Knowl Inf Syst (2010) 24:467–497
DOI 10.1007/s10115-009-0253-8

REGULAR PAPER

Periodic subgraph mining in dynamic networks

Mayank Lahiri · Tanya Y. Berger-Wolf

Received: 16 March 2009 / Revised: 30 June 2009 / Accepted: 1 August 2009 /
Published online: 22 September 2009

Abstract In systems of interacting entities such as social networks, interactions that occur regularly typically correspond to significant, yet often infrequent and hard to detect, interaction patterns. To identify such regular behavior in streams of dynamic interaction data, we propose a new mining problem of finding a minimal set of periodically recurring subgraphs to capture all periodic behavior in a dynamic network. We analyze the computational complexity of the problem and show that it is polynomial, unlike many related subgraph or itemset mining problems. We propose an efficient and scalable algorithm to mine all periodic subgraphs in a dynamic network. The algorithm makes a single pass over the data and is also capable of accommodating imperfect periodicity. We demonstrate the applicability of our approach on several real-world networks and extract interesting and insightful periodic interaction patterns. We also show that periodic subgraphs can be an effective way to uncover and characterize the natural periodicities in a system.

Keywords Graph mining · Dynamic social networks · Periodic patterns · Frequent closed subgraphs · Parsimony

1 Introduction

Many natural and artificial systems can be modeled as a set of individual actors or entities, such as humans, animals or computers, interacting among themselves. Network analysis is the study of the structural and dynamic aspects of these interactions, in an effort to better understand the nature of the underlying system. In this paper, we deal with the detection of a type of predictable behavior in such systems, namely periodically recurring interaction patterns in networks that change over time. Our goal is to detect periodic behavior even if it persists only for a short period of time, since such *locally periodic* behavior often holds a

M. Lahiri (✉) · T. Y. Berger-Wolf
Department of Computer Science, University of Illinois at Chicago,
Chicago, IL, USA
e-mail: mlahiri@gmail.com

special meaning in real-world systems. As the simplest form of predictable behavior, periodic interaction patterns can indicate interesting relationships between the individuals involved in the interactions. Furthermore, with the right formal definition of what constitutes periodic behavior, the aggregate periodicities of an entire set of mined interaction patterns can yield insight about the global dynamics of the system being observed. We define the periodic pattern mining problem for dynamic networks as a step towards this goal, and describe an efficient algorithm to mine all such patterns from a stream of dynamic interaction data.[1]

Part of the motivation for our work is the fact that streams of time-varying interaction data are being collected in very diverse settings, which makes efficient, principled methods for analyzing such data imperative. Although the best known example of network analysis is perhaps *social network analysis* [35], network analysis has more recently been used in a variety of fields to analyze systems as diverse as the Internet [15], animal behavior [16,33], e-mail habits [7,11], mobile phone usage patterns [27], and co-authorship patterns in research publications [3,28]. There is also a recognized, emerging need to analyze the dynamic aspects of interaction data in other fields. For example, ecologists often tag wild animals with GPS or proximity sensors to study behavioral and social association patterns of the animals [16,22,33]. This results in a continuous stream of interaction data, where periodically recurring patterns might correspond to seasonal or other recurrent association patterns. The same methodology has been used in human behavior experiments, with location-aware cellphones naturally replacing tracking collars [12]. Analyzing the local periodicities in such datasets presents opportunities for social science research, as well as commercial applications such as recommender systems, traffic analysis and user modeling. The method presented in this paper helps to answer two questions: what are the typical periodicities present in a dataset, and what are the specific interaction patterns that occur at these periodicities?

Our definition of the periodic pattern mining problem is specifically tailored for the analysis of dynamic networks, and is generic enough to handle all the situations just mentioned. It differs from earlier work in periodic pattern mining primarily in the use of two related concepts: (a) the concept of closed subgraphs, and (b) the principle of parsimony. Closed subgraph mining has been extensively explored in the context of a related problem of *frequent pattern mining* [18]. It draws from the areas of formal concept analysis and lattice theory to reduce redundancy in the definition of a frequent pattern, and thus reduces the potentially exponential (in the size of the input) number of output patterns that must be computed [6,30]. The principle of parsimony is commonly known as Occam's Razor, and is a widely practiced guideline that suggests favoring the simplest hypothesis that is consistent with a phenomenon. Combining these two concepts allows us to define periodic patterns in a way that avoids any redundant information, is more amenable to analysis, and allows the development of a provably efficient online mining algorithm. Furthermore, all the information contained in earlier definitions of periodic pattern mining is contained in ours in a more compact form, i.e., the output of earlier algorithms can be deterministically generated from the output of our algorithm, but such a process would only add redundant information to the output.

We demonstrate the usefulness of mining periodic patterns on four diverse real-world datasets. Mirroring the increasing diversity of network analysis domains, we examine datasets of wild zebra association patterns, geographical movement patterns of university students, and the sightings of celebrities associated with the entertainment industry, among others. In

[1] A shorter version of this paper appeared as [25]. The major additions to this version are as follows: (a) we formally describe the framework of mining parsimonious periodic patterns, (b) we describe and prove the correctness of a more efficient version of the algorithm in [25], (c) we use smoothing instead of jitter to handle noise, since the former is better defined, and (d) we evaluate the performance of our algorithm compared to the SMCA algorithm [20].

addition to demonstrating the practical efficiency of our algorithm, we find that analyzing the collective periodicities of all mined patterns is indeed informative about the dynamics of the system being studied, yielding highly intuitive results about the specific systems we analyzed. We also found a number of interesting patterns which are intriguing because of a combination of their structure and periodicity. Some of these patterns occur relatively infrequently and might not have stood out had only their frequency of occurrence been considered, as is the case in frequent pattern mining.

This paper is organized as follows. In the next section, we present some preliminary definitions related to dynamic networks, as well as some graph theoretic properties that are key to the inherent complexity of the problem. In Sect. 3, we formally define the mining problem, which incorporates the concepts of closed subgraphs and parsimony. This is followed by a discussion of related literature in Sect. 4. In Sect. 5, we analyze the inherent complexity of the problem and derive an exact upper bound on the maximum number of possible periodic subgraphs in any dynamic network. We show that the mining problem is in the computational complexity class P (polynomial), in contrast to the closely related frequent pattern mining problem [4,37]. The complexity analysis of the problem is then used in Sect. 6 to build an efficient, online mining algorithm. The results of our experimental evaluation are presented in Sect. 7, followed by some concluding remarks and possible future research directions.

2 Preliminaries

Dynamic networks are a representation for a time series of interactions between a set of unique entities. Let $\mathcal{V} \in \mathbb{N}$ represents this set of entities. Interactions between entities can be either directed or undirected, and are assumed to have been recorded over a period of T discrete timesteps. The question of how much real time should constitute a timestep is beyond the scope of this paper; we use natural quantizations specific to each of our datasets, such as 1 day per timestep. The only requirement is that a timestep should correspond to a meaningful amount of real time, as the periodicities of mined subgraphs will be in multiples of the chosen timestep.

Definition 2.1 (*Dynamic network*) A *dynamic network* $\mathcal{G} = \langle G_1, \ldots, G_T \rangle$ is a time-series of graphs, where $G_t = (V_t, E_t)$ is a simple graph of interactions E_t observed at timestep t among the subset of entities $V_t \subseteq \mathcal{V}$ at timestep t.

Figure 1 is an example of a dynamic network with five timesteps. Definition 2.1 implies a convenient graph theoretic property that reduces the high computational complexity of many algorithmic tasks on graphs: since a vertex represents a unique entity, each vertex v in a particular timestep's graph G_t has a *unique vertex label*. This constitutes a class of graphs that can be represented as sets of integers, resulting in a reduction to quadratic computational complexity (in the number of vertices) for certain hard graph problems, such as maximal common subgraph and subgraph isomorphism [10,24,25].

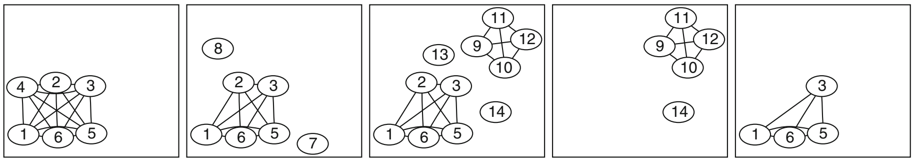


Fig. 1 An example of a dynamic network with five timesteps

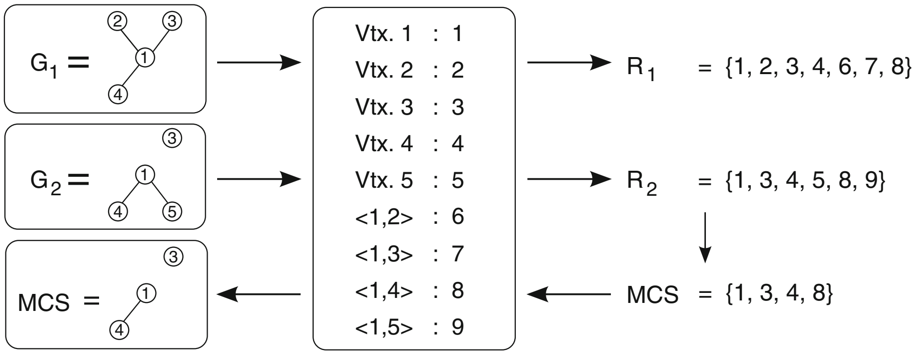


Fig. 2 The correspondence between graph and set representations for graphs with unique *vertex labels*. The example demonstrates the computation of the maximal common subgraph of two graphs using set representation

Property 2.1 (Set representation) *For a graph $G = (V, E)$ with unique vertex labels, the* set representation *R for G is formed by mapping each vertex and edge to a unique element in R, where $R \subset \mathbb{N}$.*

Since each vertex is uniquely identifiable by its label, it follows that each edge is also uniquely identifiable by its endpoints. This allows each vertex and edge to be coded as a unique integer, even across different graphs over the same vertex set. It can trivially be shown that two graphs (or timesteps) will result in the same set R if and only if they have identical vertex and edge sets. Although connectivity information is lost in the set representation, it is a useful transformation for the following algorithmic tasks, which are key to the development of our algorithm.

Property 2.2 (Subgraph testing) *For two graphs G_1 and G_2 with unique vertex labels, testing whether G_1 is a subgraph of G_2 or vice versa is equivalent to checking whether the corresponding set representations R_1 and R_2 are subsets of each other. For this reason, we use the subset operator $\subseteq$ to denote a subgraph relationship between G_1 and G_2.*

Property 2.3 (Maximal common subgraph) *For a set f graphs with vertex unique labels, finding the* maximal common subgraph *(MCS) is equivalent to the maximal intersection of their set representations. For a set of graphs $G_1, \ldots, G_T$, a vertex or an edge is part of the MCS if it is part of every G_t. As a result, the MCS always exists, is unique and well-defined, but could possibly be the empty graph with no vertices or edges. We use the intersection operator $\cap$ to denote the MCS of two or more graphs.*

Property 2.4 (Hashing) *A hashing function exists for graphs since the set representation R has a global ordering by virtue of $R \subset \mathbb{N}$.*

Figure 2 demonstrates the use of Property 2.1 to calculate the MCS of two graphs using set representation. A further implication of the set representation is that a dynamic network can be represented as a transaction database (also known as 'market-basket' data [1]) for certain data mining tasks like frequent subgraph mining[2] [21,23]. Although mining for periodic

[2] Since connectivity information is lost in the set representation, frequent *connected* subgraphs and subgraphs with other specific graph-theoretic properties cannot be extracted from the set representation.

patterns in time-ordered transaction databases has been studied in different contexts [18–20,29,38], one of the main advantages of our framework is the ability to handle structured data like dynamic networks (with connectivity information) while also being applicable to unstructured data like transaction databases.

We now introduce some terminology from the frequent pattern mining problem to be used in our problem definition and analysis.

