Superimposing Periodic Subgraph Mining in Dynamic Social Network

S. Vairachilai

Abstract Nowadays, complex network has been inspired by the observation of many real-world network patterns such as pattern of mobile phone usage, gene expression, e-mail habits, protein interactions in networks, regional interaction in archaeological contexts and co-authorship in research publications. The detection of interaction patterns that occur at fixed time periods can be mined in a complex network. This article proposes an algorithm to predict an interaction pattern hidden behind these networks, namely periodic subgraphs embedded in a complex network. The primary data structures used in tracking the periodic subgraphs are lists, tables, data frames, matrices and vectors. The algorithm can be implemented using analytical tools that do not require advanced data structures such as pattern trees and indexing techniques such as hash maps. In this paper, our algorithm processes a dynamic network in two stages. The first stage of the algorithm extracts all maximal subgraphs which appear between successive time steps. In second stage, it maintains the periodicities of the maximal subgraphs identified and flushes out those graphs whose periodicities ceased to exist. The algorithm is tested by generating a dynamic network using random graph generating function and by superimposing periodic subgraphs on the network at different time interval periods.

Keywords Dynamic network · Subgraph mining · Maximal subgraph · R programming

1 Introduction

Dynamic networks are very powerful model for representing time-varying structure in the network. The interaction among these networks varies with respect to time. In dynamic network analysis, frequent and periodic substructures are the basic patterns which can be discovered in a temporal collection of graphs. The several methods and

S. Vairachilai (✉)
Department of Computer Science and Engineering, CVR College Of Engineering, Hyderabad 501510, India
e-mail: vairachilai2676@gmail.com

V. Bhateja et al. (eds.), *Data Engineering and Intelligent Computing*,
Advances in Intelligent Systems and Computing 1,
https://doi.org/10.1007/978-981-16-0171-2_58

algorithms have been developed to analyse and discover interesting patterns in graph data applications [1]. Mining the communities on temporal graphs has more attention due to the several real-time applications [2, 3]. Elfeky et al. [4] developed an algorithm to find the periodic patterns in time series database. They found the periodic patterns of unknown periods without affecting the time complexity in database. Cao et al. [5] proposed a solution to mine periodic patterns in spatiotemporal data. They proposed an efficient algorithm for retrieving maximal periodic patterns. Lahiri and Berger-Wolf [6, 7] introduced periodic patterns mining in dynamic networks. They proposed PSEMiner algorithm that finds periodic patterns in polynomial time. Tanbeer et al. [8] introduced periodic–frequent patterns that the patterns occur frequently and regularly in a database. Apostolico et al. [9] introduced ListMiner algorithm to solve unessential tree node creation problem and speed up PSEMiner algorithm.

The periodic and frequent interactions among the elements in a data set can be modelled as a graph. In graph, the set of vertices are represented as individual elements and the set of edges are represented as interactions between them. The interactions between the individual elements are noted over a period of time. The time periods are divided into discrete time steps of equal duration. These data formulate a dynamic network. The size and the complexity of the network increase from social network analysis (SNA) to dynamic network analysis (DNA). The algorithm is also difficult to predict and hard to detect patterns hidden behind these networks. In the last few years, the discovery of graph structures that occur a significant number of times in a dynamic network has been an active research. It focuses on dynamic state and dynamic event networks [10]. In dynamic state networks, links have a tendency to endure. These applications are found in protein network, social network and collaboration networks. In dynamic state networks, dynamic event networks have links which represent brief events which may not endure for a prolonged period. The examples of such networks include e-mail and telephone call graphs. The proposed superimposing periodic subgraph mining time complexity is not increased exponentially.

The rest of this paper is organized as follows. In Sect. 2, periodic subgraph mining has been discussed. In Sect. 3, the simulated network and the periodic subgraphs superimposed at different time intervals are discussed. The proposed algorithms are discussed in Sect. 4. In Sects. 5 and 6, the time complexity and experimental results are discussed. The conclusion and the direction of future work are discussed in Sect. 7.

