Mining Evolving Patterns in Dynamic Relational Networks

Rezwan Ahmed and George Karypis

Abstract Dynamic networks have recently been recognized as a powerful abstraction to model and represent the temporal changes and dynamic aspects of the data underlying many complex systems. This recognition has resulted in a burst of research activity related to modeling, analyzing, and understanding the properties, characteristics, and evolution of such dynamic networks. The focus of this growing research has been on mainly defining important recurrent structural patterns and developing algorithms for their identification. Most of these tools are not designed to identify time-persistent relational patterns or do not focus on tracking the changes of these relational patterns over time. Analysis of temporal aspects of the entity relations in these networks can provide significant insight in determining the conserved relational patterns and the evolution of such patterns over time. In this chapter we present new data mining methods for analyzing the temporal evolution of relations between entities of relational networks. A qualitative analysis of the results shows that the discovered patterns are able to capture network characteristics that can be used as features for modeling the underlying dynamic network in the context of a classification task.

1 Introduction

As the capacity to exchange and store information has soared, so has the amount and diversity of available data. This massive growth in scale, combined with the increasing complexity of the data, has spawned the need to develop efficient methods to process and extract useful information from this data. Within the research dedicated to solving this problem, a significant amount of attention has been given to develop efficient mining tools to analyze graphs and networks modeling relations between objects or entities. Due to their flexibility and availability of theoretical and applied tools for efficient analysis, networks have been used as generic model to represent the relations between various entities in diverse applications. Examples

R. Ahmed (✉) • G. Karypis
Department of Computer Science and Engineering, University of Minnesota, Minneapolis, MN 55455, USA
e-mail: ahmed@cs.umn.edu; karypis@cs.umn.edu

M.E. Celebi, K. Aydin (eds.), *Unsupervised Learning Algorithms*,
DOI 10.1007/978-3-319-24211-8_17

of some widely studied networks include the friend-networks of popular social networking sites like Facebook, the Enron email network, co-authorship and citation networks, and protein–protein interaction networks. Until recently, the focus of the research has been on analyzing graphs and networks modeling static data. However, with the emergence of new application areas, it has become apparent that static models are inappropriate for representing the temporal changes underlying many of these systems, and that it is imperative to develop models capable of capturing the dynamic aspects of the data, as well as tools to analyze and process this data.

In recent years there has been a burst of research activity related to modeling, analyzing, and understanding the properties and characteristics of such dynamic networks. The growing research has focused on finding frequent patterns [5], clustering [8], characterizing network evolution rules [4], detecting related cliques [7, 21], finding subgraph subsequences [13], and identifying co-evolution patterns capturing attribute trends [10, 15] in dynamic networks. Although the existing techniques can detect the frequent patterns in a dynamic network, most of them are not designed to identify time-persistent relational patterns and do not focus on tracking the changes of these conserved relational patterns over time. Analysis of temporal aspects of the entity relations in these networks can provide significant insight determining the conserved relational patterns and the evolution of such patterns over time.

The objective of this chapter is to present new data mining methods that are designed to analyze and identify how the relations between the entities of relational networks evolve over time. The key motivation underlying this research is that significant insights can be gained from dynamic relational networks by analyzing the temporal evolution of their relations. Such analysis can provide evidence to the existence of, possibly unknown, coordination mechanisms by identifying the relational motifs that evolve and move in a similar and highly conserved fashion, and to the existence of external factors that are responsible for changing the stable relational patterns in these networks. Different classes of evolving relational patterns are introduced that are motivated by considering two distinct aspects of relational pattern evolution. The first is the notion of state transition and seeks to identify sets of entities whose time-persistent relations change over time and space. The second is the notion of coevolution and seeks to identify recurring sets of entities whose relations change in a consistent way over time and space.

The rest of the chapter is organized as follows. Section 2 reviews some graph-related definitions and introduces notation used throughout the chapter. Section 3 presents a new class of patterns, referred to as the evolving induced relational states (EIRS), which is designed to analyze the time-persistent relations or states between the entities of the dynamic networks. These patterns can help identify the transitions from one conserved state to the next and may provide evidence to the existence of external factors that are responsible for changing the stable relational patterns in these networks. We developed an algorithm to efficiently mine all maximal non-redundant evolution paths of the stable relational states of a dynamic network. Next in Sect. 4, we introduce a class of patterns, referred to as coevolving relational motifs (CRM), which is designed to identify recurring

sets of entities whose relations change in a consistent way over time. CRMs can provide evidence to the existence of, possibly unknown, coordination mechanisms by identifying the relational motifs that evolve in a similar and highly conserved fashion. An algorithm is presented to efficiently analyze the frequent relational changes between the entities of the dynamic networks and capture all frequent coevolutions as CRMs. In Sect. 5, we define a new class of patterns built upon the concepts of CRMs, referred to as coevolving induced relational motifs (CIRM), is designed to represent patterns in which all the relations among recurring sets of nodes are captured and some of the relations undergo changes in a consistent way across different snapshots of the network. We also present an algorithm to efficiently mine all frequent coevolving induced relational motifs. In Sect. 6 we provide a qualitative analysis of the information captured by each class of the discovered patterns. For example, we show that some of the discovered CRMs capture relational changes between the nodes that are thematically different (i.e., edge labels transition between two clusters of topics that have very low similarity). Moreover, some of these patterns are able to capture network characteristics that can be used as features for modeling the underlying dynamic network. A comprehensive evaluation of the performance and scalability of all the algorithms is presented through extensive experiments using multiple dynamic networks derived from real-world datasets from various application domains. The detailed analysis of the performance and scalability results can be found in [1–3]. Section 7 summarizes the conclusions and presents some future research directions.

2 Definitions and Notation

A relational network is represented via labeled graphs. A *graph* $G = (V, E)$ consists of a set of vertices V and a set of edges E that contain pairs of vertices. A graph G is called *labeled* if there are functions χ and ψ that assign labels to the vertices and edges, respectively. That is, there is a set of vertex labels λ_v and edge labels λ_e such that $\chi : V \rightarrow \lambda_v$ and $\psi : E \rightarrow \lambda_e$. For simplicity, we will use l_u and l_v to denote the labels of vertices u and v, respectively, and use l_e to denote the label of edge e. The labels assigned to the vertices (edges) do not need to be distinct and the same label can be assigned to different vertices (edges). A graph $G' = (V', E')$ is a *subgraph* of G if $V' \subseteq V$ and $E' \subseteq E$. A subgraph $G'' = (V'', E'')$ of G is an *induced subgraph* if $\{(u, v)|(u, v) \in V'' \times V''\} \cap E = E''$.

Given a connected graph $G = (V, E)$ and a depth-first search (DFS) traversal of G, its *depth-first search tree* T is the tree formed by the forward edges of G. All nodes of G are encoded with subscripts to order them according to their discovery time. Given a DFS tree T of graph G containing n nodes, the root node is labeled as (v_0) and the last discovered node is labeled as (v_{n-1}). The *rightmost path* of the DFS tree T is the path from vertex v_0 to vertex v_{n-1}.

A *dynamic network* $\mathcal{N} = \langle G_1, G_2, \ldots, G_T \rangle$ is modeled as a finite sequence of labeled undirected graphs, where $G_t = (V, E_t)$ describes the state of the system at

a discrete time interval t. Each of these labeled undirected graphs will be called a *snapshot*. Note that the set of vertices in each snapshot is assumed to be the same whereas the set of edges can change across the different snapshots. The set of vertices is referred to as the nodes of $\mathscr{N}$, denoted by $V_{\mathscr{N}}$. When nodes appear or disappear over time, the set of nodes of each snapshot is the union of all the nodes over all snapshots. The nodes across the different snapshots are numbered consistently, so that the ith node of G_k ($1 \leq k \leq T$) will always correspond to the same ith node of $\mathscr{N}$. In addition, each of these snapshots is assumed to have their own vertex- and edge-labeling functions (χ_t and ψ_t), which allows for the label of each vertex/edge to change across the snapshots.

An edge (u, v) is in a *consistent state* over a maximal time interval $s{:}e$ if it is present in all snapshots $G_s, \ldots, G_e$ with the same label and it is different in both G_{s-1} and G_{e+1} (assuming $s > 1$ and $e < T$). We define the *span sequence* of an edge as the sequence of maximal-length time intervals in which an edge is present in a consistent state. The span sequence of an edge will be described by a sequence of vertex labels, edge labels, and time intervals of the form $\langle (l_{u_1}, l_{e_1}, l_{v_1}, s_1{:}e_1), \ldots, (l_{u_n}, l_{e_n}, l_{v_n}, s_n{:}e_n) \rangle$, where $s_i \leq e_i$ and $e_i \leq s_{i+1}$.

A *persistent dynamic network* $\mathscr{N}^{\phi}$ is derived from a dynamic network $\mathscr{N}$ by removing all the edges from the snapshots of $\mathscr{N}$ that do not occur in a consistent state in at least ϕ consecutive snapshots, where $1 \leq \phi \leq T$. It can be seen that $\mathscr{N}^{\phi}$ can be derived from $\mathscr{N}$ by removing from the span sequence of each edge all the intervals whose length is less than ϕ.

An *injection* is defined as a function $f : A \rightarrow B$ such that $\forall a, b \in A$, if $f(a) = f(b)$, then $a = b$. A function $f : A \rightarrow B$ is a bijective function or *bijection*, *iff* $\forall b \in B$, there is a unique $a \in A$ such that $f(a) = b$. A function composition $g \circ f$ implies that for function $f : X \rightarrow Y$ and $g : Y \rightarrow Z$, then *composite* function $g \circ f : X \rightarrow Z$.

An induced subgraph that involves the same set of vertices and whose edges and their labels remain conserved across a sequence of snapshots will be referred to as the *induced relational state* (IRS). The IRS definition is illustrated in Fig. 1. The three-vertex induced subgraph consisting of the dark shaded nodes that are connected via the directed edges corresponds to an induced relational state as it remains conserved in the three consecutive snapshots. The key attribute of an IRS is that the set of vertices and edges that compose the induced subgraph must remain

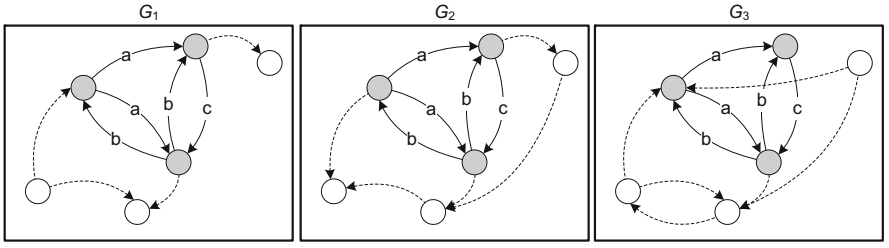


Fig. 1 Examples of relational states

the same in the consecutive snapshots. From this definition we see that an IRS corresponds to a time-conserved pattern of relations among a fixed set of entities (i.e., nodes) and as such can be thought as corresponding to a *stable* relational pattern. An IRS S_i will be denoted by the tuple $S_i = (V_i, s_i{:}e_i)$, where V_i is the set of vertices of the induced subgraph that persists from snapshot G_{s_i} to snapshot G_{e_i} ($s_i \leq e_i$) and it does not persist in G_{s_i-1} and G_{e_i+1} (assuming $s_i > 1$ and $E_i < T$). We will refer to the time interval $s_i{:}e_i$ as the *span* of S_i, and to the set of consecutive snapshots $G_{s_i}, \ldots, G_{e_i}$ as its *supporting set*. By its definition, the span of an IRS and the length of its supporting set are *maximal*. Note that for the rest of the thesis, any references to subsequences of snapshots will be assumed to be consecutive. The induced subgraph corresponding to an IRS S_i will be denoted as $g(S_i)$. A snapshot G_t supports an induced relational state S_i, if $g(S_i)$ is an induced subgraph of G_t. The size of an IRS S_i is defined in terms of the number of vertices and represented as k, where $k = |V_i|$. Based on the type of the dynamic network, a low k value may generate a large number of IRSs capturing trivial relational information, whereas a large k value may not detect any IRS.

A *relational motif* is a subgraph that occurs frequently in a single snapshot or a collection of snapshots. It is similar to the traditional definition of frequently occurring graph patterns [16]. An *embedding* of a relational motif in a snapshot is determined by performing subgraph isomorphism operation of the motif's graph pattern. A motif *occurs* in a snapshot (i.e., a snapshot supports a motif) means that there exists an embedding of the motif in to the snapshot. In order to determine, how many times a motif occurs in a collection of snapshots, we count the total number of embeddings of the motif in that collection of snapshots. In Fig. 2, the three-vertex subgraph consisting of the shaded nodes that are connected via the labeled edges corresponds to a relational motif that occurs a total of four times (twice in G_1 and once in each of G_2 and G_3). Note that the set of nodes that support the multiple occurrences of a relational motif do not need to be the same. In Fig. 2, there are three non-identical sets of nodes supporting the four occurrences, since the shaded

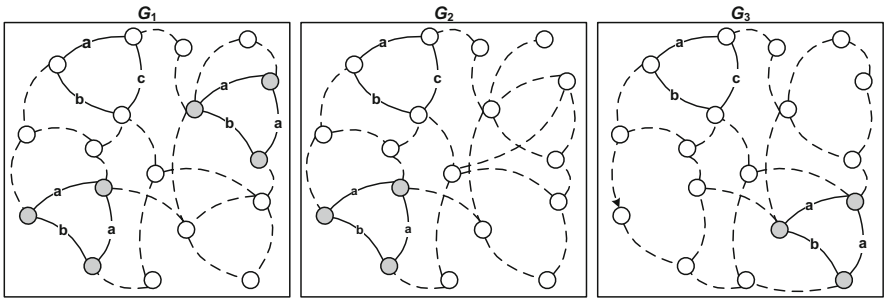


Fig. 2 Examples of relational motifs. The *shaded nodes* that are connected via the *labeled edges* correspond to a relational motif. The three-vertex subgraph consisting of the *shaded nodes* that are connected via the *labeled edges* corresponds to a relational motif that occurs a total of four times (twice in G_1 and once in each of G_2 and G_3)

node set in G_2 is same as one of the node set of G_1 (lower left one). We will use M to denote a relational motif and the underlying subgraph will be denoted by the tuple (N, A, L_N, L_A), where N is the set of nodes, A is the set of edges (arcs), and the functions L_N and L_A that assign labels to the nodes and edges, respectively.

Note that a key difference between relational states and relational motifs is that the set of nodes that support the relational state is the same in the consecutive snapshots. On the other hand, the set of nodes that support the multiple occurrences of a relational motif do not need to be the same. This difference changes the computational complexity required to identify these two types of relational patterns. Specifically, since the sets of vertices involved in the relational state remain fixed across its supporting set of snapshots, we can determine if a snapshot supports a relational state by simply checking if the relational state's graph pattern is a subgraph of that snapshot. On the other hand, in order to determine if a snapshot supports a relational motif (and how many times), we need to perform subgraph isomorphism operations (i.e., identify the embeddings of the relational motif's graph pattern), which can be expensive if the number of distinct vertex labels in the motif and/or snapshot is small.

An *induced relational motif* is an induced subgraph that occurs frequently in a single snapshot or a collection of snapshots. In order to determine if a snapshot supports an induced relational motif (and how many times), we need to perform induced subgraph isomorphism operations (i.e., identify the embeddings of the relational motif's graph pattern).

3 Mining the Evolution of Conserved Relational States

Significant insights regarding the stable relational patterns among the entities can be gained by analyzing temporal evolution of the complex entity relations. This can help identify the transitions from one conserved state to the next and may provide evidence to the existence of external factors that are responsible for changing the stable relational patterns in these networks.

3.1 Evolving Induced Relational State

An example of the type of evolving patterns in dynamic networks that the work in this section is designed to identify is illustrated in Fig. 3. This figure shows a hypothetical dynamic network consisting of 14 consecutive snapshots, each modeling the annual relations among a set of entities. The four consecutive snapshots for years 1990 through 1993 show an induced relational state S_1 that consists of nodes {a, b, e, f}. This state was evolved to the induced relational state S_2 that occurs in years 1995–1999 that contains nodes {a, b, d, e, h}. Finally, in years 2000–2003,

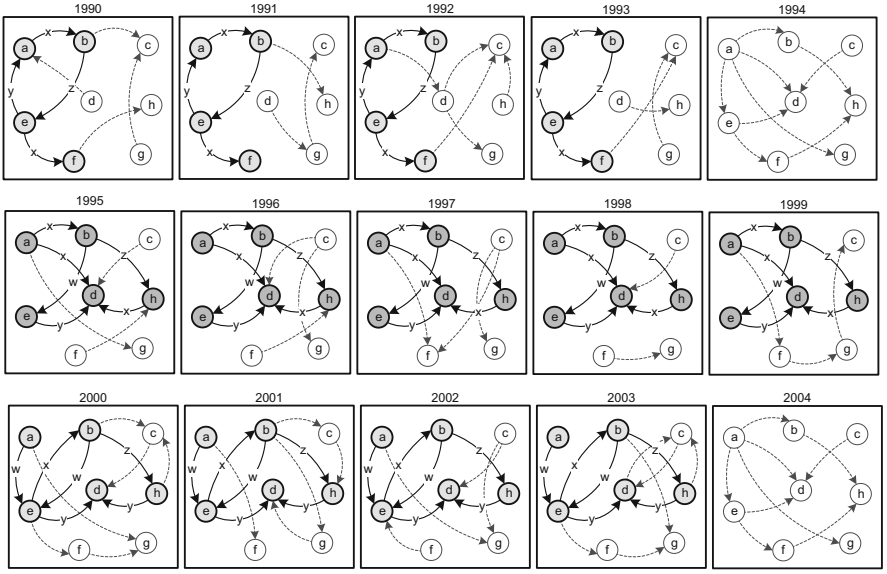


Fig. 3 An example of an evolving relational state

the induced relational state S_2 was further evolved to the induced relational state S_3 that contains the same set of nodes but has a different set of relations. Note that even though the sets of nodes involved in S_1 and S_2 are different, there is a high degree of overlap between them. Moreover, the transition from S_1 to S_2 did not happen in consecutive years, but there was a 1 year gap, as the snapshot for 1994 does not contain either S_1 or S_2. Such a sequence of induced relational states $S_1 \rightsquigarrow S_2 \rightsquigarrow S_3$ represents an instance of what we refer to as an *evolving induced relational state* (EIRS) and represents the types of patterns that the work in this section is designed to identify.

