Algorithms for Mining Evolving Patterns in Dynamic Relational Networks

A DISSERTATION
SUBMITTED TO THE FACULTY OF THE GRADUATE SCHOOL
OF THE UNIVERSITY OF MINNESOTA
BY

Rezwan Ahmed

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
Doctor of Philosophy

Dr. George Karypis, Advisor

September, 2014

© Rezwan Ahmed 2014
ALL RIGHTS RESERVED

Acknowledgements

I would like to begin by thanking Almighty for all his countless blessings and guidance at every step of my life.

I owe all that I have achieved in my graduate research to the constant support and motivation of my advisor Professor George Karypis. He has been a great mentor. I have always aspired to his exceptional work ethic. I am ever grateful to him for accepting me as a part-time student and for going above and beyond in helping me succeed in my research.

I would like to thank my wife, Bashi, and my daughter, Inaaya, for all their love, support, and inspiration. Bashi, thank you for all the sacrifices you made to support my plans, thank you for excusing me from all weekend duties so that I can work on my research, and thank you for always being there to help me get through the stressful times. Inaaya, thank you for brightening every day with your hugs and smiles.

My deepest gratitude goes to my father, Emaduddin Ahmad, and my mother, Monowara Sultana, for their love and countless sacrifices. Thank you for believing in me and encouraging me in my second attempt at building a career. My heartfelt thanks to my sisters, Sabrina and Farhana, for their love and support. Thank you to my friend and brother-in-law, Rezwanul Haque, for his constant support. Thank you to my in-laws, Mr. and Mrs. Raveendranathan, for their unconditional love and care. Thanks also to my brothers-in-law, Pratheep, Senthu, and Gajen, for keeping me smiling even during the most difficult times.

I would like to express my gratitude to Professors Daniel Boley, John Carlis, Gediminas Adomavicius, and Jaideep Srivastava for serving on my thesis committee and providing valuable feedback. A special thanks to Professor Raj Karim for his enthusiastic encouragement regarding my studies since I arrived in Minnesota. I am grateful to

all my colleagues at Boston Scientific for being supportive during my graduate studies.

I was fortunate enough to be among the intelligent and highly motivated students of the Karypis lab: Asmaa, Agoritsa, Chris, David, Dominique, Eva, Eugene, Fan, Huzefa, Jeremy, Kevin, Mohit, Santosh, Sara, Shaden, and Xia. Thank you all for your friendship, advice, and support over the years.

I would also like to thank the cooperative staff at the Department of Computer Science, the Digital Technology Center, and the Minnesota Supercomputing Institute at the University of Minnesota for providing state of-the-art research facilities and for assisting in my research.

Lastly, I wish to thank my all other friends and family, who are too numerous to name, for their support and prayers.

Dedication

To my caring parents, my loving wife, and my amazing daughter.

Abstract

Dynamic networks have recently being 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 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. Computational methods and tools that can efficiently and effectively analyze the changes in dynamic relational networks can substantially expand the types and diversity of dynamic networks that can be analyzed and the information that can be gained from such analysis. This may provide information on the recurrence and the stability of its relational patterns, help in the detection of abnormal patterns potentially indicating fraudulent or other malevolent behaviors, and improve the ability to predict the relations and their changes in these networks.

In this dissertation we present new data mining methods for analyzing the temporal evolution of relations between entities of relational 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.

We first present a new class of patterns, referred 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 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. At last, we define a new class of patterns built upon the concepts of CRMs, referred 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.

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 results from the experiments illustrate the efficiency of these algorithms. In addition, 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.

The new classes of evolving patterns and the algorithms can enable the effective analysis of complex relational networks towards the goal of better understanding of their temporal changes. Knowing these patterns provides strong observational evidence of the existence of mechanisms that control, coordinate, and trigger these evolutionary changes, which can be used to focus further studies and analysis.

Contents

List of Tables

List of Figures

Chapter 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 of some widely studied networks includes 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 [1], clustering [2], characterizing network evolution rules [3], detecting related cliques [4, 5], finding subgraph

subsequences [6], and identifying co-evolution patterns capturing attribute trends [7, 8] 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 the research work presented in this thesis is to develop new data mining methods that are designed to analyze and identify how the relations between the entities of complex 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.

1.1 Key Contributions

1.1.1 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.

In this thesis (Chapter 5), we present a new class of pattern that captures the evolution of the stable relational states in dynamic network. Our contribution is two fold. First, we introduce a new class of patterns referred as the evolving induced relational states (EIRS) that are designed to analyze the time-persistent relations or states between the entities of the dynamic networks. Second, we present an algorithm to efficiently mine all maximal non-redundant evolution paths of the stable relational states of a dynamic network.

We experimentally evaluate our algorithm using four real world datasets. First, we evaluate the performance and scalability of the algorithm on a large patent citation network and a co-authorship network by varying different input parameters. Second, we investigate some of the discovered evolving induced relational states from a trade network, an email communication network, a patent citation network and a co-authorship network and provide a qualitative analysis of the information captured in them. In addition, we introduce some interestingness measures to efficiently identify the evolving induced relational states that capture highest changes in their stable relations over the years.

1.1.2 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.

In this thesis, we introduce two new classes of dynamic graph patterns to identify recurring sets of entities whose relations change in a consistent way over time. The first class aims to find all coevovling patterns (i.e., evolving subgraphs) that meet the user specified minimum frequency threshold. The second class focuses on a subset of the first class to identify the coevovling patterns that take into account all relations (i.e., evolving induced subgraphs) among the groups of entities that have changed over time.

Mining All Coevolving Relational Motifs We introduce a new class of patterns (in Chapter 6), referred to as coevolving relational motifs (CRM), designed to capture patterns that change in a consistent way in a dynamic networks. CRMs identify consistent patterns of relational motif evolution that can provide valuable insights on the processes of the underlying networks. In addition, we present an algorithm, referred to as CRMminer, to efficiently mine a subclass of these patterns by identifying all frequent coevolving relational motifs such that the motifs that make up the CRM share at least one relation (i.e., an edge) that changes over time. Our algorithm follows a depth-first

exploration of the frequent CRM lattice and incorporates canonical labeling for redundancy elimination. We impose both frequency based and node overlap based constraints for pruning the search space to increase efficiency and an approximate pruning to reduce the complexity of our algorithm.

We provide a comprehensive evaluation of the performance of CRMminer and the usefulness of the discovered patterns. We experimentally evaluate the performance and scalability of CRMminer on a large co-authorship network, on a cell culture bioprocess network (multivariate time series data), and on a market sales network (multivariate time series data) by varying different input parameters. Furthermore, our experiments show that the approximate version of CRMminer is able to identify the majority of the CRMs while reducing the amount of time that is required. In addition, we investigate some discovered coevolving relational motifs from all three datasets and provide a qualitative analysis of the information captured in them. 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). In addition, we investigate the extent to which these discovered CRMs can lead to high quality features to build a predictive model. Our results, in the context of a bioprocess dataset, show that the discovered CRMs are present mostly as part of the high yield production runs.

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.

In this thesis (Chapter 7), a new class of dynamic graph patterns is presented that is built upon the early work for CRMs and focuses on all relations among a set of

nodes. 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 impose a frequency based constraint for pruning of the search space to increase efficiency and a node overlap based constraints for pruning of the output space to control the degree of change among the entities involved in a particular pattern.

We experimentally evaluate our algorithm on dynamic networks derived from three real world datasets: a large co-authorship network, a cell culture bioprocess network (multivariate time series data), and a market sales network (multivariate time series data). 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 significantly reduce the number of discovered patterns and the corresponding runtime reduced three orders of magnitude compared to CRMminer. We also provide a qualitative analysis of the information captured by the discovered CIRMs. We show that CIRMs can capture relational changes that are highly unlikely. Our analysis in the context of a bioprocess network shows that the small number of discovered CIRMs can equally capture network characteristics as CRMs.

1.2 Outline

- Chapter 2 provides definitions and notation used throughout this thesis.
- Chapter 3 presents different sources of dynamic networks and provides summary of research focused on mining patterns on static and dynamic networks.
- Chapter 4 describes different dynamic network datasets that were derived from real world applications to be used in experimental evaluation of different mining algorithms presented in this thesis.

- In Chapter 5, a new class of pattern (EIRS) that captures the evolution of the stable relational states and corresponding mining algorithms are presented.
- Chapter 6 presents a new class of dynamic graph patterns (CRM) to identify recurring sets of entities whose relations change in a consistent way over time. It also describes an algorithm to efficiently mine all CRMs and provides comprehensive evaluation of the performance of the mining algorithm.
- Chapter 7 describes a new class of dynamic graph patterns that is built upon the work for CRMs and focuses on all relations among a set of nodes and provides an algorithm to efficiently find all such patterns. It also presents experimental evaluation of the mining algorithm in term of the performance and scalability.
- In Chapter 8 a set of discovered patterns (EIRS, CRMs, and CIRMs) are investigated to provide a qualitative analysis of the information captured in them.
- Chapter 9 summarizes the conclusions of the research presented in this thesis and some future research directions.

1.3 Related Publications

The work presented in this thesis has been published in leading journals and conferences in data mining and knowledge discovery. The related publications are listed as follows:

- **Rezwan Ahmed** and George Karypis. Algorithms for Mining the Evolution of Conserved Relational States in Dynamic Networks. In *Proceedings of 11th IEEE International Conference on Data Mining*, pages 1–10, 2011.
- **Rezwan Ahmed** and George Karypis. Algorithms for Mining the Evolution of Conserved Relational States in Dynamic Networks. In *Knowledge and information systems*, volume 33(3), pages 603–630, Springer, 2012.
- **Rezwan Ahmed** and George Karypis. Algorithms for Mining the Coevolving Relational Motifs in Dynamic Networks. Technical Report 14-008, University of Minnesota, 2014.

- **Rezwan Ahmed** and George Karypis. Algorithms for Mining the Coevolving Relational Motifs in Dynamic Networks. In *The Transactions on Knowledge Discovery from Data*, 2014. (Under Review)

- **Rezwan Ahmed** and George Karypis. Mining Coevolving Induced Relational Motifs in Dynamic Networks. In *Proceedings of The IEEE International Conference on Data Mining*, 2014. (Under Review)

Chapter 2

Definitions and Notation

A relational network is represented via labeled 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 ($|V| = |L[V]|$), and a set of edge labels $L[E]$ modeling the type of the relations ($|E| = |L[E]|$). The labels assigned to the vertices (edges) are typically not unique and multiple vertices (edges) can have the same label. The relations between entities can either have a direction or not, leading to directed or undirected edges. 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 $u \in V''$, $(u, v) \in E''$.

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 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 maximal time interval $s{:}e$ if it is present in all snapshots $G_s, \ldots, G_e$ with the same label l and it is different in both G_{s-1} and G_{e+1} (assuming $s > 1$ and $e < T$). 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 $e_i \leq s_{i+1}$. In addition, we define the *span sequence* of a vertex as the sequence of maximal-length time intervals in which the vertex has at least one edge incident on it. The span sequence of the vertex will be represented as a sequence of time intervals in a similar fashion that of an edge.

A *persistent dynamic network* $\mathcal{N}^\phi$ is derived from a dynamic network $\mathcal{N}$ by removing all the edges from the snapshots of $\mathcal{N}$ 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 $\mathcal{N}^\phi$ can be derived from $\mathcal{N}$ 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 $f : A \to B$ such that $\forall a, b \in A$, if $f(a) = f(b)$, then $a = b$. A function $f : A \to B$ 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 $f : X \to Y$ and $g : Y \to Z$, then *composite* function $g \circ f : X \to Z$.

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 Figure 2.1. 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 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

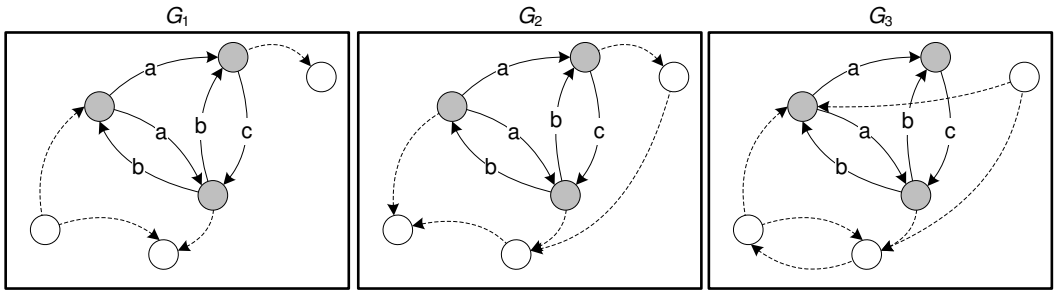

Figure 2.1: Examples of relational states.

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. In Figure 2.2, 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). The set of nodes that support the multiple occurrences of a relational motif do not need to be the same. 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). We will use M to denote a relational motif and the underlying subgraphs will be denoted by the tuple $(N, A, L[N], L[A])$, where N is

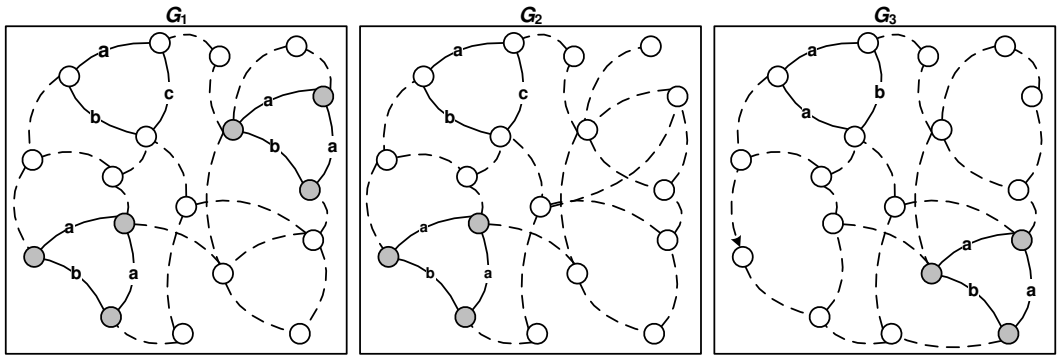

Figure 2.2: Examples of relational motifs. The shaded nodes that are connected via the labeled edges corresponds to a relational motif.

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.

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 states 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 motifs 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).

Table 2.1: Notations used throughout the thesis

Notation	Description
G	A labeled graph
V	Vertices of a graph
E	Edges of a graph
$L[V]$	A set of vertex labels
$L[E]$	A set of edge labels
$\mathcal{N}$	A dynamic network
$\langle G_1, G_2, \ldots, G_T \rangle$	A finite sequence of graphs
G_t	A labeled graph capturing the state of the system at time t
$\mathcal{N}^{\phi}$	A persistent dynamic network
S	An induced relational state
$g(S_i)$	An induced subgraph corresponding to an IRS S_i
G^{RS}	An induced relational state graph
N	A set of vertices corresponding to a CRM
M	A relational motif
$\boldsymbol{l_u}$	A vector containing the labels of vertex u of a CRM
$\boldsymbol{l_{u,v}}$	A vector containing the labels of edge (u, v) of a CRM
ϕ	A user defined minimum support threshold
β	An inter-state similarity or inter-motif overlap threshold
m_{min}	A minimum number of motifs per CRM
k_{min}	A minimum size (i.e. number of vertices/edges) of a pattern
k_{max}	A maximum size (i.e. number of vertices/edges) of a pattern
ω	A missing vertex or edge label
T_c	A DFS tree of a CRM C
$E_{T_c,fw}$	The forward edge set of DFS tree T_c
$E_{T_c,bw}$	The backward edge set of DFS tree T_c
$\#T_{\text{CRM}}$	The total number of discovered CRMs
$\#Q_{\text{CRM}}$	The number of CRMs that meet the β threshold (valid CRMs)
#A	The number of anchors
T_{CIRM}	The total number of discovered CIRMs
Q_{CIRM}	The number of CIRMs that meet the β threshold (valid CIRMs)

Chapter 3

Background and Related Work

Networks are generic models used in various applications from different domains to model the relations between various entities. Examples of some widely studied networks includes the friends-networks of popular social networking sites like Facebook and Myspace [9], the Enron email network [10, 11, 12, 1, 13], phone calling networks [14, 15, 16], co-authorship and citation networks [17, 18, 19, 20, 16], web-page linking networks [21, 22, 23], protein-protein interaction networks [24, 25, 26], transcription regulatory networks [27, 28, 29, 30], and networks derived from the IMDB movie database [31, 32, 16]. Figure 3.1 presents a type of regulatory network extracted from low-throughput studies reported in the stem-cell literature. The network is created by combining data from 271 publications and contains cell-signaling and gene-regulatory links that can be direct or indirect.