Definition 2.2 (*Support*) Given a dynamic network $\mathcal{G}$ of T timesteps and an arbitrary graph $F = (V, E)$, the *support set* $S(F)$ of F in $\mathcal{G}$ is the set of all timesteps t in $\mathcal{G}$ where F is a subgraph of G_t, which we denote $F \subseteq G_t$. The *support* of F is the cardinality of its support set, $|S(F)|$:

$$S(F) = \{t_i, \ldots, t_j\} \quad \text{such that} \quad \forall t \quad (t \in S(F) \leftrightarrow F \subseteq G_t).$$

Definition 2.3 (*Frequent subgraph*) Given a dynamic network $\mathcal{G}$ of T timesteps, an arbitrary graph $F = (V, E)$ is **frequent** if its support exceeds a user-defined *minimum support threshold* $\sigma \leq T$.

Definition 2.3 is the basis of the well-known frequent pattern mining problem, which deals with the extraction of all subgraphs F where $|S(F)| \geq \sigma$. An implication of the naïve definition of a frequent subgraph is the *downward closure* property, which states that every subgraph of a frequent subgraph F is itself frequent. This serves as the underpinning of Agrawal and Srikant's classic Apriori algorithm, which searches for large frequent patterns by iteratively concatenating the smaller, frequent sub-patterns implied by the downward closure, relying on the sparsity of larger frequent patterns [1]. The downward closure is what makes a principled, incremental search through pattern space tractable, but is also a double-edged sword. Although many improvements have been made to the classic Apriori algorithm [8,18], any mining algorithm required to explicitly enumerate every frequent pattern in a dataset would, in doing so, have to enumerate the exponential number of subgraphs of every frequent subgraph which is a redundant and resource expensive process. The cornerstone of a solution to this problem is the use of closed subgraphs [6,18,30].

Definition 2.4 (*Closed subgraph*) Given a dynamic network $\mathcal{G}$ of T timesteps and an arbitrary graph $F = (V, E)$, F is *closed* if it is maximal for its support set: no vertex or edge can be added to F while maintaining its support.

Mining frequent closed subgraphs is an elegant solution to the redundancy of the general frequent pattern mining problem. It captures all the information of the more general formulation, but can result in output that is exponentially smaller in size without any loss of information. We therefore adopt it as an integral part of our problem definition, which is described in the next section.

3 Problem definition

We formally define the periodic subgraph mining for dynamic networks as a special case of frequent closed pattern mining with important additional computational properties. These properties allow the development of efficient mining algorithms and justify an independent treatment of the problem, rather than an approach that would, for example, push constraints into a conventional frequent pattern mining algorithm [17,31,32,41]. The relation to frequent pattern mining also highlights the fact that we are searching for locally periodic patterns, i.e., those that exhibit periodic behavior in a contiguous subsequence of the entire data stream. These are also known as *partially periodic* patterns [19,20,26]. We begin with a basic formulation of the problem and then develop it into a parsimonious formulation. We end this section by describing mechanisms to rank periodic patterns and handle imperfect periodicity in real-world datasets.

3.1 Basic formulation

Definition 3.1 (*Periodic support set*) Given a dynamic network $\mathcal{G}$ and an arbitrary subgraph $F = (V, E)$, a *periodic support set* of F in $\mathcal{G}$, denoted $S_P = (i, p, s)$, is a maximal, ordered set of s timesteps starting at t_i with every two consecutive timesteps being p steps apart.

$$S_P = (i, p, s) = \langle t_i, t_{i+p}, \ldots, t_{i+p(s-1)} \rangle$$

subject to the following constraints:

1. **Existence in $\mathcal{G}$:** F must exist at all timesteps in S_P, i.e., $\forall t\ (t \in S_P \rightarrow F \subseteq G_t)$. Note that the implication in the constraint is only in the forward direction, unlike Definition 2.3.
2. **Minimum size:** A periodic support set has to have at least two elements, i.e., $|S_P| = s \geq 2$.
3. **Temporal maximality:** The support set cannot be extended in time to contain F and still be periodic, i.e., $F \nsubseteq G_{t_{(i-p)}}$ and $F \nsubseteq G_{t_{(i+p\cdot s)}}$.

The *phase offset* of a periodic support set is defined as $m = (t_i - 1)$ **mod** p, since indices start from 1. Thus, $0 \leq m < p$.

A key difference in the definitions of a support set for frequent pattern mining and periodic pattern mining is that a single graph F can have multiple periodic support sets to allow for multiple, disjoint, or overlapping periodic behavior. Thus, we require the extraction of all periodic subgraph *embeddings*, rather than just the periodic subgraphs themselves. This is encompassed in the following definition.

Definition 3.2 (*Periodic subgraph embedding*) Given a dynamic network $\mathcal{G}$, a *periodic subgraph embedding (PSE)* is a pair $\langle F, S_P \rangle$, where F is an arbitrary graph that is closed over a periodic support set S_P with $|S_P| \geq \sigma$. The following list summarizes the properties of a PSE:

1. **Minimum support:** $|S_P| \geq \sigma \geq 2$, from Definition 3.1.
2. **Structural maximality:** F is maximal over S_P, i.e., F is the MCS of S_P, from Definition 2.4.
3. **Temporal maximality:** S_P is temporally maximal for F, from Definition 3.1.

Figure 3 shows an example of a dynamic network with two PSEs at $\sigma = 3$. The first is the subgraph $\{(1, 4), (1, 5)\}$ with a period of 2 and support set of $\langle 1, 3, 5 \rangle$, and the second is the singleton vertex $\{1\}$ with a period of 1 and a support set of $\langle 3, 4, 5 \rangle$. Note that the subgraph $\{(1, 2), (1, 3)\}$ is frequent but not periodic at $\sigma = 3$.

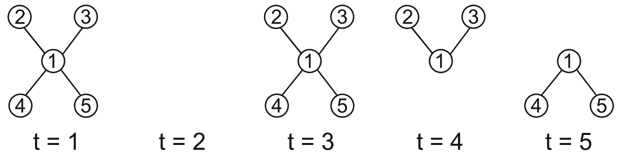


Fig. 3 An example of a dynamic network with two PSEs at $\sigma = 3$

3.2 Parsimonious formulation

We now address the issue of redundant information in the output. If we think of a PSE from Definition 3.2 as communicating a set of timesteps at which a particular subgraph exhibits its periodic behavior, a PSE which communicates information that is already contained in another PSE is redundant. For example, a subgraph F of period 2 with adequate support will also be output as a subgraph of period 4, and so on. This will continue for a fixed number of multiples of the base period, depending on the support of the pattern and the minimum support, in spite of the fact that the higher multiples communicate no new information about the subgraph in question. Furthermore, when analyzing periodic behavior in terms of the periodicities of mined patterns, there is no justifiable reason *prima facie* (or in keeping with Occam's Razor) to count multiples of a base pattern's period, unless those multiples extend beyond the support of the base pattern.

Although the use of closed subgraphs reduces much of the redundancy associated with the output of an Apriori style algorithm, the basic definition of a PSE still retains some of it. To eliminate all such redundancy, we pose our problem as that of mining a *minimal set of patterns* to cover all periodic occurrences of all periodic subgraphs. Keeping in line with the principle of parsimony, this eliminates patterns with periods that are multiples of a base period, unless they convey some new information about a periodic occurrence. In order to describe this concept formally, we first define the notion of *subsumption* of PSEs.

Definition 3.3 (*Subsumption*) For two periodic subgraphs F_1 and F_2 with respective periodic support sets $S_{P,1} = (i_1, p_1, s_1)$ and $S_{P,2} = (i_2, p_2, s_2)$, $\langle F_1, S_{P,1}\rangle$ completely contains or *subsumes* $\langle F_2, S_{P,2}\rangle$ if all of the following conditions hold:

1. $F_2 \subseteq F_1$
2. $t_{i_2} \geq t_{i_1}$
3. $t_{i_2+p_2\cdot(s_2-1)} \leq t_{i_1+p_1\cdot(s_1-1)}$
4. $p_2 = k \cdot p_1$ for some integer $k > 0$
5. $t_{i,2} = t_{i,1} + l \cdot p_1$ for some integer $l \geq 0$

We prove that all conditions listed above are necessary for subsumption. Condition 1 is trivially required to ensure that no information is lost. Let $f_1(l) = t_{i,1} + l \cdot p_1$ and $f_2(l) = t_{i,2} + l \cdot p_2$ be the lth occurrence of F_1 and F_2, respectively, for some integer l. For subsumption, we require that the support set $S_{P,2}$ is completely contained within the support set $S_{P,1}$. Conditions 2 and 3 require that the support set of F_2 is contained within the bounds of the support set of F_1, although they could be of different phase offsets and not overlapping at all, or partially overlapping but of different periods. Condition 4 requires that the period of F' is an integer multiple of F, and condition 5 requires that F_1 and F_2 have compatible phase offsets, which ensures that they overlap. This is handled by requiring that the first occurrence of F_2 overlap with any occurrence of F_1. Thus, $t_{i,2} = f_1(l)$, which yields the final condition $t_{i,2} = t_{i,1} + l \cdot p_1$.

Definition 3.4 (*Parsimonious PSE*) A PSE that is not subsumed by any another PSE is a *parsimonious periodic subgraph embedding* (PPSE).

As an example to motivate the mining of PPSEs, consider a system in which all the nodes only interact periodically with either period 2 or 4, starting at arbitrary times and continuing for an arbitrary number of repetitions. Suppose that we want to discover these unknown periodicities by observing the system for a period of time. With non-parsimonious PSEs, duplicates of each true periodic pattern would be reported for a fixed number of multiples of either 2 or 4, depending on the specific pattern. If we were to plot a histogram of the periodicities of all mined patterns, we would see various artifacts from the higher order periodicities, which could obscure the true periodicities. On the other hand, with parsimonious PSEs and enough data, the true periodicities of 2 and 4 would, with high probability, be the most prominent peaks.

Definition 3.5 (*Periodic subgraph mining problem*) Given a dynamic network $\mathcal{G}$ and a minimum support threshold $\sigma \geq 2$, the PERIODIC SUBGRAPH MINING problem is to list all parsimonious periodic subgraphs embeddings in $\mathcal{G}$ that satisfy the minimum support.

3.3 Practical considerations

3.3.1 Handling noise by smoothing

Since real-world networks are unlikely to always contain perfectly periodic patterns, we use smoothing as a mechanism for accommodating imperfect periodicity. Given a user-defined smoothing parameter $S \geq 1$, we transform the dynamic network by considering a sliding window over its timesteps. In other words, we transform the dynamic network $\mathcal{G}$ in the following manner,[3] where $G_i \in \mathcal{G}$:

$$\mathcal{G}' = \langle G_1 \cup \ldots \cup G_S, G_2 \cup \ldots \cup G_{S+1}, \ldots \rangle$$

In addition, the following two conditions handle the removal of artifacts introduced by the smoothing process.

1. The minimum period $P_{\min}$ is set to S.
2. PSEs of the same subgraph that share the same period and differ in their starting positions by at most $S - 1$ timesteps are merged. In other words, the PSE with the highest support is retained. This can be done as a post-processing step or incorporated into the mining algorithm itself.

By introducing this smoothing mechanism, we allow a window of timesteps within which the order of events does not matter. No smoothing is performed at $S = 1$.

3.3.2 Purity: a measure for ranking periodic subgraphs

A periodically recurring subgraph is not necessarily representative of an interaction pattern that occurs *only* periodically, as shown in Fig. 4. The *purity* measure expresses how likely it is that a PSE occurs only periodically over its support set.

Definition 3.6 (*Purity*) Given a PSE $\langle F, S_P \rangle$ with period p, starting at timestep t_i and with support $s = |\langle t_i, \ldots, t_j \rangle|$, the purity of F is the ratio of its periodic support to its total support in the timestep range $[t_i, t_j]$.

$$\text{Purity}(F) = \frac{s}{|\{t : F \subseteq G_t, t_i \leq t \leq t_j\}|}$$

It is sometimes advantageous to define the purity of a subgraph as the average purity of its edges. Doing so is more representative of the temporal characteristics of the entire subgraph. We use the term ‘purity’ to refer to average purity for the remainder of this paper. Figure 4 shows an example of the purity measure.

[3] Blank timesteps are appended to the beginning and end of the dynamic network as necessary to handle boundary conditions.