2 Related Work

2.1 Preliminaries

Dynamic Network: In dynamic network, <G1,…,GT> consisting of simple graphs Gt = <Vt,Et> over a unique labelled set of vertices {Vt} and a minimum support threshold which represents the minimum number of time steps in which a periodic

subgraph appears, the algorithm to detect periodic subgraphs will list all periodic subgraphs having different periods appearing at different time steps of the dynamic network.

Periodic Subgraph: A subgraph is periodic if its support (occurrence frequency) in a given dynamic network is not less than a minimum support threshold and it occurs in a regular time interval (step). Support is the number of subgraphs that contain the pattern (occurrence frequency). The minimum support values (σ) lie between $1 \leq \sigma \leq t$, where 't' is the number of time steps in the dynamic network.

Maximal Common Subgraph (MCS): The MCS is obtained by taking the intersection of their set representations. The MCS in graph is unique and could possibly be the empty graph with no vertices or edges.

2.2 Periodic Subgraph

A sequence of subgraphs uniquely labelled vertices in a dynamic network, the periodic subgraph is a graph which reappears at regular intervals. Apostolico et al. [11] discussed the periodic subgraph mining problem that consists of discovering maximal subgraphs that repeat at regular intervals. When a series of observations were captured as temporal graphs, detecting periodically recurrent sequence of subgraphs having uniquely labelled edges demands efficient algorithms.

Lahiri and Berger-Wolf [6] have proposed an algorithm to detect interaction patterns hidden behind such dynamic networks by identifying subgraphs that occur periodically in the network. The primary data structures used in tracking the subgraphs are pattern tree, subgraph hash map and an auxiliary data structure known as a descriptor to generate support set. The algorithm was implemented using a pattern tree that maintains information about all patterns that are either currently periodic or could become periodic at a future time step. As each time step of the dynamic network is read, the pattern tree is traversed and updated by flushing out the patterns that are no longer periodic, and also identifying new possible periodic subgraphs. In this article, an attempt was made to develop a similar algorithm using the concepts of data sets and data structures such as lists, tables, data frames, matrices and vectors in R and SAS programming languages. The developed algorithm is tested using randomly generated graphs and by superimposing periodic subgraphs that appear over time periods. The algorithm was implemented using R environment which provides an appropriate data structure to handle graph databases and also provides functions to generate random graphs.

3 Simulated Network

One can use existing graphs captured at a sequence of time steps from real-world problems and detect maximal common subgraphs that occur either frequently or periodically. However, gathering such data is extremely time-consuming. Alternatively, one can choose to use graph generators to create and simulate real-world problems. In this article, an attempt was made to generate a time series of random graphs as a dynamic network and superimpose periodic subgraphs that occur with different periodicities during different time intervals. Many of the studies of random graphs have their origin back to the work of Erdős and Rényi [12–14] whose fundamental work has laid the foundation for the theory of random graph generation. There are two standard models for Erdős random graph generation [14], in which two parameters were used one to control the number of nodes while generating the random graph and the other to control the density or a number of edges in the network. For example, the random graph model $G(N, M)$ assigns a uniform probability to all graphs with N nodes and M number of edges, while the other graph model $G(N, p)$ generates a graph with N number of nodes and the edges were chosen randomly with probability p. The later model is used in this research to generate random graphs at each time step of the dynamic network using Erdos–Renyi graph function. To maintain a unique set of labels for the vertices over different time periods, we have assigned unique numbers to the vertices after generating the graphs at each time step. In real-world applications such as social media networks, all nodes are not likely to be active simultaneously. To achieve this condition, we tested the model with different probability values and fixed a low value for p and eliminating nodes with zero vertex degrees.

To superimpose periodic subgraphs with different periodicities, we have considered a couple of standard connected graphs such as star and ring. We superimpose such graphs having a small number of nodes and edges to the randomly generated graphs over different time periods of the network. The entire sequence of graphs generated was stored as data sets, and an index file was maintained to retrieve any graph at any selected time step randomly. As an example, we have shown the graphs at time step 10 before and after superimposing subgraphs in Figs. 1, 2 and 3. Figure 1 shows the randomly generated graph at time step 10. This randomly generated graph consists of 50 nodes. Figure 2 depicts periodic subgraphs superimposed at different time intervals. In this figure, the periodic subgraphs with periods 1 and 2 are superimposed between steps 2–8 and 4–10, respectively. Figure 3 shows the random graph along with the period's graphs embedded at time step 10. The standard connected star graph with 6 nodes is generated.