The nodes of a network model the entities of a system and the edges model the relations between these entities. The relations can either represent direct interactions between the entities or they can represent indirect interactions often involving entities that are not explicitly present in these networks (e.g., in the case of co-authorship networks, the interactions are facilitated by the co-authored publications that are not included in the network). The types of entities and relations are usually captured by giving the nodes and edges some attributes in the form of labels and, in many cases, edges are also given a weight that models the strength of the corresponding relations. Depending on the richness of the set of labels assigned to the nodes and edges, the networks can either model simple or complex entities and relations. In *simple*

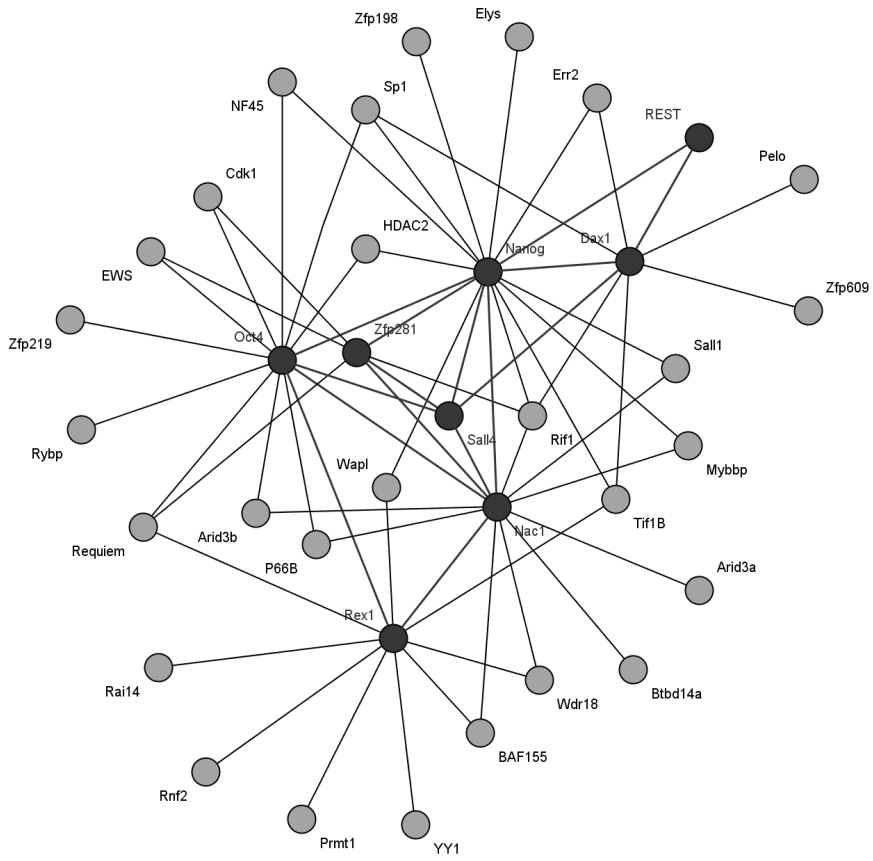

Figure 3.1: A regulatory network extracted by Huilei Xu [30] from low-throughput studies reported in the stem-cell literature.

relational networks, the sizes of the node- and edge-label sets are small (usually just one node and one edge label), whereas in *complex relational networks*, the sets of labels are considerably larger. Moreover, in complex networks the labels can be more complex data types such as sets, sets-of-sets, vectors, or matrices. Simple networks are used to model relatively uniform sets of entities and relations, whereas complex networks are used to model complex entities connected via complex relations. Many systems modeled via networks are dynamic in nature as the entities and relations that need to be modeled change over time. To model the dynamic characteristics present in such complex systems, the above *static* network model has been extended to a sequence of

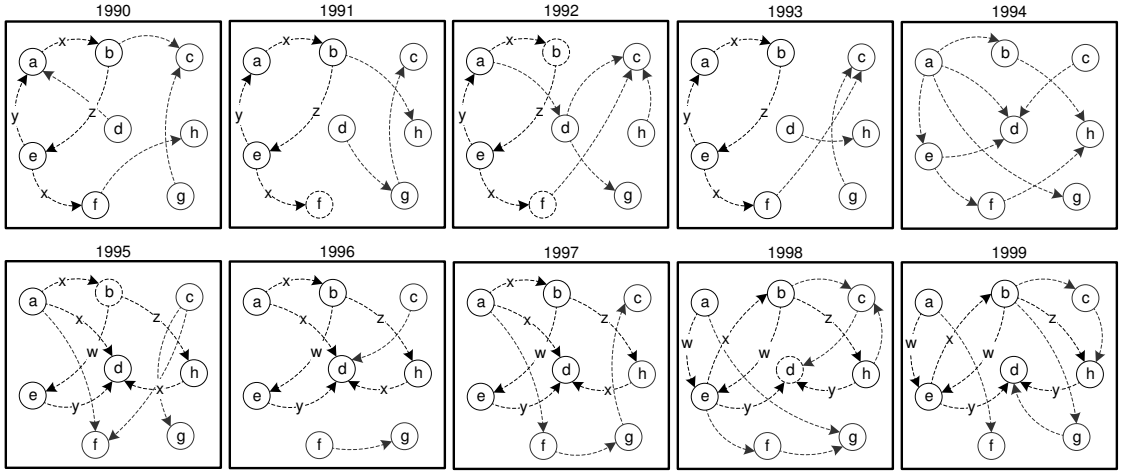

Figure 3.2: An example of a dynamic network that captures the changes of the entities and their relations over time. A sequence of networks each one representing the state of the system at a given time (snapshot).

networks [2, 33, 34, 35, 36, 37, 38, 39, 1, 7], each one representing the state of the system at a given time (snapshot). Figure 3.2 presents a dynamic network that captures the changes of the entities and their relations over time. Depending on the underlying domain and application, the snapshots in these *dynamic* networks either represent an *aggregate* network that contains all relations up to the current time point (e.g., co-authorship network) [17, 18, 19, 20, 16] or a *differential* network that contains only the relations between successive snapshots (e.g., phone calling networks) [14, 15, 16, 40].

3.1 Sources of Dynamic Relational Networks

There are a number of applications in different domains that give rise to dynamic relational networks. This section describes five classes of such networks in order to illustrate the broad applicability of the various methods that will be developed. These classes of networks and the underlying domains will also serve as the primary sources for the networks that will be used to validate the applicability and assess the performance

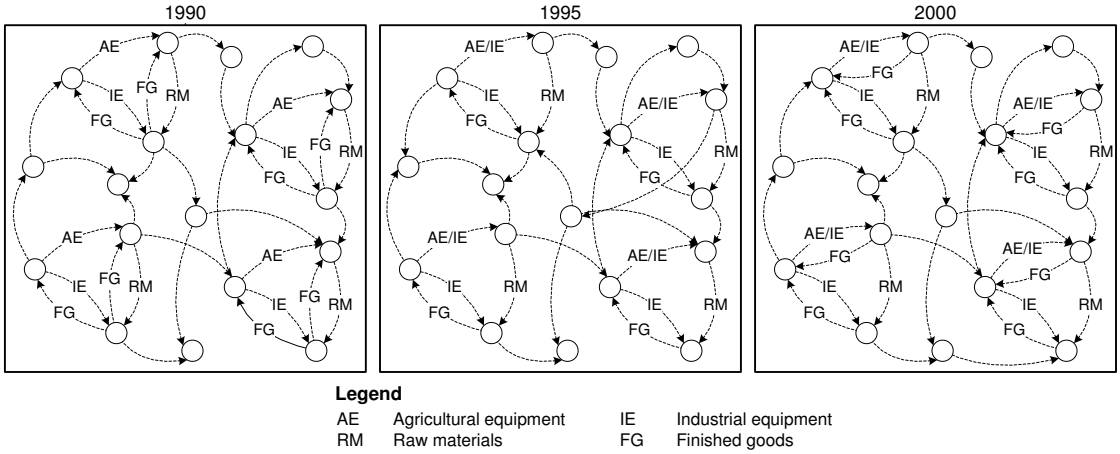

Figure 3.3: A hypothetical country-to-country trading network where labels represent the commodities been traded.

of the proposed methods both. The diversity of these networks in terms of their size, relational complexity, and span of time will facilitate the evaluation of different aspects of the proposed methods.

3.1.1 Trading Networks

These networks are used to model the trading of goods among a set of entities. The nodes model the trading entities (e.g., countries, states, businesses, individuals) and directed edges model the transfer of goods from one entity to another. The node-labels model information describing various problem- and application-relevant characteristics of the entities (e.g., political system, GDP, population age-distribution, industry group) and the edge-labels model various characteristics describing the goods being traded (e.g., commodity types, quantity). The dynamic nature of these complex networks arise from the fact that in most cases, the trading partners and the traded goods change over time, leading to a sequence of network snapshots each modeling the trading activity over a certain period of time. Figure 3.3 presents a hypothetical country-to-country trading network that 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[41].

3.1.2 Communication Networks

These networks are used to model the communication of information among a set of entities over periods of time. Examples of such networks include email networks, Internet traffic, and telephone calls. The nodes and their labels model the entities and their various characteristics, whereas the directed edges model communication of information between the entities. The edge-labels model the information that is being communicated (e.g., distribution of the themes in the emails or the network protocols used to carry the network traffic), which often is high-dimensional.

3.1.3 Health-care Networks

This includes various networks that can be created by modeling doctor-patient data over periods of time. One such example is the doctor-doctor network that can be formed by taking into account their common patients in successive periods of time. In this network, the nodes will correspond to the doctors and two doctors will be connected via an edge if they share a sufficient number of patients. The node-labels will model information about the doctors (e.g., practice type and areas, medical specialty, training, health-care networks, etc.), whereas the edge-labels will model information about their common patients (e.g., current and prior medical conditions, treatments, medications, demographic information, etc.). Note since the edge-labels describe characteristics of a population of individuals, the labels will be in the form of distributions. Other networks that can be derived include among others the doctor-treatment, treatment-treatment, and medical condition-medical condition networks.

3.1.4 Authorship Networks

This includes various networks that can be created by modeling authorship data such as scientific publications and blogging and includes various networks such as co-authorship, co-conference, and co-posting that have been previously studied by the data mining research community[17, 18, 19, 20, 16]. In this project information about the characteristics of the networks' entities and the characteristics about the co-authored content

or publication venues will be used to derive the node and edge labels.

3.1.5 Citation Networks

This includes various directed networks that can be formed based on the references that occur in various publications. The nodes in these networks correspond to individual publications or their aggregates along different dimensions. For example, in a citation network based on US patents, the nodes can correspond to the individual patents, the art areas associated with the patents based on USPTO's classification, or the patents' assignees. The edges will correspond to the individual citations or their aggregations when the nodes represent sets of patents. Depending on the network, the node- and edge-labels can model information such as the assignees' industry areas or the patents' art areas.

3.2 Review of Relevant Literature

3.2.1 Pattern Discovery in Static Graphs & Networks

Over the last ten years, considerable research effort has been devoted to developing algorithms to find patterns in graphs and networks. This research has resulted in the development of algorithms for finding different types of patterns such as paths [42, 43], trees [44, 45, 46], induced subgraphs [47], arbitrarily connected subgraphs [48, 49, 50, 51, 52], and various types of cliques [53, 54, 55, 56]. While most of these methods have been developed for mining databases containing relatively small graphs, algorithms have also been developed to identify subgraphs with a large number of embeddings in a single large graph (i.e., network) [57, 58, 59, 60]. Methods using heuristics to select relevant patterns based on criteria other than occurrence frequency, such as the minimum description length [61, 62], have also been investigated. A consequence of all this research is that there are now a wide range of methods available to efficiently mine large graph databases and networks to identify frequently occurring graph patterns. A brief overview of some state of the art algorithms related to frequent subgraph mining on static graphs is provided below.

The AGM [47, 63] algorithm was first designed to find frequent induced subgraphs

and later extended to find arbitrary subgraphs. It starts with frequent single-vertex subgraphs and grows the frequent subgraphs one vertex-at-a-time using a breadth first approach. At each iteration of expansion, two k-vertex frequent subgraphs are joined when they share the same $(k-1)$-vertex subgraph. Canonical labeling scheme based on vertex sorted adjacency matrices is used to determine similar subgraphs. The FSG [64, 52] algorithm also uses a breadth-first approach to find all frequent arbitrary connected subgraphs. It grows the frequent subgraphs one-edge-at-a-time and two k-edge subgraphs are merged if they share the same $(k-1)$-edge subgraph, called the *core*. FSG achieved significant improvement in runtime compared to AGM by introducing efficient canonical labeling scheme, efficient pattern growth algorithm, and efficient frequency counting technique.

Though the breadth-first approach provides a fast and complete solution, these methods consumes high memory space for massive graph datasets. More recently, DFS-based approaches have been introduced to overcome this limitation. The gSpan [65, 66] algorithm finds all frequent arbitrary connected subgraphs following a depth-first approach and uses an efficient canonical labeling scheme based on the depth-first traversal. gSpan was able to maintain a low memory footprint and outperformed FSG on all types of graph data.

Similar to gSpan, there are other DFS approaches that find all frequent arbitrary connected subgraphs from graph databases, such as MoFa [67], FFSM [68], SPIN [69], and Gaston [70]. Most of these algorithms are motivated by different DFS based sequence and tree mining algorithms such as PrefixSpan [71], TreeMinerV [44], and FREQT [45]. Among these algorithms, Gaston is the fastest algorithm due to smart partitioning of the problem space. Gaston divided and ordered the frequent subgraph mining problem into frequent path mining, then frequent subtree mining, and at last frequent subgraph mining. since enumeration of paths and trees are much more efficient, it operates on the expensive subgraph enumeration last. To eliminate unnecessary isomorphism checks, it stores all embeddings so that it grows patterns that actually appear.

Another group of algorithms focus on mining frequent subgraphs in a large single graph such that the subgraphs have at least t embeddings in that large graph. SUBDUE [72, 62, 73] is one of the first and well-known algorithm for finding recurring

subgraphs in a single large graph. It is an approximate algorithm that tries to minimize the minimum description length (MDL) of a graph by compressing the frequent subgraphs and uses a heuristic beam search to narrow the search space. GREW [57] is another heuristic algorithm that finds connected subgraphs with large number of vertex disjoint embeddings in a large single graph. It iteratively discovers frequent subgraphs by first identifying vertex disjoint embeddings of subgraphs that were frequent in previous iterations and merging certain subgraphs that are connected to each other via one or multiple edges. The iterative merging process continues until it can no longer merge candidate subgraphs. The hSIGRAM and vSIGRAM [74, 58] algorithms find all frequent subgraphs in a large sparse graph. These algorithms build a lattice of frequent subgraphs where hSIGRAM builds the lattice in breadth-first manner and vSIGRAM build it in depth-first manner. The goal is to identify the various nodes (i.e., subgraphs) of this lattice and the frequency of the associated subgraphs. The support of each subgraph is determined by the overlap graph based MIS measure and different MIS measures are used, such as exact and approximate MIS measures. Experimental results show that vSIGRAM was faster than hSIGRAM and both algorithms outperformed SUBDUE.

3.2.2 Pattern Discovery in Dynamic Networks

Since dynamic networks have only recently emerged as an important research area, there is a growing research on formally defining important recurrent structural patterns and developing algorithms for their identification. We provide a brief overview of different pattern mining methods focusing on dynamic network, since the research presented in this thesis is closely related to this topic.

Desikan et. al. [75] analyzed the importance of mining temporally evolving Web graphs, but did not provide any algorithmic solution for detecting the stable patterns or their evolution. Borgwardt et. al. [1] introduced the notion of *dynamic subgraph*, which extends the traditional notion of the subgraph to include the sequence of subgraphs that exist in a consecutive sequence of snapshots and developed algorithms to identify the set of dynamic subgraphs that occur frequently in a dynamic network. Given a sequence G_{ts} of n graphs $G_1, \ldots, G_n$, the *dynamic graph* was defined as $DG(G_{ts}) = (V_{DG}, E_{DG}, L[V_{DG}], es)$, where $V_{DG} = V_i$ for all $1 \leq i \leq n$, $E_{DG} = \cup_{i=1}^{n} E_i$, L as

the labels of V_{DG} and the mapping $es : E_{DG} \rightarrow \{0|1\}^n$ maps each edge e in E_{DG} to a binary string $es(e)$ of length n. A graph $DSG_1 = (V_1, E_1, L_1, es_1, start_1)$ is a *dynamic subgraph* of length k of a dynamic graph $DG_2 = (V_2, E_2, L_2, es_2)$ if $V_1 = V_2$, $E_1 = E_2$, $L_1 = L_2$, and for all corresponding edges e_1 and e_2 from E_1 and E_2: $es(e_1) = substr(es_2(e_2), start_1, start_1 + (k-1))$, where $start_1$ is the state of DG_2 in which DSG_1 starts with $1 \leq start_1 \leq n-(k-1)$. The dynamic subgraph DSG_1 qualifies as a frequent dynamic subgraph (FDS) of DG_2 if DG_2 contains at least t identical embeddings of DSG_1, where t is a user defined frequency threshold parameter. This algorithm extends the GREW [57] algorithm to detect the frequent dynamic subgraphs (FDS). ENRON [13] email communication dataset was used to evaluate the algorithm.

Jin et. al. [7] focused on the problem of finding recurrent patterns in dynamic networks in which a time series is associated with each node as its weight and the topology of the network remains same. They introduced the notion of the *trend motif*, which is a connected induced subgraph in which each node's time series exhibits a consistent increasing/decreasing trend over a time interval. Give a graph G, a trend motif $G^f(V_s)$ is defined as $(V_s, [t_s, t_e], f)$ with $(t_s < t_e)$, where $G(V_s)$ is a connected induced subgraph of G, f is a function $F : V_s \rightarrow \{+, -\}$, and $[t_s, t_e] = [t_s^1, t_e^1] \cap [t_s^2, t_e^2] \cap \cdots \cap [t_s^n, t_e^n]$, where $[t_s^i, t_e^i]$ is a maximal interval of trend for vertex $v_i \in V_s$, and n is the number of vertices in V_s. To reduce the search space and find frequent motifs, they use three user defined parameters and define a motif as $G_s^f(l, w.\theta)$, where l is the minimal internal length for a trend interval of each vertex in the motif, w is the minimal length for the intersection of the motif occurrence and θ is the minimum number of distinct subset of vertices that are equivalent to G_s^f. To evaluate their algorithm they used protein interaction network for Yeast and its microarray expression data to capture time series, financial stock market and normalized GDP data.

