Mining Coevolving Induced Relational Motifs in Dynamic Networks

Rezwan Ahmed[*] George Karypis[†]

Abstract

A fundamental task associated with the analysis of a dynamic network is to study and understand how the network changes over time. Co-evolution of patterns, where all the relations among a set of entities change in a consistent way over time, can provide evidence of possibly unknown coordination mechanism among the entities of a dynamic network. This paper introduces a new class of dynamic network patterns, referred to as coevolving induced relational motifs (CIRMs), which are designed to identify a recurring set of nodes whose complete set of intra-relations undergo some changes in a consistent way over time. We develop an algorithm to analyze all relational changes between entities and find all frequent coevolving induced relational motifs. Experimental results based on multiple dynamic networks derived from real world datasets show that the algorithm is able to identify all frequent CIRMs in small amount of time. In addition, a qualitative analysis of the results shows that the discovered CIRMs 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

Dynamic networks are widely used to model the relationships between various entities that evolve over time and to capture the dynamic aspects of the data from many complex real world applications. In this paper we focus on finding patterns that capture frequent conserved changes over time among the relationships of the entities involved (i.e., co-evolution). Such patterns can indicate, possibly unknown, coordination among the entities and may expose the factors that influence their consistent change. In addition, in the context of classification tasks, these patterns can be used as features for exploring the processes that lead to certain outcomes. For example, the co-evolving patterns found in [1] from a bioprocess network were able to capture sufficient network characteristics to identify the high-yielding production runs.

[*]Dept. of Computer Science & Engineering, Uni. of Minnesota

[†]Dept. of Computer Science & Engineering, Uni. of Minnesota

The growing research of mining evolving patterns is primarily focused on finding dynamic subgraphs [2], finding subgraph subsequences [5], identifying co-evolution patterns capturing attribute trends [4], and finding co-evolving relational motifs (CRMs) that consists of recurring subgraphs changing in a consistent way [1]. Most of the existing methods target patterns that are arbitrary in nature (i.e., subgraphs). Such patterns allow the algorithms to ignore some of the relations that exist among the nodes and cause them to generate a large number of patterns. However, in order to understand how the relations between groups of entities have changed over time, we need to take into account all of their relations. For example, when analyzing the changes in a co-authorship network over time, it is important to focus on how all the relations between a set of authors have changed to effectively understand the development path of their research area and popular scientific research trends.

In this paper, our contribution is two fold: First, we define a new class of patterns, referred as *coevolving induced relational motifs (CIRMs)*, 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. Second, we present an algorithm, referred to as CIRMminer, to efficiently mine all frequent coevolving induced relational motifs. CIRMminer follows a depth-first exploration approach that uses canonical labeling for redundancy elimination and ensures induced isomorphism of the motifs.

We experimentally evaluate our algorithm on dynamic networks derived from three real world datasets: a large co-authorship network, a bioprocess network, and a market sales network. We perform comprehensive evaluation of the performance and scalability of CIRMminer on all three datasets by varying different input parameters. Our experiments show that CIRMminer is able to identify all frequent patterns in a small amount of time and scales linearly to the output space (i.e., based on number of frequent patterns present in a dataset). Our qualitative analysis of the discovered CIRMs in the context of a bioprocess network shows that CIRMs can capture network characteristics that can be used as features for classification tasks.

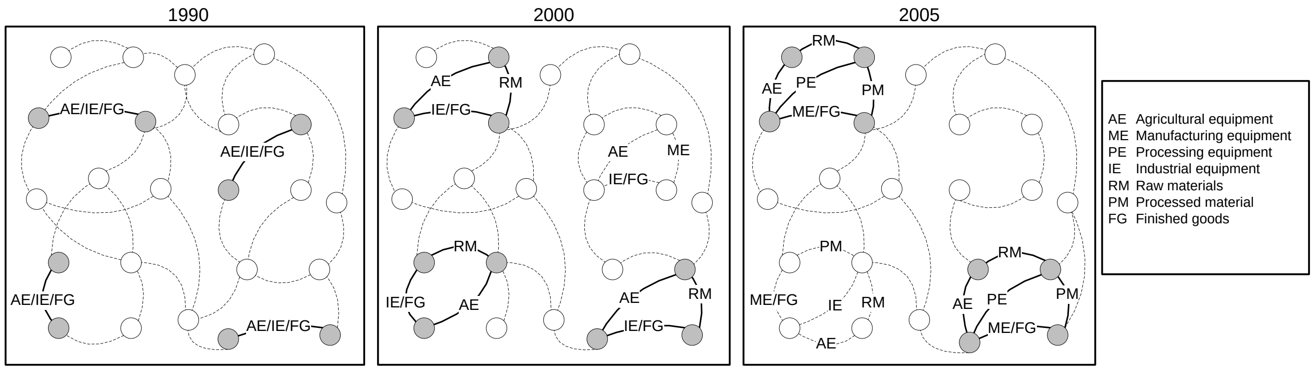


Figure 1: An example of a coevolving induced relational motif in the context of a hypothetical country-to-country trading network where labels represent the commodities been traded. Assume the minimum support threshold (ϕ) for the CIRM is 2.