EIRSs identify entities whose relations transition through a sequence of time-persistent relational patterns and as such can provide evidence of the existence of external factors responsible for these relational changes. For example, consider a dynamic network that captures the trading patterns between a set of entities. The nodes in this *trading network* model the trading entities (e.g., countries, states, businesses, individuals) and the directed edges model the transfer of goods and their types from one entity to another. An EIRS in this trading network can potentially identify how the trading patterns change over time (e.g., addition/deletion of edges or inclusion of new trading partners) signalling the existence of significant economic, political, and environmental factors that drive such changes (see Sect. 6 for some examples of such patterns in an inter-country trading network). Similarly, in a dynamic network that captures the annual citation structure of U.S. Patents (or other scientific publications), EIRSs can identify how stable knowledge transfer

or knowledge sharing sub-networks among different science areas have evolved and thus facilitate the identification of transformative science developments that changed the state of these sub-networks.

The formal definition of an EIRS is as follows.

Definition 1 (Evolving Induced Relational State). Given a dynamic network $\mathcal{N}$ containing T snapshots, a value ϕ ($1 \leq \phi \leq T$), and a value β ($0 < \beta \leq 1$), an evolving induced relational state of length m is a sequence of induced relational states $\langle S_1, S_2, \ldots, S_m \rangle$ that satisfies the following constraints:

(i) the supporting set of each induced relational state S_i contains at least ϕ consecutive snapshots in the persistent dynamic network $\mathcal{N}^{\phi}$ of $\mathcal{N}$,
(ii) for each $1 \leq i < m$, the first snapshot in S_{i+1}'s supporting set follows the last snapshot in S_i's supporting set,
(iii) for each $1 \leq i < m$, $g(S_i)$ is different from $g(S_{i+1})$, and
(iv) for each $1 \leq i < m$, $|V_i \cap V_{i+1}|/|V_i \cup V_{i+1}| \geq \beta$.

The value ϕ, referred to as the support of the EIRS, is used to capture the requirement that each induced relational state occurs in a sufficiently large number of consecutive snapshots and as such it represents a set of relations among the entities involved that are stable. The value β, referred to as the *inter-state similarity*, is used to enforce the minimum vertex-level similarity between the selected relational states. This ensures that the EIRS captures the relational transitions of a consistent set of vertices (i.e., they do not jump on entirely different parts of the network in successive relational states) but at the same time allows for the inclusion of new vertices and/or the elimination of existing vertices, if they are required to describe the new relational state. The inter-state similarity can be also defined in terms of the maximum number of vertices that can be different between the selected relational states. The third constraint in the above definition is used to eliminate EIRSs that contain consecutive IRSs with identical induced subgraphs. This is motivated by our desire to find EIRSs that capture changes in the time-persistent relations. However, the above definition allows for the same induced subgraph to occur multiple times in the same EIRS, as long as these occurrences do not happen one after the other (e.g., it allows for EIRSs of the form $\langle S_1, S_2, S_3 \rangle$ in which $g(S_1) = g(S_3)$).

An important aspect of the definition of an EIRS is that it is defined with respect to the persistent dynamic network $\mathcal{N}^{\phi}$ of $\mathcal{N}$ and not $\mathcal{N}$ itself. This is because we are interested in finding how the persistent relations among a set of entities have changed over time and as such we first eliminate the set of relations that appear for a short period of time. Note that we focus on finding the maximal EIRSs, since they represent the non-redundant set of EIRSs.

Given the above definition, the work in this section is designed to develop efficient algorithms for solving the following problem:

Problem 1 (Maximal Evolving Induced Relational State Mining). Given a dynamic network $\mathcal{N}$ containing T snapshots, a user defined support ϕ ($1 \leq \phi \leq T$), an

inter-state similarity β ($0 < \beta \leq 1$), a minimum size of $k_{\min}$ and a maximum size of $k_{\max}$ vertices per IRS, and a minimum EIRS length $m_{\min}$, find all EIRSs such that no EIRS is a subsequence of another EIRS.

Since the set of maximal EIRSs contains all non-maximal EIRSs, the above problem will produce a succinct set of results. Also, the minimum and maximum constraints on the size of the IRSs involved is introduced to allow an application to focus on IRSs of meaningful size, whereas the minimum constraint on the EIRS length is introduced in order to eliminate short paths.

3.2 Finding Evolving Induced Relational States

The algorithm that we developed for finding all maximal EIRSs (Problem 1) follows a two-step approach. In the first step, the dynamic network $\mathcal{N}$ is transformed into its persistent dynamic network $\mathcal{N}^{\phi}$ and a recursive enumeration algorithm is used to identify all the IRSs $\mathcal{S}$ whose supporting set is at least ϕ in $\mathcal{N}^{\phi}$. The $\mathcal{N}$ to $\mathcal{N}^{\phi}$ transformation is done by removing spans that are less than ϕ from each edge's span sequence and then removing the edges with empty span sequences. In the second step, the set of IRSs are mined in order to identify their maximal non-redundant sequences that satisfy the constraints of EIRS's definition.

3.2.1 Step 1: Mining of Induced Relational States

The algorithm that we developed to mine all induced relational states is based on a recursive approach to enumerate all (connected) induced subgraphs of a graph that satisfy minimum and maximum size constraints. In the rest of this section we first describe the recursive algorithm to enumerate all induced subgraphs in a simple graph and then describe how we modified this approach to mine the induced relational states in a dynamic network. The enumeration algorithm was inspired by the recursive algorithm to enumerate all spanning trees [23]. Our discussion initially assumes that the graph is undirected and the necessary modifications that apply for directed graphs are described afterwards. Also, any references to induced subgraphs assumes *connected* induced subgraphs.

Induced Subgraph Enumeration

Given a graph $G = (V, E, L[E])$, let $G_i = (V_i, E_i, L[E_i])$ be an induced subgraph of G (V_i can also be empty), V_f be a subset of vertices of V satisfying $V_i \cap V_f = \emptyset$, and let $F(V_i, V_f)$ be the set of induced subgraphs of G that contain V_i and zero or more vertices from V_f. Given these definitions, the complete set of induced subgraphs of G is given by $F(\emptyset, V)$. The set $F(V_i, V_f)$ can be computed using the recurrence relation.

$$F(V_i, V_f) = \begin{cases} V_i, & \text{if } \text{sgadj}(V_i, V_f) = \emptyset \\ & \text{or } V_f = \emptyset \\ F(\{u\}, V_f \setminus u) \cup F(\emptyset, V_f \setminus u), & \text{if } V_i = \emptyset \wedge V_f \neq \emptyset \\ \text{where } u \in V_f & \\ F(V_i \cup \{u\}, V_f \setminus u) \cup F(V_i, V_f \setminus u), & \text{otherwise,} \\ \text{where } u \in \text{sgadj}(V_i, V_f) & \end{cases} \tag{1}$$

where $\text{sgadj}(V_i, V_f)$ (*subgraph-adjacent*) denotes the vertices in V_f that are adjacent to at least one of the vertices in V_i.

To show that Eq. (1) correctly generates the complete set of induced subgraphs, it is sufficient to consider the three conditions of the recurrence relation. The first condition, which corresponds to the initial condition of the recurrence relation, covers the situations in which either (1) none of the vertices in V_f are adjacent to any of the vertices in V_i and as such V_i is the only induced subgraphs that can be generated, or (2) V_f is empty and as such V_i cannot be extended further. The second condition, which covers the situation in which V_i is empty and V_f is not empty, decomposes $F(\emptyset, V_f)$ as the union of two sets of induced subgraphs based on an arbitrarily selected vertex $u \in V_f$. The first is the set of induced subgraphs that contain vertex u (corresponding to $F(\{u\}, V_f \setminus u)$) and the second is the set of induced subgraphs that do not contain u (corresponding to $F(\emptyset, V_f \setminus u)$). Since any of the induced subgraphs in $F(\emptyset, V_f)$ will either contain u or not contain u, the above decomposition covers all possible cases and it correctly generates $F(\emptyset, V_f)$. Finally, the third condition, which corresponds to the general case, decomposes $F(V_i, V_f)$ as the union of two sets of induced subgraphs based on an arbitrarily selected vertex $u \in V_f$ that is adjacent to at least one vertex in V_i. The first is the set of induced subgraphs that contain V_i and u (corresponding to $F(V_i \cup \{u\}, V_f \setminus u)$) and the second is the set of induced subgraphs that contain V_i but not u (corresponding to $F(V_i, V_f \setminus u)$). Similarly to the second condition, this decomposition covers all the cases with respect to u and it correctly generates $F(V_i, V_f)$. Also the requirement that $u \in \text{sgadj}(V_i, V_f)$ ensures that this condition enumerates only the connected induced subgraphs[1]. Since each recursive call in Eq. (1) removes a vertex from V_f, the recurrence relation will terminate due to the first condition. Finally, since the three conditions in Eq. (1) cover all possible cases, the overall recurrence relation is correct.

In addition to correctness, it can be seen that the recurrence relation of Eq. (1) does not have any overlapping sub-problems, and as such, each induced subgraph of $F(V_i, V_f)$ is generated only once, leading to an efficient approach for generating $F(V_i, V_f)$. Constraints on the minimum and maximum size of the induced subgraphs

[1]Given a connected (induced) subgraph g, it can be grown by adding one vertex at a time while still maintaining connectivity; e.g., an MST of g (which exists due to its connectivity) can be used to guide the order by which vertices are added.

can be easily incorporated in Eq. (1) by returning $\emptyset$ in the first condition when $|V_i|$ is less than the minimum size and not performing the recursive exploration for other two cases.

Induced Relational State Enumeration

There are two key challenges in extending the induced subgraph enumeration approach of Eq. (1) in order to enumerate the IRSs in a dynamic network. First, the addition of a vertex to an IRS is different from adding a vertex to an induced subgraph as it can result in multiple IRSs depending on the overlapping spans between the vertex being added and the original IRS. Consider an IRS $S_i = (V_i, s_i : e_i)$, a set of vertices V_f such that $V_i \cap V_f = \emptyset$, and a vertex $v \in V_f$ that is adjacent to at least one of the vertices in V_i. If v's span sequence contains multiple spans that have overlaps greater than or equal to ϕ with S_i's span, then the inclusion of v leads to multiple IRSs, each supported by different disjoint spans.

Figure 4 illustrates the vertex addition process during an IRS expansion. Figure 4a shows a simple case of vertex addition where an IRS $S_1 = (\{a, b, e\}, 1:7)$ is expanded by adding adjacent vertex c having a single span of $2:4$. The resultant IRS is $S_2 = (\{a, b, e, c\}, 2:4)$ that contains all the vertices and only the overlapping span of S_1 and c. Figure 4b shows a more complex case where the vertex h that is added to S_1 has the span sequence of $\langle 1:3, 5:9 \rangle$. In this case, the overlapping spans $1:3$ and $5:7$ form two separate IRSs $S_3 = (\{a, b, e, h\}, 1:3)$ and $S_4 =$

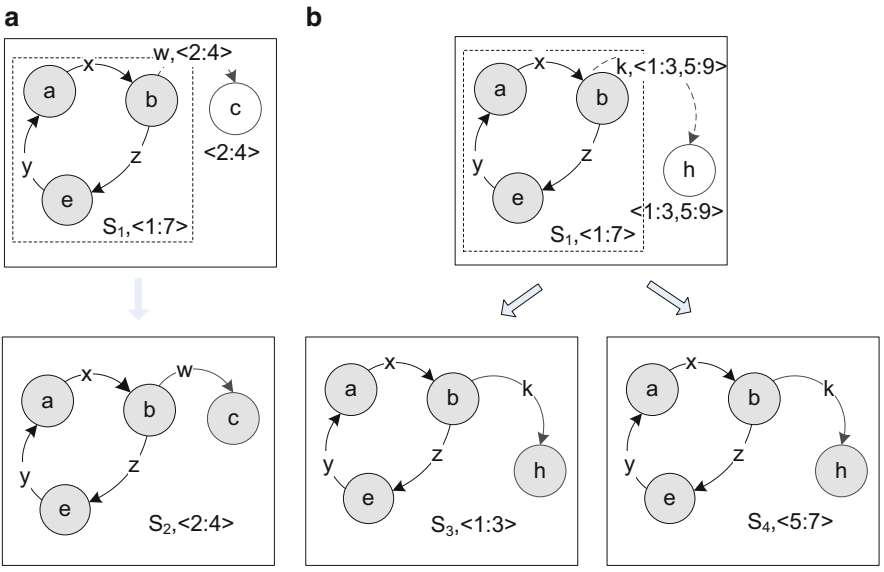


Fig. 4 Adding a vertex to an induced relational state

$(\{a, b, e, h\}, 5{:}7)$, each of which needs to be considered for future expansions in order to discover the complete set of IRSs.

Second, the concept of removing a vertex from V_f used in Eq. (1) to decompose the set of induced subgraphs needs to be re-visited so that to account for the temporal nature of dynamic networks. Failure to do so will lead to an IRS discovery algorithm that will not discover the complete set of IRSs and the set of IRSs that it discovers will be different based on the order that it chooses to add vertices in the IRS under consideration.

This is illustrated in the example of Fig. 5. The IRS $S_1 = (\{a, b\}, 0{:}12)$ is expanded to $S_2 = (\{a, b, c\}, 1{:}5)$ by adding the adjacent vertex c. In terms of Eq. (1), this corresponds to the third condition and leads to the recursive calls of $F(\{a, b, c\}, V_f \setminus c)$ and $F(\{a, b\}, V_f \setminus c)$ (i.e., expand S_2 and expand S_1). It is easy to see that the set of IRSs that will be generated from these recursions will not contain $S_4 = (\{a, b, c, d\}, 6{:}11)$, since it can only be generated from S_2 but its span does not overlap with S_4's span. On the other hand, if S_1 is initially expanded by adding vertex d, resulting in the IRS $S_3 = (\{a, b, d\}, 6{:}11)$, then the recursive calls of $F(\{a, b, d\}, V_f \setminus d)$ and $F(\{a, b\}, V_f \setminus d)$ will generate S_4 and S_2, respectively. Thus, based on the order by which vertices are selected and included in an IRS, some IRSs may be missed. Moreover, different vertex inclusion orders can potentially miss different IRSs.

To address both of these issues the algorithm that we developed for enumerating the complete set of IRSs utilizes a recursive decomposition approach that extends Eq. (1) by utilizing two key concepts. The first is the notion of the set of vertex-span tuples that can be used to grow a given IRS. Formally, given an IRS $S_i = (V_i, s_i{:}e_i)$

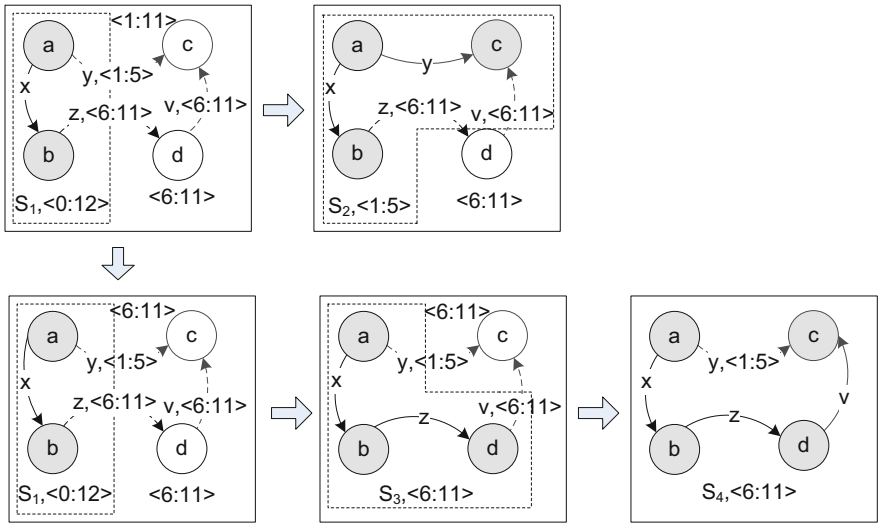


Fig. 5 Updating span sequence of a vertex

and a set of vertices V_f in $\mathcal{N}^{\phi}$ with $V_i \cap V_f = \emptyset$, the irsadj(S_i, V_f) is the set of vertex-span tuples of the form $(u, s_{u_j} : e_{u_j})$ such that $u \in V_f$ and $(V_i \cup \{u\}, s_{u_j} : e_{u_j})$ is an IRS whose span is at least ϕ. Note that irsadj(S_i, V_f) can contain multiple tuples for the same vertex if that vertex can extend S_i in multiple ways (each having a different span and possibly an induced subgraph with different sets of edges). The tuples in irsadj(S_i, V_f) represent possible extensions of S_i and since a vertex can occur multiple times, it allows for the generation of IRS with the same set of vertices but different spans (addressing the first challenge). To illustrate how this operation can address the first challenge, consider the example of Fig. 4b. When vertex h containing the span sequence of $\langle 1:3, 5:9 \rangle$ is added into $S_1 = (\{a, b, e\}, 1:7)$, each vertex-span tuple of h generates separate overlapping spans $1:3$ and $5:7$. This results in forming two separate IRSs $S_3 = (\{a, b, e, h\}, 1:3)$ and $S_4 = (\{a, b, e, h\}, 5:7)$. Now, S_3 and S_4 are further expanded in order to discover the complete set of IRSs.