Lahiri et. al. [76] proposed a mining problem of finding periodic or near periodic subgraphs in dynamic networks. They combine two aspects; i) frequent pattern mining in transactional and graph databases, and ii) periodic pattern mining in unidimensional and multidimensional sequences. They employ a ranking measure *average purity* to express how likely it is that a periodic subgraph embedding occurs only at it's periodically predictable timestep. The algorithm uses a pattern tree that maintains information about all patterns that are either currently periodic or could become periodic at a future

timestep. As each new timestep is read, the pattern tree is traversed and updated. Any patterns that are no longer periodic are flushed and new periodic patterns are added. They used ENRON, Plains Zebra, Reality Mining and IMDB Celebrities data set to analyze inherent periodicity.

Berlingerio et. al. [3] provided an algorithm to detect frequent connected subgraphs in time-evolving graphs for deriving graph-evolution rules that satisfy a minimum confidence constraint. They assume that as the graph evolves, the nodes and edges are only added and never deleted. To calculate the confidence for a certain pattern, they decomposed each pattern into sequence of steps (head $\rightarrow$ body) and measured the confidence for each transition. The algorithm, called GERM, is an adaptation of gSpan and uses minimum-image-based support computation. They used Flickr, Y!360, DBLP and arXiv datasets to evaluate the algorithm and collect interesting rules.

Robardet [5] represented the frequent patterns of a graph as *pseudo-cliques* where the density of the edges among the vertices of an induced subgraph is above a user defined threshold and proposed an algorithm that first mines each graph snapshot of a dynamic graph for local patterns and then combines these with patterns from previous snapshot based on some constraints to form evolving patterns. To reduce the search space, the algorithm selects only the maximal and isolated patterns that have less number of external links per vertex. Five basic temporal relationships between a pair of subgraphs from consecutive time stamps: Stability, Growth, Diminution, Extinction and Emergence. They use two dynamic sensor network and a dynamic mobility network to evaluate the algorithm.

Wackersreuther et. al. [77] defined a framework to perform frequent subgraph discovery in dynamic networks. This approach is very similar to [1] and uses suffix trees. The algorithm first finds frequent subgraphs in the union graph of a time series of graphs and then searches the resulting static frequent subgraphs for frequent dynamic patterns. The frequent static subgraphs are canonicalized to be represented in string format and the string algorithm is used to determine the longest common substrings to classify as dynamic pattern. They use Protein Protein Interaction network data from yeast and a time series of yeast gene expression levels to evaluate the algorithm.

Sun et. al. [78] presented a parameter-free framework, called GraphScope that finds communities on time-evolving graphs along with the change-points. This algorithm

is based on Minimum Description Length (MDL) principle. Given a dynamic graph, GraphScope finds a good partition of source and destination nodes to summarize the fundamental community structure and also incrementally constructs graph segments by selecting good change points. Experimental results are shown based on University network traffic, ENRON email traffic, Cell phone call logs and Bluetooth network datasets.

You et. al. [79] analyzed a sequence of graphs to discover graph rewrite rules that capture the changes that occur between pairs of graphs in the sequence and then mined the graph rewrite rules to learn general transformation rules that capture repeated patterns in the rewrite rules. The algorithm uses MDL principle to detect maximum common subgraph between two consecutive graphs while identifying graph rewrite rules and also used to find common frequent patterns on graph rewrite rule to learn general transformation rules. They used biological network data (KEGG pathways data in combination with artificially generated time-series data and microarray data) to build dynamic graphs to evaluate their algorithm.

Inokuchi et. al. [80] presented a framework to mine a complete set of frequent subsequences embedded in a given set of observed graph sequences. They proposed a graph grammatical framework that compactly describes a graph sequence by introducing graph transformation rules to capture gradual changes such as insertion, deletion, and relabeling of vertices and edges between consecutive graph snapshots. Then they mined subsequences called frequent transformation subsequences *FTS*s represented by the transformation rules of the graph sequence data. They presented a mining tool, called GTRACE, to enumerate relevant FTSs from the graph transformation rules. The experiments were done on synthetic data and real work data (ENRON, MIT Reality Mining).

Liu et. al. [81] analyzed the problem of spotting significant changing induced subgraphs in an evolving graph. They propose an incremental algorithm to compute the neighborhood random walk distance to measure the vertex closeness and use the closeness measure to further define a vertex importance score. The significant changing subgraphs are defined to contain all the vertices above a certain threshold and most of the vertices whose closeness with the important vertices change a lot. Regions are expanded using an extension of density clustering algorithm that include the vertices whose closeness difference with the important vertices are high. Experimental results

are captured using DBLP co-authorship dataset and ENRON dataset.

Shoubridge et. al. [82] surveyed various graph distance measures that are sensitive to detect abnormal changes between two graph snapshots. They also presented two methods to identify regions of significant change. First method, called *symmetric difference*, constructs a change matrix between two graphs and permutates it to rank all vertices in ascending order of the number of network topology change event experienced by individual vertices. Second method constructs k-neighborhoods for each vertex and compare neighborhood graph distances using a graph distance measure to track significant changes.

Chan et. al. [83] presented an algorithm, called *cSTAG* (Clustering for Spatio-Temporal Analysis of Graphs), to discover the regions of correlated spatio-temporal change in evolving graphs and use these regions to characterize the events that caused them. It represents structural changes to the graph as binary waveforms where a change waveform is associated with each edge of the graph. Temporal correlation is measured using the distance between associated binary waveforms. Edges that have changed over time and are topologically connected are considered spatially close. They have provided two clustering solutions that use both spatial and temporal distance information to find regions and use a region association method to find regions with long term correlation. Both synthetic and real world data (network routing graph and website traffic network) are used to evaluate cSTAG.

Duan et. al. [84] presented an algorithm *Stream-Group* to solve community mining on dynamic weighted directed graph (DWDG). Their algorithm first constructs compact communities by computing graph's relevance matrix that indicates the degree of a node belonging to a community using Random Walk with Restart technique. Then it merges the compact communities along the direction of maximum increment of the modularity. At last a similarity measure between partitions of continuous time segments is devised to detect the change-points and an incremental algorithm updates graph segment partitions as the new graphs are added to the graph segment. This paper is similar to GraphScope [78]. They used both synthetic and real world (ENRON) datasets to evaluate their algorithm.

Inokuchi et. al. [6] presented the problem of finding frequent, relevant, and induced

subgraph subsequences, called *FRISS*s, from graph sequence data. These induced subgraph subsequences capture the changes of a subgraph over the subsequence. Their 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 induce 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 lot of short and trivial frequent sequences.

Bogdanov et. al. [85] presented the problem of finding the highest-scoring connected temporal subgraph, referred as Heaviest Dynamic Subgraph (HDS), in an edge weighted network whose weights change over time. HDS is an optimization problem where the scoring function is maximized over all possible subgraphs and all possible sub-intervals of the whole interval. They provided a filter-and-verify based algorithm, called MEDEN, that uses tight upper bounds of the optimal solution to prune the quadratic sub-interval space and an efficient heuristic for verifying promising sub-intervals. This algorithm is able to find regions of congestion in a large road network from Los Angeles.

Desmier et. al. [8] presented an algorithm to analyze and mine patterns from dynamic attributed graphs. Similar to Trend Motifs [7], they mine patterns, referred as cohesive co-evolution patterns, that are composed of tri-sets of vertices, timestamps, and signed attributes. These patterns identify sets of vertices that are similar from the point of view of their attribute values and of the vertices in their neighborhood. They presented an algorithm that mines all cohesive co-evolution patterns in a depth-first manner. Both synthetic and real world data (DBLP, Brazil landslide images) were used to evaluate their algorithm. Later authors presented an extension to their work in [86], where the algorithm, called MINTAG, mines maximal dynamic attributed sub-graphs that satisfy some constraints on the graph topology and on the attribute values. This paper applied several interestingness measures used in itemset mining techniques to the dynamic attributed graph settings and devised MINTAG to use these additional constraints to guide the search toward relevant patterns.

Mongiovì et. al. [87] introduced the problem of mining significant smoothly evolving processes in a dynamic network, where a process corresponds to a smoothly evolving subgraph in time that includes high-weighted edges. This problem is similar to the HDS problem [85], but focuses on subgraphs that evolve smoothly in time. They presented an

approximate algorithm, called LEGATO, that first filters the nodes that cannot be part of the optimal solution and then performs a heuristic search to locate evolving subgraph candidates and refines them to get the smooth solution. Synthetic and two real world datasets (Traffic and Wikipedia) are used to evaluate LEGATO performances.

3.2.3 Evolution of Dynamic Networks

A related body of research has investigated the task of identifying and tracking evolving communities in social networks. One of the problems studied in this field, known as evolutionary clustering[2, 33, 34], is to find clusters in a dynamic network in the form of snapshots, such that clusters at any given time groups similar entities while also preserving the grouping of entities in past snapshots. Closely related problems concern tracking the proximity of two entities in the network[35], detecting important events such as the merging or split of existing communities, as well as the formation of new ones [36], and identifying stable or persistent communities and their long-lasting members[37, 38, 39]. The methods developed for these problems have been used in a wide range of applications including disease modeling[88, 89], cultural and information transmission[90, 91, 92, 93], intelligence and surveillance[94, 95], business management[96, 97], conservation biology and behavioral ecology[98, 99]. Even though these methods provide valuable insights on the evolution of dynamic networks[93, 89, 91, 96, 95, 94, 88, 98, 99, 97], the nature of the evolution that they study is to a large extent orthogonal to those that is studied in this thesis.

Chapter 4

Dynamic Network Datasets

4.1 Datasets used for EIRS Evaluation

We evaluated the algorithm for mining EIRSs using datasets from a patent citation network, a trade network, an email communication network and a co-authorship network. The scalability of the algorithm was assessed on the patent citation dataset and co-authorship dataset whereas all four datasets were used in the qualitative assessment of the identified EIRSs.

Patents Citation Network This is a citation network derived from the United States Patent and Trademark office's (USPTO) bibliographic information for the patents

Table 4.1: Dynamic Network Datasets.

Dataset	#Vertices	Avg. #Edges	Span
Patent N2	84152	45465	34
Patent N3	84152	63633	34
Patent N4	84152	79358	34
Trade	192	23	53
Enron	130	88	30
DBLP	1057524	99615	75

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

granted from 1976 to 2009. The nodes correspond to the primary art areas associated with the patents based on USPTO's classification and the edges correspond to aggregated citations between art areas. For example, if patent A of art area α cites patent B of art area β in year x (A's issue year), then a directed edge is added from α to β in the graph representing year x. Citations that produce self references are removed. The classification of USPTO art areas is hierarchical and forms a tree structure that can be up to 16 levels deep. The patents are assigned the classes corresponding to the leaf nodes of the tree. In our experiments, in order to obtain art areas that contain a sufficiently large number of patents, we rolled up the classification tree to the third level, and all the patents underneath each third-level node was assigned the class of that node. The snapshots of the dynamic network that we created correspond to the citation network of each year, leading to a dynamic network consisting of $2009 - 1976 = 34$ snapshots. Since the vertices at each snapshot can potentially be connected to all other vertices, we pre-processed each snapshot in order to derive a set of dynamic networks that contain the most important set of outgoing edges (i.e. references) from each node. This is done as follows. For each vertex of each snapshot, we first choose the 20 most frequent edges. The frequency of an edge (a, b) is defined as the number of patent-to-patent references (P_1, P_2) such that P_1's class is a and P_2's class is b. Then, for each edge (a, b) we calculate its lift (i.e., $w(a, b) = p(b|a) \backslash p(b)$) to use as its weight. Based on these weights, we construct three dynamic network datasets N2, N3 and N4 by selecting the highest weighted 2, 3 and 4 edges for each vertex in each snapshot of the network. The size and density of the networks is presented in Table 4.1.

Trade Network This is a trade network that models the yearly export and import relations of 192 countries from 1948 to 2000 based on the Expanded Trade and GDP Data [100]. 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 network that we created corresponds 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.

Email Communication Network This is a communication network that models the email traffic between employees of the ENRON company [13]. The nodes model the employees who are labeled by their rank and a node id to uniquely identify two nodes with same label (i.e. multiple employees with the same rank). The directed edges model the communications between the employees. To represent the dataset as a dynamic network, the entire time span was divided into 30 equal-size intervals (approximately 30 days). If an email was sent from node a to node b at a certain time interval, a directed edge $a \rightarrow b$ is added to the graph representing that time interval. This representation contains 130 nodes and about 90 edges per snapshot.

Co-Authorship Network This is a co-authorship network that models the yearly co-authorship relations from 1938 to 2012 based on the DBLP Computer Science Bibliography Data [101]. The nodes model the authors of the publications from various conferences, journals, books, lecture series, etc. and the undirected edges between the authors model the collaboration between two scientists at a certain year. If an author A co-authors a publication with an author B in a given year, the edge $A - B$ is added to that year's co-authorship graph. Multiple publications by the same authors in a year are counted as a single relation. The snapshots of the dynamic network that we created corresponds to the co-authorship network of each year, leading to a dynamic network consisting of $2012 - 1938 = 75$ snapshots. The edge distribution among the snapshots is skewed. The minimum edge count is 2, the maximum edge count is 1.2 million and the average is 99615.

4.2 Datasets used for CRM & CIRM Evaluation

We have used two different types of datasets to evaluate CRMminer. The DBLP co-authorship network is a real world dynamic network that captures yearly co-authorship relations. The bioprocess network (GT) and the sales network (Sales) datasets are based on multivariate time-series data. To characterize the relations among different variables and understand changes over time, we represent the time-series data as a dynamic network. The CRMs discovered from these networks can be used to characterize the overall network, as shown in Chapter 8.

Table 4.2: Dynamic Network Datasets.

Dataset	#Vertices	#Edges	Span	Avg. #Edges	Avg. Degree
DBLP	1,057,524	4,841,084	55	88,020	0.08
GT	3458	83,454	47	1776	0.51
Sales	2697	138,044	66	2092	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.

Co-Authorship Network (DBLP)

This is a co-authorship network that models the yearly co-authorship relations from 1958 to 2012 based on the DBLP Computer Science Bibliography Data [101]. The snapshots of the dynamic network that we created corresponds 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 from various conferences, journals, books, lecture series, etc. and the undirected edges between the authors model the collaboration between two scientists at a certain year. If an author A co-authors a publication with an author B in a given year, the edge (A, B) is added to that year's co-authorship graph. Multiple publications by the same authors in a year are counted as a single relation.

The edge distribution among the snapshots is skewed. In Figure 4.1, we show the edge distribution of the DBLP network. To determine co-authorship relations between authors with significant contribution, we removed authors who published less than 5 different years. We also removed all the relations of an author if he/she had co-authored with more than 50 other authors in a single year. To assign edge labels, we used the CLUTO software[1] to cluster the publication titles into 50 groups. These title groups can be thought as the research areas or publication topics that the authors focused on their collaboration.

[1] `http://www.cs.umn.edu/~cluto`

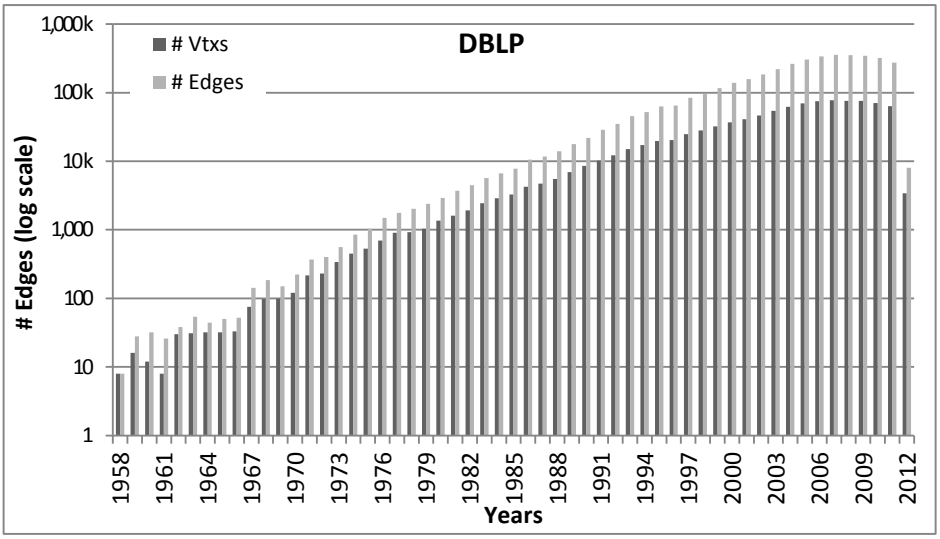

Figure 4.1: The edge distribution of the DBLP co-authorship network.

Bioprocess Network (GT)

This is a cell culture bioprocess data [102] captured from the production bioreactors at Genentech's manufacturing facility. This multivariate time-series data tracks the dynamics of various process parameters at every minute over 11 days period for a 247 production runs. In Table 4.3, we show the parameters that are used to construct the GT network. Note that some of the related variables are assigned the same vertex label. To represent this data as a dynamic network, we computed correlation matrices for 14 of the process parameters using a sliding window of 12 hours interval with 50% overlap. This process resulted in 47 correlation matrices per production run. 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 $+0.9$ or below -0.8 threshold. These threshold values are chosen to select equal number of positive and negative labeled edges. This way we construct 47 network snapshots where each snapshot contains a 3458 vertices (14 parameter * 247 runs) and the edges between them.

Table 4.3: Bioprocess Network Dataset.