2 Related Work

The problem of finding patterns in static network is a well studied research area [10, 7, 11]. In recent years, there has been a lot of research work focused on the problem of mining dynamic networks. The notion of the *dynamic subgraph* was introduced in [2] to capture a sequence of subgraphs that exist in a consecutive sequence of snapshots. Their algorithm identifies the set of dynamic subgraphs that occur frequently in a dynamic network. Jin et. al. [6] introduced *trend motif*, which was later [4] defined in a more generic way to capture the problem of mining cohesive co-evolving patterns that represent the local co-evolution of similar vertices at several timestamps based on their attributes trends. Inokuchi et. al. [5] solved the problem of finding frequent induced subgraph subsequences from graph sequence data and capturing the changes of a subgraph over the subsequence. Recently, Ahmed et. al. [1] introduced coevolving relational motifs (CRM), that consist of recurring sets of entities (i.e., nodes) whose relations change in a consistent way across the different snapshots of the network.

All the above patterns except [5] correspond to arbitrary subgraphs that may ignore some of the relations that exist among the nodes. This forces these algorithms to identify a lot of trivial and redundant patterns where most of them may fail to capture the complete set of relations that exist and have co-evolved among the nodes. In case of [5], the algorithm first forms a union graph based on the graph sequence, then identifies all frequent induced subgraphs within the union graph, and lastly uses a frequent sequence miner to determine each induced subgraph sequences in the graph sequence. Their method does not allow users to control the length (i.e., number of graphs in a subsequence) or the size of the patterns. This may lead to generating a lot of short and trivial frequent sequences. Our work in this paper is directly related to CRMs and designed to focus on all relations among the nodes and identify patterns in significantly less time. The fundamental differences between a CRM and the CIRM that is the focus of this paper are: i) CIRMs are composed of induced relational motifs, and ii) a CIRM can potentially have more motifs than the number of motifs in the anchor (split extensions).

3 Definitions and Notation

A dynamic network is represented via a sequence of labeled undirected graphs. A *labeled graph* $G = (V, E, L[V], L[E])$ is composed of a set of nodes V modeling the entities of the network, a set of edges E modeling the relations between these entities, a set of node labels $L[V]$ modeling the type of the entities ($|V| \geq |L[V]|$), and a set of edge labels $L[E]$ modeling the type of the relations ($|E| \geq |L[E]|$). The labels assigned to the vertices (edges) are typically not unique and multiple vertices (edges) can have the same label. A *subgraph* $G' = (V', E', L[V'], L[E'])$ of G is a graph such that $V' \subseteq V$, $E' \subseteq E \cap (V' \times V')$. An *induced subgraph* $G'' = (V'', E'', L[V''], L[E''])$ of G is a graph such that $V'' \subseteq V$, $E'' \subseteq E$ and $\forall (u, v) \in E$ such that $v \in V''$ and $u \in V''$, $(u, v) \in E''$.

A *dynamic network* $\mathcal{N} = \langle G_1, G_2, \ldots, G_T \rangle$ is a finite sequence of graphs, where each $G_t = (V, E_t, L_t[V], L_t[E_t])$ is a labeled graph describing the state of the system at a discrete time interval t. The term *snapshot* will be used to refer to each of the graphs in the sequence. Snapshots are assumed to contain the same set of nodes, which will also be referred to as the nodes of $\mathcal{N}$, denoted by $V_{\mathcal{N}}$, but potentially different sets of edges and node/edge labels. When nodes appear or disappear over time, the set of nodes of each snapshot is the union of all the nodes over all snapshots. Also, 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 $\mathcal{N}$.

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. An edge (u, v) is in a *consistent state* over a time interval $s\!:\!e$ if it is present in all snapshots $G_s, \ldots, G_e$ with the same label l. 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 $l_{u_i}, l_{v_i} \in L[V], l_{e_i} \in L[E]$, $s_i \leq e_i$, and $e_i \leq s_{i+1}$.

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). We will use M to denote an induced relational motif and the underlying induced subgraphs 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), $L[N]$ is the set of node labels, and $L[A]$ is the set of edge labels.

