrence Sequence	Weight Sequence	Direction Sequence
1	2	110111001101	pq0qpq00pq0q	fb0ffb00fb0f
1	3	111111111111	pqrprrpqspqt	ffbbffbbffbb
1	4	110111011101	pq0qpq0qpq0q	ff0bfb0fff0f
2	3	101101011111	q0rp0r0ppqrs	b0bb0b0bbbbb
3	4	111111111111	pqrspqtsprsq	bbbbffffbbbb

3.2. Finding Quasi Regular Patterns

Here we have utilized the basic concept that an edge will be quasi regular on direction or quasi regular on weight if and only if the edge is quasi regular on structure. Here, we first mine quasi regular pattern on structure. We proceed to find quasi regular patterns in weight and direction only on those edges in which quasi regular patterns on structure are found. So we first find quasi regular pattern on structure by applying the algorithm (presented in Section 3.2.1) on each edge at occurrence sequence of summary graph. To find quasi regular pattern on weight or quasi regular pattern on direction we apply the same algorithm on only those edges that are regular on structure. Since weight and direction sequences are independent of each other, we can apply the algorithm on them in parallel to find quasi regular patterns on weight and on direction as shown in Fig. 2.

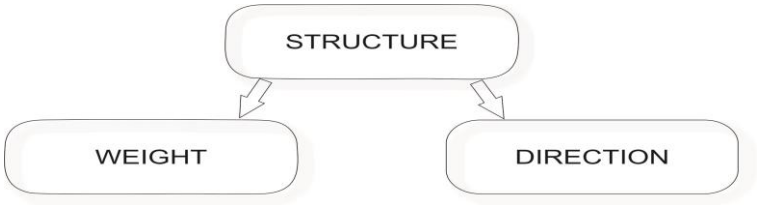


Fig. 2. Pictorial representation of the basic concept.

3.2.1. Algorithm

The function PatternSearch()is applied on each of occurrence, weight, direction sequences to obtain the corresponding pattern (*l, p, start*) for each edge.

Input: String such as occurrence sequence, weight sequence or direction sequence of an edge and threshold *th*

Output: The pattern found with jitter *k*

Function PatternSearch(*string, th*)

1. *strlen* = number of characters in *string*
2. *len*= floor(*strlen / th*)
3. **for** *n* in *len*downto 2 **do**
4. *k* = floor(*n/2*)
5. *i* = 1

6. **while**(*i*< = *strlen*) **do**
7. *NextIndex* = 1
8. *A* = substring from index *i* to *n* + *i* - *1*
9. **If** *A* consists of all zeros, discard it.
10. Set *i* = *i* + *n* and **continue**
11. **end if**
12. *count* = 1
13. *notEqual*= 0
14. **if**((*i*+*n*)>(*strlen*-*n*+*1*)) **then**
15. **Break**
16. **end if**
17. **for**(*j* in(*i*+*n*)to *strlen*-*n*+*1*,incrementing *j* by *n*)
18. *B* = substring from index *j* to *j* + *n* - *1*
19. **if**(*A* is not equal to *B*) **then**
20. **if**(*notEqual* is 0) **then**
21. *notEqual* = 1
22. *ncount* = 0
23. **for**(*t* from 0 to *n-1)*
24. **If**(character at *i*+*t* !=character at *notEqual*+*t*) **then**
25. *ncount*= *ncount*+ 1
26. **end if**
27. **end for**
28. **if**(*ncount*<= *k*) **then**
29. *count* = *count* +1
30. **else break**
31. **end if**
32. **else break**
33. **end if**
34. **else** *count* = *count* + 1
35. **end if**
36. **end for**
37. **if**(*count* != 1 and *count*>= *th*) **then**
38. **if**(for all patterns found so far with same starting timestep i, the current pattern found has maximum length)

39. Write *A* in the list of patterns found
40. *NextIndex = count*n*
41. **end if**
42. *i = i + NextIndex*
43. **end while**
44. **end for**
45. **end function**

The Algorithm takes as input the types of occurrence sequence on which the patterns need to be found as string and the threshold *th*. It finds the required substrings subject to the condition that the substrings must be repeated consecutively at least *th* times, allowing a jitter at most once in the entire sequence.