4 Proposed Algorithm

The algorithm proposed in this article was implemented using R environment. The advantage of using R is that it gives an additional leverage to use statistical computing

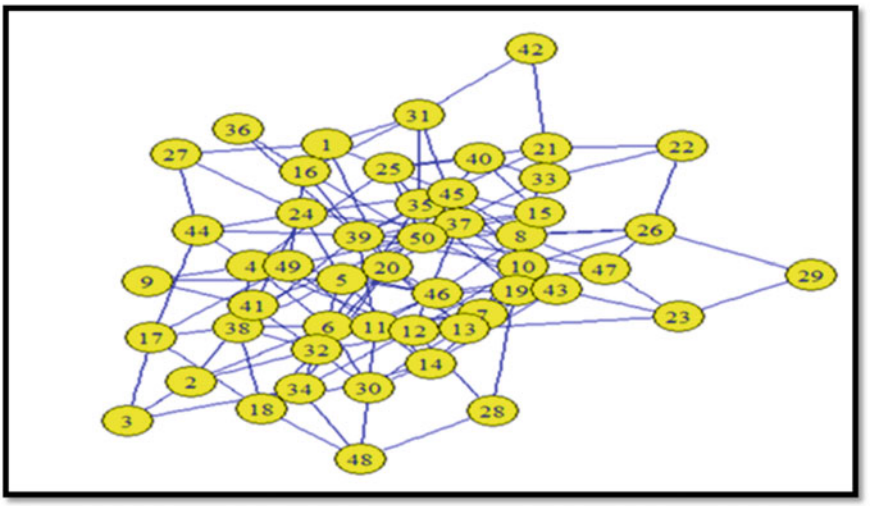

Fig. 1 Random graph at time step 10

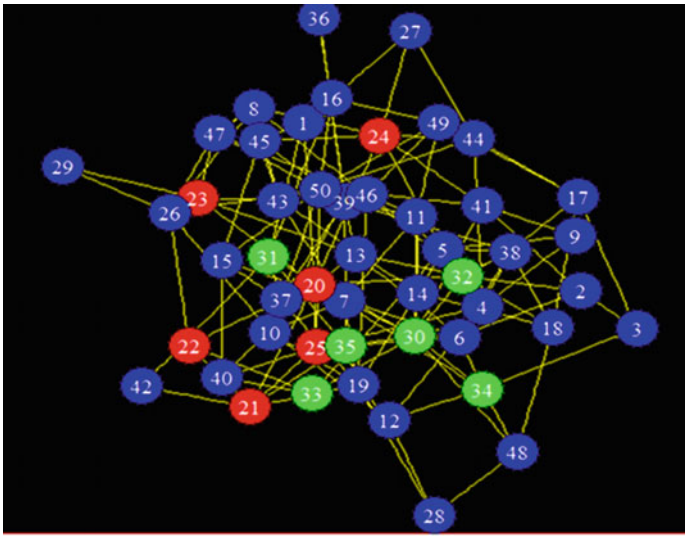

Fig. 2 Periodic subgraphs superimposed at different time intervals

features and also allows us to plot the graphs resulted at various stages of computations. The built-in functions allow us to intersect and decompose the subgraphs and maintain in appropriate lists. To implement the algorithm, we need to have appropriate data structures to store the temporal series of graphs and the intermediate subgraphs resulting from intersections of graphs between various time steps. The