4 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 Figure 1. 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 paper is as follows:

Definition 1 *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,

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. A 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:

i) there is a bijection ξ from N to B

ii) the injection $\xi \circ \xi_j$ is an embedding of M_j in G_{i_j}

iii) 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 paper, we focus on identifying a subclass of CIRMs, called anchored CIRMs, such that in addition to the conditions of Definition 1, 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 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 1, 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 paper is designed to develop an efficient algorithm for solving the following problem:

Problem 1 *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 1 is referred as a *frequent* CIRM and it is *valid* if it also satisfies the minimum node overlap constraint (Definition 1(iv)).

5 Coevolving Induced Relational Motif Mining

We developed an algorithm for solving Problem 1, 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.

5.1 CIRM Representation We adopt the method of [1] to represent a CIRM $c = \{N, \langle M_1, \ldots, M_m \rangle\}$ as a graph $G_c = (N, E_c)$, such that an edge $(u, v) \in E_c$ is a 5-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 CIRM 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 an edge (u, v), or any of the vertices u, or v, is not present in the M_k motif, then ω is inserted at the kth entry of that label vector. Note that the value of ω is lexicographically greater than the maximum edge and vertex label. This representation is illustrated in Figure 2.

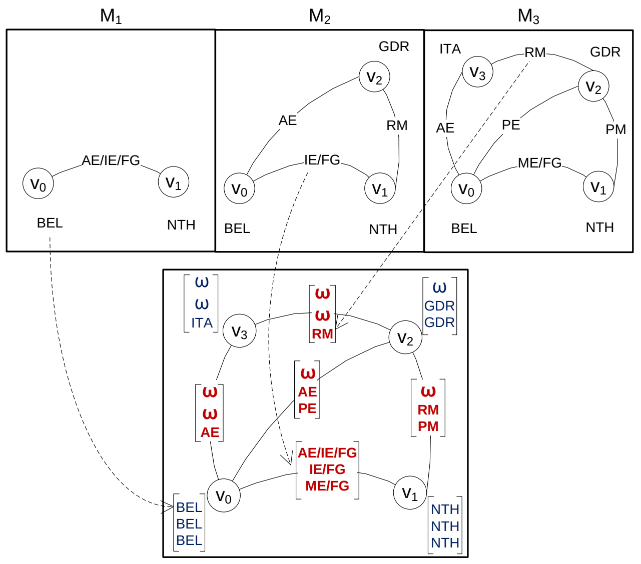


Figure 2: A CIRM Representation. The CIRM c consists of 3 motifs $\langle M_1, M_2, M_3 \rangle$ and represents relations among vertices $N = \{v_0, v_1, v_2, v_3\}$ using 5 edges. G_c (bottom graph) shows the CIRM representation capturing vertex and edge label vectors.

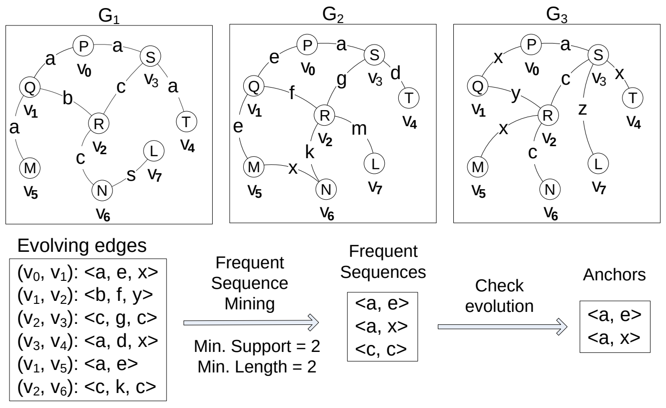
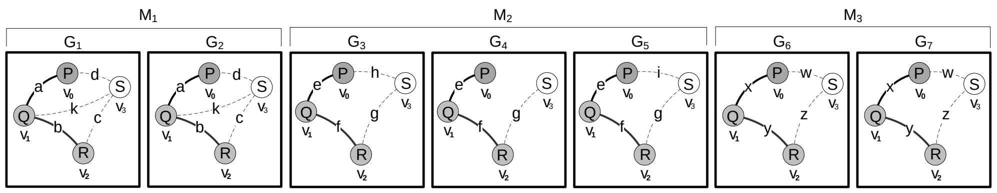


Figure 3: The process of mining anchors from the network $\langle G_1, G_2, G_3 \rangle$. Since all vertex labels remained consistent over time, we listed the edge label sequences as the span sequence of the evolving edges.