The second is the notion of vertex-span deletion, which is used to eliminate the order dependency described earlier and generates the complete set of IRSs. The key idea is when a tuple $(u, s_{u_j} : e_{u_j})$ is added into S_i, instead of removing u from V_f, only remove the span $s_{u_j} : e_{u_j}$ from u's span sequence in V_f. Vertex u will only be removed from V_f iff after the removal of $s_{u_j} : e_{u_j}$ its span sequence becomes empty or the remaining spans have lengths that are smaller than ϕ. Formally, given V_f, a vertex $u \in V_f$, and a vertex-span tuple $(u, s_{u_j} : e_{u_j})$, the *span-deletion* operation, denoted by $V_f|(u, s_{u_j} : e_{u_j})$, updates the span sequence of u by removing the span $s_{u_j} : e_{u_j}$ from its span sequence, eliminating any of its spans that become shorter than ϕ, and eliminating u if its updated span sequence becomes empty. The span-deletion operation is the analogous operation to vertex removal of Eq. (1). To illustrate how this operation can address the second challenge, consider again the example of Fig. 5. Once c is added into S_1, the span that it used (i.e., $1:5$) is deleted from its span sequence, resulting in a new span sequence containing $\langle 6:11 \rangle$. Now the recursive call corresponding to $F(\{a, b\}, V_f|(c, 1:5))$ will be able to identify S_4 as c is still part of V_f.

Given the above definitions, the recursive approach for enumerating the complete set of IRSs can now be formally defined. Let $S_i = (V_i, s_i : e_i)$ be an IRS, V_f a set of vertices in $\mathcal{N}^{\phi}$ with their corresponding span-sequences in $\mathcal{N}^{\phi}$ such that $V_i \cap V_f = \emptyset$, and $H(S_i, V_f)$ be the set of IRSs that (1) contain V_i and zero or more vertices from V_f and (2) their span is a sub-span[2] of $s_i : e_i$. Given the above, the complete set of IRSs in $\mathcal{N}^{\phi}$ is given by $H((\emptyset, 1:T), V(\mathcal{N}^{\phi}))$. The recurrence relation for $H(S_i, V_f)$ is:

[2]The *sub-span* of a span corresponds to a time interval that is either identical to the span or is contained within it.

$$H(S_i, V_f) = \begin{cases} S_i, & \text{if irsadj}(S_i, V_f) = \emptyset \text{ or } V_f = \emptyset \\ H((\{u\}, s_u{:}e_u), V_f | (u, s_u{:}e_u)) \cup H((\emptyset, 1{:}T), V_f | (u, s_u{:}e_u)), & \text{if } V_i = \emptyset \wedge V_f \neq \emptyset \\ \text{where } u \in V_f \text{ and } s_u{:}e_u \text{ is a span of } u & \\ H((V_i \cup \{u\}, s_u{:}e_u), V_f | (u, s_u{:}e_u)) \cup H(S_i, V_f | (u, s_u{:}e_u)), & \text{otherwise.} \\ \text{where } (u, s_u{:}e_u) \in \text{irsadj}(S_i, V_f) & \end{cases} \quad (2)$$

The above recurrence relation shares the same overall structure with the corresponding recurrence relation for enumerating the induced subgraphs [Eq. (1)] and its correctness can be shown in a way similar to that used for Eq. (1). To eliminate redundancy, we omit the complete proof of Eq. (2), and only focus on discussing the third condition, which represents the general case. This condition decomposes $H(S_i, V_f)$ as the union of two sets of IRSs based on an arbitrarily selected vertex-span tuple $(u, s_u{:}e_u) \in \text{irsadj}(S_i, V_f)$. Since the span $s_u{:}e_u$ is a maximal overlapping span between u and S_i, the set of vertices $V_i \cup \{u\}$ with the span of $s_u{:}e_u$ is an IRS. With respect to vertex-span tuple $(u, s_u{:}e_u)$, the set of IRSs in $H(S_i, V_f)$ can belong to one of the following three groups: (i) the set of IRSs that contain u and have a span that is a sub-span of $s_u{:}e_u$; (ii) the set of IRSs that contain u and have a span that is disjoint with $s_u{:}e_u$, and (iii) the set of IRSs that do not contain u. The $H((V_i \cup \{u\}, s_u{:}e_u), V_f | (u, s_u{:}e_u))$ part of the third condition generates (i), whereas the $H(S_i, V_f | (u, s_u{:}e_u)$ part generates (ii) and (iii). What is missing from the above groups is the group corresponding to the set of IRSs that contain u and have a span that partially overlaps with $s_u{:}e_u$. The claim is that this cannot happen. Consider an IRS $S_j = (V_j, s_j{:}e_j) \in H(S_i, V_f)$ that contain u and without loss of generality, assume that $s_u < s_j < e_u < e_j$. Since we are dealing with induced subgraphs and stable topologies (i.e., from the definition of an IRS), the connectivity of u to the vertices in V_i remains the same during the span of $s_u{:}e_u$ and also during the span of $s_j{:}e_j$, which means that the connectivity of u to the vertices in V_i remains the same during the entire span of $s_u{:}e_j$. This is a contradiction, since $(u, s_u{:}e_u) \in \text{irsadj}(S_i, V_f)$ and as such is a maximal length span of stable relations due to the fact that $(V_i \cup \{u\}, s_u{:}e_u)$ is an IRS and the span of an IRS is maximal. Thus, the two cases of the third condition in Eq. (2) cover all possible cases and it correctly generate the complete set of IRSs. Also, since each recursive call modifies at least the set V_f, none of the recursive calls lead to overlapping subproblems, ensuring that each IRS is only generated once.

The high-level structure of the IRS enumeration algorithm is shown in Algorithm 1. A relational state S_i, the set of available vertices V_f that are not in S_i, and a list of all identified IRSs $\mathscr{S}$ are passed to *enumerate* to incrementally grow S_i. To generate all IRSs, the enumeration process starts with enumerate($\emptyset$,V_f, $\mathscr{S}$) where V_f includes all vertices for $\mathscr{N}^{\phi}$. Lines 1–6 initiate separate enumeration for each vertex. Line 7 selects an adjacent vertex of S_i from V_f. If no adjacent vertex is found,

Algorithm 1 enumerate(S_i, V_f, $\mathscr{S}$)

```
 1: if S_i is ∅ then
        for each vertex v in V_f do
 2:         S_i ← construct a relational state using v and its span
 3:         remove v from V_f
 4:         enumerate(S_i, V_f, 𝒮)
 5:     return
 6: v ← select an adjacent vertex of S_i from V_f
 7: if there is no v found for expansion of S_i then
 8:     return
 9: rslist ← detect all relational states by including v in S_i
    for each relational state s in rslist do
10:     if s contains at least minimum number of required vertices then
11:         add s to 𝒮
12: v' ← update v's span sequence by removing the span of each relational state in rslist
13: if v' is not empty then
14:     replace v by v' in V_f
15: else
16:     remove v from V_f
17: enumerate(S_i, V_f, 𝒮)
18: if v' is not empty then
19:     remove v' from V_f
    for each relational state s in rslist do
20:     if s can be expanded then
21:         enumerate(s, V_f, 𝒮)
22: add v to V_f
23: return
```

recursion terminates (Lines 8–9). Once the vertex v is identified, a list of new IRS *rslist* is constructed by including v in S_i. This is the step (Line 10) in which multiple IRSs are generated. For each new relational state s in *rslist* (Lines 11–13), if s satisfies the minimum size requirements, it is recorded as part of $\mathscr{S}$. Then v's span sequence is updated by removing the span of each IRS in *rslist* and stored as v'. v is removed from V_f and v' is added to V_f (Lines 14–18). At this point (Line 19), the algorithm recursively grows S_i with updated V_f. When the recursion completes enumerating S_i, v' is removed from V_f (Lines 20–21). For each new relational state s in *rslist* (Lines 22–24), if s can be expanded further, a new recursion is started with s being the IRS and V_f. When the recursion returns(Line 25), v is added back to V_f to restore original list of V_f. When the initial call to *enumerate* terminates, the list $\mathscr{S}$ contains all qualified IRSs.

Handling Directed Edges

To handle directed edges, we consider each direction of an edge separately, such that a directed edge $a \rightarrow b$ is listed separately from $a \leftarrow b$. The direction of an edge is stored as part of the label along with the span sequence of that edge. The

Algorithm 2 mpm($G^{RS}, u, t, d[], p$)

```
1:  /* u is the current node */
2:  /* t is the current time */
3:  /* d[] is the discovery array */
4:  /* p is the current path */
5:  d[u] = t++
6:  push u into p
7:  if adj(u) = ∅ and |p| > minimum EIRS-length then
8:      record p
9:  else
        for each node v in (adj(u) sorted in increasing end-time order) do
10:         if d[v] < d[u] then
11:             mpm(G^RS, v, t, d, p)
12: pop p
```

direction of an edge at a certain span is coded as 0, 1 or 2 to represent $a \rightarrow b$, $a \leftarrow b$ or $a \leftrightarrow b$. Note that the ordering of the vertices in an edge (a, b) are stored in increasing vertex-number order (i.e, $a < b$). Using the above representation of an edge, we determine the direction of all the edges of an IRS during its span duration.

3.2.2 Step 2: Mining of Maximal Evolution Paths

The algorithm that we developed to identify the sequence of IRSs that correspond to the maximal EIRSs is based on a modified DFS traversal of a directed acyclic graph that is referred to as the *induced relational state graph* and will be denoted by G^{RS}. G^{RS} contains a node for each of the discovered IRSs and a virtual root node r which is connected to all nodes. For each pair of IRSs $S_i = (V_i, s_i : e_i)$ and $S_j = (V_j, s_j : e_j)$, G^{RS} contains a directed edge from the node corresponding to S_i to the node corresponding to S_j iff $V_i \neq V_j$, $|V_i \cap V_j|/|V_i \cup V_j| \geq \beta$ and $e_i < s_j$ (i.e., constraints (ii)–(iv) of Definition 1).

The algorithm, referred to as *mpm* (Maximal Path Miner), uses a discovery array $d[]$ to record the discovery times of each node during traversal and all discovery times are initially set to -1. The traversal starts from the root node and proceeds to visit the rest of the nodes. Given a node u, the *mpm* algorithm selects among its adjacent nodes the node v that has the earliest end-time and $d[v] < d[u]$. The *mpm* algorithm also keeps track of the current path from the root to the node that it is currently at. If that node (i.e., node v) has no outgoing edges, then it outputs that path (i.e., u, v). The sequence of the relational states corresponding to the nodes of that path (the root node is excluded) represents an EIRS. The pseudocode is shown in Algorithm 2.

A close inspection of the *mpm* algorithm reveals that it is similar in nature to a traditional depth-first traversal with two key differences. First, the adjacent nodes of each node are visited in non-decreasing end-time order (line 10) and second, the

condition on line 11 prevents the traversal of what are essentially forward edges in the depth-first tree but allows for the traversal of cross edges. To see that the *mpm* algorithm generates the complete set of maximal paths in a non-redundant fashion (i.e., each maximal path is output once), it is sufficient to consider the following. First, *mpm* without the condition on line 11 will generate all paths and each path will be generated once and it will terminate. This follows directly from the fact that G^{RS} is a directed acyclic graph. Second, the condition on line 11 eliminates paths that are contained within another path. To see this, consider the case in which while exploring the nodes adjacent to u, a vertex $v \in \mathrm{adj}(u)$ is encountered such that $d[v] > d[u]$. The fact that $d[v] > d[u]$ indicates that vertex v was encountered from another path from u that traversed a vertex $v' \in \mathrm{adj}(u)$ that was explored earlier. Thus, there is a path $p = u \rightarrow v' \rightsquigarrow v$ whose length is greater than one edge. As a result, the length of all paths that contain the edge $u \rightarrow v$ can be increased by replacing the edge with p. Third, the paths generated are maximal. Let $p_1 = (u_{i_1} \rightsquigarrow u_{i_j} \rightarrow u_{i_{j+1}} \rightsquigarrow u_{i_k})$ be a path discovered by *mpm* and assume that is not maximal. Without loss of generality, let $p_2 = (u_{i_1} \rightsquigarrow u_{i_j} \rightarrow u_{i_{j'}} \rightsquigarrow u_{i_{j+1}} \rightsquigarrow u_{i_k})$ be the maximal path that contains p_1 as a sub-path. Since path p_2 contains $u_{i_{j'}}$ before $u_{i_{j+1}}$, the end-time of $u_{i_{j'}}$ has to be less than the end-time of $u_{i_{j+1}}$. While exploring the nodes adjacent to u_{i_j}, the *mpm* algorithm would have selected $u_{i_{j'}}$ prior to $u_{i_{j+1}}$ (since adjacent nodes are visited in increasing end-time order) and in the course of the recursion would have visited $u_{i_{j+1}}$ from $u_{i_{j'}}$. Consequently, when $u_{i_{j+1}}$ is considered at a later point as an adjacent node of u_{i_j}, $d[u_{i_{j+1}}] > d[u_{i_j}]$ will prevent *mpm* from any further exploration. Thus, *mpm* will not generate the non-maximal path $(u_{i_1} \rightsquigarrow u_{i_j} \rightarrow u_{i_{j+1}} \rightsquigarrow u_{i_k})$.

Selective Materialization

The discussion so far assumes that G^{RS} has been fully materialized. However, this can be expensive as it requires pairwise comparisons between a large number of IRSs in order to determine if they satisfy constraints (ii)–(iv) of Definition 1. For this reason, the *mpm* algorithm materializes the portions of G^{RS} that it needs during the traversal. This allows it to reduce the rather expensive computations associated with some of the constraints by not having to visit forward edges. Moreover it utilizes the minimum EIRS length constraint to further prune the parts of G^{RS} that it needs to generate.

For a given node u, the *mpm* algorithm needs the adjacent node of u that has the earliest end-time. Let e_u be the end-time of node u. Since a node (i.e., an IRS) is required to have at least a span of length ϕ, we start u's adjacent node search among the nodes in G^{RS} that have the end-time of $e_k = e_u + \phi$. According to constraint (iv) of Definition 1, a certain minimum threshold of similarity is desired between two IRSs of an EIRS. Thus, it is sufficient to compare u with only those nodes that have at least a common vertex with u. We index the nodes of G^{RS} based on the vertices so that the similar nodes of u can be accessed by looking up all the nodes that have

at least one vertex in common with u. If a node v is similar to u and $d[v] < d[u]$, we add the node to the adjacency list. If the search fails to detect any adjacent node, we initiate another search looking for nodes that have an end-time of $e_k + 1$ and continue such incremental search until an adjacent node of u is found or all possible end-times have been explored.

4 Mining the Coevolving Relational Motifs

Computational methods and tools that can efficiently and effectively analyze the temporal changes in dynamic complex relational networks enable us to gain significant insights regarding the relations between the entities and how these relations have evolved over time. Such *coevolving* patterns can provide evidence to the existence of, possibly unknown, coordination mechanisms by identifying the relational motifs that evolve in a similar and highly conserved fashion.

4.1 Coevolving Relational Motifs

Coevolving relational motifs are designed to identify the relational patterns that change in a consistent way over time. An example of this type of conservation is illustrated in Fig. 6, in the context of a hypothetical country-to-country trading network where labels represent the commodities being traded. The network for 1990 shows a simple relational motif (M_1) between pairs of nodes that occurs four times (shaded nodes and solid labeled edges). This relational motif has evolved in the network for 2000 in such a way so that in all four cases, a new motif (M_2) that includes an additional node has emerged. Finally, in the network for 2005 we see that three out of these four occurrences have evolved to a new motif (M_3) that now involves four nodes. This example shows that the initial relational motif among the four sets of nodes has changed in a fairly consistent fashion over time (i.e., it *coevolved*) and such a sequence of motifs $M_1 \rightsquigarrow M_2 \rightsquigarrow M_3$ represents an instance of a CRM.

CRMs identify consistent patterns of relational motif evolution that can provide valuable insights on the processes of the underlying networks. For example, the CRM of Fig. 6 captures the well-known phenomenon of production specialization due to economic globalization, in which the production of goods have been broken down into different components performed by different countries [11]. Similarly, CRMs in health-care networks can capture how the set of medical specialties required to treat certain medical conditions have changed over the years, in communication networks CRMs can capture the evolution of themes being discussed among groups of individuals as their lifestyles change, whereas CRMs in corporate email networks can capture how the discussion related to certain topics moves through the companies' hierarchies.

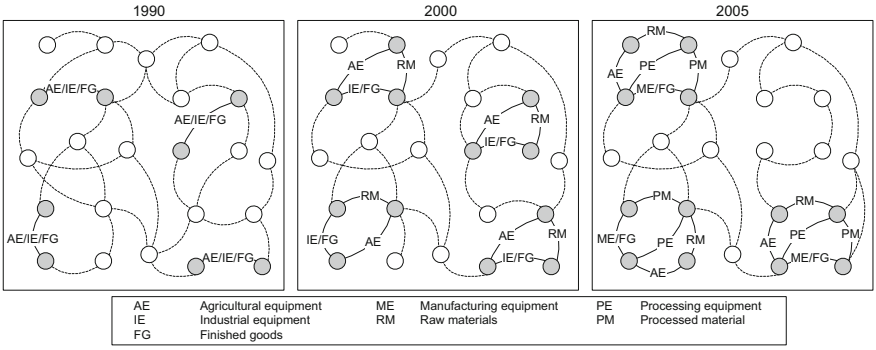


Fig. 6 An example of a coevolving relational motif in the context of a hypothetical country-to-country trading network where labels represent the commodities being traded

The formal definition of a CRM that is used in this chapter is as follows:

Definition 2. A CRM of length m is a tuple $\{N, \langle M_1, \ldots, M_m \rangle\}$, where N is a set of vertices, m is the number of relational motifs, and each relational motif $M_j = (N_j, A_j, L_{N_j}, L_{A_j})$ is defined over a subset of vertices N if there is an injection ξ_j from N_j to N.

4.1.1 CRM Embedding/Occurrence

An m-length CRM occurs in a dynamic network $\mathcal{N}$ whose node set is V, if there is a sequence of n snapshots $\langle G_{i_1}, G_{i_2}, \ldots, G_{i_n} \rangle$, where $m \leq n$, and a subset of vertices B of V (i.e., $B \subseteq V$) such that:

(1) there is a bijection ξ from N to B
(2) the injection $\xi \circ \xi_j$ is an embedding of M_j in G_{i_j}
(3) there is no embedding of M_j via the injection $\xi \circ \xi_j$ in $G_{i_{j+1}}$ or no embedding of M_{j+1} via the injection $\xi \circ \xi_{j+1}$ in G_{i_j}.