Vertex label	Variable(s) Description
ASX	Air sparge rate, Air sparge total
BT	Base totalized
CO2	CO_2 sparge rate, CO_2 sparge total
DOX	Dissolved oxygen controller output, Dissolved oxygen primary
FRO	Flowrate overlay
O2X	O_2 sparge rate, O_2 sparge total
PHX	pH Controller output, pH Online
ST	Sparge total
WLC	Weight load cell

The vertex labels of the 14 parameters used to construct the Bioprocess Network Dataset (GT) dataset. The labels ASX, CO2, DOX, O2X, and PHX represent two parameters each.

Table 4.4: Sales Dataset.

Vertex label	Variable(s) Description
ANA	Analgesics
BER	Beer
CEX	Cereals, Oatmeal
CHE	Cheeses
CIG	Cigarettes
CRX	Cookies, Crackers, Snack Crackers
CSO	Canned Soup
DID	Dish Detergents
DTX	Laundry Detergents, Fabric Softeners
FEC	Front-end-candies
FRX	Frozen Dinners, Entree, Juices
GRO	Grooming Products
JUX	Bottled Juices, Refrigerated Juices
PTW	Paper Towels
SDR	Soft Drinks
SPX	Bath Soap, Soaps, Shampoos
TEX	Toothbrushes, Toothpastes
TNA	Canned Tuna
TTI	Bathroom Tissues

The vertex labels of the 29 parameters used to construct the Sales dataset. The labels CRX, FRX, and SPX represent three parameters each. The labels CEX, DTX, JUX, and TEX represent two parameters each.

Store Transaction Network (Sales)

This dynamic network is constructed using Dominicks Finer Foods store sales data collected from the James M. Kilts Center, University of Chicago Booth School of Business[2] . The data captures weekly store-level sales from 93 stores collected over a period of more than seven years (400 weeks). We considered the sales data as multivariate time-series data for each store where 29 variables are the different categories of the product sold. In Table 4.4, we show the different categories of the product considered in the Sales network. Note that some of the related variables are assigned the same vertex label. To represent this data as a dynamic network, we considered the total number of items sold per category in a week at a store and computed correlation matrices between 29 categories using a sliding window of 12 weeks interval with 50% overlap. This process resulted in 66 correlation matrices per store. To construct a network, 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 $+0.85$ or below -0.4 respectively. These threshold values are chosen to select equal number of positive and negative labeled edges. This way we construct 66 network snapshots where each snapshot contains a 2697 vertices (29 parameters * 93 stores) and the edges between them.

[2] `http://research.chicagobooth.edu/kilts/marketing-databases/dominicks/dataset`

Chapter 5

Mining the Evolution of Conserved Relational States

5.1 Evolving Induced Relational State (EIRS)

An example of the type of evolving patterns in dynamic networks that the work in this chapter is designed to identify is illustrated in Figure 5.1. 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, 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 one 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 chapter is designed to identify.

EIRSs identify entities whose relations transition through a sequence of time-persistent

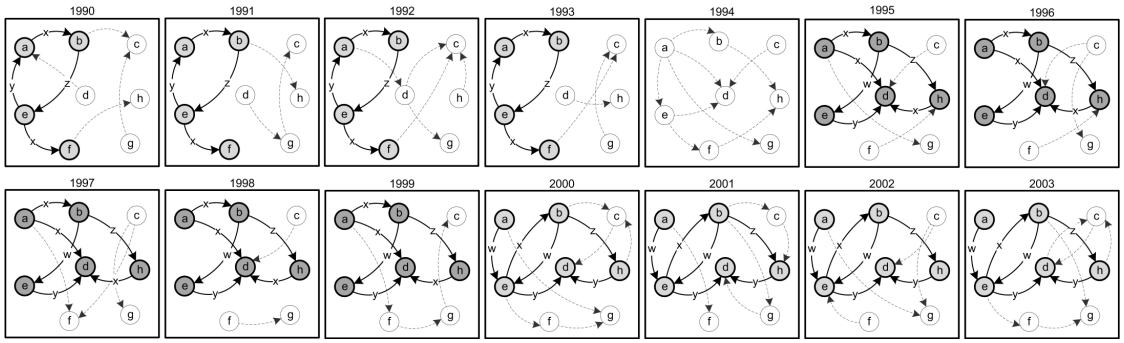

Figure 5.1: An example of an evolving relational state.

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 Chapter 8 for some examples of such patterns in a 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 chapter 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 $(1 \leq \phi \leq T)$*, a 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.

5.2 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.

5.2.1 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 [103]. 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.

$$F(V_i, V_f) =$$

$$\begin{cases} V_i, & \text{if } \text{sgadj}(V_i, V_f) = \emptyset \\ & \text{or } V_f = \emptyset \\ F(\{u\}, V_f \setminus u) \cup F(\emptyset, V_f \setminus u), & \\ \text{where } u \in V_f & \text{if } V_i = \emptyset \wedge V_f \neq \emptyset \\ F(V_i \cup \{u\}, V_f \setminus u) \cup F(V_i, V_f \setminus u), & \\ \text{where } u \in \text{sgadj}(V_i, V_f) & \text{otherwise,} \end{cases} \tag{5.1}$$

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 Equation 5.1 correctly generates the complete set of induced subgraphs, 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 (i) 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 (ii) 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 Equation 5.1 removes a vertex from V_f, the recurrence relation will terminate due to the first condition. Finally, since the three conditions in Equation 5.1 cover all possible cases, the overall recurrence relation is correct.

In addition to correctness, it can be seen that the recurrence relation of Equation 5.1 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 can be easily incorporated in Equation 5.1 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 Equation 5.1 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 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 5.2 illustrates the vertex addition process during an IRS expansion. Figure 5.2(a) shows a simple case of vertex addition where an IRS $S_1 = (\{a, b, e\}, 1{:}7)$ is

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

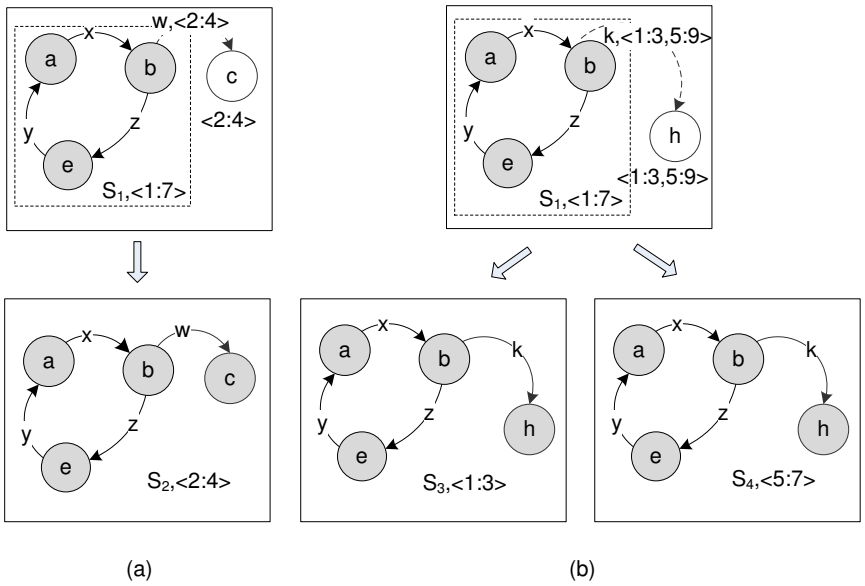

Figure 5.2: Adding a vertex to an induced relational state.

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 5.2(b) 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 $S_4 = (\{a, b, e, h\}, 5{:}7)$, 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 Equation 5.1 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 Figure 5.3. 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 Equation 5.1, this corresponds to the third condition and leads to the recursive calls of

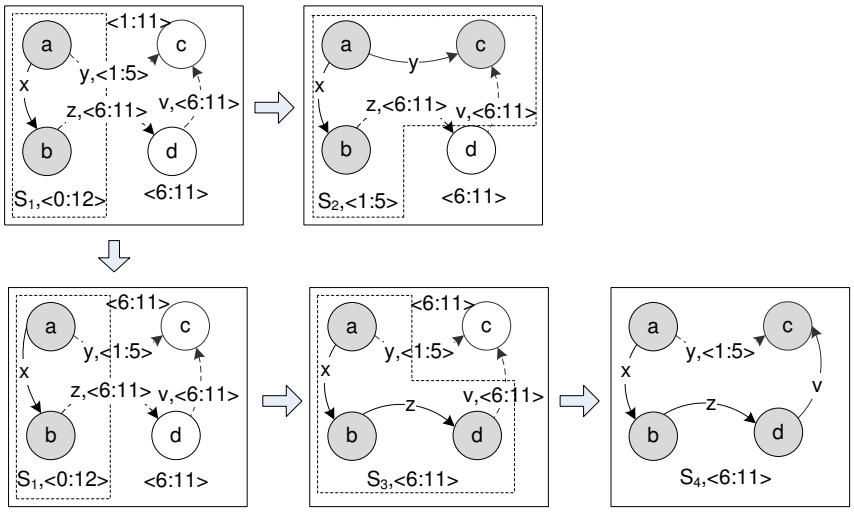

Figure 5.3: Updating span sequence of a vertex.

$F(\{a,b,c\}, V_f \setminus c)$ 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. One 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 Equation 5.1 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 $S_i = (V_i, s_i\!:\!e_i)$ 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 Figure 5.2(b). 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 generate 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 Equation 5.1. To illustrate how this operation can address the second challenge, consider again the example of Figure 5.3. 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 (i) contain V_i and zero or more vertices from V_f and (ii) their span is a sub-span[2] of $s_i{:}e_i$. Given the above, the complete set of IRSs

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

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:

$$H(S_i, V_f) =$$

$$\begin{cases} S_i, & \text{if irsadj}(S_i, V_f) = \emptyset \\ & \text{or } V_f = \emptyset \\ H((\{u\}, s_u{:}e_u), V_f | (u, s_u{:}e_u)) & \\ \cup\, H((\emptyset, 1{:}T), V_f | (u, s_u{:}e_u)), & \text{if } V_i = \emptyset \wedge V_f \neq \emptyset \\ \text{where } u \in V_f \text{ and } s_u{:}e_u \text{ is a span of } u & \\ H((V_i \cup \{u\}, s_u{:}e_u), V_f | (u, s_u{:}e_u)) & \\ \cup\, H(S_i, V_f | (u, s_u{:}e_u)), & \text{otherwise.} \\ \text{where } (u, s_u{:}e_u) \in \text{irsadj}(S_i, V_f) & \end{cases} \tag{5.2}$$

The above recurrence relation shares the same overall structure with the corresponding recurrence relation for enumerating the induced subgraphs (Equation 5.1) and its correctness can be shown in a way similar to that used for Equation 5.1. To eliminate redundancy, we omit the complete proof of Equation 5.2, 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

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

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

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 Equation 5.2 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 $\mathcal{S}$ are passed to *enumerate* to incrementally grow S_i. To generate all IRSs, the enumeration process starts with enumerate($\emptyset$,$V_f, \mathcal{S}$) where V_f includes all vertices for $\mathcal{N}^{\phi}$. 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, 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 $\mathcal{S}$. 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 $\mathcal{S}$ 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 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.