5.2 Mining Anchors The search for CIRMs is initiated by locating the frequent anchors that satisfy the CIRM definition and the restrictions defined in Problem 1. This process is illustrated in Figure 3. 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 [10] 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 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.

5.3 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 [11], which we have extended for the problem of CIRM mining. gSpan



	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	

Figure 4: 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.

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 the problem of efficiently mining CIRMs, we defined the minimum DFS code of a CIRM folllowing the definition of the minimum DFS code of a CRM [1]. 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 [1]. Once properly defined, the correctness and completeness of frequent CIRM enumeration follows directly from the corresponding proofs of gSpan.

5.3.1 CIRM growth The CIRM enumeration process follows the rightmost extension rule [11] 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 (i) connect to the nodes on the rightmost path and (ii) 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 determines 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 4 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_1), (v_0, v_2), and (v_0, 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 apply frequent sequence mining technique [10] 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, the subsequence $\langle I_1, I_2, I_5\rangle$ from Figure 4 is frequent, then the following set of edges is considered for vertex v_3: $(0, 3, \langle P, P, P\rangle, \langle d, h, w\rangle, \langle S, S, S\rangle)$, $(1, 3, \langle Q, Q, Q\rangle, \langle k, \omega, \omega\rangle, \langle S, S, S\rangle)$, $(2, 3, \langle R, R, R\rangle, \langle c, g, z\rangle, \langle S, S, S\rangle)$

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 Figure 4), both IDs I_2 and I_4 contains the span that belongs to motif M_2. To identify CIRMs according to Definition 1, it needs to divide these set of edges as multiple candidate sets where each set contains only one of the overlapping span 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 1 (i.e., $M_j \neq M_{j+1}$). It discards such CIRM split extensions as a post-processing step. Note that each frequent candidate set of edges are 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.

Algorithm 1 CIRMminer($\mathcal{N}$, $\mathcal{C}$)

```
1: $C_{anchor} \leftarrow$ find all frequent anchors from $\mathcal{N}$
2: for each $c$ in $C_{anchor}$ do
3:     $e_c \leftarrow$ all embeddings of $c$ in $\mathcal{N}$
4:     $ExpandCIRM(\mathcal{N}, \mathcal{C}, c, e_c)$
5: return
```

Algorithm 2 ExpandCIRM($\mathcal{N}$, $\mathcal{C}$, c, e_c)

```
1: if $support(c, e_c) < \phi$ then
2:     return
3: if $size(c) \geq k_{min}$ then
4:     if $overlap(c) \geq \beta$ then
5:         $\mathcal{C} \leftarrow \mathcal{C} \cup c$
6:     if $size(c) = k_{max}$ then
7:         return
8: $V \leftarrow$ all frequent adjacent nodes of $c$ in $\mathcal{N}$
9: for each candidate $v$ in $V$ do
10:     $S \leftarrow$ find all frequent edge sets that connect $v$ to $c$
11:     for each edge set $s$ in $S$ do
12:         $c' \leftarrow$ add $s$ to $c$
13:         if $c' = CanonicalLabel(c')$ then
14:             $e_{c'} \leftarrow$ all embeddings of $c'$ in $\mathcal{N}$
15:             ExpandCIRM($\mathcal{N}$, $\mathcal{C}$, $c'$, $e_{c'}$)
16: return
```

5.4 CIRMminer Algorithm The high-level structure of CIRMminer is shown in Algorithm 1. It first finds all frequent anchors according to Section 5.2. 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 $\mathcal{C}$ 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 Section 5.3.1. 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'.

5.5 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. We adopted the minimum image based support measure [3] to calculate the minimum support of a CIRM in a dynamic network.

As defined in Section 3, 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. Given a CIRM $c = \{N_c, \langle M_1, M_2, \ldots, M_m\rangle\}$ in a dynamic network $\mathcal{N} = \{V_{\mathcal{N}}, \langle G_0, G_1, \ldots, G_T\rangle\}$ where $m \leq T$, this measure identifies the vertex in c which is mapped to the least number of unique vertices in $\mathcal{N}$ and uses this number as the frequency of c in $\mathcal{N}$. To formally define, the minimum image based support of c is defined as:

$$\sigma(c, \mathcal{N}) = \min_{v \in V_c} \mid \{\varphi_i(v)\colon\ \varphi_i \text{ is an occurrence of } c \text{ in } \mathcal{N}\} \mid .$$

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.