4.1.2 CRM Constraints

A CRM needs to satisfy the following constraints:

(1) it occurs in $\mathcal{N}$ at least ϕ times,
(2) each occurrence of the CRM in $\mathcal{N}$ uses a non-identical set of nodes,
(3) $M_j \neq M_{j+1}$, and
(4) $|N_j| \geq \beta |N|$ where $0 < \beta \leq 1$.

Note that the third condition of the CRM embedding in the above definition is designed to ensure that for each pair of successive motifs at least one of them is not

supported by the snapshot-nodes pair that supported the other motif. This is done to ensure that there is a relational change between the nodes associated with those embeddings in each others snapshot.

The purpose of the CRM constraints in Definition 2 is as follows: First, a CRM *occurs* in the dynamic network when there exists an embedding of that CRM (i.e., an embedding of the evolving motif sequence) in to the dynamic network. Thus, the number of times a CRM occurs in a dynamic network can be used as a metric to evaluate the significance of a certain evolving pattern. For example, the number of occurrences of the CRM in Fig. 6 is 3. The parameter ϕ is used to eliminate sequences of evolving motifs that are not frequent enough to indicate the existence of an underlying process driving these changes. Second, two occurrences of a CRM are considered to use non-identical sets of nodes if their motifs use distinct embeddings in the snapshots that support them. This constraint is used to eliminate CRMs that converge to the same sets of nodes for a particular motif (i.e., CRMs sharing embeddings). The third constraint of Definition 2 limits the discovered CRMs to only an evolving sequence of motifs and not a sequence that remains the same. Finally, the parameter β is used to control the degree of change between the sets of nodes involved in each motif of a CRM and enforces a minimum node overlap among all motifs of CRM. Note that the frequent dynamic subgraphs introduced by Borgwardt et al. [5] correspond to CRMs in which the snapshots supporting each set of nodes are restricted to be consecutive and $\beta = 1$.

An edge of a CRM captures the relational changes between two nodes over time. Given a pair of vertices u and v of a CRM, the edge (u, v) may change label over time or may not be present in all motifs of the CRM. The number of edges in a CRM can be thought as the union of all edges between the nodes of the CRM. For example, an edge of the CRM captured in Fig. 6 is $AE/IE/FG \rightsquigarrow IE/FG \rightsquigarrow ME/FG$. Also, the total number of edges of the CRM is 5, since the last motif contains five edges between the four shaded nodes and all edges of previous motifs are earlier versions of the edges existing in the last motif.

In this chapter we focus on developing an efficient algorithm to mine a subclass of the CRMs, such that in addition to the conditions mentioned in Definition 2, the motifs that make up the CRM, also share at least one edge that itself is a CRM. Formally, we focus on identifying the CRMs $c = \{N_c, \langle M_1, \ldots, M_m \rangle\}$ that contain at least a pair of vertices $\{u, v\} \in N_C$ such that each induced subgraph M'_i of M_i on $\{u, v\}$ is connected and $x = \{\{u, v\}, \langle M'_1, \ldots, M'_m \rangle\}$ is a CRM. A CRM like x that contains only one edge and two vertices will be called an *anchor*. We focus our CRM enumeration problem around the anchors, since they ensure that the CRM's motifs contain at least a pair of nodes in common irrespective of the specified overlap constraint that evolves in a conserved way. It also characterizes how the network around these core set of entities coevolved with them. In addition, we enforced the anchor constraint to simplify the enumeration problem and handle the exponential complexity of the CRM enumeration in various dynamic networks. We will refer to the class of CRMs that contain an anchor as *anchored CRMs*. For the rest of the discussion, any references to a CRM will assume it is an anchored CRM.

Given the above definition, the work in this chapter is designed to develop efficient algorithm for solving the following problem:

Problem 2. Given a dynamic network $\mathcal{N}$ containing T snapshots, a user defined minimum support ϕ ($1 \leq \phi$), a minimum number of edges $k_{\min}$ per CRM, and a minimum number of motifs $m_{\min}$ per CRM, find all CRMs such that the motifs that make up the CRM, also share an anchor CRM.

A CRM that meets the requirements specified in Problem 2 is referred to as a *frequent* CRM and it is *valid* if it also satisfies the minimum node overlap constraint (Definition 2(iv)).

4.2 Finding Coevolving Relational Motifs

A consequent of the way anchored CRMs are defined is that the number of motifs that they contain is exactly the same as the number of motifs that exist in the anchor(s) that they contain. As a result, the CRMs can be identified by starting from all the available anchors and try to grow them by repeatedly adding edges as long as the newly derived CRMs satisfy the constraints and have exactly the same number of motifs as the anchor. Since a CRM can contain more than one anchor, this approach may identify redundant CRMs by generating the same CRM from multiple anchors. Therefore, the challenge is to design a strategy that is complete and non-redundant. To achieve this, we develop an approach that generates each CRM from a unique anchor. Given a CRM, the anchor from which it will be generated is referred to as its *seed anchor*.

The algorithm that we developed, named CRMminer, for finding all non-redundant valid CRMs (Problem 2) initially identifies all frequent anchors and then performs a depth-first exploration of each anchor pattern space along with a canonical labeling that is derived by extending the ideas of the minimum DFS code [24] to the case of CRMs for redundancy elimination. We impose frequency-based constraints by stopping any further exploration of a CRM when the pattern does not occur at least ϕ times in the dynamic network $\mathcal{N}$.

4.2.1 CRM Representation

A CRM $c = \{N, \langle M_1, \ldots, M_m \rangle\}$ is represented as a graph $G_c = (N, E_c)$, such that an edge $(u, v) \in E_c$ is a five-item tuple $(u, v, \boldsymbol{l_u}, \boldsymbol{l_{u,v}}, \boldsymbol{l_v})$, where $u, v \in N$, the vectors $\boldsymbol{l_u}$ and $\boldsymbol{l_v}$ contain the vertex labels and $\boldsymbol{l_{u,v}}$ contains the edge labels of all motifs. If the CRM consists of m motifs, then $\boldsymbol{l_u} = \langle l_{u_1}, \ldots, l_{u_m} \rangle$, $\boldsymbol{l_v} = \langle l_{v_1}, \ldots, l_{v_m} \rangle$ and $\boldsymbol{l_{u,v}} = \langle l_{u_1,v_1}, \ldots, l_{u_m,v_m} \rangle$. The kth entry in each vector $\boldsymbol{l_u}$, $\boldsymbol{l_{u,v}}$, and $\boldsymbol{l_v}$ records the connectivity information among the vertices of the kth motif (M_k). If an edge (u, v) is part of motif M_k, then the kth entry of $\boldsymbol{l_u}$, $\boldsymbol{l_v}$, and $\boldsymbol{l_{u,v}}$ are set to the labels of u and v vertices, and the label of the (u, v) edge, respectively. If both vertices u and v are

part of motif M_k, but the (u, v) edge is not, or at least one of the vertices u or v is not part of the M_k motif (i.e., no (u, v) edge is possible), then ω is inserted at the kth entry of the $\boldsymbol{l}_{u,v}$ to capture the disconnected state. Similarly, if u or v does not have any incident edges in the M_k motif (i.e., the vertices are not present in that motif), then ω is added as the vertex label at the kth entry of $\boldsymbol{l}_u$ or $\boldsymbol{l}_v$. Note that the value of ω is lexicographically greater than the maximum edge and vertex label.

This representation is illustrated in Fig. 7. The CRM consists of three motifs $\langle M_1, M_2, M_3 \rangle$ and represents relations among vertices $N=\{v_0, v_1, v_2, v_3\}$ using five edges. The edge between (v_0, v_1) exists in all three motifs capturing changes in relation as the edge label changes from $AE/IE/FG \rightsquigarrow IE/FG \rightsquigarrow ME/FG$. It is represented as $\boldsymbol{l}_{v_0}=\langle BEL, BEL, BEL \rangle$, $\boldsymbol{l}_{v_1}=\langle NTH, NTH, NTH \rangle$, and $\boldsymbol{l}_{v_0,v_1}=\langle AE/IE/FG, IE/FG, \ ME/FG \rangle$. The next edge between (v_1, v_2) appears in two motifs and the label vectors are represented as $\boldsymbol{l}_{v_1}=\langle NTH, NTH, NTH \rangle$, $\boldsymbol{l}_{v_2}=\langle \omega, GDR, GDR \rangle$, and $\boldsymbol{l}_{v_0,v_1}=\langle \omega, RM, \ PM \rangle$. Following similar process, we can represent edges (v_2, v_0), (v_2, v_3), and (v_3, v_0).

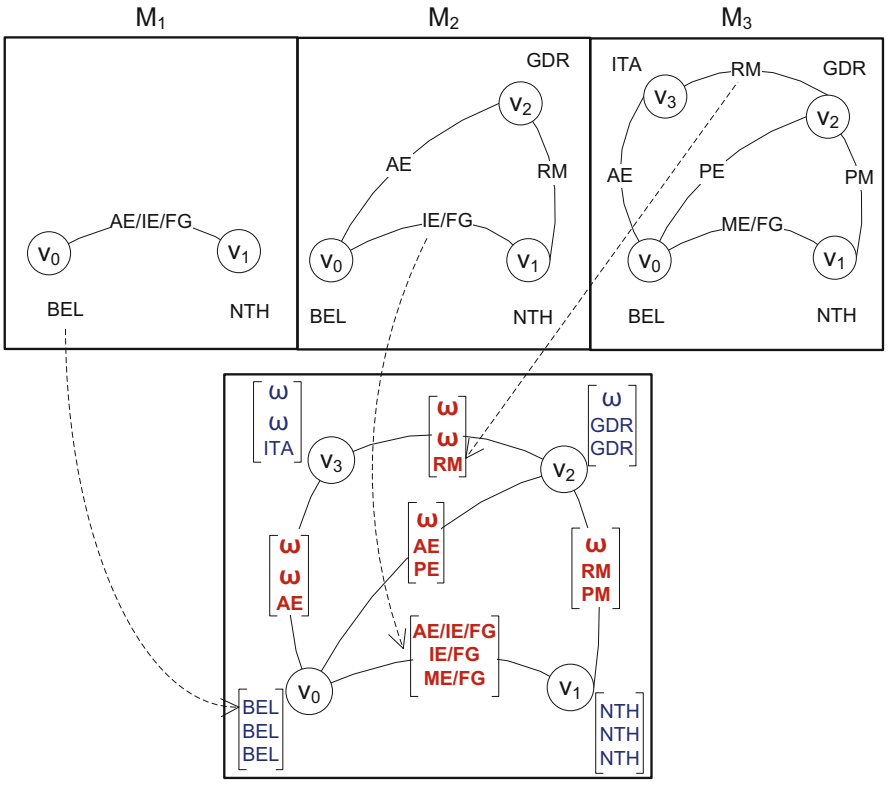


Fig. 7 A CRM representation. The CRM c consists of three motifs $\langle M_1, M_2, M_3 \rangle$ and represents relations among vertices $N = \{v_0, v_1, v_2, v_3\}$ using five edges. G_c (*bottom graph*) shows the CRM representation capturing vertex and edge label vectors

4.2.2 Mining Anchors

The search for CRMs is initiated by locating the frequent anchors that satisfy the CRM definition and the restrictions defined in Problem 2. This is done as following: Given a dynamic network $\mathscr{N}$, we sort all the vertices and edges by their label frequency and remove all infrequent vertices and edges. Remaining vertices and edges are relabeled in decreasing frequency. We determine the span sequences of each edge and list every edge's span sequence if that sequence contains at least a span with an edge label that is different from the rest of the spans. At this point, we use the sequential pattern mining technique prefixSpan [19] to determine all frequent span sequences. Since the frequent sequences can be partial sequences of the original input span sequences, it is not guaranteed that they all contain consecutive spans with different labels. Thus, the frequent sequences that contain different consecutive spans in terms of label are considered as the anchors. The number of spans in a frequent span sequence corresponds to the total number of motifs in the anchor.

4.2.3 CRM Enumeration

Given an anchor c, we generate the set of desired CRMs by growing the size of the current CRM one edge at a time following a depth-first approach. To ensure that each CRM is generated only once in the depth-first exploration, we use an approach similar to the gSpan algorithm [24], which we have extended for the problem of CRM mining.

gSpan explores the frequent pattern lattice in a depth-first fashion. The pattern lattice is represented as a hierarchical search space where each node corresponds to a connected frequent pattern, the highest node being an empty pattern (i.e., a single vertex), the next level nodes represent one-edge patterns, and so on. The nth level nodes, which represent n-edge patterns, contain one more edge than the corresponding $(n-1)$ level nodes. To ensure that each frequent pattern in this lattice is visited exactly once, gSpan's exploration amounts to visiting the nodes of the lattice (i.e., frequent patterns) by traversing a set of edges that form a spanning tree of the lattice. This spanning tree is defined by assigning to each node of the lattice a canonical label, called the minimum DFS code. A DFS code of a graph is a unique label that is formed based on the sequence of edges added to that node during a depth-first exploration. The minimum DFS code is the DFS code that is lexicographically the smallest. Given this canonical labeling, the set of lattice edges that are used to form the spanning tree correspond to the edges between two successive nodes of the lattice (parent and child) such that the minimum DFS code of the child can be obtained by simply appending the extra edge to the minimum DFS code of the parent. For example, given a DFS code $\alpha = \langle a_0, a_1, \cdots, a_m \rangle$, a valid child DFS code is $\gamma = \langle a_0, a_1, \ldots, a_m, b \rangle$, where b is the new edge. This spanning tree guarantees that each node has a unique parent and all nodes are

connected [24]. To efficiently grow a node, gSpan generates the child nodes by only adding those edges that originate from the vertices on the rightmost path of the DFS-tree representation of the parent node. It then checks whether the resulting DFS code of the child node corresponds to the minimum DFS code. Construction of the child nodes generated by adding other edges (i.e., not from the rightmost path) is skipped, since such child nodes will never contain a DFS code that corresponds to the minimum DFS code.

In order to apply the ideas introduced by gSpan to the problem of efficiently mining CRMs, we need to develop approaches for (1) representing the DFS code of a CRM, (2) ordering the DFS codes of a CRM using the DFS lexicographic ordering, (3) representing the minimum DFS code of a CRM to use as the canonical label, and (4) extending a DFS code of a CRM by adding an edge. Once properly defined, the correctness and completeness of frequent CRM enumeration follows directly from the corresponding proofs of gSpan.

DFS Code of a CRM

In order to derive a DFS code of a CRM, we need to develop a way of ordering the edges. Given a CRM c, represented as a graph $G_c = (N, E_c)$, we perform a depth-first search in G_c to build a DFS tree T_c. The vertices (N) are assigned subscripts from 0 to $n-1$ for $|N| = n$ according to their discovery time. The edges (E_c) are grouped into two sets: the forward edge set contains all edges in the DFS tree and denoted as $E_{T_c,fw} = \{(v_i, v_j) \in E_c |\ i < j\}$, and the backward edge set contains all edges which are not in the DFS tree and denoted as $E_{T_c,bw} = \{(v_i, v_j) \in E_c |\ i > j\}$. Let us denote a partial order on $E_{T_c,fw}$ as $\prec_{T_c,fw}$, a partial order on $E_{T_c,bw}$ as $\prec_{T_c,bw}$, and a partial order on E_c as $\prec_{T_c,bw+fw}$. Given two edges $e_1 = (v_{i_1}, v_{j_1})$ and $e_2 = (v_{i_2}, v_{j_2})$, the partial order relations are defined as:

(a) $\forall e_1, e_2 \in E_{T_c,fw}$, if $j_1 < j_2$, then $e_1 \prec_{T_c,fw} e_2$.
(b) $\forall e_1, e_2 \in E_{T_c,bw}$, if $i_1 < i_2$ or ($i_1 = i_2$ and $j_1 < j_2$), then $e_1 \prec_{T_c,bw} e_2$.
(c) $\forall e_1 \in E_{T_c,bw}$ and $\forall e_2 \in E_{T_c,fw}$, if $i_1 < j_2$, then $e_1 \prec_{T_c,bw+fw} e_2$.
(d) $\forall e_1 \in E_{T_c,fw}$ and $\forall e_2 \in E_{T_c,bw}$, if $j_1 \leq i_2$, then $e_1 \prec_{T_c,bw+fw} e_2$.

The combination of the three partial orders defined above enforces a linear order $\prec_{T_c,E_c}$ on E_c.

Given this linear order $\prec_{T_c,E_c}$, we can order all edges in G_c and construct an edge sequence to form a DFS code of a CRM, denoted as code(c, T_c). An edge of the DFS code of a CRM is represented similar to the CRM edge definition and uses a 5-tuple representation $(i, j, \boldsymbol{l_i}, \boldsymbol{l_{i,j}}, \boldsymbol{l_j})$, where i and j are the DFS subscripts (i.e., the discovery time) of the vertices, and $\boldsymbol{l_i}$, $\boldsymbol{l_j}$, and $\boldsymbol{l_{i,j}}$ are the label vectors of the vertices and edge, respectively. The kth entry in each vector $\boldsymbol{l_i}$, $\boldsymbol{l_j}$, and $\boldsymbol{l_{i,j}}$ contains the labels of vertices and edges of motif M_k.