5.2.2 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}

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
10:    for each node v in (adj(u) sorted in increasing end-time order) do
11:       if d[v] < d[u] then
12:          mpm(G^RS, v, t, d, p)
13: pop p
```

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 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), represent 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 $v \in adj(u)$ 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 $v' \in adj(u)$ 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.

5.3 Interestingness Measures for EIRSs

Depending on the characteristics of the underlying dynamic networks and the parameters ϕ, β, $k_{\min}$, $k_{\max}$, and $m_{\min}$ used to define the EIRSs being mined, the EIRS discovery algorithm can find a large number of EIRSs. To efficiently identify interesting relational changes or specific types of changes, we developed three methods for determining the interestingness of an EIRS. These methods focus on identifying the EIRS whose constituent IRSs are undergoing the highest degree of change. Note that this is just one of the many ways that can be used to assign a quantitative interestingness measure to an EIRS and other measures can be derived by looking at the nodes, relational inversions, cyclicality, etc.

The first measure referenced to as the *total drift (TD)* is designed to identify the EIRSs that show the highest change between the different IRS involved. In particular, given an EIRS, we computed the ratio of the total number of unique edges (i.e., relations) in all of its constituent IRSs over the total number of edges in the same IRSs. This measure is defined as

$$TD = \frac{\text{total \# of unique edges in all states}}{\text{total \# of edges in all states}}. \tag{5.3}$$

If the relational states of an EIRS contain the same edges in all states, this measure will assign a low score for that path.

The second measure referenced to as the *mean drift (MD)* is designed to identify the EIRSs that show the highest change between each successive pair of IRSs. In particular, given an EIRS, we computed the ratio of the total number of unique edges (i.e., relations) between each pair of successive IRSs over the total number of edges in

all the IRSs. This measure is defined as

$$MD = \frac{1}{N-1}\sum_{i=1}^{N-1}\frac{\text{total \# of unique edges in } S_i \text{ and } S_{i+1}}{\text{total \# of edges in } S_i \text{ and } S_{i+1}}. \tag{5.4}$$

The third measure referenced to as the *mean drift from initial state (MDIS)* is designed to identify the EIRS that show the highest change between the first IRS and all subsequent IRSs. In particular, given an EIRS, we computed the ratio of the total number of unique edges (i.e., relations) between each pair of the first IRS and any other IRSs over the total number of edges in all the IRSs. This measure is defined as

$$MDIS = \frac{1}{N-1}\sum_{i=2}^{N}\frac{\text{total \# of unique edges in } S_1 \text{ and } S_i}{\text{total \# of edges in } S_1 \text{ and } S_i}. \tag{5.5}$$

The TD measure provides a summary of all the changes that occur among all IRSs of an EIRS. When users want to identify the EIRSs that capture highly dynamic relationship among the nodes, the TD measure can robustly provide such ranking. However, it ignores the incremental changes between the consecutive IRSs of an EIRS. On the other hand, both the MD and the MDIS measures are sensitive to such incremental changes and allows users to locate such relational changes.

5.4 Experimental Design & Results

5.4.1 Datasets

We evaluated our algorithm using datasets from a patent citation network, a trade network, an email communication network and a co-authorship network. The scalability of our algorithm was assessed on the patent citation dataset and co-authorship dataset whereas all four datasets were used in the qualitative assessment of the identified EIRSs. All datasets used for evaluation have been presented in Section 4.1.

5.4.2 Performance Results

We evaluated the performance and scalability of our algorithm for mining the maximal EIRSs using the N2, N3, and N4 datasets from the patent citation network and DBLP dataset. Our evaluation is designed to assess how the density of the networks and the

Table 5.1: Network Density.

Dataset	#IRS	#EIRS	ETime	PTime	TotalTime
N2	56,725	2,495	5	0	13
N3	1,864,660	45,448	287	160	458
N4	16,696,859	103,335	3,226	16,662	19,901

The number of vertices in an IRS is 4 to 8, the maximum allowed vertex difference between two successive IRSs is 1, and ϕ is 4. Run times are in seconds. #IRS denotes the number of discovered IRSs. #EIRS denotes the number of discovered EIRSs. Etime denotes the amount of time spent to enumerate IRSs. PTime denotes the amount of time spent for path enumeration. TotalTime denotes the total amount of time spent to determine all EIRSs in the network.

various parameters associated with the EIRS definition impacts the performance of the algorithm. All experiments are conducted on a Linux cluster with 6-core Intel Xeon X7542 "Westmere" processors at 2.66 GHz.

Note that in order to better assess how the interstate similarity component in the definition of the EIRS impacts the performance of the algorithm in all the experiments presented in this section, instead of using $|V_i \cap V_j|/|V_i \cup V_j|$ as a measure of inter-state similarity (constraint (iv) of Definition 1), we used the number of different vertices between V_i and V_j as a measure of distance. This allows us to explicitly increase/decrease the complexity of the mining problem by changing the number of different vertices that is allowed between successive IRSs.

Network Density

The performance of the algorithm for the three datasets is shown in Table 5.1. The datasets N2, N3 and N4 contain the same number of vertices $84,152$, but their density in terms of the number of edges present in the network increases by ~ 1.4 times from N2 to N3 and by ~ 1.2 times from N3 to N4. The results in Table 5.1 show that as the graph density increases the number of EIRSs found increases (i.e. the number of EIRSs increased from $2,495$ in N2 to $45,448$ in N3 to $103,335$ in N4). At the same time, the total runtime to discover the EIRSs increases from 13 seconds to process N2 to 458 seconds for N3 and $19,901$ seconds for N4. Even though the IRS enumeration step is mostly the time consuming process, as the number of IRS increases the time needed to traverse the IRS graph and discover the EIRSs starts to increase. For N4, about 84% of the total runtime was spent discovering the maximal paths. For sparse graphs, IRS

Table 5.2: Inter-state similarity based on N4 dataset.

ISDiff	$k_{\min}$	$k_{\max}$	#IRS	#EIRS	ETime	PTime	TotalTime
1	5	8	3,037,440	315	343	51	407
2	5	”	”	86,731	342	116	472
3	5	”	”	75,592,341	346	409	768
1	6	”	2,962,777	2	338	46	398
2	6	”	”	5,289	338	93	444
3	6	”	”	9,781,632	337	223	574

Dataset N4 is used for this experiment and $\phi = 5$. ISDiff denotes the inter-state distance capturing the maximum allowed vertex difference between two IRSs. $k_{\min}$ denotes the minimum number of vertices allowed in an IRS. $k_{\max}$ denotes the maximum number of vertices allowed in an IRS. Rest of the column labels are described in Table 5.1

Table 5.3: Inter-state similarity based on DBLP dataset.

ISDiff	$k_{\min}$	$k_{\max}$	#IRS	#EIRS	ETime	PTime	TotalTime
1	4	5	2,491,924	121	1,482	5	1,729
2	4	5	”	199,741	1,935	9	2,173
1	4	6	13,451,662	170	22,078	56	22,459
2	4	6	”	235,545	23,989	80	24,363

Dataset DBLP is used for this experiment and $\phi = 4$. ISDiff denotes the inter-state distance capturing the maximum allowed vertex difference between two IRSs. $k_{\min}$ denotes the minimum number of vertices allowed in an IRS. $k_{\max}$ denotes the maximum number of vertices allowed in an IRS. Rest of the column labels are described in Table 5.1

enumeration time dominates the computation. For denser graphs, the direction graph building requires the most amount of computing.

Inter-state Distance

Table 5.2 shows the performance of the algorithm using the N4 dataset for different values of the maximum allowed number of different vertices between two consequitive states on an EIRS. The number of different vertices is varied from 1 to 3 for two different sets of IRSs (i.e., a set of IRSs with 5 to 8 vertices and other set of IRSs with 6 to 8 vertices). We observed that the number of discovered EIRSs increases as the maximum allowed vertex difference increases (i.e. the similarity threshold decreases). For the $5-8$ set, the increase in the maximum allowed vertex difference from 1 to 3 causes an increase in discovered EIRSs from 315 to $75,592,341$.

Table 5.3 shows similar performance results based on the DBLP dataset. The number of different vertices is varied from 1 to 2 for two different sets of IRSs (i.e., a set of IRSs with 4 to 5 vertices and other set of IRSs with 4 to 6 vertices). Similar to Table 5.2, the number of discovered EIRSs increases as the maximum allowed vertex different increases. For the $4-6$ set, the increase in the maximum allowed vertex difference from 1 to 2 causes an increase in discovered EIRSs from 170 to $235,545$. However, as we increase the maximum allowed vertex difference, the EIRSs will start containing unrelated IRSs in their path, since the similarity threshold between the IRSs are lower, which may represent less interesting EIRS. The decrease in similarity threshold also increases the total runtime, the path enumeration step takes longer to process more edges between IRSs.

Table 5.4: Minimum Span (ϕ) study based on N3 dataset.

ϕ	QEdges	#IRS	#EIRS	ETime	PTime	TotalTime
5	5,521	190,491	112	21	1	34
4	9,515	744,053	18,788	105	33	148
3	21,726	5,761,948	1,190,067	1,289	4,729	6,028

Dataset N3 is used for this experiment, the number of vertices in an IRS is 5 to 7, and the maximum allowed vertex difference between two linked IRSs is 1. QEdges denotes the number of edges in $\mathcal{N}^{\phi}$, and rest of the column labels are described in Table 5.1

Table 5.5: Minimum Span (ϕ) study based on DBLP.

ϕ	QEdges	#IRS	#EIRS	ETime	PTime	TotalTime
5	79,688	762,156	125	337	1	571
4	164,958	2,624,519	1,114	1,550	8	1,791
3	382,200	11,405,118	47,811	22,543	1,975	24,815

Dataset DBLP is used for this experiment, the number of vertices in an IRS is 3 to 5, and the maximum allowed vertex difference between two linked IRSs is 1. QEdges denotes the number of edges in $\mathcal{N}^{\phi}$, and rest of the column labels are described in Table 5.1

Minimum Span

The performance of the algorithm for different values of minimum span (ϕ) is shown in Table 5.4 and Table 5.5. The value of ϕ represents the minimum length requirement for an induced relational state to be in consistent state and for this experiments is in

terms of years.

From these results, we observed that as the value of ϕ decreases, the number of discovered EIRSs and the runtime increases. The value of ϕ controls the number of edges that can qualify to be part of the $\mathcal{N}^{\phi}$ and the lower the value of ϕ is the more number of edges will qualify. In Table 5.4, we see that the number of qualified edges increase from $5,521$ to $21,726$ for $\phi = 5$ and $\phi = 3$. For DBLP dataset (Table 5.5), we see that the number of qualified edges increase from $79,688$ to $382,200$ for $\phi = 5$ and $\phi = 3$. As the number of qualified edges in $\mathcal{N}^{\phi}$ increase, the number of IRSs increases and resulting in discovering higher number of EIRSs.

We also observed in Table 5.4 that the runtime increase as the value of ϕ decreases from 34 seconds to $6,028$ seconds for $\phi = 5$ and $\phi = 3$, since the algorithm needs to process more number of IRSs to find relations and their evolution paths. Similarly, in Table 5.5, the runtime increase as the value of ϕ decreases from 571 seconds to $24,815$ seconds for $\phi = 5$ and $\phi = 3$. This parameter is an important factor in finding EIRSs in different datasets, since the conserved state of a pattern is likely to be different depending on the type of the data.