5.6 Minimum Overlap (β) Each motif of a CIRM needs to contain at least a minimum percentage of the nodes from all the nodes of the CIRM. This minimum node overlap threshold (defined as β in Section 3) controls the degree of change that is allowed between the sets of nodes in each motif of a CIRM. Given a CIRM $c = \{N_c, \langle M_1, M_2, \ldots, M_m\rangle\}$ containing m motifs, the *minimum overlap* of a CIRM is defined as:

$$\rho(c) = \frac{\min_{1 \leq i \leq m} \{|V_{M_i}|\}}{|N_c|},$$

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 CIRM are coherent, it is not anti-monotonic [12]. Thus, to generate a complete set of CIRMs that meet user specified thresholds of support and overlap, we cannot prune CIRMs that do not satisfy this constraint as CIRMs derived from it can satisfy the constraint. Hence, we need to enumerate all CIRMs that meet the support threshold and then search the output space for CIRMs that meet the minimum overlap requirement.

6 Experimental Methodology & Datasets

All experiments are conducted on a 64-bit Linux desktop with 8-core Intel® Core$^{\text{TM}}$ i7-3770 processor at 3.40GHz and 16GB of RAM.

Table 1: Dynamic Network Datasets.

Dataset	#Vertices	#Edges	Span	Avg. #Edges	Avg. Degree
DBLP	1,057,524	3,971,100	55	72,202	0.07
GT	3,458	83,454	47	1,776	0.51
Sales	2,697	138,044	66	2,092	0.78

#Vertices denotes the total number of vertices in the dynamic network, #Edges denotes the total number of edges in the dynamic network. Span denotes the total number of snapshots in the dynamic network. Avg. #Edges denotes the average number of edges per snapshot in the dynamic network. Avg. Degree denotes the avg. number of edges per node in a snapshot.

6.1 Co-Authorship Network (DBLP) This network models the yearly co-authorship relations from 1958 to 2012 (Table 1) based on the DBLP Computer Science Bibliography Database [9]. 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.

6.2 Bioprocess Network (GT) This is a cell culture bioprocess dataset [8] 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 (Table 1), 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.

6.3 Store Transaction Network (Sales) This network is constructed using Dominicks Finer Foods store sales data that captures weekly store-level sales from 93 stores collected over a period of more than seven years (400 weeks). To represent the dynamic network, we generate 66 correlation matrices between 29 product categories using a sliding window of 12 weeks interval with 50% overlap. To construct a network snapshot, we use each category as a vertex and two vertices are connected with an edge labeled as positive or negative if their correlation is above or below a threshold. Data from each store sales forms a small graph based on the correlation matrix at a certain time interval and the union of the graphs from all 93 stores at a certain time interval represents the snapshot.

7 Results & Discussion

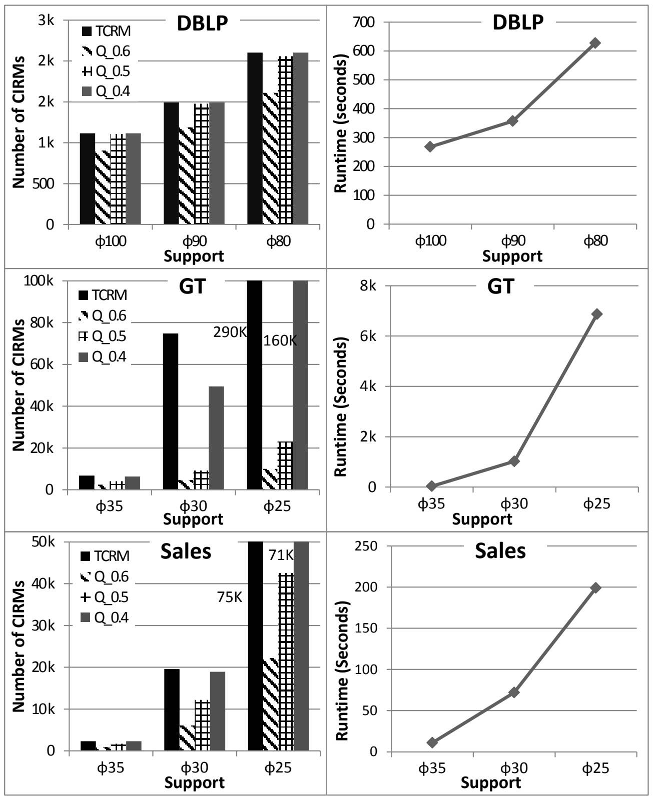


Figure 5: CIRMminer performance for different values of the minimum support (ϕ) and overlap threshold (β). $TCRM$ denotes the frequent CIRMs (T_{CIRM}). $Q\_0.6$, $Q\_0.5$, and $Q\_0.4$ denotes the valid CIRMs (Q_{CIRM}) for the specified β threshold.