From line 3-41 the algorithm finds the required pattern of lengths varying from *len* down to 2. Patterns with all zero values are not considered. The consecutive substrings of length n are compared to substring A. Lines 15- 34 account for the jitter/noise in the pattern. If the substrings are not equal, the number of mismatching characters are counted, which should be less than or equal to *k*, the maximum allowed length of jitter. Also, if for the same starting timestep, more than one pattern is found, the one with maximum length is considered to resolve this conflict.

3.3. Finding pattern subgraphs

To find the pattern subgraphs, the edges with the same value of the three tuple (*l*,*p*,*start*) are grouped together. The connected components found in these edges are considered as pattern subgraphs having four-tuple (*g*, *l*, *p*, *start*).

Considering the dynamic network shown in Fig. 1, it can be seen that a pattern subgraph found for weight is a subgraph of the pattern subgraph found for structure. Similarly, a pattern subgraph found for direction is a subgraph of the pattern subgraph found for structure. This is shown in Fig. 3.

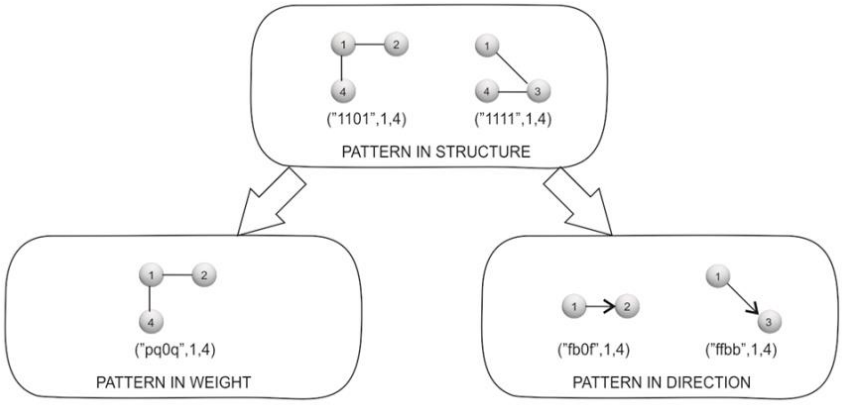


Fig. 3. Pattern subgraph.

3.4. Time analysis

Let *strlen* be the length of the input string. Then, the worst case time complexity of PatternSearch(*string*,*th*) is $O(strlen^4/th^2)$. Let e be the number of edges in the dynamic network. Then the time required to mine regular patterns in structure is of the order $O(e*strlen^4/th^2)$. Now, suppose the number of edges on which regular patterns in structure have been found is e_s. Then, the time required to find patterns in weight and direction individually is $O(e_s*strlen^4/th^2)$. Thus, the total time required is $O(e*strlen^4/th^2 + 2* e_s*strlen^4/th^2)$.On the other hand, if patterns on structure, weight and direction had been found separately on all edges, the running time would be of the order $O(3*e*strlen^4/th^2)$. Since $e_s \leq e$(in most practical applications $e_s << e$),the presented approach is a significant improvement over the latter.

To show the practical feasibility of the approach, we conduct experiments on an Enron emails and a synthetic dataset, the results of which are presented in Section 4.

4. Experimental result and notion of dynamic centrality

We have performed experiment on Enron email dataset. Since Enron email dataset is sparse in nature so we have conducted experiment on synthetic dataset also to evaluate performance on dense dataset. The experiment is conducted in R programming language and run on a Intel® Core™ i5 CPU- M430 @ 2.27Ghz 32-bit system with 4 GB RAM and Windows 7 Professional Operating System.

4.1. Experiment on two datasets

Enron email dataset: In the present paper Enron email dataset has been used to show the practical feasibility of the algorithm on a real world application. The dataset can be downloaded from http://www.cs.cmu.edu/~enron/. The Enron email communication network covers all the email communication within a dataset of around half million emails. Nodes of the network are email addresses and if an address *i* has sent at least one email to address *j*, the graph contains a directed edge from *i* to *j*. The weight of the edge depicts the number of characters sent from the sender to the receiver. Based on the above structure, graphs are constructed from the communications between email addresses from December 1999 to March 2002. Considering one month as one time-step, we get a dynamic network of 28 such graphs, consisting of a total of 2,342 nodes and 73,592 edges. Only those users (i.e., email addresses) are considered that exchange at least 3 emails. Self-loops (i.e., the edges from a node to itself) are ignored. The weights of edges between the same nodes at a timestep are added.