For example, Fig. 8 presents two CRMs and their corresponding DFS codes. The DFS codes are listed below to show the sequence of edges and their differences (i.e.,

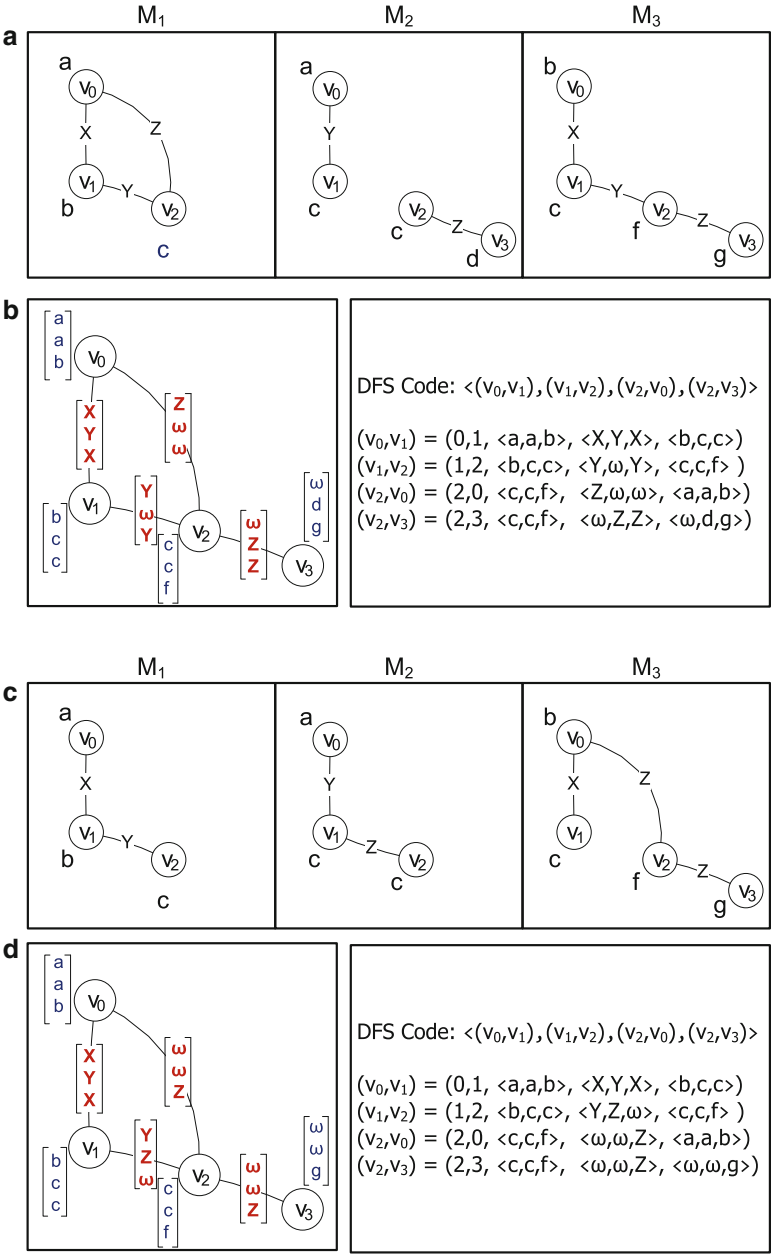


Fig. 8 DFS code for two CRMs represented as a sequence of edges (v_0, v_1), (v_1, v_2), (v_2, v_0), and (v_2, v_3). (**a**) Presents CRM C_1 consisting of three motifs, four vertices, and four edges. (**b**) Presents G_{C_1} and the corresponding DFS code for CRM C_1. (**c**) Presents CRM C_2 consisting of three motifs, four vertices, and four edges. (**d**) Presents G_{C_2} and the corresponding DFS code for CRM C_2

CRM C_1—Fig. 8a	CRM C_2—Fig. 8c
$\langle$(0, 1, ⟨a, a, b⟩, ⟨X, Y, X⟩, ⟨b, c, c⟩),	$\langle$(0, 1, ⟨a, a, b⟩, ⟨X, Y, X⟩, ⟨b, c, c⟩),
(1, 2, ⟨b, c, c⟩, ⟨Y, ω, Y⟩, ⟨c, c, f⟩),	(1, 2, ⟨b, c, c⟩, ⟨Y, Z, ω⟩, ⟨c, c, f⟩),
(2, 0, ⟨c, c, f⟩, ⟨Z, ω, ω⟩, ⟨a, a, b⟩),	(2, 0, ⟨c, c, f⟩, ⟨ω, ω, Z⟩, ⟨a, a, b⟩),
(2, 3, ⟨c, c, f⟩, ⟨ω, Z, Z⟩, ⟨ω, d, g⟩)$\rangle$	(2, 3, ⟨c, c, f⟩, ⟨ω, ω, Z⟩, ⟨ω, d, g⟩)$\rangle$

the edge labels): Note that the kth entry of a vertex and edge label vector of a DFS code is filled with ω if the corresponding vertex or edge is not present in motif M_k.

To maintain the edge ordering of a DFS code similar to gSpan, our algorithm follows the DFS code's neighborhood restriction rules [24]. Given a DFS code of a CRM code(c, T_c), assume two neighboring edges $a_k = (i_k, j_k, \boldsymbol{l}_{i_k}, \boldsymbol{l}_{i_k,j_k}, \boldsymbol{l}_{j_k})$ and $a_{k+1} = (i_{k+1}, j_{k+1}, \boldsymbol{l}_{i_{k+1}}, \boldsymbol{l}_{i_{k+1},j_{k+1}}, \boldsymbol{l}_{j_{k+1}})$. Then, a_k and a_{k+1} must meet the following rules:

(a) If $a_k \in E_{T_c,bw}$, then one of the following must be true:

(1) when $a_{k+1} \in E_{T_c,fw}$, $i_{k+1} \leq i_k$ and $j_{k+1} = i_{k+1}$.
(2) when $a_{k+1} \in E_{T_c,bw}$, $i_{k+1} = i_k$ and $j_k < j_{k+1}$.

(b) If $a_k \in E_{T_c,fw}$, then one of the following must be true:

(1) when $a_{k+1} \in E_{T_c,fw}$, $i_{k+1} \leq j_k$ and $j_{k+1} = j_k + 1$.
(2) when $a_{k+1} \in E_{T_c,bw}$, $i_{k+1} = j_k$ and $j_{k+1} < i_k$.

Note that the above stated rules ensure that the edge (v_2, v_0) appears before the edge (v_2, v_3) in Fig. 8b, d.

DFS Lexicographic Ordering

To establish a canonical labeling system for a CRM, CRMminer defines the DFS lexicographical ordering based on the CRM's DFS code definition. The linear ordering is defined as follows. Let two DFS codes of a CRM consisting of m motifs be $\alpha = \text{code}(c_\alpha, T_\alpha) = \langle e_\alpha^0, e_\alpha^1, \ldots, e_\alpha^p \rangle$ and $\delta = \text{code}(c_\delta, T_\delta) = \langle e_\delta^0, e_\delta^1, \ldots, e_\delta^q \rangle$, where p and q are the number of edges in α and δ, and $0 \leq p, q$. Let the forward and backward edge set for T_α and T_δ be $E_{\alpha,fw}$, $E_{\alpha,bw}$, $E_{\delta,fw}$, and $E_{\delta,bw}$, respectively. Also, let $e_\alpha^x = (i_\alpha^x, j_\alpha^x, \boldsymbol{l}_{i_\alpha^x}, \boldsymbol{l}_{i_\alpha^x,j_\alpha^x}, \boldsymbol{l}_{j_\alpha^x})$ and $e_\delta^y = (i_\delta^y, j_\delta^y, \boldsymbol{l}_{i_\delta^y}, \boldsymbol{l}_{i_\delta^y,j_\delta^y}, \boldsymbol{l}_{j_\delta^y})$ be two edges, one in each of the α and δ DFS codes. We define a check $PA(e)$ to determine whether a CRM edge is a potential anchor edge that contains no ω label and changes between consecutive motifs with respect to its edge/vertex labels. Now, $e_\alpha^x < e_\delta^y$, if any of the following is true:

(1) $F_\omega(\boldsymbol{l}_{i_\alpha^x,j_\alpha^x}) \leq F_\omega(\boldsymbol{l}_{i_\delta^y,j_\delta^y})$, where $F_\omega(\boldsymbol{A})$ is the number of ω's in vector $\boldsymbol{A}$, or
(2) $PA(e_\alpha^x)$ = true and $PA(e_\delta^y)$ = false, or
(3) $e_\alpha^x \in E_{\alpha,bw}$ and $e_\delta^y \in E_{\delta,fw}$, or
(4) $e_\alpha^x \in E_{\alpha,bw}$, $e_\delta^y \in E_{\delta,bw}$, and $j_\alpha^x < j_\delta^y$, or

(5) $e_\alpha^x \in E_{\alpha,bw}, e_\delta^y \in E_{\delta,bw}, j_\alpha^x = j_\delta^y$ and $\boldsymbol{l}_{i_\alpha^x,j_\alpha^x} < \boldsymbol{l}_{i_\delta^y,j_\delta^y}$, or
(6) $e_\alpha^x \in E_{\alpha,fw}, e_\delta^y \in E_{\delta,fw}$, and $i_\delta^y < i_\alpha^x$, or
(7) $e_\alpha^x \in E_{\alpha,fw}, e_\delta^y \in E_{\delta,fw}, i_\alpha^x = i_\delta^y$ and $\boldsymbol{l}_{i_\alpha^x} < \boldsymbol{l}_{i_\delta^y}$, or
(8) $e_\alpha^x \in E_{\alpha,fw}, e_\delta^y \in E_{\delta,fw}, i_\alpha^x = i_\delta^y, \boldsymbol{l}_{i_\alpha^x} = \boldsymbol{l}_{i_\delta^y}$ and $\boldsymbol{l}_{i_\alpha^x,j_\alpha^x} < \boldsymbol{l}_{i_\delta^y,j_\delta^y}$, or
(9) $e_\alpha^x \in E_{\alpha,fw}, e_\delta^y \in E_{\delta,fw}, i_\alpha^x = i_\delta^y, \boldsymbol{l}_{i_\alpha^x} = \boldsymbol{l}_{i_\delta^y}, \boldsymbol{l}_{i_\alpha^x,j_\alpha^x} = \boldsymbol{l}_{i_\delta^y,j_\delta^y}$, and $\boldsymbol{l}_{j_\alpha^x} < \boldsymbol{l}_{j_\delta^y}$.

Based on the above definitions, we derive the following conditions to compare two DFS codes of a CRM. We define $\alpha \leq \delta$, iff any of the following conditions is true:

(a) $\exists t, 0 \leq t \leq \min(p, q), e_\alpha^k = e_\delta^k$ for $k < t$, and $e_\alpha^t < e_\delta^t$, or
(b) $e_\alpha^k = e_\delta^k$ for $0 \leq k \leq p$ and $p \leq q$.

Note that the label vectors (i.e., $\boldsymbol{l}_{i_\alpha^x,j_\alpha^x}$ and $\boldsymbol{l}_{i_\delta^y,j_\delta^y}$) are compared lexicographically. The DFS lexicographical ordering ranks edges with label vectors containing no ω labels higher than the edges which does. To define the relation between ω and valid vertex/edge label, the value of ω is set to a lexicographically higher value than the maximum edge and vertex label. Furthermore, we rank an anchor edge over an edge that remains same. This is important as we show later in Sect. 4.2.5.

In order to provide a detailed example of the DFS lexicographical ordering, let us compare the DFS codes of CRMs C_1 and C_2 presented in Fig. 8 following the rules presented above. For both the DFS codes, the first edge (v_0, v_1) is the same. In case of the second edge (v_1, v_2), both the DFS codes contain the same vertex label vectors. However, the edge label vectors are different and the edge label vector $\langle Y, Z, \omega \rangle$ of DFS code 2 is smaller than $\langle Y, \omega, Y \rangle$ DFS code 1. Thus, DFS code 2 is lexicographically smaller than DFS code 1.

Minimum DFS Code

A CRM can be represented by different DFS trees resulting in different DFS codes. To define a canonical label for a CRM, we select the minimum DFS code according to the DFS lexicographic order, represented by min code(c). To efficiently determine the minimum DFS code of a CRM, we start from the smallest edge according to the DFS lexicographic order. To grow the code by adding another edge, we select all adjacent edges following the rightmost extension rules and select the smallest one according to the DFS lexicographic order. Similar to simple graph isomorphism, given two CRMs c and c', c is isomorphic to c' if and only if min code(c) = min code(c'). Thus, by searching frequent minimum DFS codes, we can identify the corresponding frequent CRMs.

Pattern Lattice Growth

The difference between a simple graph pattern and a CRM is the vertex/edge label representation. One contains a single label and the other contains a label vector (i.e.,

sequence of labels including an empty label ω). The growth of a simple graph by one edge results in a number of simple graphs based on the number of unique labels of the frequent edges on the rightmost path. However, a CRM extended by an edge can generate large number of CRMs, since these are formed based on the combination of the label vectors of the frequent edges on the rightmost path.

In Fig. 9, we illustrated one edge growth of a simple graph and a CRM according to rightmost extension. Both the simple graph and the CRM are expanded by adding a frequent edge (v_2, v_3). In case of the simple graph (Fig. 9a), the original DFS code consisted sequence of two edges (v_0, v_1) and (v_1, v_2) represented as $(0, 1, a, X, b)$ and $(1, 2, b, Y, c)$. After the one edge extension, the new edge $(2, 3, c, Z, d)$ connected the new vertex v_3 to the rightmost vertex v_2. When the CRM is considered (Fig. 9c), the original DFS code consisted of the same sequence of edges $\{(v_0, v_1), (v_1, v_2)\}$ represented as: $\mathbf{(}0, 1, \langle a, a, b\rangle, \langle X, Y, X\rangle, \langle b, c, c\rangle\mathbf{)}$, $\mathbf{(}1, 2, \langle b, c, c\rangle, \langle Y, \omega, Y\rangle, \langle c, \omega, f\rangle\mathbf{)}$. Based on the combination of the label vectors of vertices v_2 and v_3, and edge (v_2, v_3), we have the following options for the (v_2, v_3) edge: $\mathbf{(}2, 3, \langle c, c, f\rangle, \langle \omega, Z, Z\rangle, \langle \omega, d, g\rangle\mathbf{)}$, $\mathbf{(}2, 3, \langle c, c, f\rangle, \langle \omega, \omega, Z\rangle, \langle \omega, \omega, g\rangle\mathbf{)}$, and $\mathbf{(}2, 3, \langle c, c, f\rangle, \langle \omega, Z, \omega\rangle, \langle \omega, d, \omega\rangle\mathbf{)}$. Note that we allow a CRM to

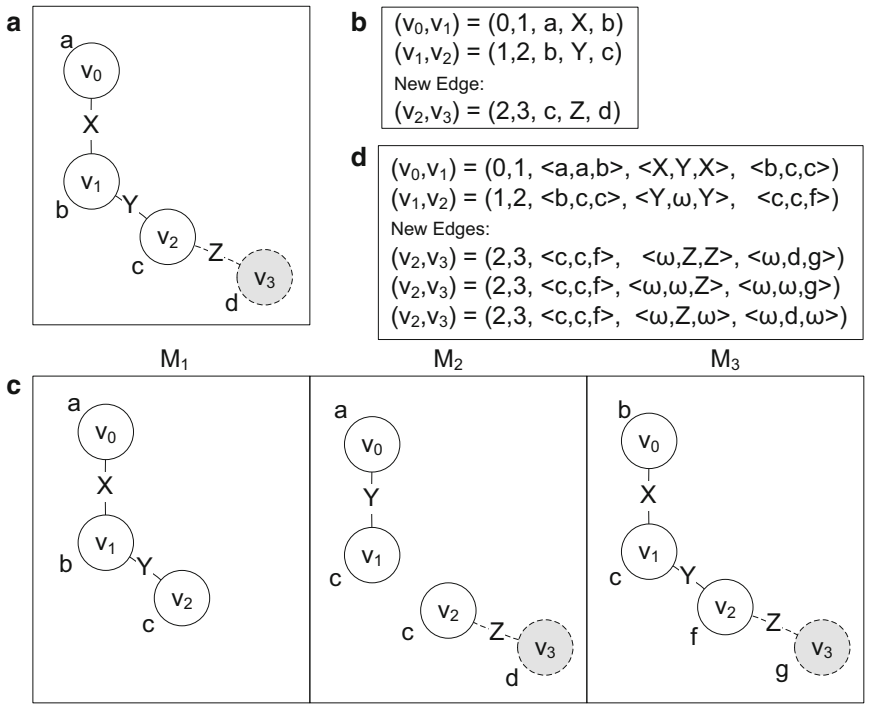


Fig. 9 Adding an edge according to rightmost extension rules. (**a**) Extending a simple graph, (**b**) DFS code of the simple graph, (**c**) extending a CRM, and (**d**) DFS code of the CRM

grow by an edge that may not be present or frequent in all motifs of that CRM to ensure complete set of the results.

To efficiently determine the frequent candidate edges during the rightmost extension, we apply the sequential pattern mining technique [19]. Even though there can be significantly more number of child CRMs from one edge extension of a CRM than a simple graph, the extended lexicographic ordering is able to order all candidates and enable us to perform pre-order search on the pattern lattice. Since traversal of the pattern lattice of a CRM remains similar to a simple graph, it allows CRMminer to prune the pattern with non-minimum DFS codes and their descendants similar to gSpan without impacting the completeness of the results.

4.2.4 CRMminer Algorithm

The high-level structure of CRMminer is shown in Algorithm 3. It first finds all frequent anchors according to Sect. 4.2.2. After locating all embeddings of an anchor c, the algorithm grows it recursively by adding one frequent adjacent edge at a time. The recursive function *ExpandCRM* first checks the support of c to prune any infrequent expansion. If c meets the minimum size ($k_{\min}$) and overlap requirement (β), it is added to the $\mathscr{C}$ to be recorded as a valid CRM. It terminates expansion when c reaches the maximum size ($k_{\max}$). To grow c further, the algorithm collects all adjacent edges of c in $\mathscr{N}$ following the DFS rightmost extension rules in Sect. 4.2.3. Each candidate edge e is added to c to construct its child c'. To prevent redundancy and ensure completeness, it only grows c' if it represents the canonical label of c', i.e., the minimum DFS code of c'.