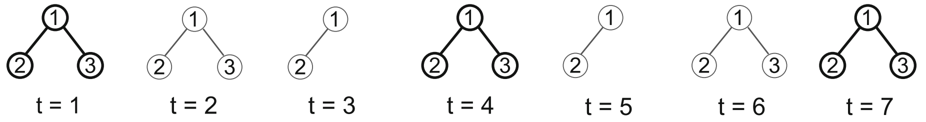


Fig. 4 A periodic subgraph embedding (*bold*) with non-periodic occurrences. The purity of this periodic subgraph is 3/5, whereas its average purity is $\frac{1}{2}(\frac{3}{5} + \frac{3}{7}) \sim 0.51$

Definition 3.7 (*Average purity*) The *average purity* of a subgraph $F = (V, E)$ is the average purity of all of its edges.

$$\text{AvgPurity}(F) = \frac{1}{|E|} \sum_{e \in E} \text{purity}(e)$$

4 Related work

Searching for periodicity and periodic patterns have appeared in different contexts in data mining. In this section, we review relevant literature concerning periodic pattern mining, as well as the closely related problem of frequent pattern mining. We omit certain earlier antecedents to this line of research, such as mining cyclic association rules [29] and frequent sequential patterns [2], as they are not directly relevant. Also, omitted for the same reason are periodic pattern mining approaches that require or assume that the entire input is at least approximately periodic, including techniques that use Fast Fourier Transforms [13,14].

Most algorithms for mining periodic patterns deal with unstructured data such as a sequence or multiple, aligned sequences. In the most general formulation of the problem, the input consists of a sequence of symbols sets $S = \langle a_1, \ldots, a_T \rangle$, where each symbol set a_i is drawn from a finite universal set $\mathcal{L}$. A pattern is a sequence $P = \langle b_1, \ldots, b_p \rangle$ of length p, where p is the period of the pattern and each $b_i \subseteq \mathcal{L} \cup \{*\}$. The '*' character is a wild card that matches any symbol. Less general versions consider only a single sequence as the input, so each $a_i \in \mathcal{L}$ and $b_i \in \mathcal{L} \cup \{*\}$. The pattern mining problem is to extract all such patterns from the input sequence, subject to constraints such as a minimum support. Algorithms for this task are generally variants of the classic Apriori algorithm of Agrawal and Srikant [1], in which larger patterns are iteratively built from smaller ones. Note that the definition of a periodic pattern in this line of research is essentially a sequence with wildcards, whereas our definition is closer to concepts from frequent pattern mining.

Han et al. introduced one of the first algorithms to mine partial periodic patterns in multidimensional sequences [19]. They adopt an Apriori-inspired search through pattern space using a novel prefix-based data structure called a *max-subpattern tree*. Ma and Hellerstein [26] propose a similar, Apriori-inspired approach consisting of two level-wise algorithms for mining periodic patterns in the presence of both partial periodicity as well as imperfect periodicity. They also propose an interesting statistical (as opposed to combinatorial) foundation for defining periodicity.

Yang et al. [38,39] proposed another level-wise mining algorithm for detecting 'surprising' periodic patterns, i.e., those judged to be interesting based on deviation from their expected frequency. This is intended to overcome limitations of using the support of a pattern as the sole measure of its worth. They devise two variants of information gain as measures of interest: *bounded information gain* [38] and *generalized information gain* [39], the second of which obeys the triangle inequality. However, a number of independence assumptions are made, such as the probability of occurrence of an event being the same at any point in time, and these might not hold in dynamic networks.

Yang et al. [40] propose a level-wise mining algorithm that allows imperfect (or 'asynchronous') periodic patterns to be discovered. They do this by introducing two user-defined parameters into the mining process to specify the minimum number of repetitions of a pattern and the maximum amount of disruption allowed. Huang and Chang [20] build on this in their description of SMCA, a suite of four algorithms for mining periodic patterns [20]. The fundamental idea is still to conduct a level-wise search through pattern space, but augmented with more efficient data structures and algorithms than earlier approaches. Each algorithm enumerates more complex patterns from the output of an earlier stage.

Finally, our work is inspired by frequent pattern mining, which is concerned with the discovery of patterns that occur more frequently than a user-defined threshold. A relatively young offshoot of this line of research is frequent subgraph mining [21,23], which was originally devised to search for common structures in databases of chemical compounds represented as graphs. A detailed overview of this field is beyond the scope of this paper, but may be found in [18] and [8]. There are, however, a number of recent complexity results for frequent pattern mining that are relevant. Specifically, given a set of maximal frequent itemsets, Boros et al. [4] show that it is NP-complete to decide if there is a further maximal frequent itemset. Yang [37] shows that different variants of maximal frequent pattern mining, including itemsets and subgraphs with unique vertex labels, are either #P-hard or #P-complete in terms of counting the number of satisfying solutions. Thus, many variants of frequent pattern mining are computationally intractable in the worst case.

5 Complexity analysis of the mining problem

We now analyze the computational complexity of the periodic subgraph mining problem as defined in Sect. 3. In order to do this, we derive an exact upper bound on the number of PSEs that can exist in any dynamic network of T timesteps. We prove that this upper bound is a polynomial function of the number of timesteps and the minimum support value. We show that the upper bound is sharp by constructing a 'worst-case' dynamic network.[4] The proof leads to the conclusion that mining all closed PSEs can be done in polynomial time in the size of the input, proving that the mining (enumeration) problem is in the complexity class P, when the graphs have unique vertex labels. This is in contrast to the more general frequent subgraph mining problem, which is NP-hard for enumeration and #P-complete for counting, even with unique vertex labels [4,37]. We take advantage of the intrinsic polynomial complexity of the problem to design an efficient single-pass mining algorithm in Sect. 6. We do not include smoothing in the following analysis, and purely algebraic manipulations are omitted for brevity.

Theorem 5.1 PERIODIC SUBGRAPH MINING *in dynamic networks is in P.*

To prove Theorem 5.1, we first construct a class of worst-case dynamic networks and show that any member of this class has the maximum possible number of PSEs. We utilize the concept of a *projection* of a discrete time sequence to count the maximum number of PSEs in this class of dynamic networks [13].[5]

[4] An alternate version of this proof in terms of maximal subgraphs, but with the same outcome, can be found in [25].

[5] In principle, any combinatorial technique can be used to count the number of PSEs. Projections are convenient for Definition 3.2 and some extensions to it.

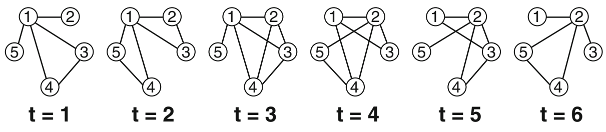


Fig. 5 An example of a worst case dynamic network for mining PSEs at $\sigma = 3$

Definition 5.1 Given a dynamic network $\mathcal{G}$, a **projection** $\pi_{m,p}$ of $\mathcal{G}$ is a subsequence of graphs

$$\pi_{m,p} = \langle G_{1+m}, G_{1+m+p}, G_{1+m+2p}, \ldots\rangle,$$

where p is the *period* of the projection and $0 \leq m < p$ is the phase offset.

It should be clear from the definitions of periodicity and projection that any periodic support set at minimum support σ is embedded in at least σ consecutive positions of some projection $\pi_{m,p}$.

Proposition 5.1 *Let F be the MCS of any $s \geq \sigma$ consecutive positions of any projection $\pi_{m,p}$. If F is not empty, then it is a periodic subgraph and the s consecutive timesteps from $\pi_{m,p}$ are part of a PSE for F.*

Proof A non-empty MCS F of any $s \geq \sigma$ consecutive positions implies that F is maximal over a support set of at least σ periodic timesteps, which in turn might or might not be temporally maximal for F. However, in either case, the s timesteps are part of some valid periodic support set of size at least σ. This is a sufficient condition to satisfy Definition 3.2, and thus F is a periodic subgraph. □

Corollary 5.1 *In the worst computational complexity case for mining PSEs in a dynamic network, the MCS of every $s \geq \sigma$ consecutive positions of every projection is not empty and contains a unique PSE.*

Proof Clearly, if every periodic subset of $s \geq \sigma$ timesteps of the dynamic network contains a unique MCS, then they all need to be enumerated by any mining algorithm and it is indeed the worst case input for a periodic subgraph mining problem. We now show that it is attainable using an explicit construction. We place a different edge in each $s \geq \sigma$ consecutive positions of every projection to ensure that each edge is part of a unique PSE. Let edge e be created in this way with support set S_P in some $\pi_{m,p}$. Considering only S_P, we know that it is temporally maximal for the edge e because e does not exist in any other timesteps. Furthermore, the MCS of S_P is non-empty because it contains at least the edge e. Thus, each edge is part of a unique PSE whose support set is S_P. Since a different edge was placed in every $s \geq \sigma$ consecutive positions of every projection, the number of PSEs is equal to the number of edges created. No additional PSEs can be created since every permissible support set, i.e., with support greater than σ, is already part of a unique PSE. Therefore, the described structure is a worst case instance for its size. □

Figure 5 shows an example construction of such a worst-case dynamic network with 12 PSEs at $\sigma = 3$. The next step is to explicitly calculate the upper bound on the number of PSEs in the worst-case network instances. Following from Corollary 5.1, we only need to count the number of $s \geq \sigma$ consecutive positions of every projection to derive this bound. In order to do this, we first state the bounds on several other parameters.

Proposition 5.2 *In a dynamic network with T timesteps, the* maximum period *of any periodic subgraph with support at least σ is $P = \lfloor (T-1)/(\sigma-1) \rfloor$.*

Proposition 5.3 *In a dynamic network with T timesteps, the* length of any projection *is $|\pi_{m,p}| = \lceil (T-m)/p \rceil$.*

The proofs of the Propositions 5.2 and 5.3 are straightforward and similar to those in [13]. Given the above expressions, we now derive an exact bound by construction.

Theorem 5.2 *In a dynamic network with T timesteps, there are at most $O(T^2 \ln \frac{T}{\sigma})$ closed PSEs at minimum support σ.*

Proof From Corollary 5.1, the maximum number of PSEs possible in a dynamic network at minimum support σ is equal to the number of $s \geq \sigma$ length windows over all possible projections of the network. For a given projection $\pi_{m,p}$ and value of s, it is clear that the number of length-s windows over the projection is $|\pi_{m,p}| - s + 1$, where $|\pi_{m,p}|$ is the length of the projection defined in Proposition 5.3. Thus, for a given value of s, the number of length-s windows over all projections can be obtained by substituting the expressions from Propositions 5.2 and 5.3:

$$\sum_{p=1}^{\left\lfloor \frac{T-1}{s-1} \right\rfloor} \sum_{m=0}^{p-1} \left(\left\lceil \frac{T-m}{p} \right\rceil - s + 1 \right)$$

We have replaced σ with s in the expression for the maximum period of a pattern from Proposition 5.2, since we only want projections which contain at least one length-s window for any s. This constitutes the outer summation; the inner summation is over all possible phase offset values m for a given period p. Finally, the term inside the summation is the number of length-s windows in any projection, where $|\pi_{m,p}|$ has been substituted from Proposition 5.3. We now sum this expression over all possible values of s, which run from σ to T, and relax the floor and ceiling expressions for an asymptotic closed form approximation.

$$\sum_{s=\sigma}^{T} \sum_{p=1}^{\left\lfloor \frac{T-1}{s-1} \right\rfloor} \sum_{m=0}^{p-1} \left(\left\lceil \frac{T-m}{p} \right\rceil - s + 1 \right) \tag{1}$$

$$\sim \sum_{s=\sigma}^{T} \sum_{p=1}^{\frac{T-1}{s-1}} \sum_{m=0}^{p-1} \left(\frac{T-m+p}{p} - s + 1 \right) \tag{2}$$

Expression 2 algebraically simplifies to an expression that is $O(T^2 \cdot H(\frac{T-1}{\sigma-1}))$, where $H(n) = \sum_{k=1}^{n} \frac{1}{k}$ is the nth harmonic number, asymptotically approximated by $\ln n$. Thus, the number of PSEs at minimum support σ is bounded asymptotically by $O(T^2 \ln \frac{T}{\sigma})$ (and exactly by Eq. 1). □

Proof of Theorem 1 To finally prove Theorem 5.1, consider an algorithm that outputs the MCS of every σ length window of every projection. Since the MCS of a set of graphs with unique vertex labels can be found in time $O(V+E)$ [10], in the worst case, this results in $O(T^2 \ln \frac{T}{\sigma})$ periodic 'fragments' computed in $\Theta((V+E)T^2 \ln \frac{T}{\sigma})$ time. Every pair of periodic fragments is then compared and merged if they represent overlapping embeddings of the same periodic subgraph, in time $O((V+E)(T^2 \ln \frac{T}{\sigma})^2)$, resulting in all PSEs. Another run over pairs of PSEs can eliminate all non-parsimonious PSEs, resulting in an overall time complexity of $O((V+E)T^4(\ln \frac{T}{\sigma})^2)$. Thus, the mining problem is in P, and the exact bound on the number of closed PSEs is given in summation form in Theorem 5.2. □

6 The algorithm

We now present PSEMINER,[6] our algorithm for mining all PPSEs in a dynamic network. We start by describing the most basic form of the algorithm, which mines closed (not just parsimonious) PSEs, and proving its correctness and complexity. We then describe some simple optimizations to the basic algorithm that allow it to output only PPSEs and also improve its efficiency in practice.