7.1 Performance Results In order to assess the scalability and performance of CIRMminer, we collected experimental results by running CIRMminer for different support thresholds to generate the complete set of CIRMs (T_{CIRM}) and then applied the overlap threshold to identify the valid CIRMs (Q_{CIRM}) from the output space. Figure 5 shows the performance of CIRMminer for different values of the minimum support and overlap threshold[1]. The following observations can be made from these results. First, for all datasets, the number of discovered CIRMs (T_{CIRM} and Q_{CIRM}) increases as the minimum support (ϕ) decreases. The increase in the number of CIRMs is expected due to the anti-monotonic property of the support threshold. Second, the number of valid CIRMs (Q_{CIRM}) increases as the overlap threshold (β) decreases. This is expected as the lower overlap requirement allows a greater number of different nodes in the CIRM.

Third, the time to discover the CIRMs increases, as the minimum support (ϕ) decreases. For the *DBLP*

[1]To ensure that all experiments are easily reproducible, we provided datasets and CIRMminer software at our project site `https://sites.google.com/site/cirm2014sup/`

Table 2: Comparing CRMminer vs. CIRMminer results.

Data	ϕ	CRMminer			CIRMminer		
		$\#T_{\text{CRM}}$	$\#Q_{\text{CRM}}$	Time	$\#T_{\text{CIRM}}$	$\#Q_{\text{CIRM}}$	Time
DBLP	140	103.5K	47.7K	68,303	414	412	60.24
	130	151.3K	68.7K	92,106	513	511	71.85
	120	223.5K	100.4K	129,413	651	648	91.07
	110	334.3K	148.9K	181,036	833	830	116.40
	100	512.5K	226.7K	248,101	1,115	1,106	163.85
	90	807.4K	356.6K	423,217	1,493	1,476	225.21
GT	70	547.8K	506.8K	266	209	206	0.69
	60	1.4M	1.2M	577	359	352	1.11
	50	5.3M	4.6M	1,642	732	692	2.03
	40	20.9M	17.2M	4,774	2,208	1,941	5.09
	30	86.2M	69.6M	13,966	74,855	9,236	561.52
Sales	45	10.9K	1.7K	10	41	41	0.27
	40	109.9K	9.1K	64	223	195	0.76
	35	3.4M	585.2K	1,005	2,335	1,722	5.57
	30	62.9M	17.7M	10,825	19,567	12,195	36.15
	25	259.4M	64.8M	40,128	74,908	42,576	99.89

Run times are in seconds. ϕ denotes the minimum support. $\#T_{\text{CRM}}$ denotes the total number of discovered CRMs. $\#Q_{\text{CRM}}$ denotes the number of CRMs that meet the β threshold out of $\#T_{\text{CRM}}$. $\#T_{\text{CIRM}}$ denotes the total number of discovered CIRMs. $\#Q_{\text{CIRM}}$ denotes the number of CIRMs that meet the β threshold out of $\#T_{\text{CIRM}}$. For all datasets, β is 0.50, m_{min} is 3, k_{min} is 4 and k_{max} is 8.

and the *Sales* datasets, the runtimes scale linearly to the size of the output space (i.e., the number of frequent patterns present in the dataset). For the *DBLP* dataset, as the support threshold decreases from 100 to 80, the output space increases by 1.8 times and the runtime increases by 2.3 times. Similarly, for the *Sales* dataset, as the support threshold decreases, the output space increases about 25 times when the runtime increases by 18 times. However, for the *GT* dataset, the runtime increases at a higher rate than the output space. The reason for that is for ϕ=25, CIRMminer spent a long time processing a large number of candidates ($2,748,974$ CIRMs), but most of those candidates failed to qualify later due to lack of minimum support or size requirements. Thus, this significant increase in the work associated with a decrease in support is not directly captured by simply looking at the number of discovered CIRMs.

7.2 Comparing CRMs with CIRMs Table 2 compares the results obtained by CIRMminer and CRMminer across the three datasets for different values of support. As expected, the number of discovered CIRMs is significantly smaller than the number of CRMs. For the *GT* dataset, as the support threshold decreases from 70 to 30, the number of valid CRMs increases by 137 times. In case of CIRMminer, as the support threshold decreases, the number of valid CIRMs increases by 45 times. The runtime to discover CIRMs is lower by 24 to 938 times than mining CRMs as the support threshold decreases. Even though additional computations are needed for the induced isomorphism check, CIRMminer enumerates a fraction (i.e., the induced ones) of all frequent CRMs resulting in a significant reduction in the overall execution time. Similar output and runtime ratios are observed for the *DBLP* and the *Sales* datasets.