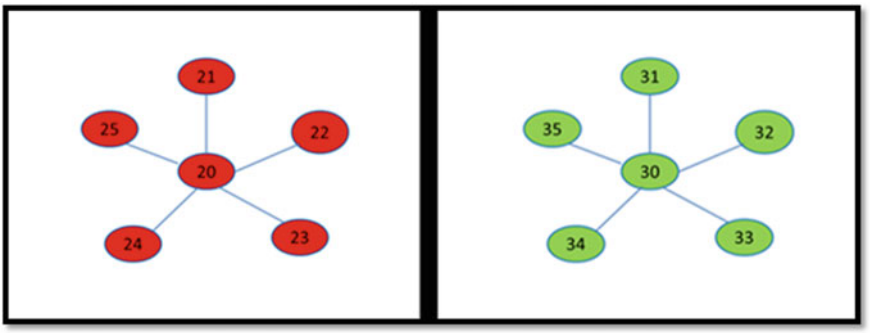

Fig. 3 Random graph along with the period's graphs embedded at time step 10

algorithm proposed by Lahiri and Berger for the current problem needs to have data structures such as link lists, pattern trees and appropriate indexing techniques such hash map. Data analysis tools such as R allow any application to use five common data structures: vectors, matrices and arrays, factors, lists and data frames. We have made use some of these data structures to implement the proposed algorithm.

The first phase of the algorithm reads the input stream of graphs sequentially. The graph at every time step is intersected with the rest of the graphs for all possible time periods. Maximal subgraphs were extracted from the intersected graphs and eliminating isolated graphs. The set of maximal subgraphs is stored in a vector for each one of the periods as a list of graphs. As we have considered randomly generated graphs at each time step, the maximal subgraphs for each time period were observed reasonably low.

4.1 Algorithm for the Phase-I

Input: Time Series of N graphs

Output: Vector of list of maximal subgraphs

Algorithm

1. for (i = 1, i < n, i++) //i denotes the time step for the first graph to be intersected
2. {
3. for (j = i, 1, i − 1) //j denotes the time step for the second graph to be intersected
4. {
5. firstGraph<-read (Gi)
6. secondGraph<-read (Gj)
7. intersectedGraph<-graph. intersection (firstGraph, secondGraph)
8. intersectedGraph<-removeIsolatedVertices(intersectedGraph)
9. maximalSubGraph[j]<-extractmaximalSubGraph(intersectedGraph)

Table 1 Output at the end of the phase-I of the algorithm

	F	S	P	N	From	To
[1]	2	1	1	3	1	6
[2]	3	2	1	4	7	12
[3]	3	1	2	5	13	18
[4]	4	3	1	5	19	26
[5]	4	2	2	6	27	30
[6]	4	1	3	7	31	34
[7]	5	4	1	6	35	41
[8]	5	3	2	7	42	51
[9]	5	2	3	8	52	57
[10]	5	1	4	9	58	65
…	…	…	…	…	…	…
[185]	20	6	14	34	1290	1296
[186]	20	5	15	35	1297	1305
[187]	20	4	16	36	1306	1309
[188]	20	3	17	37	1310	1317
[189]	20	2	18	38	1318	1326
[190]	20	1	19	39	1327	1333

10. table<-(I, k, p, nm) //p is the time period between the intersected graphs and nm is the number of maximal subgraphs stored in the List
11. }
12. }

The output at the end of phase-I of the algorithm is shown in Table 1.

The output of the phase-I maintains a table for each intersected graph the following information:

F	Time step of the first intersected graph.
S	Time step of the second intersected graph.
P	Time period at which the maximal subgraph appeared.
N	Next time the subgraph expected to appear to maintain its period (P).
From and To	The first and the last maximal subgraph stored in the list at time step S.

4.2 Algorithm for the Phase-II

Input: maximalSubGraphs [] //Vector of Lists of Maximal Subgraphs

Table [F, S, P, N, From, To] //Table of information for each maximal subgraph with minimum support threshold of 3-time steps

Output: Periodic graphs with different periods and their associated start and end time steps

Algorithm

1. for (p = 1, p < ((N/2) − 1), p++)
2. { // p is the periodicity of the maximal subgraph
3. matrix<-Table [P==p]
4. for (row = 1, row <= maxRow, row = row + p)
5. { //maxRow is the last row of the matrix
6. for(i = From(row), i < To(row), i++)
7. { // i is the graph index at the current row of the matrix
8. currentGraph<-g[i]
9. for (j = 1, j <= row + p, j++) // j is the graph index at the next row of the matrix
10. nextGraph<-g[[gindex_2]]
11. intersectedGraph<-graph. intersection (currentGraph, nextGraph)
12. if (intersectedGraph! = ø) (replace the intersected graph as the current graph for future intersection and break the loop)
13. }
14. if (intersectedGraph==ø) (flush out the currentGraph and remove the same from the list)
15. }
16. }