PSEMINER is based on the following idea: as each timestep of the dynamic network is read, we maintain a list of all PSEs seen up to timestep t. This list is maintained in a simple data structure called a *pattern tree*, which also tracks subgraphs that might become periodic at some point in the future. Once PSEs cease to be periodic, they are flushed from the tree and written to the output stream if they satisfy certain conditions like the minimum support. As each timestep G_t is read from the data stream, the pattern tree is updated with the new information, which could involve modifying, adding and deleting tree nodes. The complexity analysis in Sect. 5 allows us to prove worst-case computational time and space bounds that are polynomial in the size of the input. We describe the algorithm, its parameters, data structures and a proof of correctness in the following five sections. In Sect. 6.6, we describe optimizations that complete the description of the algorithm.

6.1 Parameters

Our algorithm is a single-pass, polynomial time and space algorithm for mining all closed PSEs in a dynamic network. It does not require any parameters, but *optionally* accepts the following:

1. Minimum support threshold $\sigma \geq 2$ (default: 2).
2. Minimum period $P_{\min}$ (default: 1).
3. Maximum period $P_{\max}$ (default: unrestricted).
4. Smoothing timesteps $S \geq 1$ (default: 1).

When the $P_{\max}$ parameter is restricted, our algorithm functions as an online algorithm, retaining only the parts of the dataset in memory that it requires to calculate periodicities. There is a natural bound on the maximum period of mined patterns if the number of timesteps T is finite and known (see Proposition 5.2). However, in many situations this information is not available or relevant, such as in streaming sensor data. In such cases, an unrestricted maximum period value places a large computational burden on the algorithm, and requires that the entire dataset be retained in memory. This is because at any timestep t, any previously observed timestep $t' < t$ could contain the initial occurrence of a periodic subgraph whose second occurrence is at timestep t. Testing for this situation requires all previously seen timesteps to be retained in memory, either explicitly or in some compressed form. The optional $P_{\max}$ parameter limits the maximum period of mined patterns, and thus eliminates the need to retain previously seen timesteps beyond a certain history.

The default parameters mine a complete set of periodic subgraphs without any smoothing, although in practice, only σ values of 3 or more are meaningful. The output of the algorithm

[6] Periodic Subgraph Embedding Miner.

is a set of closed parsimonious periodic subgraphs embeddings that satisfy the minimum support. Each embedding is written to the output stream as soon as the last possible occurrence of the subgraph has been encountered, or when the input stream has been exhausted.

6.2 Data structures

As the algorithm scans the input stream, it maintains three primary data structures to track PSEs: a *pattern tree*, a *subgraph hash map*, and an optional *timeline list* to increase efficiency. An auxiliary data structure, called a *descriptor*, is used as a compact representation of a periodic support set. We refer to nodes in the pattern tree as *treenodes* to distinguish them from nodes (vertices) in the dynamic network or in a periodic subgraph. Each treenode N is associated with a single periodic subgraph F and a set of descriptors that represent PSEs of F. We use the notation **'treenode** N/F**'** to refer to a treenode N that represents subgraph F.

6.2.1 Pattern tree, subgraph hash map and timeline list

The tree structure represents a subgraph relationship between periodic subgraphs. The structure of the pattern tree is subject to a single constraint: with the exception of the special root node, all descendants of a treenode N/F are associated with proper subgraphs of F, but not all subgraphs of F are necessarily its descendants in the tree. This property allows efficient traversal of the tree by the mining algorithm, and also allows the tree to be built and manipulated quickly and represented using very little space.[7] It also allows efficient traversal by virtue of the fact that if F is not observed at a given timestep for treenode N/F, then neither are the subgraphs represented by N's descendants (except for the root node). Direct access to treenodes is also required, which is achieved using a hash map to associate periodic subgraphs with their corresponding treenode. This can be done efficiently, as described in Property 2.4 of the set representation of dynamic networks. The timeline list is an optional component that links treenodes to the future timesteps at which they are expected to appear. Its use is discussed in Sect. 6.6.

6.2.2 Treenodes

Each treenode N/F contains a list of descriptors $\{D_1, \ldots, D_n\}$, one for each observed PSE of F. In addition, each treenode maintains a list of periods and phases of all live descriptors (see below), which is used by the tree update algorithm. Querying, adding to, and removing descriptors from this list are the primary operations on a treenode.

6.2.3 Descriptors

A descriptor D is the abbreviated representation of a periodic support set. It is associated with a treenode N/F and defines a unique PSE for F. It is formally described as a triple, since it represents a periodic support set $S_P = (i, p, s)$. The last element in the support set is defined as $t_j = t_i + p \cdot (s - 1)$ and the next expected timestep as $t_n = t_j + p$. Since descriptors are created, updated, and deleted as the input stream is read, the following definition describes the different states in which a descriptor could be at any given time.

[7] An alternative to the tree representation would be to construct a full subgraph lattice [6], with a corresponding increase in time and space complexity. Whether lattices are more efficient given the typical sparsity of dynamic networks is a question for future research.

Definition 6.1 (*Descriptor states*) At timestep t, a descriptor D for a subgraph F is *live* if $t_n > t$ or if $t_n = t$ and F is present at G_t. A descriptor that is not live is not currently exhibiting periodic behavior; it cannot change state again once it is not live. A descriptor where $t_i = t_j$ is a special case called an *anchor descriptor*, as it does not represent a periodic support set but could potentially become one if the associated subgraph F is observed at a future timestep. An anchor descriptor is defined to have a period of 0. An anchor descriptor is always live, unless $P_{\max}$ is defined and $t - t_i > P_{\max}$, in which case the anchor can never lead to a valid PSE with period at most $P_{\max}$, and is no longer needed.

6.3 Tree update algorithm

We now describe the update algorithm for the pattern tree, which is the core of the mining process. It is called once for each timestep that is read from the input. Starting with an initial pattern tree with an empty root treenode, at timestep t the algorithm traverses the pattern tree in a breadth-first search (BFS) to update treenodes with the new information contained in G_t. For each G_t, we are only interested in treenodes which might be affected by the new information. This excludes any subgraph F which has an empty MCS with G_t. In most cases, this process eliminates some branches of the pattern tree from the BFS traversal. At each treenode N/F where F has some part in common with G_t, we update descriptors at N in a manner described below. We end each tree update by ensuring that a treenode for G_t in its entirety exists in the tree with an anchor descriptor for timestep t. This accounts for the possibility that G_t in its entirety is the first occurrence of a (future) periodic subgraph. If such a treenode does not exist, it is created at a location which does not violate the subgraph property of the tree, such as the root.

During the breadth-first traversal of the tree, one of the following three conditions holds at each treenode N/F. Let $C = F \cap G_t$ be the MCS of G_t and F.

1. **Update descriptors:** If $F \subseteq G_t$, i.e., if $F = C$, then F has appeared in its entirety at timestep t. Let D be any descriptor in N and $t_n = t_j + p$ be the next expected timestep for D.
   - (a) If $t_n = t$, then D has appeared where it was expected. Timestep t is added to D's support to ensure temporal maximality.
   - (b) If $t_n < t$, then D has not appeared when expected and is thus no longer live. It is written to the output stream if its support is greater than or equal to σ, and removed from the tree.
   - (c) If $t_n > t$, then nothing is done.
   - (d) If $p = 0$, then D is an anchor descriptor. Given that timestep t is the second occurrence of F, a new descriptor D' is spawned with period $p' = t - t_i$ and phase offset $m' = (t_i - 1)$ **mod** p'. If N does not contain a live descriptor with the same period and phase offset, D' is added to the list of descriptors at N.
2. **Propagate descriptors:** If $C \neq \emptyset$ and the condition above does not hold, then a subgraph C of F is present at timestep t, instead of F in its entirety. This happens, for example, when a formerly periodic subgraph F fractures into a smaller subgraph C that continues F's periodic behavior. If a treenode for C does not already exist in the tree, determined using the subgraph hash map, it is created as a child of N (to satisfy the subgraph relationship). Let D be any descriptor at N. If $t_n = t$, then D represents a PSE which subgraph C must inherit and continue. The treenode for C receives a copy of D, if a live descriptor of the same period and phase offset does not already exist. The pattern

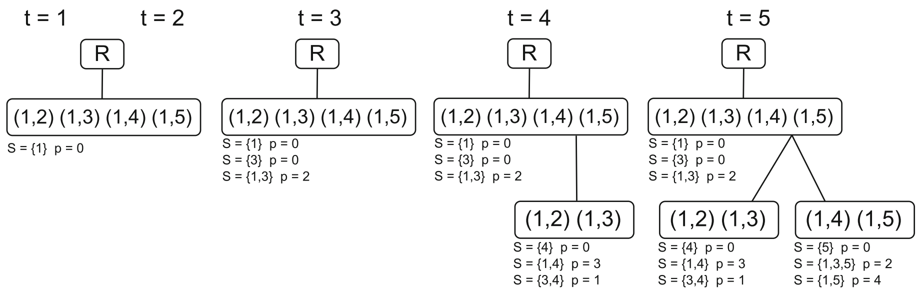


Fig. 6 The pattern tree at each timestep for the dynamic network shown in Fig. 3, considering only edges for brevity

$<F, D>$ is written to the output stream if the support of D is greater than or equal to σ, and then D is removed from treenode N.

3. **Dead subtree:** If $C = \emptyset$, then G_t and F have no common subgraph, and no descriptors at N are directly affected by the observation of G_t. Furthermore, no treenode that is a descendant of N will have any common subgraph with G_t either, since they are all subgraphs of F. The subtree rooted at N is therefore eliminated from the rest of the tree traversal.

Algorithm 1 UPDATETREE(G_t)

```
Require: $G_t$ is the graph of timestep $t$
1: $Q \leftarrow$ new queue
2: push($Q$, root.children)
3: while $N \leftarrow$ pop_front($Q$) do
4:    $C \leftarrow G_t \cap N$
5:    if $C$ is not empty then
6:       if $N \subseteq G_t$ then
7:          UPDATEDESCRIPTORS($N$)
8:       else
9:          $W \leftarrow$ FINDNODE($N$) or NEWNODE($N$, $C$)
10:         PROPAGATEDESCRIPTORS($N$, $W$)
11:      end if
12:      push($Q$, children($N$))
13:   end if
14: end while
15: $W \leftarrow$ FINDNODE($G_t$) or NEWNODE(root, $G_t$)
16: Add anchor descriptor for $G_t$ to $W$.
```

Figure 6 shows the pattern tree at each timestep during the execution of the algorithm on the network from Fig. 3. For clarity, we have described a very basic version of the algorithm. Two notable aspects of this algorithm are (1) that it outputs all PSEs, which are a superset of all PPSEs, and (2) it can dynamically calculate the purity measure. Non-parsimonious PSEs can be post-processed out of the output, but in Sect. 6.6, we show how this can be accomplished dynamically.

6.4 Correctness

The pattern tree is intended to hold all PSEs seen up to timestep t. We prove by induction that this consistent state holds at any point during the execution of the algorithm. We define a consistent state for the pattern tree as the following four conditions.