To ensure that we get regular patterns, weights w are categorized into 5 groups: $p \rightarrow w < 100$, $q \rightarrow 100 < w < 500$, $r \rightarrow 500 < w < 1000$, $s \rightarrow 1000 < w < 5000$ and $t \rightarrow w >> 5000$.To denote direction of an edge (u,v), where u and v are nodes, u being the node with lower node id. The letter f (i.e., forward edge) is used to denote a directed edge from u to v and b (i.e., backward edge) to denote a directed edge from v to u. After removing infrequent edges, i.e., the edges that occur less than 3 times, the number of edges reduces to 14,109. The experiment has been conducted in three ways:

i. Pattern subgraphs are made sequentially for structure, weight and direction known as old method.
ii. First, pattern subgraphs are made for structure. After that Pattern subgraphs are constructed sequentially for weight and direction, by considering quasi regular on structure edge only.

The efficiency of the proposed method can be shown by plotting the run times of both the methods, with varying number edges. Since the time required to find quasi regular patterns on occurrence sequences is same in both the cases, it is not considered while plotting the graph.

iii. To further reduce run time, parallel processing is used, using two cores to obtain pattern on direction and pattern on weigh parallel at quasi regular edges on structure. The graph obtained after conducting the above experiments is shown in Fig. 4.

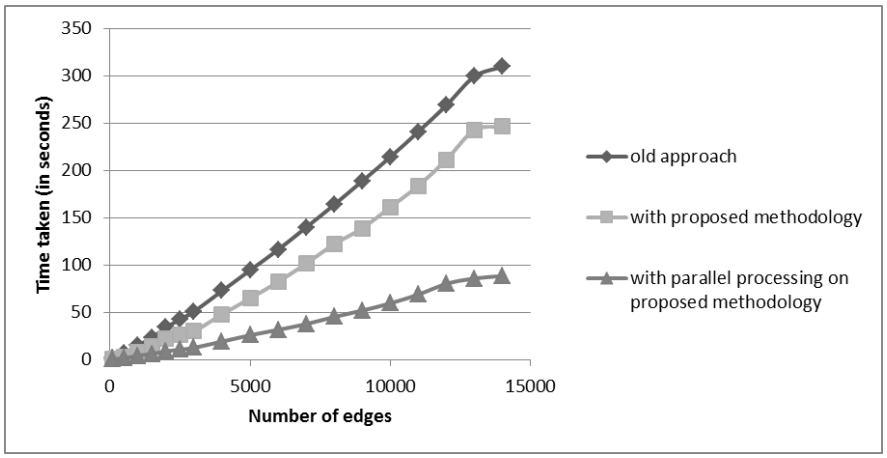
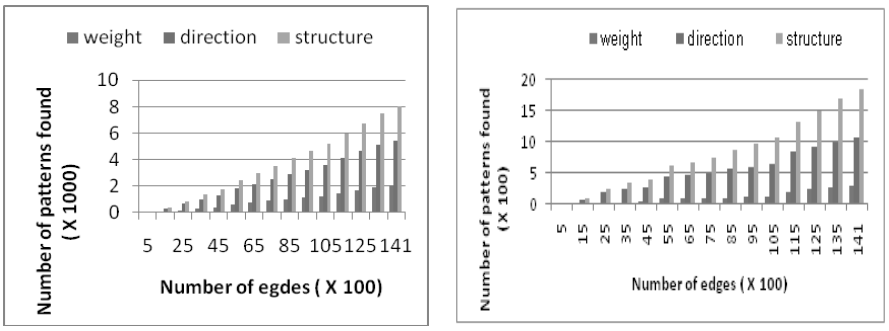


Fig. 4. Run time of algorithm in Enron email dataset.

On the Enron dataset, for an input of 14109 frequent edges, 10531, 2394, 6930 patterns are obtained on occurrence, weight and direction respectively. These patterns are obtained taking into account jitter. The numbers of patterns found with and without jitter on the Enron dataset have been shown in Figs. 5(a) and (b) respectively. It can be seen that the number of patterns found without considering jitter is significantly less. The difference between the number of patterns found with and without jitter is substantial, thus ignoring jitter might have led to loss of useful information.