Table 5.6: IRS size study based on N4 dataset.

$k_{\min}$	$k_{\max}$	#IRS	#EIRS	ETime	PTime	TotalTime
5	5	212,267	60	25	4	40
5	6	1,039,741	21,059	108	41	158
5	7	4,265,612	46,609	493	642	1,144
5	8	16,639,693	57,583	3,221	15,617	18,852

Dataset N4 is used for this experiment, the maximum allowed vertex difference between two linked IRSs is 1 and ϕ=4. The column labels are described in Table 5.1

Table 5.7: IRS size study based on DBLP dataset.

$k_{\min}$	$k_{\max}$	#IRS	#EIRS	ETime	PTime	TotalTime
3	3	132,595	4	45	1	318
3	4	561,275	661	182	2	520
3	5	2,624,519	1,114	1,550	8	1,791
3	6	13,584,257	1,163	22,671	64	22,967

Dataset DBLP is used for this experiment, the maximum allowed vertex difference between two linked IRSs is 1 and ϕ=4. The column labels are described in Table 5.1

IRS Size

We analyzed the performance of the algorithm for different sizes IRSs in Table 5.6 and Table 5.7. The size of an IRS is represented as the minimum and maximum number of vertices allowed in an IRS. For example, the size of $5-8$ means that an IRS can contain minimum of 5 vertices and maximum of 8 vertices. We observe that as the size increases, the number of discovered EIRSs increases. Since larger range in size allows more number of IRSs to be detected, the chance of finding higher number of EIRSs is increases.

In Table 5.6, the number of IRSs and EIRSs found for size 5 is $212,267$ and 60. When the size was increased to $5-8$, the number of IRSs increased to $16,639,693$ and in turn resulted with $57,583$ EIRSs. Similarly in Table 5.7, the number of IRSs and EIRSs found for size 3 is $132,595$ and 4. When the size was increased to $3-6$, the number of IRSs increased to $13,584,257$ and in turn resulted with $1,163$ EIRSs.

Chapter 6

Mining the Coevolving Relational Motifs

6.1 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 Figure 6.1, in the context of a hypothetical country-to-country trading network where labels represent the commodities been 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 Figure 6.1 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 [41]. Similarly, CRMs in

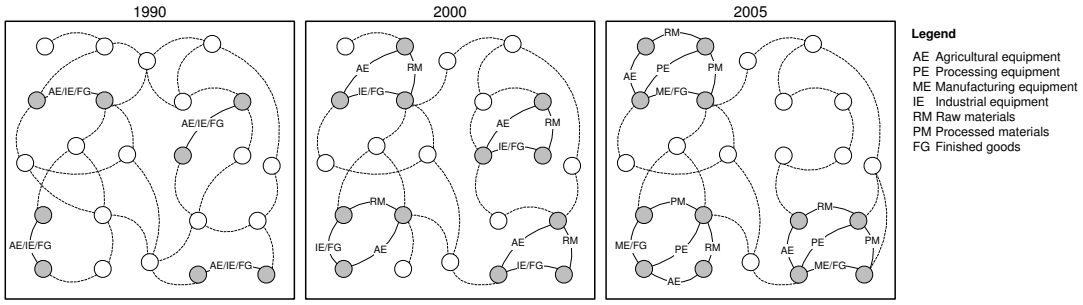

Figure 6.1: An example of a coevolving relational motif in the context of a hypothetical country-to-country trading network where labels represent the commodities been traded.

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.

The formal definition of a CRM that is used in this thesis 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 and each $M_j = (N_j, A_j)$ is a 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 $0 < \beta \leq 1$.

A relational motif M_j is *defined* over a subset of vertices N if there is a injection ξ_j from N_j to N. An m-length CRM *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}.

For example, the number of occurrences of the CRM in Figure 6.1 is 3. Note that the third condition 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 parameters ϕ and β in Definition 2 are as follows: 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. 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. Finally, 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. Note that the frequent dynamic subgraphs introduced by Borgwardt et. al. [1] (Chapter 3) correspond to CRMs in which the snapshots supporting each set of nodes are restricted to be consecutive and $\beta = 1$.

In this thesis 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. 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 thesis 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 as a *frequent* CRM and it is *valid* if it also satisfies the minimum node overlap constraint (Definition 2(iv)).

6.2 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 [65] 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 $\mathcal{N}$.

6.2.1 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 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 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

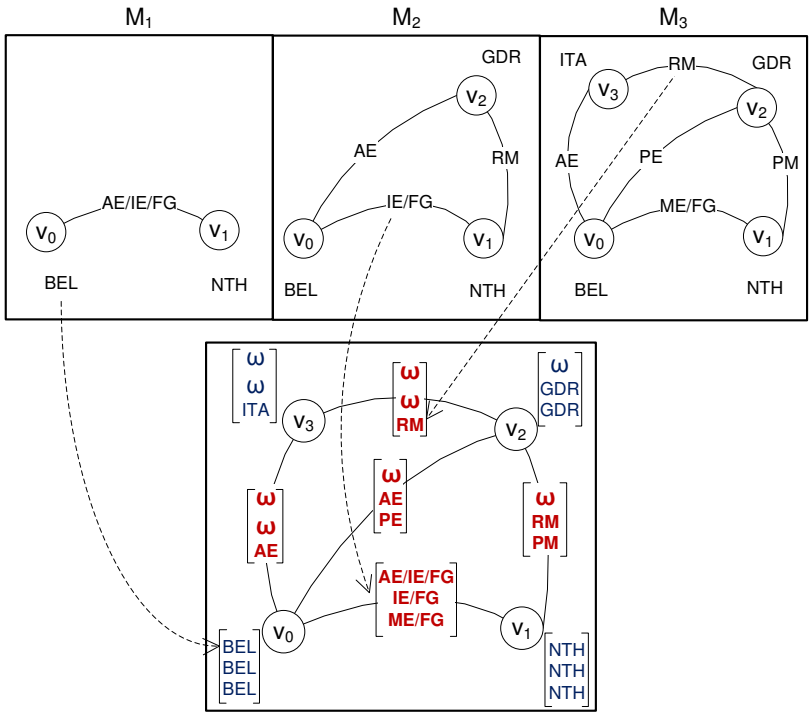

Figure 6.2: A CRM Representation. The CRM c consists of 3 motifs $\langle M_1, M_2, M_3\rangle$ and represents relations among vertices $N = \{v_0, v_1, v_2, v_3\}$ using 5 edges. G_c (bottom graph) shows the CRM representation capturing vertex and edge label vectors.

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 Figure 6.2. The CRM 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. The edge between (v_0, v_1) exists in all 3 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 2 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).

6.2.2 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 $\mathcal{N}$, 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 [71] 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.

6.2.3 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 [65], 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 1-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 [65]. 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) are 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 (i) representing the DFS code a CRM, (ii) ordering the DFS codes of a CRM using the DFS lexicographic ordering, (iii) representing the minimum DFS code of a CRM to use as the canonical label, and (iv) 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

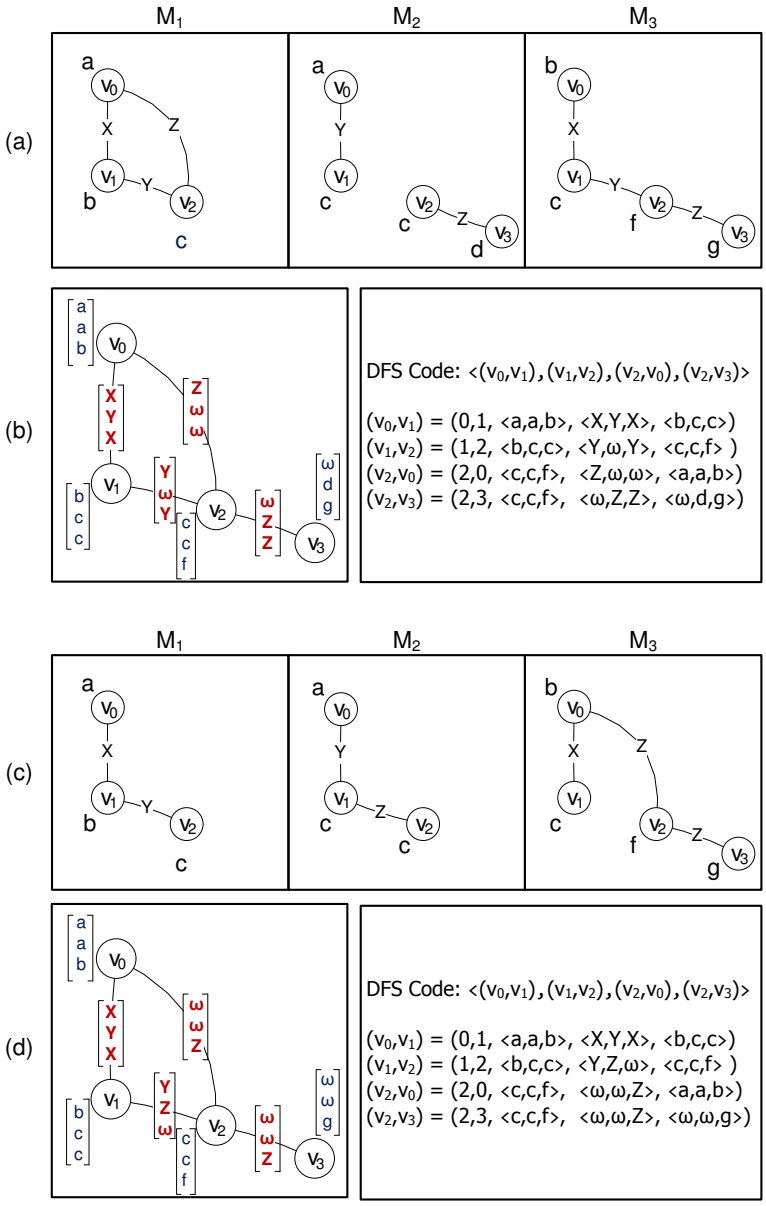

Figure 6.3: DFS codes for two CRMs represented as a sequence of edges (v_0, v_1), (v_1, v_2), (v_2, v_0), and (v_2, v_3). (a) Presents CRM C_1 consisting of 3 motifs, 4 vertices, and 4 edges. (b) Presents G_{C_1} and the corresponding DFS code for CRM C_1. (c) Presents CRM C_2 consisting of 3 motifs, 4 vertices, and 4 edges. (d) Presents G_{C_2} and the corresponding DFS code for CRM C_2.

two sets: the forward edge set $E_{T_c,fw} = \{(v_i, v_j) \in E_c |\ i < j\}$ and the backward edge set $E_{T_c,bw} = \{(v_i, v_j) \in E_c |\ i > j\}$. 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, Figure 6.3 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., the edge labels):

CRM C_1 - Figure 6.3 (a)	CRM C_2 - Figure 6.3 (c)
$\langle(0, 1, \langle \mathtt{a}, \mathtt{a}, \mathtt{b}\rangle, \langle \mathtt{X}, \mathtt{Y}, \mathtt{X}\rangle, \langle \mathtt{b}, \mathtt{c}, \mathtt{c}\rangle)$,	$\langle(0, 1, \langle \mathtt{a}, \mathtt{a}, \mathtt{b}\rangle, \langle \mathtt{X}, \mathtt{Y}, \mathtt{X}\rangle, \langle \mathtt{b}, \mathtt{c}, \mathtt{c}\rangle)$,
$(1, 2, \langle \mathtt{b}, \mathtt{c}, \mathtt{c}\rangle, \langle \mathtt{Y}, \omega, \mathtt{Y}\rangle, \langle \mathtt{c}, \mathtt{c}, \mathtt{f}\rangle)$,	$(1, 2, \langle \mathtt{b}, \mathtt{c}, \mathtt{c}\rangle, \langle \mathtt{Y}, \mathtt{Z}, \omega\rangle, \langle \mathtt{c}, \mathtt{c}, \mathtt{f}\rangle)$,
$(2, 0, \langle \mathtt{c}, \mathtt{c}, \mathtt{f}\rangle, \langle \mathtt{Z}, \omega, \omega\rangle, \langle \mathtt{a}, \mathtt{a}, \mathtt{b}\rangle)$,	$(2, 0, \langle \mathtt{c}, \mathtt{c}, \mathtt{f}\rangle, \langle \omega, \omega, \mathtt{Z}\rangle, \langle \mathtt{a}, \mathtt{a}, \mathtt{b}\rangle)$,
$(2, 3, \langle \mathtt{c}, \mathtt{c}, \mathtt{f}\rangle, \langle \omega, \mathtt{Z}, \mathtt{Z}\rangle, \langle \omega, \mathtt{d}, \mathtt{g}\rangle)\rangle$	$(2, 3, \langle \mathtt{c}, \mathtt{c}, \mathtt{f}\rangle, \langle \omega, \omega, \mathtt{Z}\rangle, \langle \omega, \mathtt{d}, \mathtt{g}\rangle)\rangle$

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. DFS code's Neighborhood restriction property defined in [65] still holds for DFS code of a CRM.

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 $\alpha = code(c_\alpha, T_\alpha) = \langle e^0_\alpha, e^1_\alpha, \ldots, e^p_\alpha \rangle$ and $\delta = code(c_\delta, T_\delta) = \langle e^0_\delta, e^1_\delta, \ldots, e^q_\delta \rangle$, 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 $e^x_\alpha = (i^x_\alpha, j^x_\alpha, \boldsymbol{l_{i^x_\alpha}}, \boldsymbol{l_{i^x_\alpha, j^x_\alpha}}, \boldsymbol{l_{j^x_\alpha}})$ and $e^y_\delta = (i^y_\delta, j^y_\delta, \boldsymbol{l_{i^y_\delta}}, \boldsymbol{l_{i^y_\delta, j^y_\delta}}, \boldsymbol{l_{j^y_\delta}})$ be two edges, one in each of the α and δ DFS codes. Now, $e^x_\alpha < e^y_\delta$, if any of the following is true:

i) $e^x_\alpha \in E_{\alpha,bw}$ and $e^y_\delta \in E_{\delta,fw}$, or

ii) $e^x_\alpha \in E_{\alpha,bw}$, $e^y_\delta \in E_{\delta,bw}$, and $j^x_\alpha < j^y_\delta$, or

iii) $e^x_\alpha \in E_{\alpha,bw}$, $e^y_\delta \in E_{\delta,bw}$, $j^x_\alpha = j^y_\delta$ and $\boldsymbol{l_{i^x_\alpha, j^x_\alpha}} < \boldsymbol{l_{i^y_\delta, j^y_\delta}}$, or

iv) $e^x_\alpha \in E_{\alpha,fw}$, $e^y_\delta \in E_{\delta,fw}$, and $i^y_\delta < i^x_\alpha$, or

v) $e^x_\alpha \in E_{\alpha,fw}$, $e^y_\delta \in E_{\delta,fw}$, $i^x_\alpha = i^y_\delta$ and $\boldsymbol{l_{i^x_\alpha}} < \boldsymbol{l_{i^y_\delta}}$, or

vi) $e^x_\alpha \in E_{\alpha,fw}$, $e^y_\delta \in E_{\delta,fw}$, $i^x_\alpha = i^y_\delta$, $\boldsymbol{l_{i^x_\alpha}} = \boldsymbol{l_{i^y_\delta}}$ and $\boldsymbol{l_{i^x_\alpha, j^x_\alpha}} < \boldsymbol{l_{i^y_\delta, j^y_\delta}}$, or

vii) $e^x_\alpha \in E_{\alpha,fw}$, $e^y_\delta \in E_{\delta,fw}$, $i^x_\alpha = i^y_\delta$, $\boldsymbol{l_{i^x_\alpha}} = \boldsymbol{l_{i^y_\delta}}$, $\boldsymbol{l_{i^x_\alpha, j^x_\alpha}} = \boldsymbol{l_{i^y_\delta, j^y_\delta}}$, and $\boldsymbol{l_{j^x_\alpha}} < \boldsymbol{l_{j^y_\delta}}$.

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) $F_\omega(\alpha) \leq F_\omega(\delta)$, where $F_\omega(x)$ is the number of edges $(\boldsymbol{l_{i^t_x, j^t_x}})$ that are set to ω, or

b) $\exists t, 0 \leq t \leq min(p, q)$, $e^k_\alpha = e^k_\delta$ for $k < t$, and $e^t_\alpha < e^t_\delta$, or

c) $e^k_\alpha = e^k_\delta$ for $0 \leq k \leq p$ and $p \leq q$.

Note that 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. This is important as we show later in Section 6.2.3. 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 Figure 6.3 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 $min\_code(c)$. 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 Figure 6.4, 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 (Figure 6.4(a)), 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 (Figure 6.4(c)), the original DFS code consisted the same sequence of edges $\{(v_0, v_1), (v_1, v_2)\}$ represented as: $\big(0, 1, \langle a, a, b \rangle, \langle X, Y, X \rangle, \langle b, c, c \rangle\big)$, $\big(1, 2, \langle b, c, c \rangle, \langle Y, \omega, Y \rangle, \langle c, \omega, f \rangle\big)$. Based on the combination of the label vectors of vertices v_2 and v_3, and edge (v_2, v_3), we have the

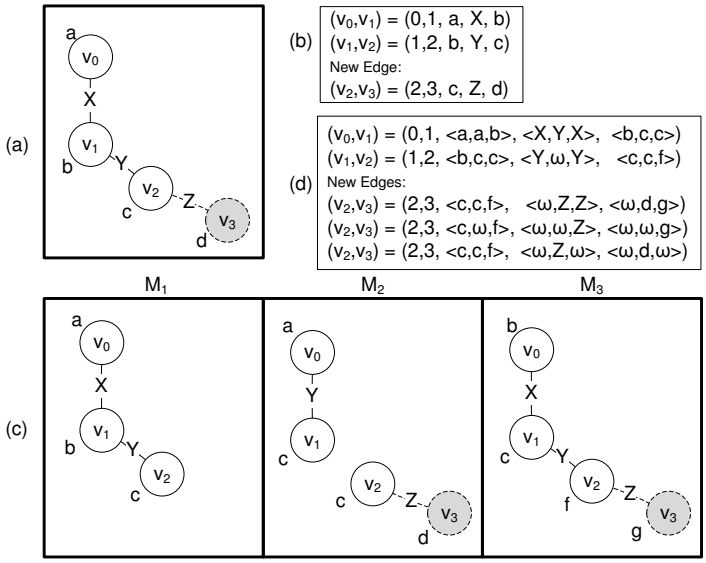

Figure 6.4: Adding an edge according to rightmost extension rules. (a) Extending a simple graph, (b) DFS code of the simple graph, (c) extending a CRM, and (d) DFS code of the CRM.

following options for the (v_2, v_3) edge: $(2, 3, \langle c, c, f\rangle, \langle \omega, Z, Z\rangle, \langle \omega, d, g\rangle)$, $(2, 3, \langle c, \omega, f\rangle, \langle \omega, \omega, Z\rangle, \langle \omega, \omega, g\rangle)$, and $(2, 3, \langle c, c, f\rangle, \langle \omega, Z, \omega\rangle, \langle \omega, d, \omega\rangle)$. Note that we allow a CRM to 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 [71]. 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.

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) $min\_code(c) = \langle e_1, e_2, \ldots, e_k \rangle$, where e_i is an 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 $code(c) = \langle e_x, e_{p_1}, \ldots, e_{p_{k-1}} \rangle$, 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 contains some ω labels. 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 right-most extension rule to add the rest of the edges one by one.