During the second phase of the algorithm, the maximal subgraphs stored in the vector of a list of graphs were further processed. The aim of this phase is to keep track whether the already appeared periodic maximal subgraph with minimum support threshold of three time steps continues to maintain its periodicity at the future time steps. If the subgraph ceased to maintain its periodicity, it is flushed out in the output stream and removed from the list of graphs maintained in the vector. Sub-tables were formed from the table resulted in phase-I to subset the information for each of the periodicities. The maximal subgraphs resulted from the phase-1 are further intersected with the subgraphs in the rest of the steps. If the intersection results in the empty graph at any point of time, it denotes that its periodicity ceased to exist and hence the graph is routed. If the intersection is not empty, upgrade the intersected graph as the current graph for the next step.

5 Complexity Analysis of Proposed Algorithm

The complexity of an algorithm indicates that how fast or slow the particular algorithm performs. The time complexity is generally calculated by counting the numbers of basic operations carried out by the algorithm. The time complexity for phase-I algorithm and phase-II algorithm (Tables 2 and 3) is tested in two ways: (i) the

Table 2 Analysis of the time complexity for phase-I and phase-II algorithm for fixed number of edges

Number of edges	Number of time steps	Phase-I		Phase-II	
		Execution time (in seconds)	Maximal subgraphs	Execution time (in seconds)	Periodic subgraphs
40	20	4.23	961	14.71	15
40	30	9.24	2318	34.89	38
40	40	22.50	4179	51.34	50
40	50	31.30	6279	64.72	57
40	60	45.58	9313	82.67	72

Table 3 Analysis of the time complexity for phase-I and phase-II algorithm for fixed number time steps

Number of edges	Number of time steps	Phase-I		Phase-II	
		Execution time (in seconds)	Maximal subgraphs	Execution time (in seconds)	Periodic subgraphs
40	20	4.23	961	14.71	15
50	20	6.15	1399	33.87	33
60	20	4.66	1740	60.95	33
70	20	5.48	2017	74.80	54
80	20	5.18	2342	108.06	78

numbers of edges are fixed, and the numbers of time steps are increased by 10. (ii) The numbers of time steps are fixed, and the numbers of edges are increased by 10. In both cases, the execution time is not increased linearly. At the same time, the execution time increase is not increased exponentially also. The time taken to intersect the two graphs and to find the maximal subgraphs depends on the R package. The structure of the maximal subgraphs depends on the number of subgraphs present in the dynamic network.

6 Experimental Results

The algorithm developed to process the dynamic network was tested on randomly generated time series of graphs. We have generated a network of different sizes and superimposed periodic subgraphs at different time intervals. A randomly generated network at twenty time steps with fifty nodes and randomly generated edges and two periodic subgraphs were superimposed with periods of one and two between ten time steps, and the phase-I algorithm has generated 1327 maximal subgraphs