Definition 6.2 (*Pattern tree consistency conditions*) The pattern tree is in a *consistent state* if the following four conditions are met:

1. The subgraph property of the pattern tree holds, i.e., all descendants of a treenode N/subgraph F contain subgraphs that are proper subgraphs of F.
2. All descriptors in the pattern tree are unique, i.e., no two descriptors D_1 and D_2 anywhere in the tree share the same subgraph and the same support set.
3. All PSEs with support $S_P \geq 2$ encountered in the data stream so far have a descriptor (and thus a treenode) in the tree.
4. All non-anchor descriptors represent PSEs that are closed up to timestep t, i.e., for a descriptor D in a treenode N/subgraph F, F is the MCS of the support set described by D, and the support set is temporally maximal at timestep t as per Definition 3.1.

If the tree is in a consistent state at timestep t, then the remaining output up to timestep t can be obtained by traversing the tree once and writing every subgraph/descriptor pair where the support of the PSE is $|S_P| \geq \sigma$. The tree is initially empty except for a dummy root node. It is therefore consistent because the four consistency conditions are vacuously true. For the inductive hypothesis, assume that the pattern tree is consistent after processing timestep G_{t-1}. Then after processing G_t, we show below that the tree is still in a consistent state, thus proving that the tree is in a consistent state during and at the end of the execution of the mining algorithm. The following is the statement and proof of the inductive step.

Theorem 6.1 *If the pattern tree is in a consistent state after processing G_{t-1}, then the pattern tree is also in a consistent state after using Algorithm 1 to process G_t.*

Proof On reading G_t from the input stream, the first two consistency conditions are not violated because no new subgraphs or descriptors have been added to the tree. Conditions 3 and 4, on the other hand, might be violated because G_t could potentially contain a previously unseen PSE, violating condition 3, or require that an existing one have its support set extended to include t, violating condition 4. Therefore, we start by focusing on events that would violate the latter two consistency conditions, while showing that the first two remain satisfied during processing. We describe each event in turn and how the consistency of the tree is violated, as well as the correctness of the actions taken to restore consistency. The following is an exhaustive list of such events, along with the action that the algorithm takes:

Case 1: G_t contains the first occurrence of a new PSE, violating condition 3; an anchor descriptor starting at timestep t is added to a treenode for G_t in its entirety.

Case 2: G_t contains the nth occurrence of a new PSE, where $n > 1$ and prior occurrences were contained within some other PSE, violating condition 3; the PROPAGATEDESCRIPTORS function is called. When $n = 1$, we have case 1 above.

Case 3: G_t contains the nth occurrence of an already existing PSE, where $n > 1$, violating condition 4; the UPDATEDESCRIPTORS function is called. Timestep t cannot be the first occurrence for an *existing* PSE, by definition.

Case 1 The first possibility is that G_t could contain the *first occurrence of a new PSE*. Since we have no way of knowing the future, we always assume that the entire graph G_t is going to become a periodic subgraph in the future with timestep t as its first timestep.[8] In Algorithm 1, a treenode W is added for G_t at the root if one does not already exist in the tree, and an anchor

[8] Incidentally, there is at least one dynamic network where each timestep contains the first occurrence of a new PSE—the worst-case construction from Sect. 5.

descriptor starting at t is added to W. Adding W at the root is a simple way to ensure that condition 1 is never violated. The descriptor is guaranteed to be unique, because no other PSE of G_t will have started at timestep t prior to G_t having been observed, and therefore condition 2 is not violated. If we are correct about the assumption that timestep t is the first occurrence of a new PSE for G_t, then we have 'presciently' added a descriptor and treenode for it at the correct time, and ensured that condition 3 is not violated. On the other hand, if G_t never occurs again, then its treenode will only contain an anchor descriptor, which is exempt from condition 4. Therefore, case 1 no longer causes the tree to be inconsistent.

Case 2 Suppose that G_t is the *nth occurrence of a new PSE, for $n > 1$*. This happens when a subgraph stops exhibiting periodic behavior, but a smaller portion of it continues to do so. The treenode for the smaller subgraph might therefore need to 'inherit' some descriptors from the treenode of the larger subgraph. For each treenode N/F, case 2 arises when $F \cap G_t \neq \emptyset$ except when $F \subseteq G_t$ (this exception is handled in the next case). Let $C = F \cap G_t$, the MCS of F and G_t. Let W be the treenode for C, which is created in the tree (at a position that does not violate condition 1) if it does not already exist.

We now need to copy descriptors where $t_j + p = t$ from N to W, since these descriptors would have been updated if F had been observed in its entirety. Let D be one such descriptor. D is now no longer live for N/F because it has failed to appear in its entirety at timestep t. The propagation process transfers D to W if W does not already have a live descriptor of the same period and phase offset D and an earlier starting position. Since treenodes N and W represent different subgraphs, copying D from N to W does not violate condition 2. Furthermore, since D was temporally maximal before, it is again temporally maximal with the addition of t to its support set. This handles conditions 4 and 3, and case 2 no longer causes the tree to be inconsistent.

Case 3 Finally, we handle the case that G_t is the *nth occurrence of an existing PSE, for $n > 1$*. This happens when a treenode N/F has $F \subseteq G_t$, which means that F has appeared in its entirety and its descriptors need to be updated. The update process scans each descriptor D in treenode N. If D is next expected at timestep t, then t is added to its support set by setting $t_j = t$. This satisfied consistency condition 4. If D is no longer exhibiting periodic behavior, i.e., if $t_j + p < t$, then D is flushed to the output stream if appropriate and then deleted. The other conditions are not violated. The final case is therefore handled correctly, and the pattern tree is again in a consistent state. □

We have inductively shown that Algorithm 1 results in a consistent tree after processing each timestep G_t in increasing order of t. This proves the correctness of the algorithm.

6.5 Time and space complexity

Given that the tree consistency conditions hold, the number of descriptors (and therefore nodes) in the tree at timestep T is bounded by Theorem 5.2 at $\sigma = 2$. As each timestep is read, the tree is traversed once. When descriptors are created or propagated, we ensure that at most one live descriptor exists at each treenode for a given period and phase offset. If the list of periods and phase offsets of live descriptors in the treenode are represented as sparse two-dimensional arrays, then lookup can be performed efficiently in amortized constant time with $O(P_{\max}^2)$ or $O(T^2)$ space complexity to hold the arrays. Thus, the worst-case time complexity of the algorithm involves traversing each descriptor in the tree once for each timestep and calculating the MCS at each treenode. From Property 2.1, the MCS of two graphs can be calculated in time $O(V+E)$. This yields a total time complexity of $O((V+E)T^3 \ln T)$ when $P_{\max}$ is not specified. When $P_{\max}$ is specified, the range of allowable periods is bounded in Theorem 5.2 and the maximum number of patterns can drop very significantly. The worst-case space complexity of our algorithm is $O((V+E+P_{\max}^2)T^2 \ln T)$ when $P_{\max}$ is specified. In practice, however, the tree size is usually several orders of magnitude smaller than the worst-case bound, as we will demonstrate.

6.6 Extensions to the basic algorithm

We have described a basic version of the mining algorithm in Sect. 6.3. A number of algorithmic refinements are possible to increase efficiency, but at the cost of conceptual simplicity. We briefly describe some of these refinements below.

6.6.1 Mining parsimonious PSEs

The most important enhancement is to make the algorithm dynamically output only parsimonious PSEs. Recall the subsumption conditions from Definition 3.3. A simple way to modify Algorithm 1 to only output parsimonious PSEs is by adding an indicator bit to each descriptor to indicate subsumption. This bit is initially cleared when the descriptor is created. When any descriptor D from treenode N/F is flushed, its subsumed bit is first checked. If it is cleared, then D is compared to all other live descriptors at N. If D is subsumed by another descriptor, it is not written to the output. On the other hand, if D subsumes (as of timestep t) some other descriptor D', the subsumed bit for D' is set. If the support of D' increases in the future, its subsumed bit is cleared since Condition 3 from Definition 3.3 is no longer true. However, if its support does not increase, then all the conditions from Definition 3.3 hold and D' is not parsimonious. It will not be flushed when the cessation of its periodic behavior is finally confirmed.

6.6.2 Sorted descriptor list

The list of descriptors at each node can be stored sorted by the next expected timestep of each descriptor. At timestep t, only descriptors which are expected at or before t will be examined, in addition to at most one descriptor that is expected after timestep t. This cuts down on the number of descriptors that need to be examined during each tree update, at the computational cost of having to sort the list of descriptors after each update. Since the number of descriptors per treenode is generally not very large, the computational overhead is minimal in practice.

6.6.3 Lazy tree updates

In practice, the algorithm spends most of its running time calculating intersections of integer sets (line 7 in Algorithm 1). Although the maximum common subgraph of two graphs is calculated in time linear in the number of vertices and edges, the size of the graphs results in a relatively expensive intersection computation. The sparsity of the network generally results in a relatively small number of treenodes, which means that many such intersections between large sets must be performed. Thus, to improve the practical efficiency of the algorithm, we can delay calculating intersections until it is absolutely necessary. This results in the lazy-intersection tree update algorithm shown in Algorithm 2. The tradeoff is that the total support of patterns, and therefore the purity measure, cannot be dynamically calculated.

Algorithm 2 LAZYUPDATETREE(G_t)

```
Require: $G_t$ is the graph of timestep $t$
1: $Q \leftarrow$ new queue
2: push($Q$, root.children)
3: while $N \leftarrow$ pop_front($Q$) do
4:   lazy $\leftarrow$ true
5:   while lazy $=$ true do
6:     $D \leftarrow$ next descriptor at $N$
7:     next $\leftarrow \texttt{last}(D) + \texttt{period}(D)$
8:     if D is an anchor or next = $T$ then
9:       lazy $\leftarrow$ false
10:    else
11:      if next $< T$ then
12:        flush $D$ to output and delete
13:      else
14:        break
15:      end if
16:    end if
17:  end while
18:  if lazy = false then
19:    $C \leftarrow G_t \cap N$
20:    if $C$ is not empty then
21:      if $N \subseteq G_t$ then
22:        UPDATEDESCRIPTORS($N$)
23:      else
24:        $W \leftarrow$ FINDNODE($N$) or NEWNODE($N$, $C$)
25:        PROPAGATEDESCRIPTORS($N$, $W$)
26:      end if
27:      push($Q$, children($N$))
28:    end if
29:  else
30:    push($Q$, children($N$))
31:  end if
32: end while
33: $W \leftarrow$ FINDNODE($G_t$) or NEWNODE(root, $G_t$)
34: Add anchor descriptor for $G_t$ to $W$.
```

6.6.4 Using a timeline to trim the tree

The timeline associates each future timestep with a list of treenodes that have at least one descriptor expected at that timestep. It can be dynamically updated at an insignificant cost (constant or logarithmic) once per treenode update, and stored in space linear in the number of treenodes. After the tree update for timestep t, all treenodes that are still associated with timestep t are guaranteed not to have been visited during the tree update, and have at least one descriptor which is no longer periodic. These treenodes can then be visited and the invalid descriptors removed, in time proportional to the number of descriptors to be removed. Thus, at the end of each tree update operation, the treenode only contains descriptors that are live at the next timestep. This ensures that the pattern tree contains a minimal number of descriptors and treenodes at any given timestep.

7 Experimental evaluation

We use four real-world dynamic social networks to evaluate our algorithm as well as some characteristics and applications of periodic subgraph mining. We also use artificial data to compare the performance of our algorithm with that of SMCA [20], a periodic pattern mining algorithm that generates periodic patterns in a level-wise search similar to Apriori and without closed or parsimonious considerations. SMCA is a four-phase algorithm and we only use the first two phases (SPMiner and MPMiner), since their combined functionality is comparable to our algorithm.[9] We first report results on the comparison with SMCA on synthetic data, before moving on to evaluating our algorithm on real dynamic networks.