Algorithm 3 CRMminer($\mathscr{N}$, $\mathscr{C}$)

```
C_anchor ← find all frequent anchors from 𝒩
for each c in C_anchor do
    embd_c ← all embeddings of c in 𝒩
    ExpandCRM(𝒩, 𝒞, c, embd_c)
return
```

4.2.5 Algorithm Completeness

To eliminate redundancy during the CRM expansion process, we use minimum DFS code as the canonical label and construct the pattern lattice to ensure that every node (i.e., a CRM) is connected to a unique parent and grown via single edge addition. This process ensures that each potential CRM is only explored once. To ensure that all discovered CRMs contain at least one anchor, we show how we can identify an anchor of a valid CRM. Given a valid CRM c and its canonical label (i.e., the minimum DFS code) $\min\text{code}(c) = \langle e_1, e_2, \ldots, e_k \rangle$, where e_i is an

Algorithm 4 ExpandCRM($\mathcal{N}$, $\mathcal{C}$, c, $embd_c$)

```
1: if support(c, embd_c) < φ then
2:     return
3: if size(c) ≥ k_min then
4:     if overlap(c) ≥ β then
5:         C ← C ∪ c
6:     if size(c) = k_max then
7:         return
8: E ← all frequent adjacent edges of c in N
   for each edge candidate e in E do
9:     c' ← add e to c
10:    if c' = CanonicalLabel(c') then
11:        embd_c' ← all embeddings of c' in N
12:        ExpandCRM(N, C, c', embd_c')
13: return
```

edge in the lexicographically ordered edge sequence. We claim that the first edge (e_1) of the canonical label of a CRM (c) is an anchor of that CRM. To prove this claim, assume e_1 is not an anchor and e_x is an anchor, where $1 < x \leq k$. Given the graph in CRM c, we can construct a DFS code for c that starts with e_x as the first edge. Assume, the new DFS code is represented as $\text{code}(c) = \langle e_x, e_{p_1}, \ldots, e_{p_{k-1}} \rangle$, where $\langle e_{p_1}, \ldots, e_{p_{k-1}} \rangle$ is an edge sequence containing a permutation of the edges $\{e_1, \ldots, e_k\} \setminus \{e_x\}$. Since e_1 is not an anchor, it either contains some ω labels or its labels remain same at least for some consecutive motifs of c. Based on the DFS lexicographic ordering and e_x being an anchor, $e_x < e_1$. Hence, we can state that $\langle e_x, e_{p_1}, \ldots, e_{p_{k-1}} \rangle < \langle e_1, e_2, \ldots, e_k \rangle$. This is a contradiction, since $\langle e_1, e_2, \ldots, e_k \rangle$ is the minimum DFS code of c. Thus, e_1 is an anchor of CRM c. Given that the first edge is an anchor, then CRMminer will generate that CRM by starting from the anchor and then following its rightmost extension rule to add the rest of the edges one by one.

4.2.6 Search Space Pruning

One of the challenges that any graph mining algorithm needs to handle is the exponential growth of the search space during enumeration. Traditionally, user specified constraints are used to prune the search space. To ensure discovery of the complete set of patterns, the pruning constraints need to have the anti-monotonicity property (i.e., a specific measure of a graph can never exceeds the measure of its subgraphs). For CRMminer, we use both support measure and minimum overlap constraints to prune the search space.

Minimum Support (ϕ)

To efficiently search patterns in a single large graph using a minimum support constraint, the support measure needs to guarantee the anti-monotonicity property. Bringmann [6] presented the minimum image based support measure to prune the search space in a single large graph. Given a pattern $p = (V_p, E_p, L_p)$ and a single large graph $G = (V_G, E_G, L_G)$, this measure identifies the vertex in p which is mapped to the least number of unique vertices in G and uses this number as the frequency of p in G. To formally define the support measure, let each subgraph g of G that is isomorphic to p be defined as an *occurrence* of p in G. For each occurrence g, there is a function $\varphi : V_p \rightarrow V_G$ that maps the nodes of p to the nodes in G such that (1) $\forall v \in V_p \Rightarrow L_p(v) = L_G(\varphi(v))$ and (2) $\forall (u, v) \in E_p \Rightarrow (\varphi(u), \varphi(v)) \in E_G$. The minimum image based support of a pattern p in G is defined as:

$$\sigma(p, G) = \min_{v \in V_p} | \ \{\varphi_i(v) : \ \varphi_i \text{ is an occurrence of } p \text{ in } G\} \ |. \tag{3}$$

This minimum image based support is anti-monotonic [6].

We adopted the minimum image based support measure to calculate the minimum support of a CRM in a dynamic network. As defined in Sect. 2, a dynamic network $\mathcal{N}$ can be represented as a single large graph where the nodes $\mathcal{N}$ are considered as the vertices of the large graph. Hence, it is possible to calculate the least number of unique vertices of the dynamic network that are mapped to a particular vertex of a CRM. Given a CRM $c = \{N_c, \langle M_1, M_2, \ldots, M_m \rangle\}$ in a dynamic network $\mathcal{N} = \{V_{\mathcal{N}}, \langle G_0, G_1, \ldots, G_T \rangle\}$ where $m \leq T$, the minimum image based support of c is defined as:

$$\sigma(c, \mathcal{N}) = \min_{v \in V_c} | \ \{\varphi_i(v) : \varphi_i \text{ is an occurrence of } c \text{ in } \mathcal{N}\} \ | \ . \tag{4}$$

Similar to the support measure of a pattern in a single large graph, by selecting the support of the vertex in c that has the least number of unique mapping in $\mathcal{N}$, we maintain the anti-monotonicity property.

Recall from Sect. 4.2.3 that the frequent candidate edges are identified using frequent sequence mining. Since we use the minimum image based support measure, the frequent edges detected by the sequence mining tool may not have sufficient support when calculated using such measure. Thus, we compute the minimum image based support for all candidate edges to only consider the edges that ensures the CRM extension to have sufficient minimum image based support.

Minimum Overlap (β)

Each motif of a CRM needs to contain at least a minimum percentage of the nodes from all the nodes of the CRM. This minimum node overlap threshold (defined as β in Sect. 4) controls the degree of change that is allowed between the sets of nodes

in each motif of a CRM. Given a CRM $c = \{N_c, \langle M_1, M_2, \ldots, M_m \rangle\}$ containing m motifs, the *minimum overlap* of a CRM is defined as:

$$\rho(c) = \frac{\min\limits_{1 \leq i \leq m} \{|V_{M_i}|\}}{|N_c|}, \tag{5}$$

where V_{M_i} is the set of nodes in motif M_i. Even though the minimum overlap constraint is a reasonable approach to ensure that the motifs that make the CRM are coherent, it is not anti-monotonic [26]. Thus, to generate a complete set of CRMs that meet user specified thresholds of support and overlap, we cannot prune CRMs that do not satisfy this constraint as CRMs derived from it can satisfy the constraint.

For example, let us assume the minimum overlap threshold is 60 % in Fig. 10. In step (1), motif M_1 contains two out of three nodes of the CRM (i.e., the minimum). Therefore, the minimum overlap at step (1) ($2/3 > 60\,\%$) is valid. We added vertex v_3 by including edge (v_2, V_3) to the CRM at step (2). This dropped the minimum overlap ($2/4$) below the threshold. However, in step (3), inclusion of edge (v_3, v_0) adds vertex v_3 to motif M_1. This increases the minimum overlap to be $3/4$ and makes the CRM valid again. Hence, we need to enumerate all CRMs that meet the support threshold and then search the output space for CRMs that meet the minimum overlap requirement.

To improve the performance, we developed an approximate version of our algorithm, named CRMminer$_x$, that discovers a subset of the valid CRMs (i.e., meet the constraints of Definition 2) by pruning the pattern lattice using the minimum overlap threshold. We first check whether a CRM meets the overlap threshold. If it does, we continue the enumeration process. If it does not, then we check whether any of the patterns at the previous level of the pattern lattice from which the current

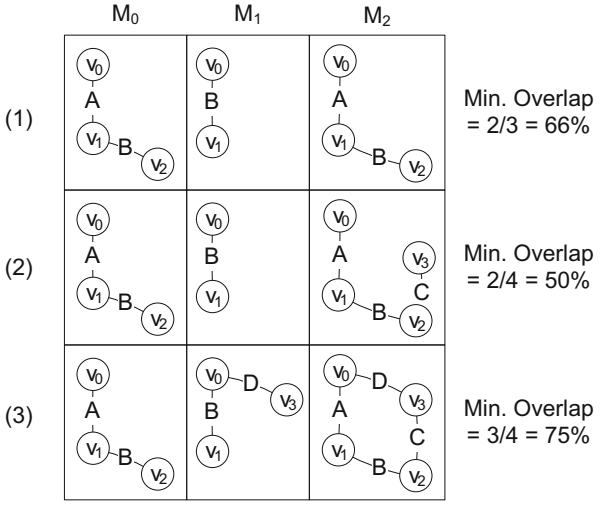


Fig. 10 Example of a CRM growth when the minimum overlap constraint is not anti-monotonic. Assume the minimum overlap constraint is 60 %

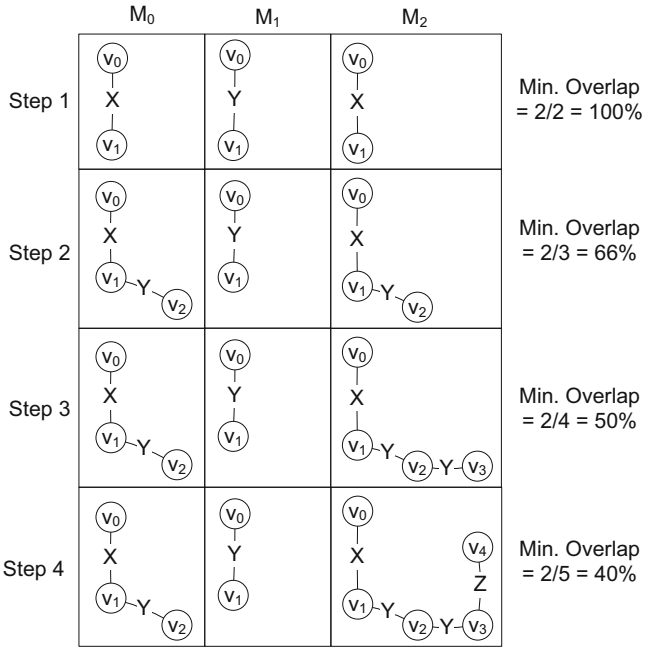


Fig. 11 Minimum overlap calculation based on (6) is used for search space pruning during the CRM enumeration process. Assuming $\beta = 60\,\%$, this CRM enumeration terminates at step 4

pattern was derived, referred to as parent CRMs, meet the overlap threshold. If at least one does, we continue enumeration. If none of the parent CRMs meet the overlap threshold, we prune that CRM.

To approximately calculate the minimum overlap of the parent CRMs, we do not generate all possible parent patterns. The parent is defined to contain one less node than the current CRM; thus, we remove a node $v \in N_c$. In such case, the parent pattern will contain $(|N_c| - 1)$ nodes and each motif M_i may contain $(|V_{M_i}| - 1)$ nodes if $v \in V_{M_i}$ or $(|V_{M_i}|)$ nodes if $v \notin V_{M_i}$. For at least one parent CRM to meet the overlap threshold, we consider the best case when $v \notin V_{M_i}$. Therefore, the minimum overlap threshold of a parent CRM c_p of the CRM $c = \{N_c, \langle M_1, M_2, \ldots, M_m \rangle\}$ is defined as:

$$\rho(c_p) = \frac{\min\limits_{1 \le i \le m} \{|V_{M_i}|\}}{|N_c| - 1}. \tag{6}$$

In Fig. 11, we illustrate the minimum overlap calculation during search space pruning. Assume the user specified $\beta = 60\,\%$. We start with the anchor at step 1 when the minimum overlap threshold is $100\,\%$. Next the edge (v_1, v_2) is added for motif M_0 and M_2 and the minimum overlap based on motif M_1 is $(2/3) = 66\,\%$. Since the β threshold is met, we continue the enumeration process. At step 3, an

edge (v_2, v_3) is added to motif M_2. Since motif M_1 contains the lowest number of nodes, the minimum overlap is $(2/4) = 50\,\%$, which does not meet the β threshold. At this point, we check whether any of the parent CRM meets the β threshold and the minimum threshold for its parent is $(2/(4-1)) = 66\,\% > \beta$. Thus, we continue the enumeration by adding an edge (v_3, v_4) to motif M_2 at step 4. The minimum overlap is $(2/5) = 40\,\%$ and its parent overlap threshold is $(2/(5-1)) = 50\,\%$. Both thresholds are lower than β, hence we stop enumerating this CRM any further.

5 Mining the Coevolving Induced Relational Motifs

Based on our work on CRMs, we realized that to understand how the relations between groups of entities have changed over time, we need to take into account all their relations. For example, the analysis of a co-authorship network to identify the popular topics of collaboration among authors can help us understand the current scientific research trends. By exploring how a group of authors collaborate over the years and observing how their relations change, we may understand the developmental path of their research area. By thoroughly analyzing the relations among different groups of authors, we may find some common factors that encourage them to collaborate. Patterns that include all relations among a set of entities are well suited to address such problem.

5.1 Coevolving Induced Relational Motifs

Coevolving Induced Relational Motifs (CIRMs) are designed to capture frequent patterns that include all relations among a set of entities (i.e., induced) at a certain time in the dynamic network and change in a consistent way over time. To illustrate this type of patterns consider the network of Fig. 12. The network for 1990 shows an induced relational motif (M_1) between pairs of nodes that occurs four times (shaded nodes and solid labeled edges). Three out of the four occurrences have evolved into a new motif (M_2) that includes an additional node in the network for 2000. Finally, in 2005 we see a new motif (M_3) that now involves four nodes and occurs two times. This example shows that the initial relational motif has changed in a fairly consistent fashion over time (i.e., it *coevolved*) and such a sequence of motifs $M_1 \rightsquigarrow M_2 \rightsquigarrow M_3$ that captures all relations among a set of entities represents an instance of a CIRM whose frequency is two (determined by M_3).

The formal definition of a CIRM that is used in this section is as follows:

Definition 3. A CIRM of length m is a tuple $\{N, \langle M_1, \ldots, M_m \rangle\}$, where N is a set of vertices and each $M_j = (N_j, A_j)$ is an **induced** relational motif defined over a subset of the vertices of N that satisfies the following constraints:

(i) it occurs at least ϕ times,

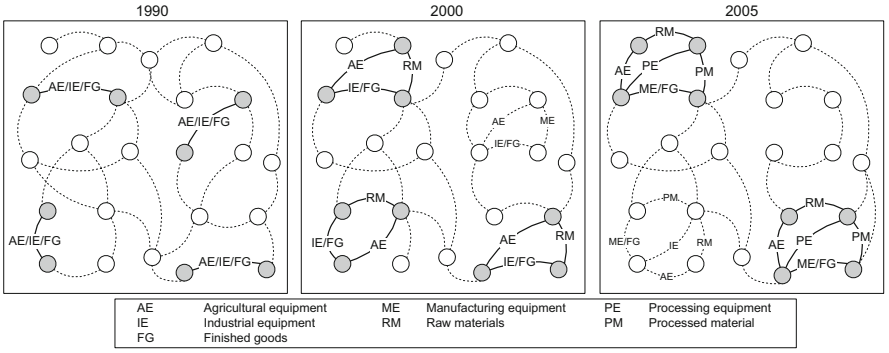


Fig. 12 An example of a coevolving induced relational motif in the context of a hypothetical country-to-country trading network where labels represent the commodities being traded. Assume the minimum support threshold (ϕ) for the CIRM is 2

ii) each occurrence uses a non-identical set of nodes,
(iii) $M_j \neq M_{j+1}$, and
(iv) $|N_j| \geq \beta |N|$ where $0 < \beta \leq 1$.

An induced relational motif M_j is defined over a subset of vertices N if there is an injection ξ_j from N_j to N. An m-length CIRM *occurs* in a dynamic network whose node set is V if there is a sequence of m snapshots $\langle G_{i_1}, G_{i_2}, \ldots, G_{i_m} \rangle$ and a subset of vertices B of V (i.e., $B \subseteq V$) such that:

(1) there is a bijection ξ from N to B
(2) the injection $\xi \circ \xi_j$ is an embedding of M_j in G_{i_j}
(3) there is no embedding of M_j via the injection $\xi \circ \xi_j$ in $G_{i_{j+1}}$ or no embedding of M_{j+1} via the injection $\xi \circ \xi_{j+1}$ in G_{i_j}.

The parameter ϕ is used to eliminate sequences of evolving motifs that are not frequent enough. Whereas the parameter β is used to control the degree of change between the sets of nodes involved in each motif of a CIRM and enforces a minimum node overlap among all motifs of CIRM.

In this section, we focus on identifying a subclass of CIRMs, called anchored CIRMs, such that in addition to the conditions of Definition 3, all motifs of the CIRM share at least one edge (anchor) that itself is a CIRM (i.e., the anchor is an evolving edge). This restriction ensures that all motifs of a CIRM contain at least a common pair of nodes and captures how these core set of entities coevolved.

Note that due to the above restriction, the number of motifs in a CIRM will be exactly the same as the number of motifs (i.e., edge spans) in its anchor. In some cases this will fail to identify evolving patterns that started from an anchor and then experience multiple relational changes between any two non-anchor nodes within the span of a motif. To address the above, we also identify a special class of CIRMs, referred to as *CIRM split extensions*, that have additional motifs than the anchor. A pattern is a CIRM split extension if:

(a) all of its motifs share an edge that satisfies properties (i), (ii), and (iv) of Definition 3, and
(b) for each maximal run of edge-spans $(x_1, x_2, \ldots, x_k)$ with the same label, there is another edge-span in the network that starts at the first snapshot of x_1 and ends at the last snapshot of x_k such that this edge-span is supported by the snapshots starting at the first snapshot that supports x_1 and ends at the last snapshot that supports x_k.

Given the above definition, the work in this section is designed to develop an efficient algorithm for solving the following problem:

Problem 3. Given a dynamic network $\mathcal{N}$ containing T snapshots, a user defined minimum support ϕ $(1 \leq \phi)$, a minimum number of vertices $k_{\min}$ per CIRM, and a minimum number of motifs $m_{\min}$ per CIRM, find all frequent anchored CIRMs and all CIRM split extensions.

A CIRM that meets the requirements specified in Problem 3 is referred to as a *frequent* CIRM and it is *valid* if it also satisfies the minimum node overlap constraint [Definition 3(iv)].