6.2.4 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. 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 [104] 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 (i) $\forall v \in V_p \Rightarrow L_p(v) = L_G(\varphi(v))$ and (ii) $\forall (u, v) \in E_p \Rightarrow (\varphi(u), \varphi(v)) \in E_G$. The minimum image based support of a pattern p in G is defined as:

$$\sigma(p, G) = \min_{v \in V_p} \mid \{\varphi_i(v) \text{: } \varphi_i \text{ is an occurrence of } p \text{ in } G\} \mid. \tag{6.1}$$

This minimum image based support is anti-monotonic [104].

We adopted the minimum image based support measure to calculate the minimum support of a CRM in a dynamic network. As defined in Chapter 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:

$$\sigma(c, \mathcal{N}) = \min_{v \in V_c} \mid \{\varphi_i(v) \text{: } \varphi_i \text{ is an occurrence of } c \text{ in } \mathcal{N}\} \mid. \tag{6.2}$$

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 Section 6.2.3 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 Section 6.1) 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:

$$\rho(c) = \frac{\min\limits_{1 \leq i \leq m} \{|V_{M_i}|\}}{|N_c|}, \tag{6.3}$$

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 [105]. 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.

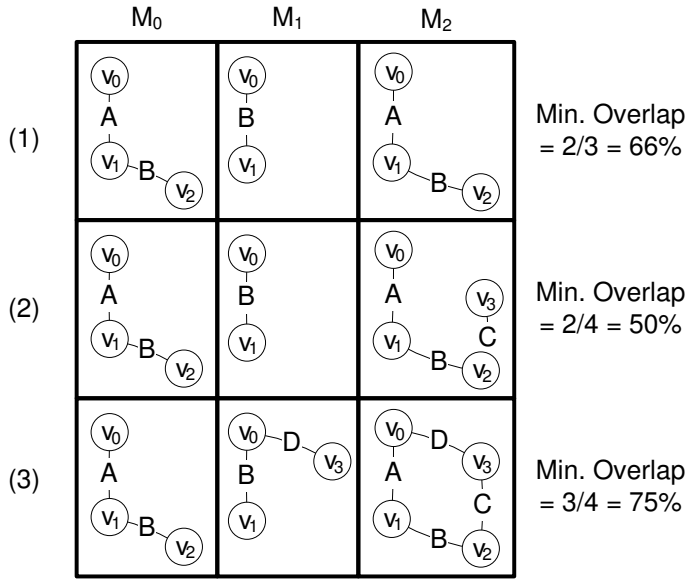

Figure 6.5: Example of a CRM growth when the minimum overlap constraint is not anti-monotonic. Assume the minimum overlap constraint is 60%.

For example, let us assume the minimum overlap threshold is 60% in Figure 6.5. In step (1), motif M_1 contains 2 out of 3 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$_{\mathrm{x}}$, 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 pattern was derived, referred 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:

$$\rho(c_p) = \frac{\min\limits_{1 \leq i \leq m} \{|V_{M_i}|\}}{|N_c| - 1}. \tag{6.4}$$

In Figure 6.6, 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 it's 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 it's parent overlap threshold is $(2/(5-1)) = 50\%$. Both thresholds are lower than β, hence we stop enumerating this CRM any further.

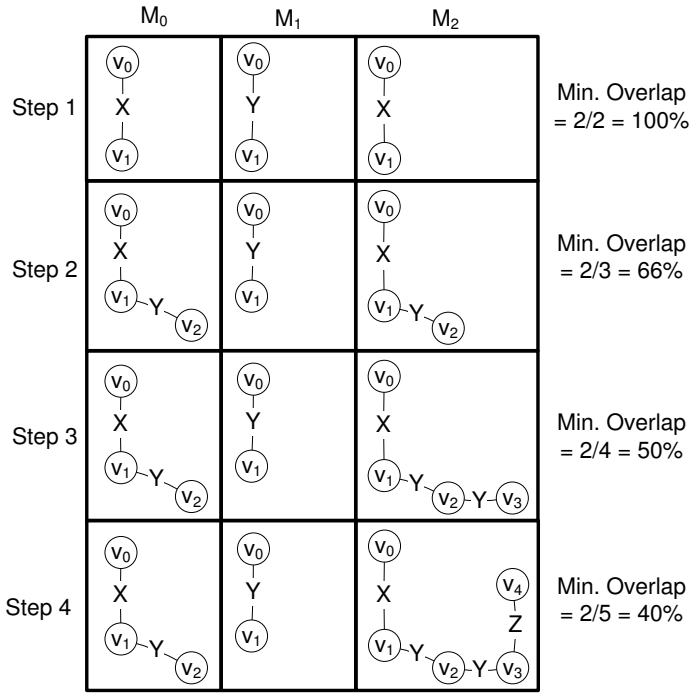

Figure 6.6: Minimum overlap calculation based on Equation (6.4) is used for search space pruning during the CRM enumeration process. Assuming $\beta = 60\%$, this CRM enumeration terminates at step 4.

6.3 Experimental Design

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

6.3.1 Datasets

We have used two different types of datasets to evaluate CRMminer. We presented all datasets details in Section 4.2. The DBLP co-authorship network is a real world dynamic network that captures yearly co-authorship relations. The bioprocess network (GT) and the sales network (Sales) datasets are based on multivariate time-series data. To characterize the relations among different variables and understand changes over time, we represent the time-series data as a dynamic network. The CRMs discovered from these networks can be used to characterize the overall network, as shown in Chapter 8.

6.3.2 Metrics

In order to asses the scalability and performance of the algorithm, we collect two sets of results for the discovered CRMs. The first set of results is collected by running CRMminer, that does not perform any overlap based pruning during CRM enumeration. This process generates the complete set of CRMs (T_{CRM}) for the specified support threshold and then applies the overlap threshold to identify the valid CRMs (Q_{CRM}) from the output space. The second set of results are collected by running CRMminer$_{\text{x}}$, that applies the overlap threshold to prune the search space during enumeration. This process uses Equation (6.4) to perform the overlap threshold check for search space pruning to collect all CRMs (T_{CRM}) and then performs a final check using Equation (7.2) to select the valid CRMs (Q_{CRM}). As discussed in Section 6.2.4, overlap based pruning does not guarantee complete set of results. Thus, the set of CRMs in T_{CRM} and Q_{CRM} collected using the CRMminer$_{\text{x}}$ are subsets of T_{CRM} and Q_{CRM} collected by using the CRMminer correspondingly.

6.4 Results

The evaluation consists of two parts. The first focuses on assessing the performance of the algorithm that we developed for finding CRMs and to assess how the different parameters associated with the definition of CRMs impacts the performance. The second focuses on assessing the information that can be extracted from the discovered CRMs by analyzing some of the patterns of coevolving relational motifs that were identified in the three datasets.

6.4.1 Performance Results

Minimum Support & Overlap

Table 6.1, Figure 6.7, Figure 6.8, and Figure 6.9 show the performance of the CRM mining algorithms and the size distribution of the discovered CRMs, respectively. These results are presented for different values of the minimum support and overlap thresholds.

The following observations can be made from these results. First, the number of discovered CRMs increases as the minimum support decreases and/or the amount of

Table 6.1: Minimum Support (ϕ) & Overlap (β) study.

Data	ϕ	β	#A	CRMminer			CRMminer$_x$		
				$\#T_{\text{CRM}}$	$\#Q_{\text{CRM}}$	Time	$\#T_{\text{CRM}}$	$\#Q_{\text{CRM}}$	Time
DBLP	100	0.70	9	261	84	27.00	261	84	26.05
		0.60	”	”	84	”	261	84	26.28
		0.50	”	”	261	”	261	261	26.49
		0.40	”	”	261	”	261	261	27.22
	90	0.70	14	548	126	61.99	496	126	49.09
		0.60	”	”	133	”	496	133	47.48
		0.50	”	”	496	”	548	496	62.06
		0.40	”	”	548	”	548	548	54.09
	80	0.70	16	1,724	203	2,298.84	1,044	203	109.78
		0.60	”	”	225	”	1,044	225	120.15
		0.50	”	”	1,044	”	1,399	1,044	239.89
		0.40	”	”	1,492	”	1,558	1,491	694.31
GT	90	0.70	7	6,255	1,370	7.10	4,553	1,370	5.92
		0.60	”	”	3,724	”	4,730	3,410	5.92
		0.50	”	”	5,173	”	6,027	5,173	7.04
		0.40	”	”	6,255	”	6,255	6,255	7.18
	80	0.70	7	240,828	145,633	200.44	216,233	144,243	182.92
		0.60	”	”	205,674	”	224,188	200,439	184.44
		0.50	”	”	231,222	”	240,376	231,222	198.87
		0.40	”	”	240,828	”	240,828	240,828	199.59
	70	0.70	9	973,116	626,877	730.21	804,228	611,206	611.04
		0.60	”	”	796,897	”	871,176	778,893	685.46
		0.50	”	”	922,324	”	958,742	919,640	712.06
		0.40	”	”	973,116	”	973,116	973,116	714.63
Sales	45	0.70	65	11,917	134	18.70	1,472	134	3.52
		0.60	”	”	134	”	1,472	134	3.49
		0.50	”	”	1,472	”	5,785	1,472	9.75
		0.40	”	”	10,480	”	11,908	10,480	18.84
	40	0.70	90	183,437	551	207.75	7,422	551	14.03
		0.60	”	”	1,027	”	7,741	1,027	14.39
		0.50	”	”	8,582	”	37,109	8,582	52.46
		0.40	”	”	99,174	”	156,320	99,174	182.53
	35	0.70	109	24,509,851	47,535	11,904.75	263,475	38,429	299.80
		0.60	”	”	552,297	”	1,205,965	351,611	904.21
		0.50	”	”	3,701,651	”	4,950,385	2,617,409	3,040.26
		0.40	”	”	13,780,709	”	18,563,088	13,427,964	8,327.78

ϕ denotes the minimum support. #A denotes the number of discovered anchors. $\#T_{\text{CRM}}$ denotes the total number of discovered CRMs. $\#Q_{\text{CRM}}$ denotes the number of CRMs that meet the β threshold out of $\#T_{\text{CRM}}$. Time denotes the amount of time spent in seconds expanding the anchors to discover Q_{CRM}. For all datasets, k_{min} is 4, k_{max} is 10, and m_{min} is 4.

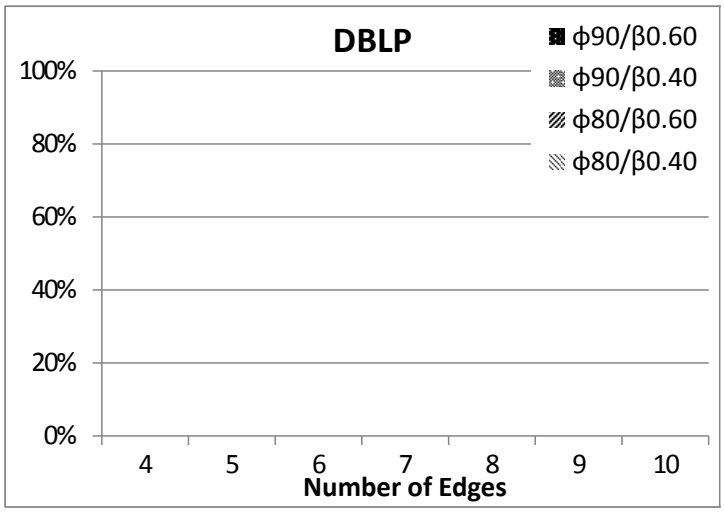

Figure 6.7: CRMs size distribution of the DBLP dataset for different minimum support and overlap thresholds. For all datasets, $k_{min} = 4$, $k_{max} = 10$, and $m_{min} = 4$.

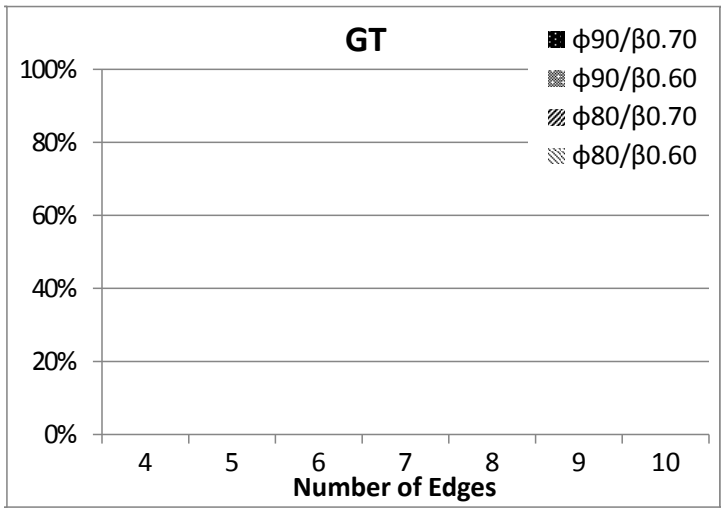

Figure 6.8: CRMs size distribution of the GT dataset for different minimum support and overlap thresholds. For all datasets, $k_{min} = 4$, $k_{max} = 10$, and $m_{min} = 4$.

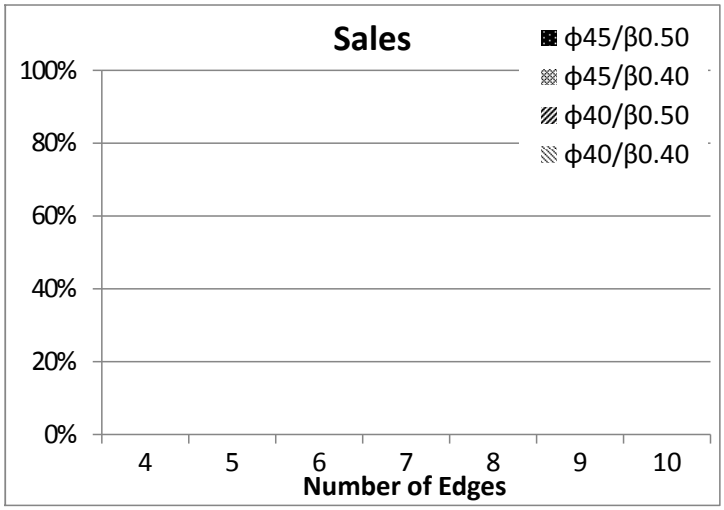

Figure 6.9: CRMs size distribution of the Sales dataset for different minimum support and overlap thresholds. For all datasets, $k_{min} = 4$, $k_{max} = 10$, and $m_{min} = 4$.

overlap decreases. Both of which were expected. Moreover, as the minimum support and/or overlap decrease, the size of the discovered CRMs increases. Second, the amount of time required by CRMminer$_x$ is lower than that required by CRMminer. Depending on the experiment, it is $1-40$ times faster than CRMminer with an average speedup of 5.8. The performance gap is higher for large values of overlap and progressively shrinks as the overlap decreases. This is expected, as CRMminer$_x$'s overlap-based pruning becomes less effective for low overlap values. Third, the number of CRMs missed by the approximate nature of CRMminer$_x$ is either none or relatively small. This indicates that CRMminer$_x$ is a viable algorithm for CRM discovery as it is both faster and also quite effective in finding most valid CRMs.

Finally, comparing how the two algorithms scale with the size of the output space (i.e., the number of valid discovered CRMs), we see that for most datasets, they either scale linearly or better than linearly. The only exception is the $DBLP$ run of CRMminer for $\phi = 80$. For this experiment, CRMminer took 37 times more time than the $\phi = 90$ experiment and depending on the specific overlap value, it only discovered $1.6 - 2.7$ times more CRMs. To better understand CRMminer's behavior for this dataset, Figure 6.10, Figure 6.11, and Figure 6.12, show various statistics about the amount of time required by the different phases of the algorithm and the number of embeddings of

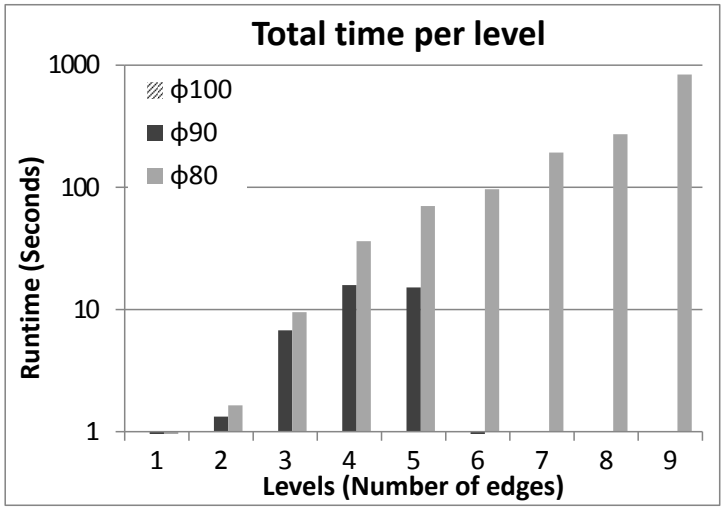

Figure 6.10: CRM enumeration details for the DBLP dataset showing the total time spent at each level. The results in the Y-axis are in log scale. For all experiments, $\beta = 0.60$ and $m_{min} = 4$.

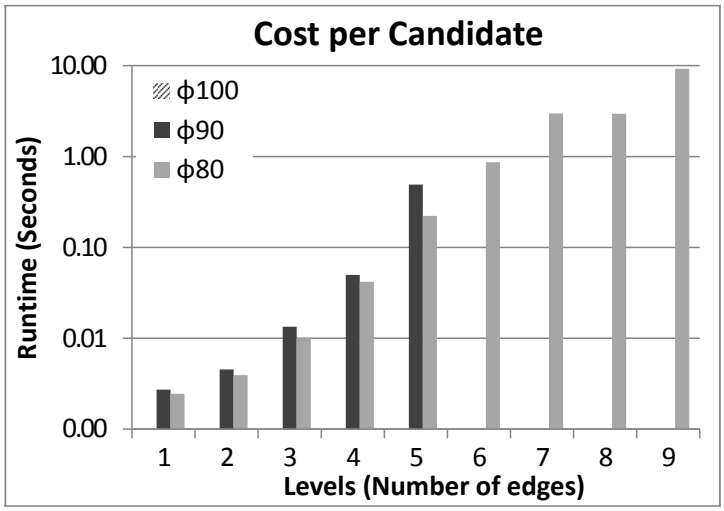

Figure 6.11: CRM enumeration details for the DBLP dataset showing the average time needed to generate a candidate CRM at each expansion level. The cost is calculated as the sum of the average time to locate a frequent edge, the average time to perform minimum DFS code check, and the average time to locate the embeddings of the candidate CRM. The results in the Y-axis are in log scale. For all experiments, $\beta = 0.60$ and $m_{min} = 4$.

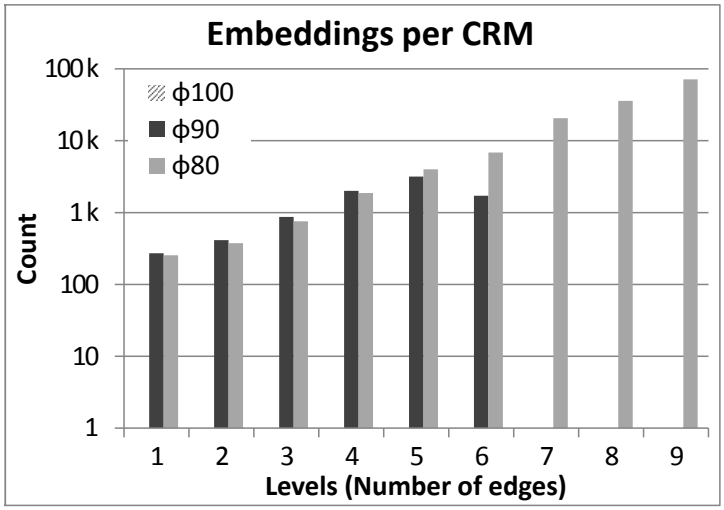

Figure 6.12: CRM enumeration details for the DBLP dataset showing the average number of embeddings maintained by each CRM at each level. The results in the Y-axis are in log scale. For all experiments, $\beta = 0.60$ and $m_{min} = 4$.

the discovered CRMs. From these results we can see that the reason for the dramatic increase in runtime is due to the fact that for $\phi = 80$ $DBLP$ contains a large number of candidate CRMs that contain many edges (Figure 6.10) and also have a large number of embeddings (Figure 6.12). As a result, CRMminer spends most of its time processing these large embedding lists, leading to its substantial increase in runtime. However, most of these candidate CRMs fail to meet the overlap constraint and this is the reason that CRMminer$_{\text{x}}$ performs better and scales better.

Understanding the missing CRMs The results presented in Table 6.1 show that CRMminer$_{\text{x}}$ is able to find almost all the valid CRMs for the $DBLP$ and the $Sales$ datasets in less time than CRMminer. For the GT dataset, even though the runtime decreased significantly, only a subset of the valid CRMs were found by CRMminer$_{\text{x}}$. As we have seen in the size distribution characteristic of the CRMs of the GT dataset presented in Figure 6.8, the CRMminer$_{\text{x}}$ fails to discover all CRMs, since the CRMs that become valid at later stage will get eliminated by the overlap threshold based pruning. The lack of anti-monotonicity property for the overlap threshold based pruning strategy eliminated a large number of potential CRMs at the early stages of the enumeration process.