We implemented our algorithm in C++, incorporating all the optimizations described in Sect. 6.6. The subgraph hash map was implemented using the Google `dense_hash_map` library,[10] optimized for speed over memory usage. The experiments with synthetic data were run on a dual-core Intel Pentium D system running at 3.2 GHz with 3 GB of RAM and Linux kernel 2.6.28. The experiments with real data were run on a quad-core Intel Xeon server running at 2.6 GHz with 24 GB of RAM and Linux kernel 2.6.22. In all cases, computation time is reported as the sum of the user (computation) and kernel (I/O, etc.) CPU time reported by the Linux `getrusage()` system call. Memory usage is the maximum resident set size reported by the Linux `proc` filesystem. The SPMiner and MPMiner components of the SMCA algorithm were implemented in C++ according to the pseudocode in [20], and use the same input, timing and output mechanisms as our algorithm.[11]

7.1 Datasets

We used dynamic networks collected from a variety of sources and covering a range of interaction dynamics. These networks are described below.

Enron e-mails. The Enron e-mail corpus is a publicly available database of e-mails sent by and to employees of the now defunct Enron corporation.[12] Timestamps, senders and lists of recipients were extracted from message headers for each e-mail on file. We chose a day as the quantization timestep, with a directed (unweighted) interaction present if at least one e-mail was sent between two individuals on a particular day.
Plains zebra. Ecologists are interested in studying the association patterns of wild Plains zebras (*Equus burchelli*) in their natural habitat. For this dataset, social interactions between animals were recorded in a nature reserve in Kenya by behavioral ecologists from Princeton University, based on direct visual observations [16,22,33]. Zebras are uniquely identifiable by the pattern of stripes on various parts of their bodies. The data were collected by ecologists making visual scans of the herds, typically once a day over periods of several months. Each entity in the dynamic network is a unique Plains zebra and an interaction represents social association, as determined by spatial proximity and the domain knowledge of ecologists.
Reality mining. Cellphones with proximity tracking technology were distributed to 100 students at the Massachusetts Institute of Technology over the course of an academic year [12]. The timestep quantization was chosen as 4 h [9].

[9] The functionality is comparable in terms of the stated goal of the algorithm only, which is to mine periodic ‘multiple event 1-patterns’. SMCA suffers from the fact that it does not generate closed or parsimonious output, thus increasing its computation time and output size relative to our algorithm, without adding any extra information.

[10] http://code.google.com/p/google-sparsehash/, version 1.4.

[11] A misprint in the pseudocode for SPMiner in [20, (Fig 3, line 12)] was corrected. For MPMiner, we used the Time-Based Enumeration (TBE) scheme, since the Segment-Based Enumeration (SBE) scheme exhausted all available system memory for the datasets we tried.

[12] Available at http://www.cs.cmu.edu/~enron/.

Table 1 Dataset characteristics, and smoothing (S) and maximum period ($P_{\max}$) values used for experimental evaluation

Dataset	Vertices	Timesteps	Avg. density	S	$P_{\max}$
Enron	82,614	2,588	0.028 ± 0.064	3	40
IMDB photos (full)	29,257	13,987	0.097 ± 0.21	3	400
Plains zebra	313	1,276	0.31 ± 0.27	6	400
Reality mining	100	2,940	0.23 ± 0.17	2	60
Server Log 1 (days)	111,108	783	0.024 ± 0.019	2	40
Server Log 2 (hours)	111,108	18,807	0.24 ± 0.3	2	960

IMDB celebrities. The Internet Movie Database (IMDB)[13] maintains a large archive of tagged, disambiguated and dated photographs of individuals associated with the production of commercial entertainment, including actors, directors, and musicians. One might reasonably assert that a degree of social (or at least professional) association exists between people photographed together by the popular press. Thus, similar to the methodology of the Plains zebra sightings, we collected metadata on 193,707 photos,[14] which collectively represent a partial structure of the social network of people associated with the entertainment industry. The quantization period was 1 day. Although the time span of the dataset is just under 40 years, most of the interactions occur in the later portion of the dataset.
Server Logs. We used the HTTP access logs from an Apache web server hosting organization and personal pages for the Laboratory of Computational Population Biology at the University of Illinois at Chicago.[15] Each vertex is either an IP address on the Internet or a file hosted on the web server. A directed edge from an IP address to a file indicates that the file was successfully accessed by a host at the IP address, creating a bipartite graph at each timestep. The log data runs from April 2007 to May 2009. We used two different quantizations of 1 day and 1 h per timestep.

7.2 Results on natural data

7.2.1 Algorithm performance

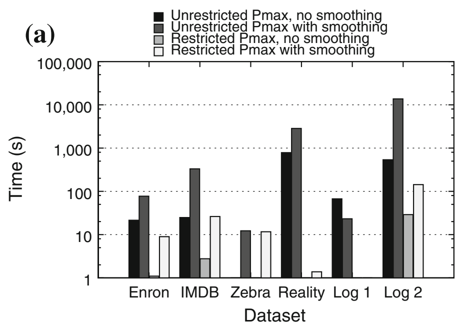
We first ran a series of experiments on our algorithm with $\sigma = 3$ and no smoothing, i.e., mining only perfectly periodic patterns. We then ran a second set of experiments with $P_{\max}$ set to restricted values, and a third set of experiments with $\sigma = 3$ and variable amounts of smoothing per dataset. Table 1 summarizes the $P_{\max}$ and smoothing values used for each dataset, based intuitively on typical periodicities and how much noise we would expect in each dataset. The second and third set of experiments demonstrate the performance of the algorithm in online and noisy situations, respectively.

Figure 7 shows the running time and memory usage of our algorithm under different circumstances. The black column shows the case when no smoothing is used and the maximum period is unrestricted. This might be considered a typical 'offline' analysis scenario. An interesting point to note is that Reality Mining takes much more time to complete mining

[13] http://www.imdb.com.

[14] In [25], we only used photos with two or more people, which is the reason for the dataset size discrepancy between versions. For this dataset, it is also informative to represent singleton (disconnected) vertices, which we have done here.

[15] http://compbio.cs.uic.edu/.



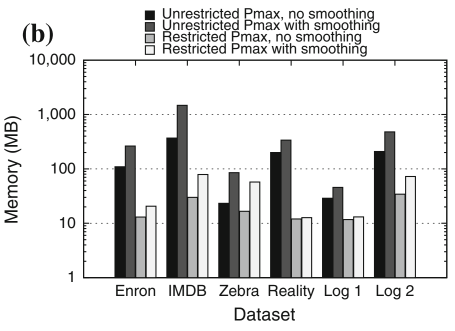
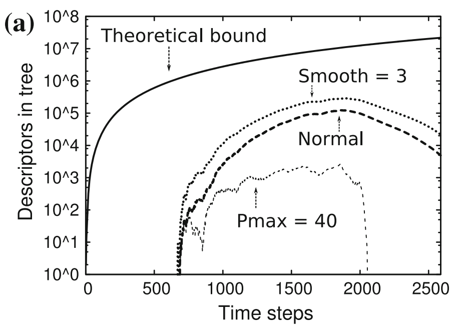


Fig. 7 Performance of the periodic subgraph mining algorithm at $\sigma = 3$, shown with an exponential y-axis. **a** Mining time. **b** Memory usage



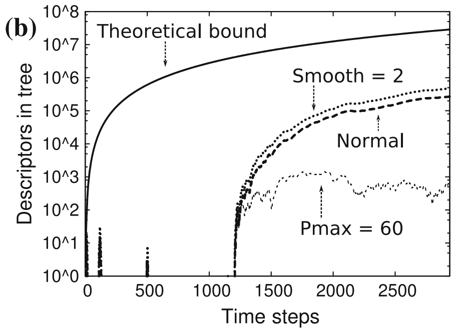


Fig. 8 Number of pattern tree descriptors with no smoothing or restrictions on period ('normal'), and for various smoothing and $P_{\max}$ values, compared to the theoretical bound. **a** Enron. **b** Reality mining

than the much larger Enron dataset, most likely due to the density of periodic patterns in it. In the typical online analysis scenario with a restricted $P_{\max}$, the algorithm took less than 30 s to execute and used less than 40 MB of memory in all cases. As expected, restricting the maximum period has a very significant effect on the performance of the algorithm.

Figure 8 shows the size of the pattern tree at each timestep for the Enron and Reality Mining datasets. It can be seen that the actual tree size is a small fraction of the theoretical upper bound. Furthermore, limiting the maximum period of mined patterns has a large impact on reducing the tree size, as expected. The Enron plot dips dramatically after about timestep 2,000 because most timesteps after that are empty. A large number of descriptors are flushed from the pattern tree when the empty timesteps are encountered. No such dip occurs in the Reality Mining dataset, which is densely periodic and continues to exhibit periodic behavior right up to the very end of the observation period.

7.2.2 *Characterizing inherent periodicity*

In addition to investigating specific periodic interaction patterns, a second goal for mining parsimonious PSEs is to analyze global periodicities in the system. In the context of dynamic networks, the goal would be to characterize the gross dynamics of the individuals in the system. Figure 9 shows histograms of the periods of patterns mined from the Enron, IMDB, Server Log and Plains Zebra datasets. For Enron, we restrict our attention to patterns with a

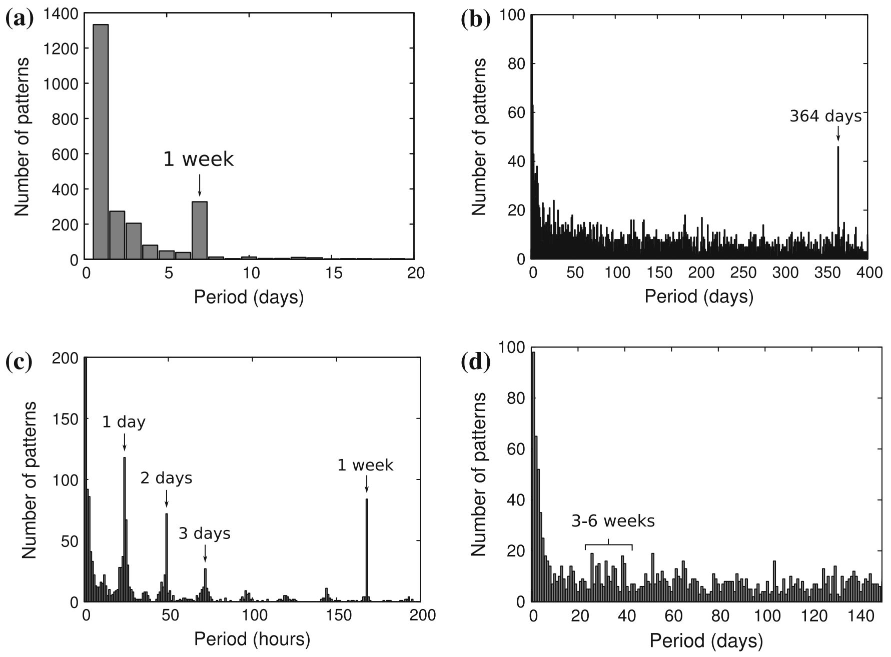


Fig. 9 Number of patterns at each period. **a** Enron, $avgPurity \geq 0.7$. **b** Internet Movie Database. **c** Server Log 2, $avgPurity \geq 0.5$. **d** Plains Zebra

high average purity, i.e., patterns which are likely to capture truly periodic behavior. Daily interaction patterns are the most prevalent periodic patterns,[16] followed by weekly patterns, as manifested by the clear peak at $p = 7$. For the IMDB dataset, we notice a similar peak at about $p = 364$. This can be explained by celebrity sightings at annual events—awards shows, for example. Thus, we are able to capture and characterize plausible natural periodicities in human interactions with no prior knowledge about the datasets. The hour-quantized Server Log dataset shows a number of interesting peaks at about 24, 48 and 168 h (the last one corresponding to a periodicity of 1 week). Note that there is also relatively little variance around the peak at 1 week, suggesting that these accesses were performed automatically. Inspecting patterns at these periods revealed the activity of various search engine crawlers, confirmed by checking ownership of IP netblocks and User-Agent strings in the HTTP requests. The Plains Zebra dataset showed a wide range of periodicities, as one might expect of animal behavior, with no strongly discernible peaks.