5.2 Coevolving Induced Relational Motif Mining

We developed an algorithm for solving Problem 3, called CIRMminer that follows a depth-first exploration approach from each anchor and identifies in a non-redundant fashion the complete set of CIRMs that occur at least ϕ times. For the rest of the discussion, any references to a relational motif or a motif will assume it is an induced relational motif. We represented CIRMs similar to CRMs in Sect. 4.2.1.

5.2.1 Mining Anchors

The search for CIRMs is initiated by locating the frequent anchors that satisfy the CIRM definition and the restrictions defined in Problem 3. This process is illustrated in Fig. 13. Given a dynamic network $\mathcal{N}$, we sort all the vertices and edges by their label frequency and remove all infrequent vertices. The remaining vertices and all edges are relabeled in decreasing frequency. We determine the span sequences of each edge and collect every edge's span sequence if that sequence contains at least a span with an edge label that is different from the rest of the spans. At this point, we use the sequential pattern mining technique prefixSpan [19] to determine all frequent span subsequences. Each of the frequent span subsequences is supported by a group of node pairs where the edge between each pair of nodes evolve in a consistent way over time. Since the frequent subsequences can be partial sequences of the original input span sequences, it is not guaranteed that they all contain consecutive spans with different labels. Thus, the frequent subsequences that

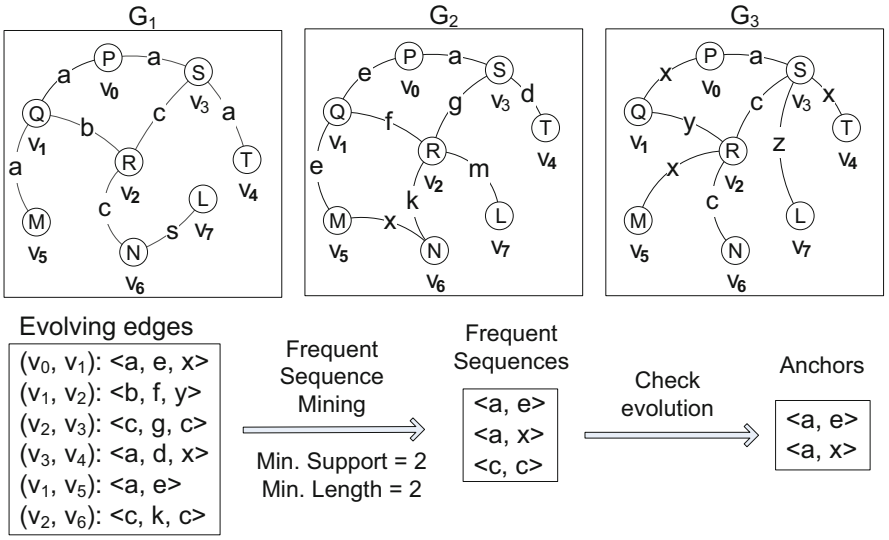


Fig. 13 The process of mining anchors from the network $\langle G_1, G_2, G_3 \rangle$. Since all vertex labels remained consistent over time, we listed the edge label sequences as the span sequence of the evolving edges

contain different consecutive spans in terms of label are considered as the anchors. The number of spans in a frequent span sequence corresponds to the total number of motifs in the anchor.

5.2.2 CIRM Enumeration

Given an anchor, CIRMminer generates the set of desired CIRMs by growing the size of the current CIRM one vertex at a time following a depth-first approach. The vertex-based CIRM growth approach was selected in order to ensure that each motif of the CIRM contains all edges among the vertices of that motif (i.e., it is an induced subgraph). To ensure that each CIRM is generated only once in the depth-first exploration, it uses an approach similar to gSpan [24], which we have extended for the problem of CIRM mining. gSpan performs a depth-first exploration of the pattern lattice avoiding redundant visits to the same node using a canonical labeling scheme, called minimum DFS code, and traverses a set of edges that form a spanning tree of the lattice. In order to apply the ideas introduced by gSpan to efficiently mine CIRMs, we defined the minimum DFS code of a CIRM following the definition of the minimum DFS code of a CRM [3]. We use the minimum DFS code of a CIRM as the canonical label. Due to the space constraint, we omit the discussion on minimum DFS code of a CIRM and encourage readers to refer to [3]. Once properly defined, the correctness and completeness of frequent CIRM enumeration follows directly from the corresponding proofs of gSpan.

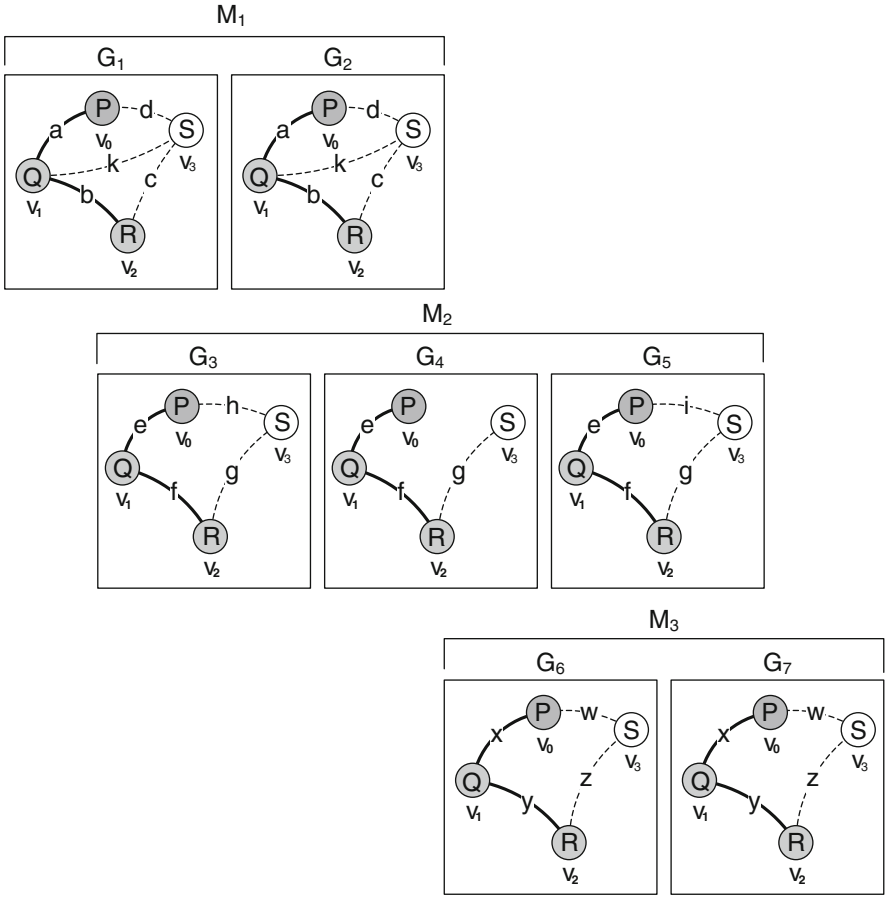
CIRM Growth

The CIRM enumeration process follows the rightmost extension rule [24] to select candidates for the next expansion and discards all CIRM extensions that do not contain a minimum DFS code. This is done by searching through all embeddings of the CIRM to identify the adjacent vertices that (1) connect to the nodes on the rightmost path and (2) the span of the vertices overlaps the span of the CIRM. For each adjacent vertex, CIRMminer goes through all embeddings of the CIRM to collect the sets of edges that connect that vertex with all existing vertices within the CIRM's span. To identify a set of edges for an embedding, it traverses through all snapshots within each motif's span and selects all maximal sets of edges connecting that vertex with the other CIRM vertices and remain in a consistent state. Each maximal set of edges along with the associated span is assigned a unique ID. The vertex and edge labels and the corresponding span determine the ID. Given these IDs, the algorithm then represents the set of edges resulting from a particular embedding of a CIRM as a sequence of IDs.

Figure 14 presents an example of generating a sequence of IDs during the process of CIRM growth. The CIRM consists of three motifs $\langle M_1, M_2, M_3 \rangle$, three vertices (shaded nodes), and two edges (solid labeled edges). An embedding of the CIRM is shown where the vertices are v_0, v_1, and v_2, and the edges are (v_0, v_1) and (v_1, v_2). We omitted vertex labels to form a simple example. To grow the CIRM by adding a new vertex, CIRMminer selects the adjacent vertex v_3 for this embedding. Hence, it needs to consider the set of edges that connects v_3 to the existing vertices v_0, v_1, and v_2. By analyzing the overlapping spans of the edges (v_0, v_3), (v_1, v_3), and (v_2, v_3), it identifies five different segments such that each one contains a maximal set of edges in a consistent state. As a result, these maximal sets are assigned unique IDs I_1 through I_5 where each ID contains a set of edges and a specific span. The set of edges is represented as: $\langle I_1, I_2, I_3, I_4, I_5 \rangle$.

CIRMminer represents the collected sets of edges from all embeddings as a collection of ID sequences and applies frequent sequence mining technique [19] to find all frequent ID sequences. Each frequent ID sequence is then considered as a frequent set of edges associated with a candidate vertex for the next CIRM extension. For example, if the subsequence $\langle I_1, I_2, I_5 \rangle$ from Fig. 14 is frequent, then the following set of edges is considered for vertex v_3: $\mathbf{(}0, 3, \langle P, P, P \rangle, \langle d, h, w \rangle, \langle S, S, S \rangle\mathbf{)}$, $\mathbf{(}1, 3, \langle Q, Q, Q \rangle, \langle k, \omega, \omega \rangle, \langle S, S, S \rangle\mathbf{)}$, $\mathbf{(}2, 3, \langle R, R, R \rangle, \langle c, g, z \rangle, \langle S, S, S \rangle\mathbf{)}$.

It is possible that a frequent ID sequence contains multiple IDs that belong to a particular motif of the CIRM. For example, if an ID sequence $\langle I_1, I_2, I_4, I_5 \rangle$ is frequent (in Fig. 14), both IDs I_2 and I_4 contain the span that belongs to motif M_2. To identify CIRMs according to Definition 3, it needs to divide these set of edges as multiple candidate sets where each set contains only one of the overlapping spans for each motif to match the total number of motifs of the original CIRM. To find the CIRM split extensions, the algorithm considers all such set of edges as valid extensions. This inclusion leads to identifying a super set of anchored CIRMs. Some of the CIRM split extensions may violate constraint (iii) of Definition 3 (i.e., $M_j \neq M_{j+1}$). It discards such CIRM split extensions as a post-processing step. Note that



	M_1		M_2			M_3	
	G_1	G_2	G_3	G_4	G_5	G_6	G_7
(v_0,v_3)	d	d	h	-	i	w	w
(v_1,v_3)	k	k	-	-	-	-	-
(v_2,v_3)	c	c	g	g	g	z	z
	I_1		I_2	I_3	I_4	I_5	

Fig. 14 Generating ID sequences from a set of edges. *Shaded vertices* and *solid line edges* are part of the existing CIRMs. Vertex v_3 is considered as the candidate vertex. For this example, since the vertex labels remain same in all snapshots, the *edges* are represented using only edge labels

Algorithm 5 CIRMminer($\mathcal{N}$, $\mathscr{C}$)

```
C_anchor ← find all frequent anchors from 𝒩
for each c in C_anchor do
    e_c ← all embeddings of c in 𝒩
    ExpandCIRM(𝒩, 𝒞, c, e_c)
return
```

Algorithm 6 ExpandCIRM($\mathcal{N}$, $\mathscr{C}$, c, e_c)

```
if support(c, e_c) < φ then
    return
if size(c) ≥ k_min then
    if overlap(c) ≥ β then
        𝒞 ← 𝒞 ∪ c
    if size(c) = k_max then
        return
V ← all frequent adjacent nodes of c in 𝒩
for each candidate v in V do
    S ← find all frequent edge sets that connect v to c
    for each edge set s in S do
        c' ← add s to c
        if c' = CanonicalLabel(c') then
            e_c' ← all embeddings of c' in 𝒩
            ExpandCIRM(𝒩, 𝒞, c', e_c')
return
```

each frequent candidate set of edges is added to the CIRM following the rightmost extension rules to determine the exact edge order ensuring that the minimum DFS code check can be performed on the extended CIRM.

5.2.3 CIRMminer Algorithm

The high-level structure of CIRMminer is shown in Algorithm 5. It first finds all frequent anchors according to Sect. 5.2.1. After locating all embeddings of an anchor c, the algorithm grows it recursively by adding one frequent adjacent node at a time. The recursive function *ExpandCIRM* first checks the support of c to prune any infrequent expansion. If c meets the minimum size and overlap requirement, it is added to the $\mathscr{C}$ to be recorded as a valid CIRM. It terminates expansion when c reaches the maximum size. To grow c further, the algorithm collects all adjacent nodes of c in $\mathcal{N}$ following the DFS rightmost extension rules. For each of the candidate node $v \in V$, it finds all frequent edge sets that connect the new node v to the existing nodes of c according to Sect. 5.2.2. Each of the candidate edge set s is added to c to construct its child c'. To prevent redundancy and ensure completeness, it only grows c' if it represents the canonical label of c', i.e., the minimum DFS code of c'.

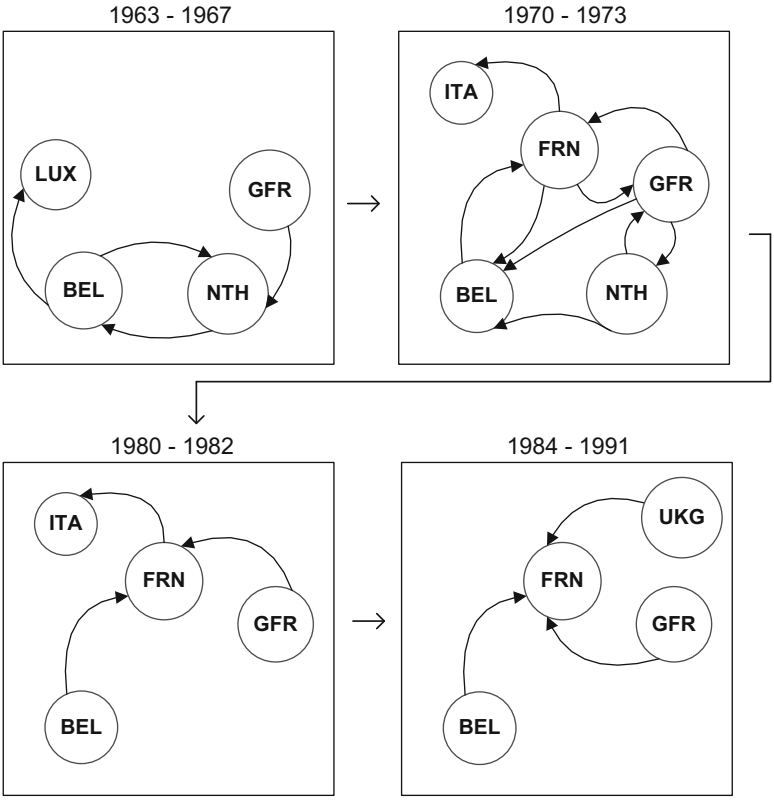


Fig. 15 An EIRS capturing a trade relation between EU countries. The nodes in the figure are *LUX*=Luxembourg, *BEL*=Belgium, *GFR*=German Federal Republic, *NTH*=Netherlands, *ITA*=Italy, and *UKG*=United Kingdom

6 Qualitative Analysis and Applications

In this section we present some of the evolving patterns that were discovered by our algorithms in order to illustrate the type of information that can be extracted from different dynamic networks.

6.1 Analysis of a Trade Network

This trade network dataset models the yearly export and import relations of 192 countries from 1948 to 2000 based on the Expanded Trade and GDP Data [12]. The nodes model the trading countries and the direct edges model the export or import activity between two countries for a certain year. The snapshots of the dynamic

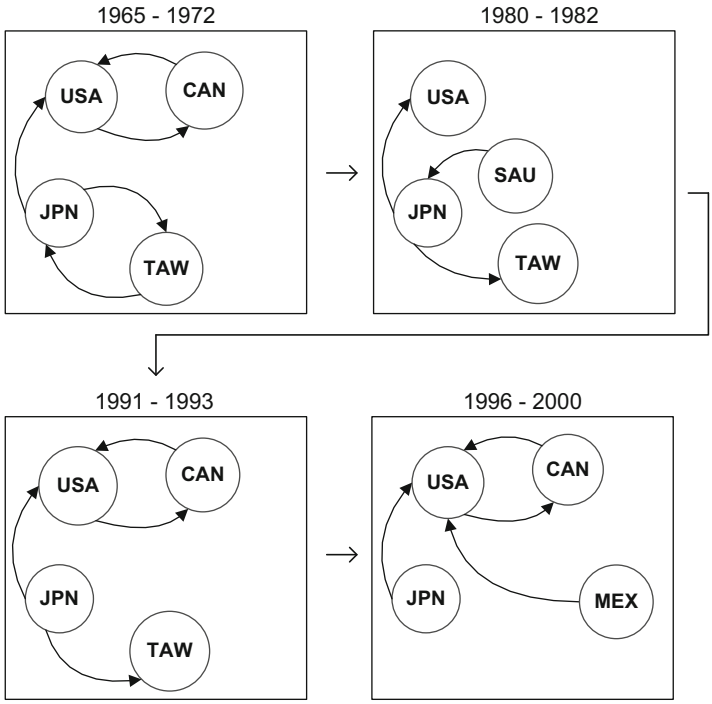


Fig. 16 An EIRS capturing a trade relation of USA. The nodes in the figure are *USA*=United States of America, *CAN*=Canada, *JPN*=Japan, *TAW*=Taiwan, *SAU*=Saudi Arabia, and *MEX*=Mexico

network that we created correspond to the trade network of each year, leading to a dynamic network consisting of $2000 - 1948 = 53$ snapshots. If the export amount from country A to country B in a given year is more than 10 % of the total export amount of A and total import amount of B for that year, a directed edge $A \rightarrow B$ is added to that year's trade graph.