Synthetic Dataset: We have generated a synthetic dynamic network dataset depicting sales trends of an e-commerce website. The dataset maintains a record of the sales of various products and their interdependencies. Graphs at 28 different timesteps are generated where each of the graphs represents one-month record. The nodes represent the various products. The Edges represent the correlation between the sales of various products. A directed edge exists from

node A to node B if more than μ (threshold) people buy product B after buying A. The network consists of 501 nodes and 73,848 edges. Here, weights *w* are categorized into 5 groups: p → 1<w<100, q →100< w<200, r → 200< w<300, s → 300< w<400 and t → w>400.The run time of the algorithm is shown in Fig. 6. Here we have taken threshold *s* equal to 2.



(a) Number of patterns with jitter. **(b) Number of patterns without jitter.**

Fig. 5. Number of patterns in Enron email dataset.

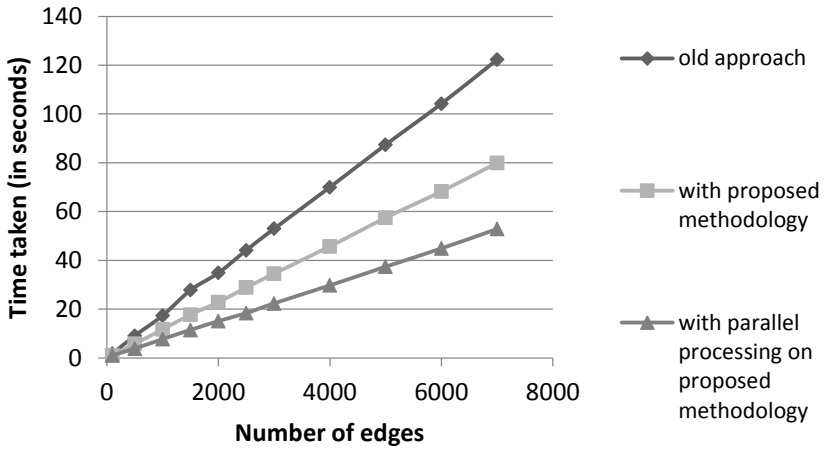
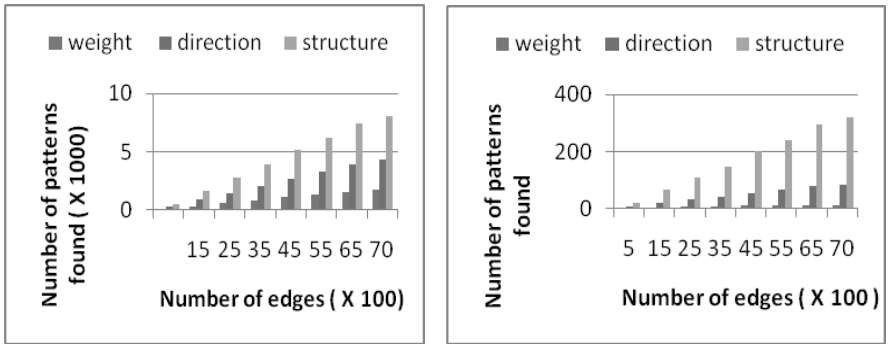


Fig. 6. Run time of algorithm pattern search (Synthetic dataset).

For synthetic product network, having an input of 7096 frequent edges, 8040, 1710, 4286 patterns are obtained on occurrence, weight and direction respectively. The numbers of patterns found with and without jitter on the synthetic dataset are shown in Figs. 7(a) and (b) respectively. Irregularities in sales of products may occur due to various reasons, like when a product goes out of stock or when some customers are unavailable. These are fairly common scenarios that might result in occurrence of jitter. The huge difference in the number of patterns found with and without jitter supports this. Thus, ignoring jitter might lead to loss of vital information about regularity in sales.



(a) Number of patterns with jitter. **(b) Number of patterns without jitter.**

Fig. 7. Number of patterns found in synthetic dataset.

4.2. Inference of Centrality

One important feature of networks is the relative centrality of individual nodes in them. Centrality is a structural characteristic of nodes in the network [20], meaning a centrality score tells us how that node fits within the overall network. Nodes with high centrality scores represent the most active or popular users.