Figure 9a and c are histograms of the periods of patterns that are above a minimum purity threshold. Clearly, changing this threshold could result in a different picture, as patterns of lower purity get included. Figure 10 shows a two-dimensional view of the histograms as a density plot. Each row represents a histogram as in Fig. 9, but thresholded by the value of the y-axis. Darker cells represent a higher concentration of patterns at that period (relative to the most concentrated cell in the row), and correspond to the peaks in Fig. 9. The top-most row is the distribution of the periods of patterns that only occur periodically, i.e., never

[16] Too much importance should not be attached to patterns of period 1 in plots thresholded by purity, since all patterns of period 1 necessarily have purity 1.

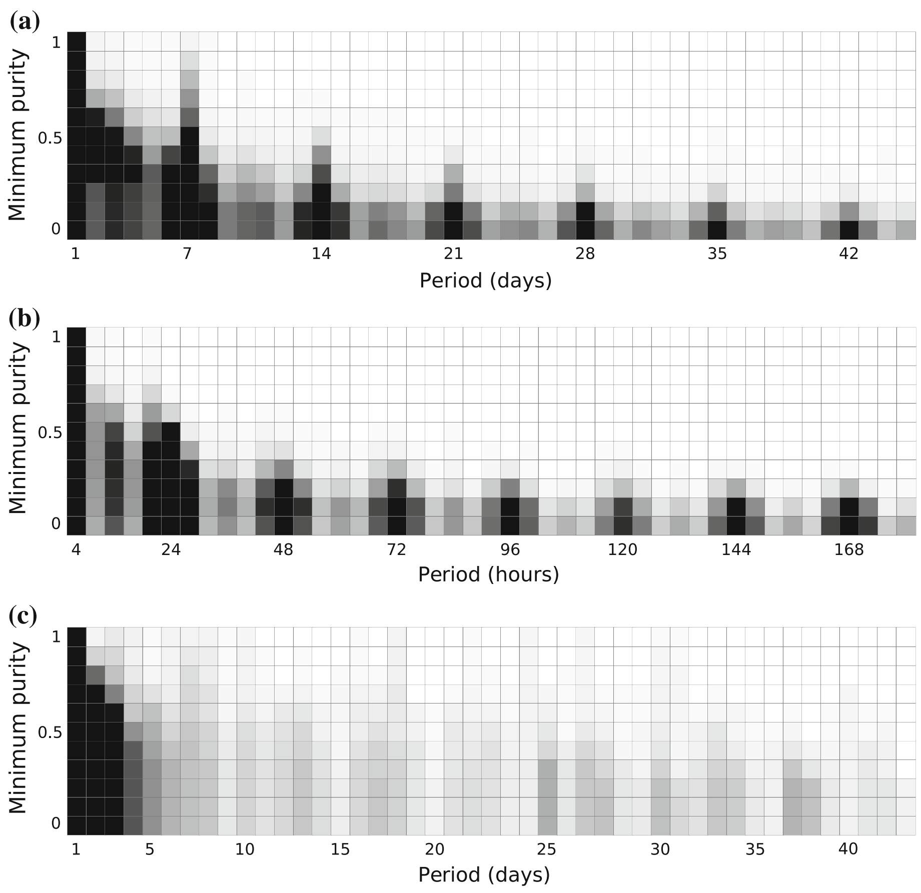


Fig. 10 Pattern density at each *minimum purity* threshold. *Each row* shows the distribution of pattern periods for patterns with purity at or greater than the *y*-axis value. *Darker cells* indicate more patterns. **a** Enron. **b** Reality mining. **c** Plains Zebra

in-between periodic occurrences, whereas the lowest row places no constraints and shows the period distribution of all mined patterns. In Fig. 10a, for example, the row corresponding to a *y*-value of 0.7 represents the histogram in Fig. 9a.

The Enron and Reality Mining datasets show strong daily and weekly periodicities, as might be expected from human interactions. This commonality is interesting because the interactions occur by different mechanisms in each dataset—by e-mail in the Enron dataset, and by physical proximity in the Reality Mining dataset. The Plains Zebra dataset, while not showing periodicities as strong as the human datasets, seems to contain relatively dense region at periods between 25 and 38. It is currently unclear whether this region indicates behavior that is ecologically meaningful, or is an artifact of the data.

7.2.3 Qualitative analysis

We now turn our attention to some qualitatively interesting periodic subgraphs discovered by our algorithm illustrating a range of periodic behavior. Figure 11a illustrates a somewhat

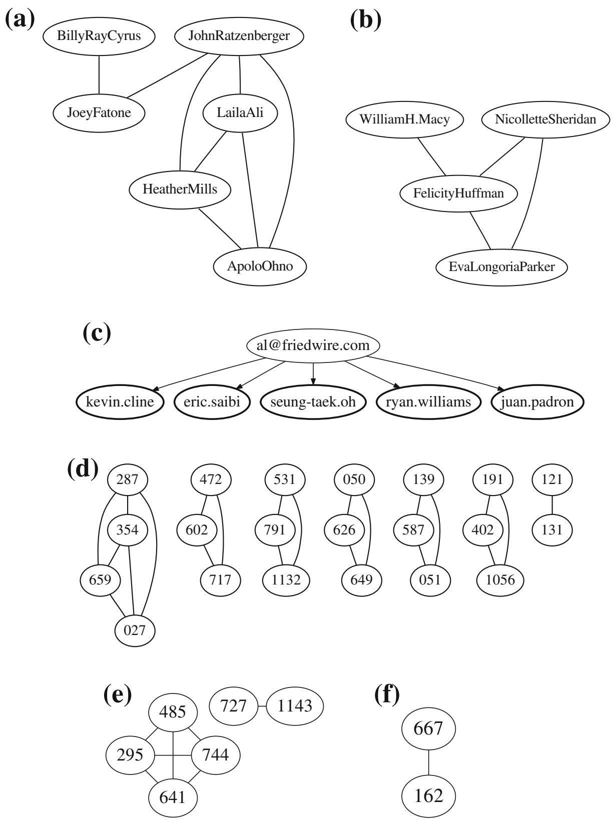


Fig. 11 Examples of some interesting periodic subgraphs. **a** IMDB: period 7 ± 2, support 3, avg. purity 1, **b** IMDB: period 364, support 3, avg. purity 0.4, **c** Enron: period 1, support 84, avg. purity 1. *Bold circles* represent @*enron.com* e-mail addresses, **d** Plains: period 7, support 4, avg. purity 0.94, **e** Plains: period 61 ± 6, support 3, avg. purity 0.71, **f** Plains: period 81 ± 6, support 4, avg. purity 1

complex pattern from the IMDB photo database that repeated approximately every week. Although the support is relatively low, what is interesting about this subgraph is the repeated non-trivial grouping of people, all of whom turned out to be contestants on a weekly 'reality television' show. Figure 11b is also from the IMDB database and is an approximately annually repeating pattern. The three individuals in the clique are actresses in a popular (circa 2004) television show, while the fourth vertex is the spouse (as of 2009) of one of the actresses. Given this context, the low average purity of the pattern is to be expected as the three actresses are very likely to have appeared together in between what are likely to be award shows. The non-trivial links in such patterns are particularly interesting and are indicative of the show's progression or relationships other than co-starring.

The subgraph shown in Fig. 11c has the highest periodic support in the Enron dataset, repeating every day for 84 consecutive days, including weekends. This is representative of a large number of similar periodic patterns in Enron, in which one person emails a group of people with periods ranging from 1 to 14 days. As shown earlier in Fig. 9, weekly emails seem to be particularly popular in a corporate setting such as this, and could be used to infer functional communities within the corporation.

Finally, we turn to the Plains Zebra dataset and to one of the most intriguing patterns shown in Fig. 11d. Although it is quite likely that the period of 7 days is an artifact of the manner in which the population was sampled, the high purity of the pattern indicates that this is a relatively stable grouping. It is also by far the largest and most repetitive such pattern, parts of which are periodic at other times as well. In contrast, the subgraphs that repeat over multiple months are shown in Figs. 11e and f. Although the support of the latter two patterns is relatively low, the high purity of Fig. 11f stands out and is representative of a large number of small but highly periodic patterns. Moreover, all the patterns are of interactions of stallion male zebras and correspond to their harems grouping for a period of time. Such groupings are indeed considered more stable for short periods of time than bachelor associations [16].

7.3 Comparison to SMCA

We generated relatively small synthetic datasets with different characteristics to compare the performance of our algorithm with the SMCA algorithm on simple interaction data. Starting with a population of 30 individuals, we generated a single graph of density d. The edges of this graph were then sampled independently at random for each of the T timesteps. Although this is not intended to be a realistic model of a social network, it allows us to control two parameters crucial to the mining process—the overall density of the dynamic network, and the number of timesteps. Since real social networks are generally sparse, we used two values for d: 0.1 and 0.15. For each of these values, T was varied from 100 to 1,000 in increments of 100.

Ten random dynamic networks were generated for each combination of T and d, and converted to their set representations. Both algorithms were run on the same set of synthetic networks with a minimum support value of $\sigma = 3$ and the maximum period unrestricted, calculated using Proposition 5.2 for each value of T. All algorithms were limited to 8 GB of disk space for storing their output, which can be considered reasonable given that this value is several orders of magnitude larger than the size of the input networks.

Figure 12 shows the performance of SMCA compared to our algorithm. The computation time used by both algorithms is comparable for density $d = 0.1$, although SMCA does not scale as well as our algorithm. For a slightly higher density of $d = 0.15$, the number of periodic patterns is expected to increase as well. The computation times are no longer comparable between algorithms, as shown in Fig. 12b. In Figs. 12b and d, there are no data points for SMCA beyond $T = 500$ since the algorithm reached the maximum output size of 8 GB prior to completion. This is partly caused by the fact that SMCA does not output closed or parsimonious patterns, which is evident from the number of patterns generated by SMCA, shown in Figs. 12c and d on a logarithmic scale.

Thus, our algorithm scales much better than SMCA. The number of patterns generated by SMCA is generally about three orders of magnitude larger than the number of parsimonious patterns output by our algorithm. The intractability of non-parsimonious periodic pattern mining is one of the main reasons we could not use SMCA on the larger natural datasets, where the number of vertices, timesteps, and the average timestep density are much higher than the values used here.

8 Conclusion

We have proposed and formalized the *periodic subgraph mining* problem for dynamic networks and analyzed the computational complexity of enumerating all periodic subgraphs. We

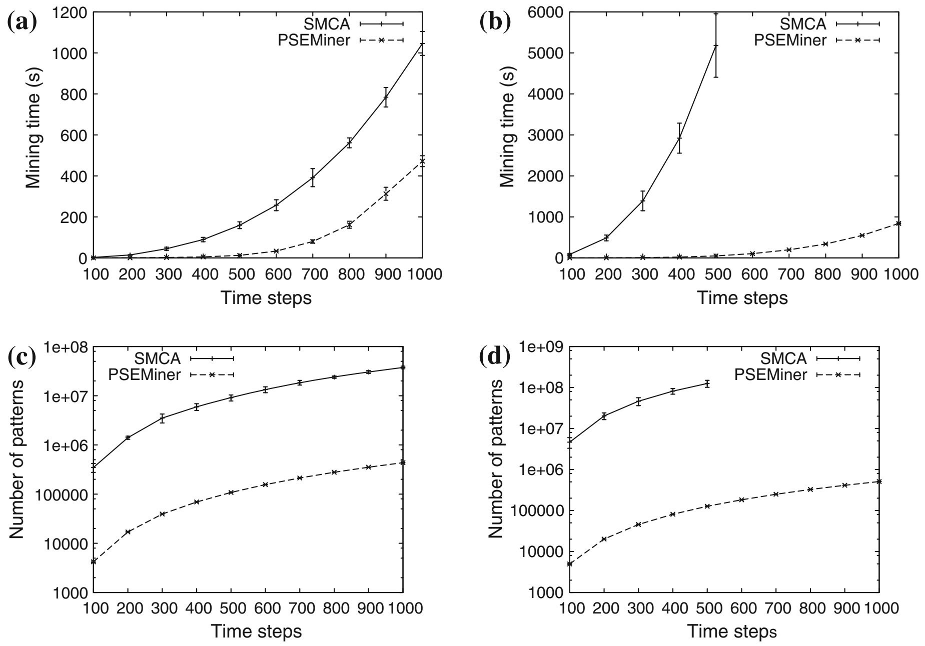


Fig. 12 The performance of SMCA compared to our algorithm, for synthetic networks of different densities d. Error bars for PSEMiner are too small to see. **a** Mining time: $d = 0.1$. **b** Mining time: $d = 0.15$. **c** Number of mined patterns: $d = 0.1$. **d** Number of mined patterns: $d = 0.15$

have shown that there are at most $O(T^2 \ln \frac{T}{\sigma})$ closed periodic subgraphs at minimum support σ in a dynamic network of T timesteps. Furthermore, we have described a polynomial time, online algorithm to mine all periodic subgraphs, including a smoothing mechanism for mining subgraphs that are not perfectly periodic. We have also proposed a new measure, *purity*, for ranking mined subgraphs according to how perfectly periodic a subgraph is. We have demonstrated our algorithm on four real-world dynamic social networks, spanning interactions between corporate executives, college students, wild zebra, and Hollywood celebrities. Our algorithm efficiently mines all periodic patterns, is provably tractable, and is a meaningful alternative to using frequent subgraph mining to look for interesting patterns in dynamic networks. We have also shown that periodic subgraphs can be used as an effective characterization of the dynamics of various systems. Our technique was able to discover plausible natural periodicities in many of the systems we examined, and shows promise as a tool for exploratory analysis of interaction dynamics.