To illustrate the type of information that can be extracted from this dynamic network, we will focus on how stable relations among the entities changed over time. We present some of the EIRSs that were discovered by our algorithm. In Fig. 15 we present an EIRS generated from the trade network capturing trade relations between some of the European countries over 30 years period and the chosen ϕ=3. The EIRS mainly captures trade relations between Belgium, Netherlands, Germany, and France. The other countries, such as Luxembourg, Italy, and United Kingdom, participate for a partial period of time. Based on the illustration, initially (during 1963–1967) Belgium and Netherlands were strong trade partners as they exported and imported from each other. The period 1970–1973 shows that the countries were heavily trading between each other. By evaluating the historical events, political and economic situation of that period, we could find the cause of higher trade activity. The periods 1980–1982 and 1984–1991 capture how France's trade relations with

Belgium and Germany became one sided as France only imported from those countries. The cause of such changes could be that France was exporting to other countries or Belgium and Germany decided to import from some other countries. In Fig. 16 we present another EIRS generated from the trade network capturing a stable trade relation of USA with other countries over 35 years period. We notice that USA and Canada have strong trade relations over a long period of time. Even though the strong tie in trading seems obvious due to the geographical co-location of the countries, it is interesting that the algorithm could discover such relation from the historical data. The EIRS also captures steady relation between the USA and Japan.

6.2 Analysis of a Co-authorship Network

This network models the yearly co-authorship relations from 1958 to 2012 based on the DBLP Computer Science Bibliography Database [18]. The snapshots correspond to the co-authorship network of each year, leading to a dynamic network consisting of $2012 - 1958 = 55$ snapshots. The nodes model the authors of the publications and the undirected edges model the collaboration between two authors at a certain year. To assign edge labels, we cluster the publication titles into 50 thematically related groups and use the cluster number as the labels. This representation contained $1,057,524$ nodes and on average $72,202$ edges per snapshot.

In order to illustrate the information that can be gathered from this dynamic network, we focus on identifying the coevolution of the relational entities. We present some of the CRMs that were discovered by our algorithm analyzing the *DBLP* dataset. The yearly co-authorship relations among the authors are divided into 50 clusters based on the title of the papers and we use the most frequent words that belong to a cluster to describe the topic it represents. To rank the discovered CRMs, we use the cosine similarity between the centroids of the clusters, referred to as *topic similarity*, that ranges from 0.02 to 0.52. For each CRM, we determine a score by calculating the average topic similarity based on all topic transitions (i.e., edge label changes) between two consecutive motifs of the CRM. The CRMs containing the least score ranks the highest. This ranking is designed to capture the frequent co-authorship relational changes that are thematically the most different.

Two of the high-ranked CRMs are shown in Fig. 17. The first CRM shows the periodic changes in research topics represented as 8 and 41 and the topic similarity between these topics is 0.22. The CRM captures the periodic transitions of the relations as author a and b collaborate with other authors c, d, and e over the time. The second CRM shows the periodic changes in research topics represented as 29 and 26 and the topic similarity between these topics is 0.20. The CRM captures similar the periodic transitions of the relations as author a and b collaborate with other authors c, d, and e over the time.

6.3 Analysis of a Multivariate Time-Series Dataset

This is a cell culture bioprocess dataset [17], referred to as the *GT* dataset that tracks the dynamics of various process parameters at every minute over 11 days period for 247 production runs. To represent this data as a dynamic network, we computed 47 correlation matrices for 14 of the process parameters using a sliding window of 12 hours interval with 50 % overlap. To construct a network snapshot based on a correlation matrix, we use each parameter as a vertex and two parameters/vertices are connected with an edge labeled as positive/negative if their correlation is above or below a certain threshold. Data from each run forms a small graph based on the correlation matrix at a certain time interval and the union of the graphs from all 247 runs at a certain time interval represents the snapshot. This dataset contains 47 snapshots where each snapshot contains 3458 nodes and on average 1776 edges.

As it is difficult to understand the relations between the nodes (i.e., parameters) without sufficient domain knowledge in Bio-Chemical processes, we decided to analyze the *GT* dataset based on discovered CRMs/CIRMs by using them as features to discriminate the production runs. For the *GT* dataset, the discovered CRMs/CIRMs display the dynamics of various process parameters during the cell culture process. Out of the 247 production runs included in the *GT* dataset, based on the quality of the yields, 48 of the runs are labeled as *Good*, 48 of the runs are labeled as *Bad*, and the remaining were not labeled. To understand the class distribution (i.e., Good or Bad) of the discovered CRMs and CIRMs, we analyzed the embeddings of the discovered CRMs and CIRMs from three different experiments using ϕ of 70, 60 and 50. Since we have 96 labeled runs, we do not count the embeddings that belong to unlabeled runs. Figure 18 shows the class distribution of the embeddings for the discovered CRMs and CIRMs. It shows that the CRMs/CIRMs were present mostly as part of the high yield runs, since more than 75 % of the embeddings belong to the Good class. The class representation is consistent for different support parameters. These results suggest that there is a consistency of what makes some thing good but runs can go bad for many reasons. Note that using such information, if we can determine the low yield runs at the early stage of experiment, we can terminate the experiment and save a lot of resources. In addition, the ratio of the embeddings supporting the Good class remains consistent between CRMs (Good c) and CIRMs (Good ic). Even though there are fewer CIRMs detected compared to CRMs, the information captured within the discovered CIRMs represents the characteristics of the underlying dynamic network as well as the CRMs.

7 Conclusion and Future Research Directions

In this chapter we presented several algorithms that can efficiently and effectively analyze the changes in dynamic relational networks. The new classes of dynamic patterns enable the identification of hidden coordination mechanisms underlying

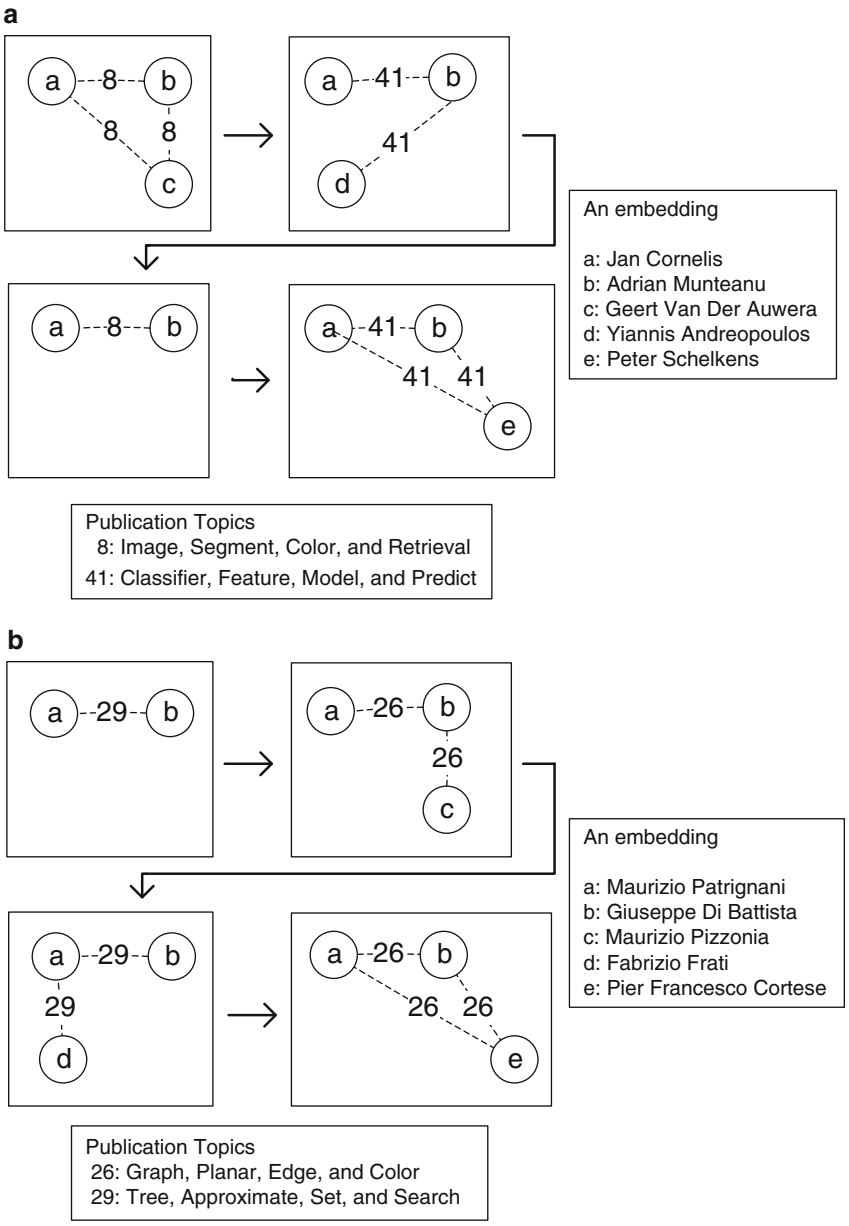


Fig. 17 Two CRMs capturing co-authorship patterns. The *edge labels* represent the domain or subject of the publications the authors were involved together. The *vertices* are labeled to show the changes in relations between the nodes. These CRMs are collected using $\phi = 90$ and $\beta = 0.60$

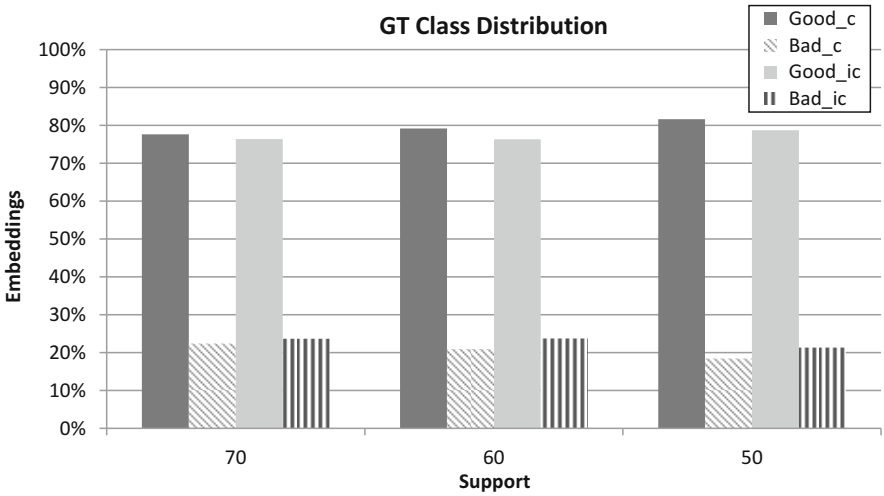


Fig. 18 A distribution of the CRM and CIRM embeddings. The Good class represents the production runs with high yield and the Bad class represents with poor yield. CIRMs were collected using ϕ = (70, 60 and 50), $\beta = 0.50$, $m_{\min} = 3$, $k_{\min} = 3$, and $k_{\max} = 8$

the networks, provide information on the recurrence and the stability of its relational patterns, and improve the ability to predict the relations and their changes in these networks. Specifically, the qualitative analysis of each class of patterns has shown the information captured by these patterns about the underlying networks and proven to be useful for building models.

There are a number of ways we can extend the existing definitions of EIRS and CRM/CIRM to address other real-world pattern mining problems. First, combine the notions of state transition and coevolution into a single pattern class in order to derive patterns that capture the coevolution of recurring stable relations. Second, relax the assumption that the various occurrences of the relational states and motifs match perfectly in terms of node and edge labels. Two general approaches can be designed to allow for flexibility in matching the types of relations and entities. The first utilizes an application-specific similarity function along with a user-supplied similarity threshold γ and the second is based on employing clustering techniques on the edge and node labels. The similarity-function based approach provides control over the degree of approximation that is allowed during the mining of evolving relational patterns. The clustering approach can be used to replace the complex edge and node labels by two distinct sets of categorical labels corresponding to the cluster number that the edges and nodes belong to.

Third, consider dynamic relational networks modeling real-life datasets that contain noisy data and missing links. Two different approaches and their combination can be considered for addressing such problems. The first solution focuses on the modeling phase of the underlying datasets. One approach is to temporally smooth the sequence of snapshots in order to eliminate noise that is localized in

time. Another approach can be to eliminate transient relations from the snapshots all together and focus the analysis on only the relations that have some degree of persistence. The second solution focuses on the pattern mining phase that allows for missing edges and nodes during the subgraph occurrence operations (i.e., the subgraph test in the case of relational states and the subgraph isomorphism in the case of relational motifs). The degree of allowed match tolerance can be controlled by a user-supplied parameter.

Fourth, utilize CRMminer/CIRMminer algorithms to mine discriminative dynamic relational patterns, whose presence or absence can help us classify a dynamic network. By combining pattern frequency and discriminative measures, it is shown that discriminative frequent patterns are very effective for classification [9]. There are several algorithms, such as LEAP [25], CORK [22], GraphSig [20], and LTS [14] that mine discriminative frequent patterns in static graph. To efficiently mine discriminative dynamic relational patterns, we need to focus on addressing two problems. The first is to design an efficient algorithm to select discriminative features among a large number of frequent dynamic patterns (i.e., CRMs/CIRMs) and to define a context aware prediction function that can measure the discriminative potential of a dynamic pattern. The second is to use both frequency and the discriminative potential function during dynamic pattern enumeration for pruning the exponential search space.

References

1. Ahmed, R., Karypis, G.: Algorithms for mining the evolution of conserved relational states in dynamic networks. Knowledge and Information Systems, **33**(3):1–28 (2012)
2. Ahmed, R., Karypis, G.: Mining coevolving induced relational motifs in dynamic networks. In: SDM Networks, the 2nd Workshop on Mining Graphs and Networks (2015)
3. Ahmed, R., Karypis, G.: Algorithms for Mining the Coevolving Relational Motifs in Dynamic Networks. In: ACM Trans. Knowl. Discov. Data, **10**(1), ACM, NY (2015)
4. Berlingerio, M., Bonchi, F., Bringmann, B., Gionis, A.: Mining graph evolution rules. In: Machine Learning pp. 115–130. Springer, Berlin/Heidelberg (2009)
5. Borgwardt, K.M., Kriegel, H.P., Wackersreuther, P.: Pattern mining in frequent dynamic subgraphs. In: IEEE ICDM, pp. 818–822 (2006)
6. Bringmann, B., Nijssen, S.: What is frequent in a single graph? In: Advances in Knowledge Discovery and Data Mining, pp. 858–863. Springer, Berlin/Heidelberg (2008)
7. Cerf, L., Nguyen, T., Boulicaut, J.: Discovering relevant cross-graph cliques in dynamic networks. In: Foundations of Intelligent Systems, pp. 513–522. Springer, Berlin/Heidelberg (2009)
8. Chakrabarti, D., Kumar, R., Tomkins, A.: Evolutionary clustering. In: ACM KDD, pp. 554–560 (2006)
9. Cheng, H., Yan, X., Han, J., Hsu, C.W.: Discriminative frequent pattern analysis for effective classification. In: IEEE 23rd International Conference on Data Engineering, 2007. ICDE 2007, pp. 716–725. IEEE, Istanbul (2007)
10. Desmier, E., Plantevit, M., Robardet, C., Boulicaut, J.F.: Cohesive co-evolution patterns in dynamic attributed graphs. In: Discovery Science, pp. 110–124. Springer, Heidelberg (2012)
11. Friedman, T.L.: The World Is Flat: A Brief History of the Twenty-First Century. Farrar, Straus & Giroux, New York (2005)

12. Gleditsch, K.S.: Expanded trade and GDP data. In: J. Confl. Resolut., **46**(5), pp. 712–724 (2002)
13. Inokuchi, A., Washio, T.: Mining frequent graph sequence patterns induced by vertices. In: Proc. of 10th SDM, pp. 466–477 (2010)
14. Jin, N., Wang, W.: LTS: Discriminative subgraph mining by learning from search history In: 2011 IEEE 27th International Conference on Data Engineering (ICDE), pp. 207–218. IEEE,Washington (2011)
15. Jin, R., McCallen, S., Almaas, E.: Trend motif: A graph mining approach for analysis of dynamic complex networks. In: IEEE ICDM, pp. 541–546 (2007)
16. Kuramochi, M., Karypis, G.: Finding frequent patterns in a large sparse graph. In: Data Mining and Knowledge Discovery, **11**(3), pp. 243–271, Springer US (2005)
17. Le, H., Kabbur, S., Pollastrini, L., Sun, Z., Mills, K., Johnson, K., Karypis, G., Hu, W.S.: Multivariate analysis of cell culture bioprocess data–lactate consumption as process indicator. In: Journal of biotechnology, 162, pp. 210–223 (2012)
18. Ley, M.: Dblp, computer science bibliography. Website, http://www.informatik.uni-trier.de/~ley/ (2008)
19. Pei, J., Han, J., Mortazavi-Asl, B., Pinto, H., Chen, Q., Dayal, U., Hsu, M.C.: PrefixSpan: Mining Sequential Patterns Efficiently by Prefix-Projected Pattern Growth. In: Proceedings IEEE International Conference on Data Engineering (ICDE), pp. 215–224 (2001)
20. Ranu, S., Singh, A.K.: Graphsig: a scalable approach to mining significant subgraphs in large graph databases. In: IEEE 25th International Conference on Data Engineering, 2009. ICDE'09, pp. 844–855. IEEE, Shanghai (2009)
21. Robardet, C.: Constraint-based pattern mining in dynamic graphs. In: IEEE ICDM, pp. 950–955 (2009)
22. Thoma, M., Cheng, H., Gretton, A., Han, J., Kriegel, H.P., Smola, A.J., Song, L., Philip, S.Y., Yan, X., Borgwardt, K.M.: Near-optimal supervised feature selection among frequent subgraphs. In: SDM, pp. 1076–1087. SIAM, Sparks (2009)
23. West, D.B.: Introduction to Graph Theory. Prentice Hall, Upper Saddle River (2001)
24. Yan, X., Han, J.: gSpan: graph-based substructure pattern mining. In: IEEE ICDM, pp. 721–724 (2002)
25. Yan, X., Cheng, H., Han, J., Yu, P.S.: Mining significant graph patterns by leap search. In: Proceedings of the 2008 ACM SIGMOD international conference on Management of data, pp. 433–444. ACM, New York (2008)
26. Zhu, F., Yan, X., Han, J., Philip, S.Y.: gPrune: a constraint pushing framework for graph pattern mining. In: Advances in Knowledge Discovery and Data Mining, pp. 388–400. Springer, Heidelberg (2007)