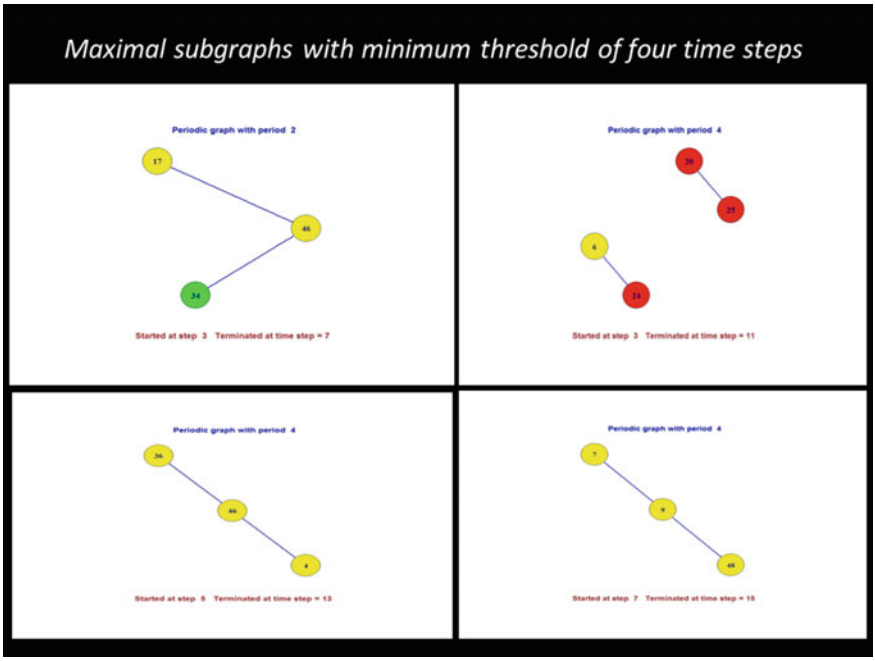

Fig. 4 Maximal subgraphs with minimum thresholds of four-time steps

with minimum support threshold of three time steps. Figure 4 shows the maximal subgraphs with minimum thresholds of four time steps. On further processing, these subgraphs, the phase-II of the algorithm flushed out 89 maximal subgraphs for more than four time steps as minimum support threshold limit. Out of these 89 graphs, more than 46 graphs had single edge graphs.

We recollect that subgraphs shown in Fig. 2 were superimposed as periodic subgraphs with periods of 1 and 2, respectively, between time steps 1 and 10. The algorithm was able to detect these graphs as periodic subgraphs with maximum periods of 4 time steps as shown in Fig. 5a, b. The graphs shown in Fig. 5c, d are the other maximal periodic subgraphs detected with periods of 4 and 2 time steps, respectively. Though the superimposed graph with period 1 remained periodic with a maximum period of 4 time steps between time steps 2 and 10, it appeared as a subgraph to another maximal subgraph with period 4 between time steps 1 and 9. Similarly, though the superimposed graph with period 2 remained periodic with a maximum period of 4 time steps between time steps 2 and 10, it appeared as a subgraph to another maximal subgraph with period 2 between time steps 6 and 10. The workflow to extract the maximal subgraph is shown in Fig. 6.

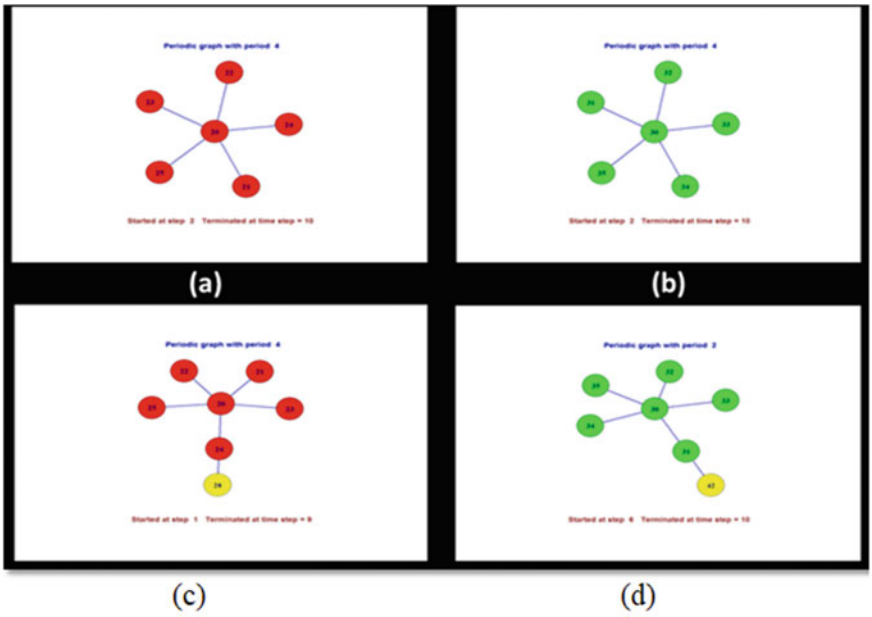

Fig. 5 Superimposed graphs detected as maximal periodic subgraphs with different periodicities