We need to derive the centrality value of the nodes based on the patterns obtained on weight sequences. To achieve this, we propose the following dynamic centrality measures-

1. Dynamic Closeness Centrality -
Dynamic Closeness Centrality ($DC_c(i)$) of node i is given by Eq. (1) where $C_C(i)$ is the Closeness Centrality of node i and $IF(p_i)$ is the Impact Factor calculated from the pattern p_i associated with node i.

$$DC_c(i) = C_c(i) \times IF(P_i) \tag{1}$$

2. Dynamic Betweenness Centrality -
Dynamic Betweenness Centrality ($DB_c(i)$) of node i is given by Eq.(2) where $B_C(i)$ is the Betweenness Centrality of node i and $IF(p_i)$ is the Impact Factor calculated from the pattern p_i associated with node i.

$$DB_c(i) = B_c(i) \times IF(P_i) \tag{2}$$

These parameters are explained in detail in the following paragraphs.

Closeness Centrality [21]**:** This measure expresses the average distance from each node to every other node in the network. The closeness centrality can be calculated by dividing 1 by the average shortest path from a node to all other nodes in the network. The closeness centrality C_c of node i is given by Eq. (3).

$$C_c(i) = \left[\sum_{j=1}^{N} d(i, j) \right]^{-1} \tag{3}$$

where, N is the number of nodes reachable from i and $d(i,j)$ is the shortest distance of node j from node i. High closeness of a node means that it is close to many other nodes (i.e., users) and thus it is an influential node, i.e., it is a key conduit of information, and an early adopter of anything that spreads in a network.

Betweenness Centrality [22]**:** This measure expresses the number of shortest paths from all vertices to all others that pass through that node. It can be calculated as Eq. (4), where g_{jk} = the number of geodesics (shortest paths) connecting j and k, and $g_{jk}(i)$ = the number of geodesics that node i lies on. Betweenness centrality measures the extent to which a user lies between other users on their geodesics.

$$B_c(i) = \sum_{(j)!=(i)!=(k)} (g_{jk}(i)/g_{jk}) \quad (4)$$

Users high on betweenness centrality, therefore, have the potential to influence others near to them in a network [23].A node with high betweenness centrality can potentially influence the spread of information through the network, by facilitating, hindering, or even altering the communication between others [21].

These measures are used on the pattern subgraphs on weight sequence. Since these pattern subgraphs are static in nature, the dynamic nature of the network is accommodated using the concept of Impact Factor, explained below.

Impact factor *IF*: Impact Factor represents the contribution of the occurrence pattern on the centrality measure of a node and thus accounts for the dynamic nature of the graph.

Let the pattern be $P = <p_1 p_2 \ldots p_L>$ where L = length of the pattern on weight sequence, then, impact factor (*IF*) is calculated as Eq.(5), where the maximum possible sum of weights = L*maximum weight possible.

From the Eq. (5), we can see that impact factor is bounded as $0 < IF \leq 1$.

$$IF = \frac{\sum_{i=1}^{L} p_i}{\text{Maximum possible sum of weights}} \quad (5)$$

Equation (5) takes into account the following factors

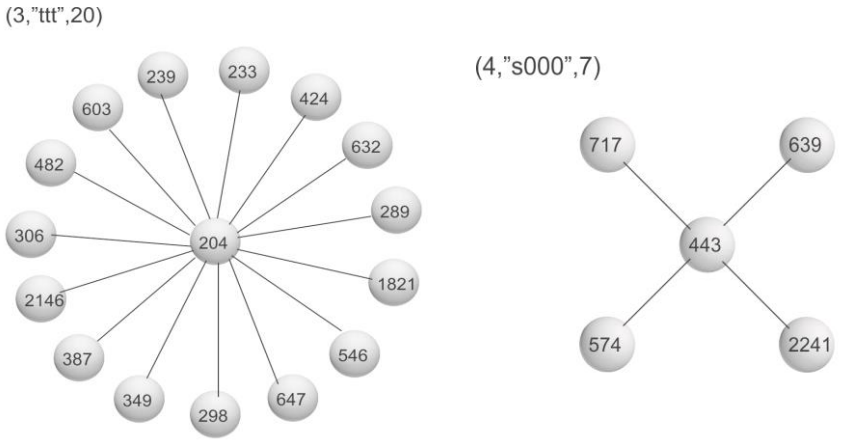
- The *IF* is directly proportional to the sum of weights in the pattern. For instance, let P_1 and P_2 be the occurrence patterns of two subgraphs, and they be "a0b" and "ac0" respectively, where a>b>c, then it means that the impact of P_1 should be greater than that of P_2.
- The IF is inversely proportional to the sparsity of the pattern. The patterns that have more number of zeros should be of less *IF* value. It is achieved by considering the maximum possible sum in the denominator. As an example, let P_1 and P_2 be the occurrence patterns of two subgraphs, and they be respectively "a00000a" and "b0b0b", where a = 20, b = 12. Applying Eq. (5), *IF* (P_1)and *IF* (P_2)are 0.285 and 0.36 respectively. Thus, *IF* (P_1) is less than *IF* (P_2). In another scenario, if *'a'* is much larger than *'b'*, the impact of P_1should be more than that of P_2. Let a = 60 and b = 12, then *IF* (P_1)is greater than *IF* (P_2).