7.3 Qualitative Analysis

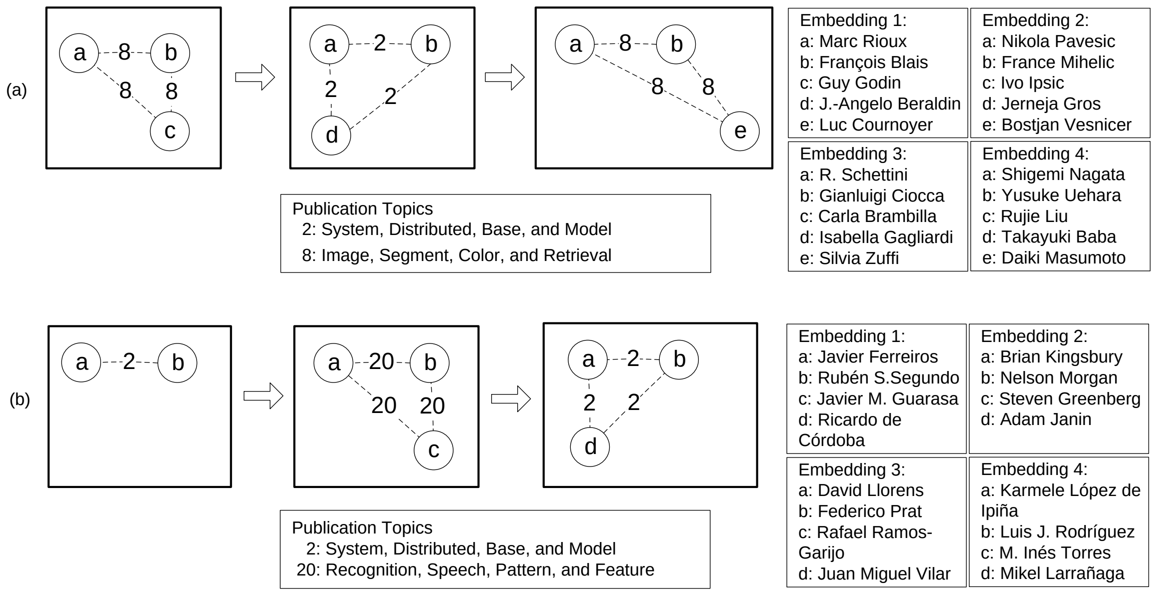
DBLP Case Studies To illustrate the types of relational changes found by CIRMminer on the *DBLP* dataset, Figure 6 shows two identified CIRMs that capture the frequent relational changes that are thematically different. The first CIRM shows the periodic changes in research topics represented as 8 and 2 and the topic similarity between these topics is 0.15. The CIRM captures the periodic transitions of the relations as author a and b collaborate with other authors c, d, and e over the time. The embeddings show relations among the four different sets of authors. The second CIRM also shows periodic changes in research topics represented as 2 and 20 and the topic similarity is 0.15.

Classification of Multivariate Time Series The authors of [1] showed that the CRMs can be used as features for building a predictive model to classify the production runs in the *GT* dataset into low and high-yielding runs. Specifically, out of the 247 runs, they identified 48 low yielding runs, and 48 high yielding runs, and then used CRMminer algorithm to find frequent CRMs and showed that most of these frequent CRMs are supported by the networks associated with the high yielding runs.

To see if CIRMs can also be used as features for this classification task, we perform a similar analysis as that reported in [1]. Figure 7 shows the class distribution of the embeddings for the discovered CIRMs and CRMs. It shows that the CIRMs are also present mostly in the high yield runs, since more than 75% of the embeddings belong to the Good class (Good_ic). Note that the ratio of the embeddings supporting the Good class remains consistent between CRMs (Good_c) and CIRMs (Good_ic). This analysis illustrates that 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.

8 Conclusion

We presented coevolving induced relational motifs to capture patterns that focus on identifying all relations between the set of entities and how that complete set of relations change in a consistent way across different snapshots of the network. The algorithm efficiently



Publication Topics
2: System, Distributed, Base, and Model
8: Image, Segment, Color, and Retrieval

Embedding 1:	Embedding 2:
a: Marc Rioux	a: Nikola Pavesic
b: François Blais	b: France Mihelic
c: Guy Godin	c: Ivo Ipsic
d: J.-Angelo Beraldin	d: Jerneja Gros
e: Luc Cournoyer	e: Bostjan Vesnicer

Embedding 3:	Embedding 4:
a: R. Schettini	a: Shigemi Nagata
b: Gianluigi Ciocca	b: Yusuke Uehara
c: Carla Brambilla	c: Rujie Liu
d: Isabella Gagliardi	d: Takayuki Baba
e: Silvia Zuffi	e: Daiki Masumoto