There are a number of interesting avenues for future research. One such direction is to incorporate probabilistic models of periodicity instead of strictly combinatorial ones. Yang et al. [39] and Ma and Hellerstein [26] are two examples of such attempts; it would be interesting to see how well they perform in dynamic networks. Along the lines of various studies on assessing the interestingness of frequent patterns [5,34,36], a method for assessing the statistical significance of mined patterns under different statistical models would be valuable in dynamic networks, especially in the context of inter-disciplinary research. A number of extensions can also be made to the algorithm we have presented in this paper. These include an extension to mine complex periodic patterns, similar to the types of patterns mined in [19,20,26,40], and different algorithms or heuristics for manipulating the structure of the pattern tree to increase efficiency. The concept of noise could also be extended to discover noisy subgraphs instead of just noisy periodicities. Finally, we believe that the capabilities of the method, especially in an inter-disciplinary context, can only be fully explored if the results of the mining process are presented or visualized in a succinct but insightful manner. This is a challenging and open question.

Acknowledgments This work is supported by NSF grants IIS-0705822 and CAREER IIS-0747369 (TB-W). We are grateful to Dan Rubenstein, Ilya Fischhoff, and Siva Sundaresan of the Department of Ecology and Evolutionary Biology at Princeton University for sharing the Plains zebra data. Their work was supported by the NSF grants CNS-025214 and IOB-9874523. We would also like to thank Dr. Kuo-Yu Huang and Dr. Chia-Hui Chang, the authors of [20], for their assistance with the SMCA algorithm. We are grateful to the anonymous reviewers for their helpful comments and to the members of the Laboratory of Computational Biology at UIC for many fruitful discussions.

References

1. Agrawal R, Srikant R (1994) Fast algorithms for mining association rules in large databases. In: Proceedings of the 20th international conference on very large data bases (San Francisco, CA, 1994). Morgan Kaufmann Publishers Inc., pp 487–499
2. Agrawal R, Srikant R (1995) Mining sequential patterns. In: Proceedings of the eleventh international conference on data engineering (Washington, DC, USA, 1995). IEEE Computer Society, pp 3–14
3. Barabási AL, Jeong H, Neda Z, Ravasz E, Schubert A, Vicsek T (2002) Evolution of the social network of scientific collaborations. Physica A: Stat Mech Appl 311(3–4):590–614
4. Boros E, Gurvich V, Khachiyan L, Makino K (2002) On the complexity of generating maximal frequent and minimal infrequent sets. In: Proceedings of the 19th annual symposium on theoretical aspects of Computer Science (London, UK, 2002). Springer-Verlag, pp 133–141
5. Bringmann B, Zimmermann A (2009) One in a million: picking the right patterns. Knowl Inf Syst 18(1):61–81
6. Carpineto C, Romano G (2004) Concept data analysis: theory and applications. Wiley, New York
7. Chapanond A, Krishnamoorthy MS, Yener B (2005) Graph theoretic and spectral analysis of Enron email data. Comput Math Organ Theory 11(3):265–281
8. Cheng J, Ke Y, Ng W (2008) A survey on algorithms for mining frequent itemsets over data streams. Knowl Inf Syst 16(1):1–27
9. Clauset A, Eagle N (2007) Persistence and periodicity in a dynamic proximity network. In: DIMACS/DyDAn workshop on computational methods for dynamic interaction networks
10. Dickinson PJ, Bunke H, Dadej A, Kraetzl M (2003) On graphs with unique node labels, vol 2726 of Lecture Notes in Computer Science. Springer, Berlin, pp 409–437
11. Diesner J, Carley KM (2005) Exploration of communication networks from the Enron Email corpus. In: Proceedings of the 2005 SIAM workshop on link analysis, counterterrorism and security, pp 3–14
12. Eagle N, Pentland A (2006) Reality mining: sensing complex social systems. Pers Ubiquitous Comput 10(4):255–268
13. Elfeky MG, Aref WG, Elmagarmid AK (2005) Periodicity detection in time series databases. IEEE Trans Knowl Data Eng 17(7):875–887
14. Elfeky MG, Aref WG, Elmagarmid AK (2005) WARP: time warping for periodicity detection. In: Proceedings of the fifth IEEE international conference on data mining (Washington, DC, USA, 2005). IEEE Computer Society, pp 138–145
15. Faloutsos M, Faloutsos P, Faloutsos C (1999) On power-law relationships of the internet topology. In: Proceedings of the conference on applications, technologies, architectures, and protocols for computer communication (New York, NY, 1999). ACM, pp 251–262
16. Fischhoff IR, Sundaresan SR, Cordingley J, Larkin HM, Sellier M-J, Rubenstein DI (2007) Social relationships and reproductive state influence leadership roles in movements of Plains zebra, *Equus burchellii*. Anim Behav 73(5):825–831
17. Garofalakis M, Rastogi R, Shim K (1999) SPIRIT: sequential pattern mining with regular expression constraints. In: Proceedings of the international conference on very large data bases, pp 223–234
18. Han J, Cheng H, Xin D, Yan X (2007) Frequent pattern mining: current status and future directions. Data Min Knowl Discov 15(1):55–86

19. Han J, Yin Y, Dong G (1999) Efficient mining of partial periodic patterns in time series database. In: Proceedings of the 15th international conference on data engineering (Los Alamitos, CA, 1999). IEEE Computer Society, pp 106–115
20. Huang K-Y, Chang C-H (2005) SMCA: a general model for mining asynchronous periodic patterns in temporal databases. IEEE Trans Knowl Data Eng 17(6):774–785
21. Inokuchi A, Washio T, Motoda H (2000) An apriori-based algorithm for mining frequent substructures from graph data. In: Proceedings of the 4th European conference on principles of data mining and knowledge discovery, pp 13–23
22. Juang P, Oki H, Wang Y, Martonosi M, Peh LS, Rubenstein DI (2002) Energy-efficient computing for wildlife tracking: design tradeoffs and early experiences with ZebraNet. ACM SIGPLAN Notices 37(10):96–107
23. Kuramochi M, Karypis G (2001) Frequent subgraph discovery. In: Proceedings of the 2001 IEEE international conference on data mining, pp 313–320
24. Lahiri M, Berger-Wolf TY (2007) Structure prediction in temporal networks using frequent subgraphs. In: Proceedings of IEEE symposium on computational intelligence and data mining, pp 35–42
25. Lahiri M, Berger-Wolf TY (2008) Mining periodic behavior in dynamic social networks. In: Proceedings of the IEEE international conference on data mining (ICDM), pp 373–382
26. Ma S, Hellerstein JL (2001) Mining partially periodic event patterns with unknown periods. In: Proceedings of the 17th international conference on data engineering (Washington, DC, USA, 2001). IEEE Computer Society, pp 205–214
27. Nanavati AA, Gurumurthy S, Das G, Chakraborty D, Dasgupta K, Mukherjea S, Joshi A (2006) On the structural properties of massive telecom call graphs: findings and implications. In: Proceedings of the 15th ACM international conference on Information and knowledge management (New York, NY, USA, 2006). ACM, pp 435–444
28. Newman MEJ (2001) From the cover: the structure of scientific collaboration networks. Proc Natl Acad Sci 98:404–409
29. Özden B, Ramaswamy S, Silberschatz A (1998) Cyclic association rules. In: Proceedings of the fourteenth international conference on data engineering (Washington, DC, USA, 1998). IEEE Computer Society, pp 412–421
30. Pasquier N, Bastide Y, Taouil R, Lakhal L (1999) Efficient mining of association rules using closed itemset lattices. Inf Syst 24(1):25–46
31. Pei J, Han J (2000) Can we push more constraints into frequent pattern mining? In: Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining. ACM, New York, pp 350–354
32. Pei J, Han J, Wang W (2002) Mining sequential patterns with constraints in large databases. In: Proceedings of the eleventh international conference on information and knowledge management. ACM, New York, pp 18–25
33. Sundaresan SR, Fischhoff IR, Dushoff J, Rubenstein DI (2007) Network metrics reveal differences in social organization between two fission–fusion species, Grevys zebra and onager. Oecologia 151(1):140–149
34. Tatti N (2008) Maximum entropy based significance of itemsets. Knowl Inf Syst 17(1):57–77
35. Wasserman S, Faust K (1994) Social network analysis: methods and applications. Cambridge University Press, Cambridge
36. Yan X, Cheng H, Han J, Yu PS (2008) Mining significant graph patterns by leap search. In: SIGMOD ’08: Proceedings of the 2008 ACM SIGMOD international conference on management of data (New York, NY, USA, 2008). ACM, pp 433–444
37. Yang G (2004) The complexity of mining maximal frequent itemsets and maximal frequent patterns. In: Proceedings of the tenth ACM SIGKDD international conference on knowledge discovery and data mining (New York, NY, 2004). ACM, pp 344–353
38. Yang J, Wang W, Yu PS (2001) InfoMiner: mining surprising periodic patterns. In: Proceedings of the 7th ACM SIGKDD international conference on Knowledge discovery and data mining (New York, NY, 2001). ACM, pp 395–400
39. Yang J, Wang W, Yu PS (2002) InfoMiner+: mining partial periodic patterns with gap penalties. In: Proceedings of the 2002 IEEE international conference on data mining (Washington, DC, 2002). IEEE Computer Society, p 725
40. Yang J, Wang W, Yu PS (2003) Mining asynchronous periodic patterns in time series data. IEEE Trans Knowl Data Eng 15(3):613–628
41. Zhu F, Yan X, Han J, Yu PS (2007) gPrune: a constraint pushing framework for graph pattern mining. Lect Notes Comput Sci 4426:388

Author Biographies

Mayank Lahiri is currently a Ph.D. candidate in the Department of Computer Science at the University of Illinois at Chicago. His research interests are primarily in dynamic network analysis, and data mining and machine learning applied to dynamic social networks. He previously received a B.S. (Hons.) degree in Computer Science in 2005 from Lafayette College in Easton, Pennsylvania, and is currently supported at the University of Illinois at Chicago by a Dean's Scholarship. He also has an eloquent proof for P = NP, but it is too long to fit in this footnote.

Tanya Y. Berger-Wolf is an assistant professor at the Department of Computer Science at the University of Illinois at Chicago. Her research interests are in applications of combinatorial optimization analysis and algorithm design techniques to problems in population biology of plants, animals, and humans, from genetics to social interactions. Dr. Berger-Wolf has received her B.Sc. in Computer Science and Mathematics from Hebrew University (Jerusalem, Israel) and her Ph.D. in Computer Science from University of Illinois at Urbana-Champaign in 2002. She has spent 2 years as a postdoctoral fellow at the University of New Mexico working in computational phylogenetics and a year at the Center for Discrete Mathematics and Theoretical Computer Science (DIMACS) doing research in computational epidemiology. She has received numerous awards for her research and mentoring, including the NSF CAREER Award in 2008.