The dynamic closeness and betweenness centralities are now calculated on two pattern subgraphs obtained from the Enron email dataset, Figs. 8(a) and (b).

To calculate the *IF*, Weight of each category is taken as mean value of the range for the category. Therefore, weight of *p*, *q*, *r*, s and t are 50, 300, 750, 3000 and 6000 respectively.

The pattern subgraph shown in Fig. 8(a) has pattern 'ttt', therefore the impact factor is equal to unity and the pattern subgraph shown in Fig. 8(b) has pattern 's000', therefore the impact factor is 0.125.

The values of closeness and betweenness centralities (calculated using the igraph package in R) of various nodes in the subgraphs has been shown in Tables 2 and 3. Here, both the dynamic centrality measure has maximum value for node 204 and node 443 as shown in Tables 2 and 3 respectively. Hence it is verified that these nodes (204, 443) are most central node.



(a) Pattern subgraph. **(b) Pattern subgraph.**

Fig. 8. Pattern subgraph Enron email dataset.

Table 2. Dynamic closeness and dynamic betweenness centralities of pattern subgraph shown in Fig. 8(a).

Node	Dynamic Closeness Centrality	Dynamic Betweenness Centrality
204	0.06666667	105
306,408,298,632,1821,424,387,2146, 647,289,603,239,349,233,546	0.03448276	0

Table 3. Dynamic closeness and dynamic betweenness centralities of pattern subgraph shown in Fig. 8(b).

Node	Dynamic Closeness Centrality	Dynamic Betweenness Centrality
443	0.03125000	0.75
717,639,2241,574	0.01785714	0

5. Conclusions

In this paper we have introduced a novel method to mine regular patterns in weighted directed dynamic network taking into account jitter of maximum length *k*. Importance of mining such patterns has been justified through analysis of results obtained after experiments on synthetically generated product network and the real-world Enron e-mail network. We find a significant difference in the number of patterns obtained with and without jitter. It shows that ignoring jitter might lead to a loss of consequential regular patterns. The charts detailing the results have also enabled meaningful deductions on local properties of the network and the patterns have provided significant details in the form of Dynamic Closeness and Dynamic Betweenness centralities through incorporation of weight in the analysis. Time response graphs have shown that the method is time-efficient and scalable for studying large networks. The experimental evaluation of the method ensures its efficiency and efficacy.