Publication Topics
2: System, Distributed, Base, and Model
20: Recognition, Speech, Pattern, and Feature

Embedding 1:	Embedding 2:
a: Javier Ferreiros	a: Brian Kingsbury
b: Rubén S.Segundo	b: Nelson Morgan
c: Javier M. Guarasa	c: Steven Greenberg
d: Ricardo de Córdoba	d: Adam Janin

Embedding 3:	Embedding 4:
a: David Llorens	a: Karmele López de Ipiña
b: Federico Prat	b: Luis J. Rodríguez
c: Rafael Ramos-Garijo	c: M. Inés Torres
d: Juan Miguel Vilar	d: Mikel Larrañaga

Figure 6: Two CIRMs capturing co-authorship patterns. The edge labels represent the domain of the publications. The vertices are labeled to identify the authors in an embedding. These CIRMs are collected using $\phi = 100$ and $\beta = 0.50$.

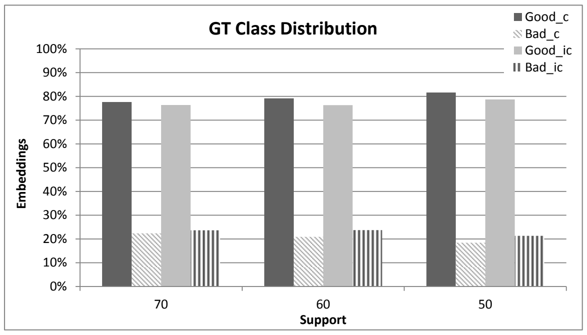


Figure 7: A distribution of the CIRM embeddings. The Good class represents the production runs with high yield and the Bad class represents with poor yield. CIRMs were collected using $\phi =$ (70, 60 and 50), $\beta = 0.50$, $m_{min} = 3$, $k_{min} = 3$, and $k_{max} = 8$.

handles the additional complexity of ensuring induced isomorphism and allows the anchored CIRMs to grow beyond the initial size. Using three real world datasets, the experimental evaluation shows the efficiency and scalability of the algorithm. Further, the qualitative analysis shows that the fewer induced evolving patterns were able to capture same level of characteristics of the underlying network as the arbitrarily evolving patterns.

References

[1] R. Ahmed and G. Karypis. Algorithms for mining the coevolving relational motifs in dynamic networks. Tech. Report 14-008, Dept. of Computer Sci. & Eng., Univ. of Minnesota, 2014.

[2] K. M. Borgwardt, H.-P. Kriegel, and P. Wackersreuther. Pattern mining in frequent dynamic subgraphs. In *IEEE ICDM*, pages 818–822, 2006.

[3] B. Bringmann and S. Nijssen. What is frequent in a single graph? *Advances in Knowledge Discovery and Data Mining*, pages 858–863, 2008.

[4] E. Desmier, M. Plantevit, C. Robardet, and J.-F. Boulicaut. Cohesive co-evolution patterns in dynamic attributed graphs. In *Discovery Science*, pages 110–124. Springer, 2012.

[5] A. Inokuchi and T. Washio. Mining frequent graph sequence patterns induced by vertices. In *Proc. of 10th SDM*, pages 466–477, 2010.

[6] R. Jin, S. McCallen, and E. Almaas. Trend motif: A graph mining approach for analysis of dynamic complex networks. In *ICDM*, pages 541–546, 2007.

[7] M. Kuramochi and G. Karypis. An efficient algorithm for discovering frequent subgraphs. *IEEE TKDE*, 16(9):1038–1051, 2004.

[8] H. Le, S. Kabbur, L. Pollastrini, Z. Sun, K. Mills, K. Johnson, G. Karypis, and W.-S. Hu. multivariate analysis of cell culture bioprocess data–lactate consumption as process indicator. *J. of Biotech.*, 2012.

[9] M. Ley. Dblp, computer science bibliography. Website, 2008. `www.informatik.uni-trier.de/\~ley/`.

[10] J. Pei, J. Han, B. Mortazavi-Asl, H. Pinto, Q. Chen, U. Dayal, and M. Hsu. PrefixSpan: Mining sequential patterns efficiently by prefix-projected pattern growth. In *ICDE*, pages 215–224, 2001.

[11] X. Yan and J. Han. gspan: Graph-based substructure pattern mining. In *IEEE ICDM*, pages 721–724, 2002.

[12] F. Zhu, X. Yan, J. Han, and S. Y. Philip. gprune: a constraint pushing framework for graph pattern mining. In *Advances in Knowledge Discovery and Data Mining*, pages 388–400. Springer, 2007.