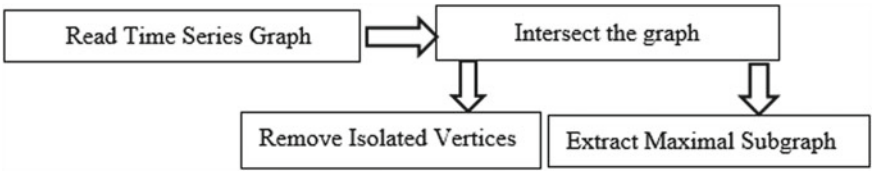

Fig. 6 Workflow to extract the maximal subgraph

7 Conclusions and Perspectives

The proposed algorithm was implemented using igraph package under R environment and tested using randomly selected streaming graphs generated by the Erdos–Renyi graph function. The implemented algorithm predicts the subgraphs superimposed on the dynamic network simulated along with other closed periodic subgraphs. The complexity of the algorithm is established as the algorithm depends on various data structures built in R environment. The developed algorithm was also tested to detect periodic subgraphs based on attributes attached to the vertices and edges of the graphs. In future, this algorithm can be tested with the social network application domains.

References

1. B. Boden, S. Gunnemann, H. Hoffmann, T. Seidl, MiMAG: mining coherent subgraphs in multi-layer graphs with edge labels. Knowl. Inf. Syst. **50**(2), 417–446 (2017)
2. R.-H. Li, J. Su, L. Qin, J.X. Yu, Q. Dai, Persistent community search in temporal networks, in *IEEE ICDE* (2005)
3. S. Ma, R. Hu, L. Wang, X. Lin, J. Huai, Fast computation of dense temporal subgraphs, in *IEEE ICDE* (2017)
4. M.G. Elfeky, W.G. Aref, A.K. Elmagarmid, Periodicity detection in time series databases. IEEE Trans. Knowl. Data Eng. **17**(7), 875–887 (2005)
5. H. Cao, N. Mamoulis, D.W. Cheung, Discovery of periodic patterns in spatiotemporal sequences. IEEE Trans. Knowl. Data Eng. **19**(4), 453–467 (2007)
6. M. Lahiri, T.Y. Berger-Wolf, Mining periodic behavior in dynamic social networks, in *IEEE International Conference on Data Mining*, pp. 373–382 (2008)
7. M. Lahiri, T. Berger-Wolf, Periodic subgraph mining in dynamic networks. J. Knowl. Inf. Syst. **24**(3), 467–497 (2010)
8. S.K. Tanbeer, C.F. Ahmed, B.S. Jeong, Y.K. Lee, Discovering periodic frequent patterns in transactional databases, in *Pacific–Asia Conference on Advances in Knowledge Discovery and Data Mining. Lecture Notes in Computer Science*, vol. 5476 (2009), pp. 242–253
9. A. Apostolico, M. Barbares, C. Pizzi, Speedup for a periodic subgraph miner. Inf. Process. Lett. **111**(11), 521–523 (2011)
10. K. Macropol, A. Singh, Reachability analysis and modeling of dynamic event networks, in *Machine Learning and Knowledge Discovery in Databases. Lecture Notes in Computer Science*, vol. 7523 (2012), pp. 442–457
11. A. Apostolico, P.L. Erdős, E. Győri, Z. Lipták, C. Pizzi, Efficient algorithms for the periodic subgraphs mining problem. J. Discrete Algorithms **17**, 24–30 (2012)
12. P. Erdős, A. Rényi, On the evolution of random graphs. Publ. Math. Inst. Hung. Acad. Sci. 17–61 (1960)
13. P. Erdős, A. Rényi, On the strength of connectedness of random graphs. Acta Math. Acad. Sci. Hungar. 267 (1961)
14. P. Erdős, A. Rényi, On random graphs I. Publicationes Math. Debrecen **6**, 290–297 (1959)