References

1. Bilgin, C.C.; and Yener, B. (2008). Dynamic network evolution: models, clustering, anomaly detection. *Technical Report*, Rensselaer University, NY.
2. Belov, N.; Martin, M. K.; Patti, Reminga, J.; Pawlowski, A.; and Carley, K.M. (2009). Dynamic networks: Rapid assessment of changing scenarios. *Proceedings of Second International Workshop on Social Computing and Behavioral Modeling, and Prediction.* Phoenix, Arizona, 33-41.
3. Borgwardt, K.M.; Kriegel, H.P.; and Wackersreuther, P. (2006). Pattern mining in frequent dynamic subgraphs. *Proceedings of the Sixth International Conference on Data Mining.* Hong Kong, China, 818-822.
4. Wackersreuther, B.; Wackersreuther, P.; Oswald, A.; Bohm, C.; and Borgwardt, K.M. (2010). Frequent subgraph discovery in dynamic networks. *Proceedings of the Eighth Workshop on Mining and Learning with Graphs in conjunction with Sixteenth ACM SIGKDD Conference on Knowledge Discovery and Data Mining* . Washington, DC, USA, 155-162.
5. Li, Y.; Lin, Q.; Zhong, G.; Duan, D.; Jin Y.; and Bi, W. (2009). A directed labelled graph frequent pattern mining algorithm based on minimum code. *Proceedings of the Third International Conference on Multimedia and Ubiquitous Engineering*. Qingdao, China, 353-359.
6. Duan , D.; Yuhua , L.; Yanan, J.; and Zhengding, L.(2009). Community mining on dynamic weighted directed graphs. *Proceedings of First ACM International Workshop on Complex Networks meet information & knowledge management*. Hong Kong, China, 11-18.
7. Robardet, C. (2009). Constraint-based pattern mining in dynamic graphs. *Proceedings of the Ninth IEEE International Conference on Data Mining.* Miami, USA, 950-955.
8. You, C.; Holder L.; and Cook, D. (2009). Learning patterns in the dynamics of biological networks. *Proceedings of the Fifteenth International Conference on Knowledge Discovery and Data Mining.* Paris, France, 977-986.

9. Lahiri, M.; Tanya, Y.; and Wolf, B.(2008). Mining periodic behavior in dynamic social networks. *Proceedings of the Eighth IEEE International Conference on Data Mining.* Pisa, Italy, 373-382.
10. Qin, G.; Gao, L.; Yang, J.; and Li, J. (2011). Evolution pattern discovery in dynamic networks. *Proceedings of the International conference on signal Processing, Communication and Computing.* Xian, Hong Kong, 933-938.
11. Gupta, A. and Thakur, H.K. (2013). A novel method to determine regular pattern in edge labelled dynamic graphs. *Proceedings of Seventh International Conference on Data Mining and Warehousing.* Bangalore, India, 39-47.
12. Gupta, A.; Thakur, H.K.; and Kishore, P. (2014). Mining regular patterns in weighted directed networks. *Proceedings of Thirteenth International Conference of Information Technology*. Bhubaneswar, India, 215-220.
13. Desikan, P.K.; and Srivastava, J. (2004). Mining temporally changing web usage graphs. *Proceedings of the Sixth International Workshop on Knowledge Discovery on the Web.* Seattle, WA, USA, 1-17.
14. Kempe, D.; Kleinberg, J.; and Kumar, A. (2000). Connectivity and inference problems for temporal networks. *Journal of Computer and System Sciences*, 64(4), 504-513.
15. Clauset, A.; Newman, M.; and Moore, C. (2004). Finding community structure in very large networks. *Journal Physical Review E*, 70(6), 066111.
16. Lerman, K.; Ghosh, R.; and Kang, J.H. (2010). Centrality metric for dynamic networks. *Proceedings of the Eighth Workshop on Mining and Learning with Graphs.* Washington, DC, USA, 70-77.
17. Yang, Y.; Jeffrey, Y.; Hong , G.; Jian, P.; and Jianzhong, L. (2014). Mining most frequently changing component in evolving graphs. *Journal of World Wide Web*, 17(3), 351-376.
18. Han, J.; Dong, G.; and Yin, Y. (1999). Efficient mining of partial periodic patterns in time series database. *Proceedings of Fifteenth International Conference on Data Engineering*. Sydney, Australia, 106-115.
19. Halder, S.; Han, Y.; and Lee, Y.K. (2013). Discovering periodic patterns using supergraph in dynamic networks. *Journal of Research Notes in Information Science* (*RNIS*), 14, 148-151.
20. Freeman, L. C. (1979). Centrality in social networks conceptual clarification. *Social Networks*, 1(3), 215-239.
21. Sabidussi, G. (1966). The centrality index of a graph. *Psychometrika*, 31, 581-603.
22. Ulrik Brandes (2008). On variants of shortest-path betweenness centrality and their generic computation. *Social Networks*, 30(2), 136-145.
23. Friedkin, N.E. (1991). Theoretical foundations for centrality measures. *American Journal of Sociology*, 96(6), 1478-1504.