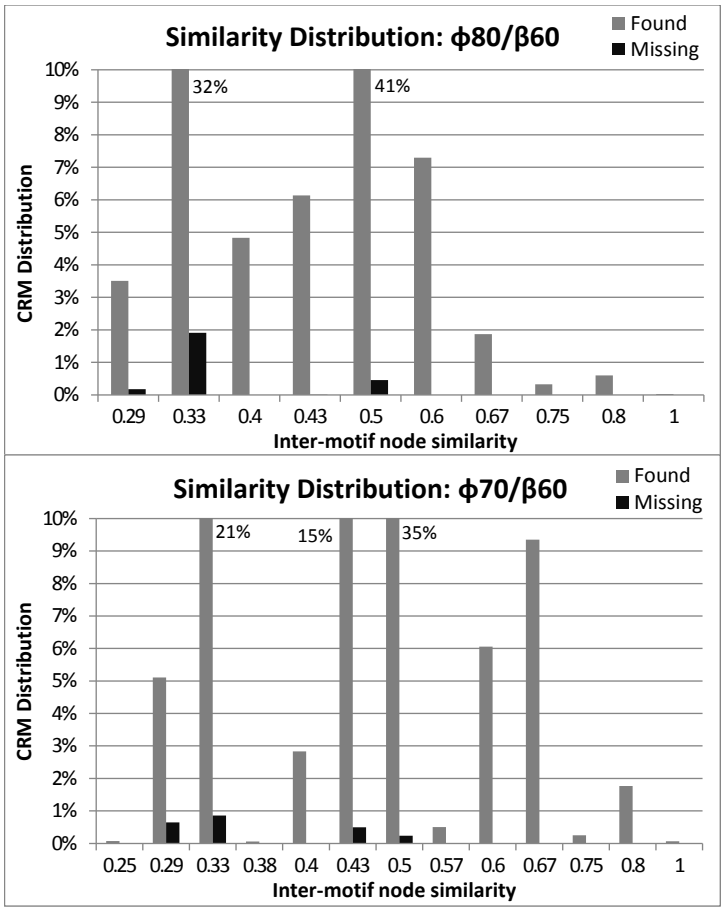

Figure 6.13: CRMs distribution based on the minimum inter-motif similarity using the *GT* dataset.

To understand the characteristics of the missing CRMs, we calculated the *minimum inter-motif similarity* of all CRMs based on the following equation:

$$\psi(c) = \frac{\min\limits_{1<i<j<m} \{|V_{M_i} \cap V_{M_j}|\}}{|N_c|}, \tag{6.5}$$

where $|V_{M_i} \cap V_{M_j}|$ is the number of common vertices between motif M_i and M_j and $|N_c|$ is the total number of vertices in CRM c.

We analyzed the *GT* dataset results presented in Table 6.1 for two different experiments ($\phi = 80, \beta = 60$ and $\phi = 70, \beta = 60$). Figure 6.13 shows the minimum inter-motif similarity distribution of all identified CRMs from two separate experiments using *GT* datasets. The *found* bars represent the CRMs that were identified by CRMminer$_x$ and

Table 6.2: Minimum Span (m_{min}) study.

		$m_{min} = 3$				$m_{min} = 4$			
Data	ϕ	#A	$\#T_{\text{CRM}}$	$\#Q_{\text{CRM}}$	Time	#A	$\#T_{\text{CRM}}$	$\#Q_{\text{CRM}}$	Time
DBLP	120	334	84,950	25,610	12,858.88	9	70	34	5.54
	110	410	116,968	35,582	17,204.75	9	140	55	10.18
	100	502	165,365	50,580	22,577.21	9	263	86	17.49
GT	40	12	1,834,449	1,210,710	381.13	1	19,797	6,490	5.92
	35	15	3,092,486	2,059,379	551.42	2	51,698	16,554	12.47
	30	20	5,150,967	3,392,215	717.02	4	123,853	47,702	23.36
Sales	35	109	425,867	46,832	135.33	11	124	7	0.26
	30	136	3,064,882	441,652	649.85	18	7,642	156	6.25
	25	175	10,840,277	1,379,708	1,817.96	33	518,390	1,308	139.32

m_{min} denotes the minimum number of motifs per CRM, and rest of the column labels are described in Table 6.1. For all datasets, β is 0.60, k_{min} is 4 and k_{max} is 6.

the *missing* bars represent the CRMs that were missed due to overlap based pruning during enumeration step. These plots show that the minimum inter-motif similarity among the missing CRMs are low. Hence, these missing CRMs contain motifs that are mostly different from each other in terms of their nodes and resulting in CRMs that may not be capturing interesting relational changes.

Minimum Span

The performance of the algorithm for different values of the minimum span (m_{min}) is shown in Table 6.2. The value of m_{min} represents the minimum number of motifs per CRM. From the reported results in Table 6.2, we observe that as the value of m_{min} decreases, the number of discovered CRMs and the runtime increases. The value of m_{min} directly impacts the number of anchors and as the number of anchors increases the total number of CRMs increases.

For the $DBLP$ dataset with $\phi = 120$, the number of anchors increases from 9 to 334 for $m_{min} = 4$ to $m_{min} = 3$. As a result, the number of valid CRMs discovered by CRMminer increases from 34 to $25,610$ and the CRMminer runtime increases by 2321 times. We observe similar increase in number of CRMs discovered and runtime for other support thresholds. For the GT dataset, the increase in the number of CRMs and the runtime is less significant than $DBLP$ dataset. As the number of anchors increased from 4 to 20 for $m_{min} = 4$ to $m_{min} = 3$ with $\phi = 30$, the total number of CRMs ($\#Q_{\text{CRM}}$) increased by 72 times and the runtime increased by 31 times. For the *Sales*

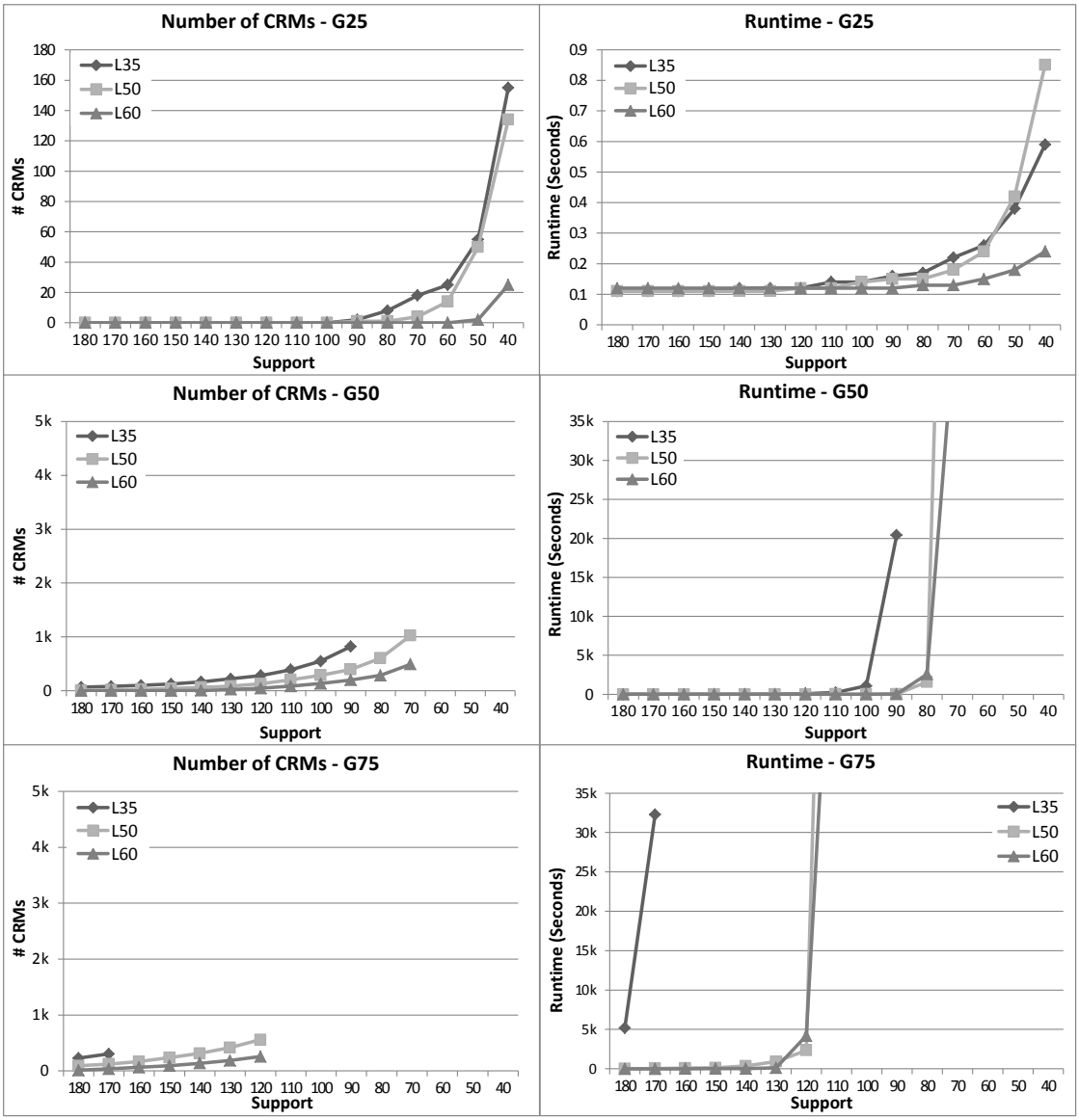

Figure 6.14: Performance of CRMminer for different versions of the $DBLP$ dataset. For all experiments, β is 0.60, m_{min} is 4, k_{min} is 3 and k_{max} is 10.

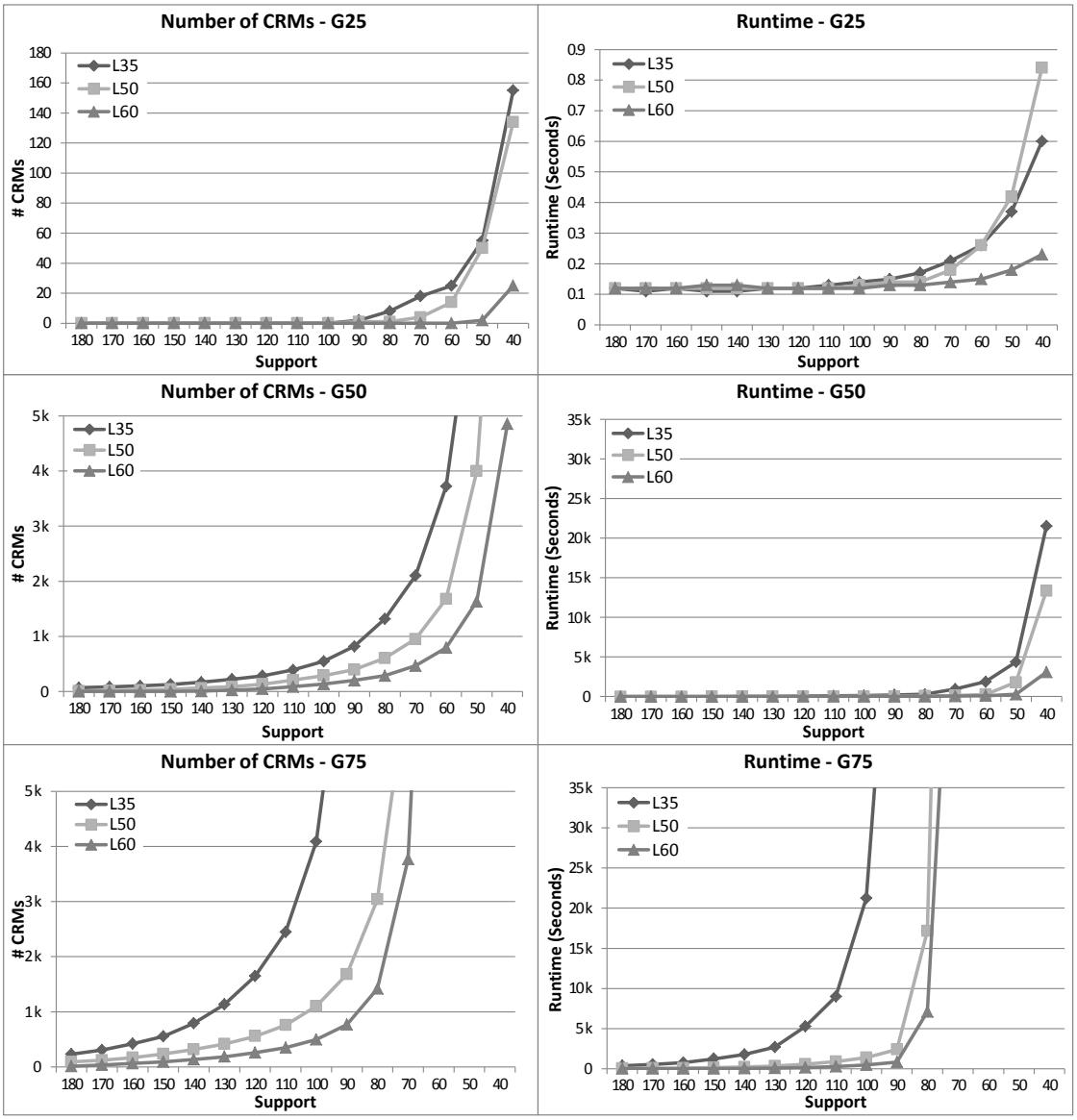

Figure 6.15: Performance of CRMminer$_x$ for different versions of the *DBLP* dataset. For all experiments, β is 0.60, m_{min} is 4, k_{min} is 3 and k_{max} is 10.

dataset, we observe similar increase in both the number of CRMs and the runtime.

Label Diversity & Network Density

The performance of a CRM mining algorithm, as well as any pattern mining algorithm is primarily impacted by the label diversity and the network density of the dataset. To evaluate the performance of our algorithms for different types of dynamic network datasets, we used the $DBLP$ dataset to generate nine different networks varying the number of edge labels and the density of the networks. First, we generated three datasets $L35$, $L50$, and $L60$ containing different number of labels by clustering the publication titles into 35, 50, and 60 groups respectively. Then for each of the three datasets, we generated three networks $G25$, $G50$, and $G75$. The $G25$ dataset was generated by removing all the relations of an author if he/she had co-authored with more than 25 other authors in a single year. Similarly, for $G50$ and $G75$, the maximum co-authorship threshold per year was set to 50 and 75 correspondingly. The total number of authors remained the same while the total number of edges increased from $G25$ to $G75$. Thus, the density is lowest in $G25$ and highest in $G75$. The number of edges in $G25$, $G50$, and $G75$ datasets are 2522412, 3973710 and 4554474 correspondingly.

Figures 12 and 13 present the results using these datasets from CRMminer and CRMminer$_{\text{x}}$, respectively. The results are displayed in increasing order of density from $G25$ to $G75$ and each graph shows the impact of support threshold in finding CRMs. Overall, as the dataset density increases, the number of discovered CRMs increases. This is expected, since the number of patterns in a denser network will most likely be greater. In addition, as the label diversity increases, the number of discovered CRMs decreases. This is because by increasing the number of labels the effective support of a CRM decreases, leading to fewer CRMs. Finally, these results show that a significant improvement in runtimes between CRMminer and CRMminer$_{\text{x}}$ is found when comparing the same graph sets in Figures 12 and 13. For example, a comparison of results between the $G75$ datasets shows that CRMminer$_{\text{x}}$ is able to complete execution for lower thresholds in about $2 - 30$ times faster than CRMminer. In addition, the increase in runtimes as the support decreases is linear and proportional to the output space. Hence, CRMminer$_{\text{x}}$ is an excellent option for processing the dense graphs with low label diversity.

Chapter 7

Mining the Coevolving Induced Relational Motifs

7.1 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 the type of patterns that CIRMs are designed to identify, consider the network of Figure 7.2. 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 four occurrences have evolved into a new motif (M_2) that includes an additional node in the network for 2000. Finally, in the network for 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.

The formal definition of a CIRM that is used in this thesis 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:*

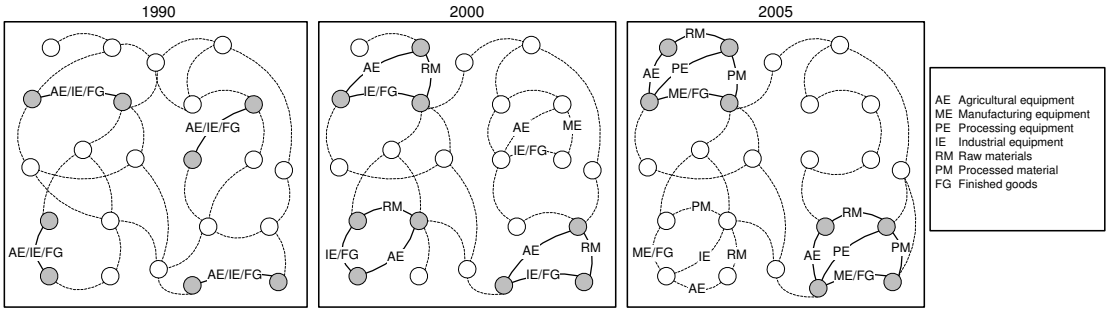

Figure 7.1: An example of a coevolving induced relational motif in the context of a hypothetical country-to-country trading network where labels represent the commodities been traded. Assume the minimum support threshold (ϕ) for the CIRM is 2.

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 thesis, we focus on identifying the CIRMs $c = \{N_c, \langle M_1, \ldots, M_m \rangle\}$ 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 CIRM. A CIRM like x that contains only one edge and two vertices will be called an *anchor*. The anchored CIRM is a subclass of CIRMs that in addition to the conditions of Definition 3, all

motifs of the CIRM share at least an edge (anchor) that itself is a CIRM (i.e., the anchor is an evolving edge). We focus on developing an efficient algorithm to mine all anchored CIRMs. 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 motif 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 thesis 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 as a *frequent* CIRM and it is *valid* if it also satisfies the minimum node overlap constraint (Definition 3(iv)).

7.2 Mining Coevolving Induced Relational Motifs

We developed an algorithm for solving Problem 3, named CIRMminer that performs a depth-first exploration of each anchor along with a canonical labeling that is derived by

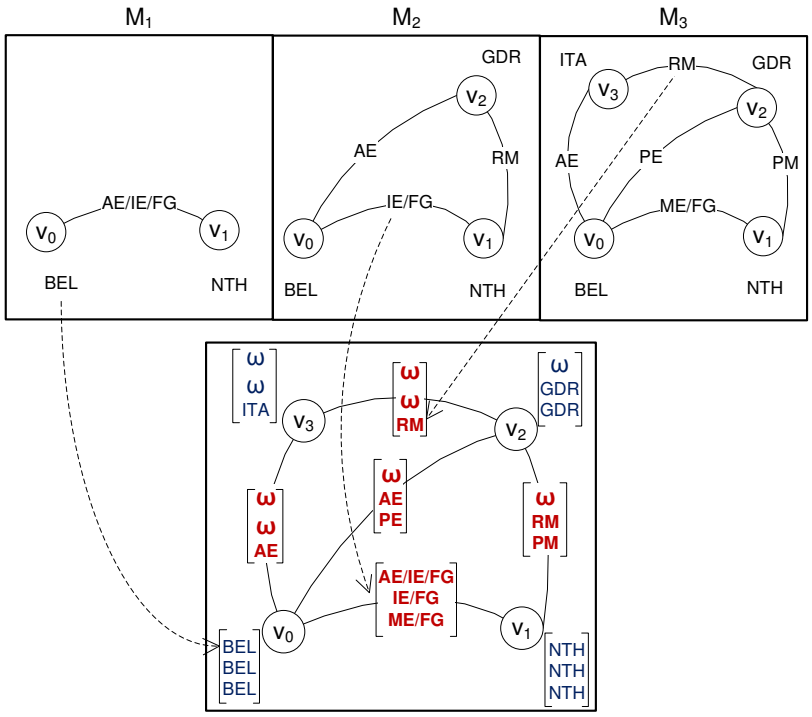

Figure 7.2: A CIRM Representation. The CIRM c consists of 3 motifs $\langle M_1, M_2, M_3\rangle$ and represents relations among vertices $N = \{v_0, v_1, v_2, v_3\}$ using 5 edges. G_c (bottom graph) shows the CIRM representation capturing vertex and edge label vectors.

extending the ideas of the minimum DFS code [65] to the case of CIRMs for redundancy elimination. We stop any further exploration of a CIRM when the pattern does not occur at least ϕ times in the dynamic network $\mathcal{N}$. For the rest of the discussion, any references to a relational motif or a motif will assume it is an induced relational motif.

7.2.1 CIRM Representation

We adopt the method presented in [106] 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).

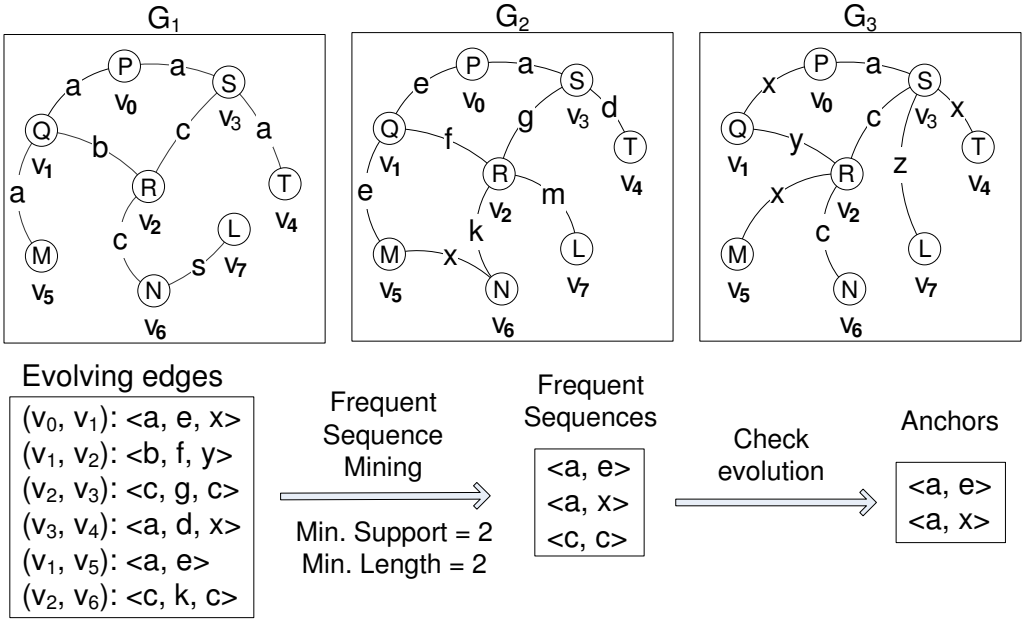

Figure 7.3: The process of mining anchors from the network $\langle G_1, G_2, G_3 \rangle$. Since all vertex labels remained consistent over time, we listed the edge label sequences as the span sequence of the evolving edges.

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 Figure 7.2.

7.2.2 Mining Anchors

The search for CIRMs is initiated by locating the frequent anchors that satisfy the CIRM definition and the restrictions defined in Problem 3. This process is illustrated in Figure 7.3. 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 [107] 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.

7.2.3 CIRM Enumeration

Given an anchor, we generate 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 because we need 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, we use an approach similar to gSpan [65], which we have extended for the problem of CIRM mining. In order to describe how gSpan ideas are adopted in CIRM enumeration process, we first provide a brief overview of gSpan and then define different components of the CIRM algorithm that extends the concepts of gSpan. Once properly defined, the correctness and completeness of frequent CIRM enumeration follows directly from the corresponding proofs of gSpan.

Overview of gSpan

To efficiently search for frequent patterns, mining algorithms use a pattern lattice which is a hierarchical representation of the search space where each node of the lattice corresponds to a pattern. The highest node is an empty pattern (i.e., a single vertex), the next level nodes represent 1-edge patterns, and the nth level nodes represent n-edge patterns. gSpan performs a depth-first exploration of the lattice avoiding redundant

visits to the same node by only traversing a set of edges that form a spanning tree of the lattice. Each node of the lattice is assigned a label, called a DFS code which is formed based on the sequence of the edges added to that node during the depth-first exploration. The lexicographically smallest DFS code of a graph is considered as the canonical label and called the minimum DFS code. Given this labeling process, the set of lattice edges that are used by gSpan to form the spanning tree correspond to edges between nodes with the minimum DFS code. An edge between two successive nodes of the lattice (parent and child) is selected as part of the spanning tree, if the minimum DFS code of the child can be obtained by simply appending the edge to the minimum DFS code of the parent. For example, given a DFS code $A = \langle a_0, a_1, \cdots, a_m \rangle$, a valid child DFS code is $A' = \langle a_0, a_1, \ldots, a_m, b \rangle$, where b is the new edge. This spanning tree guarantees that each node has a unique parent and all nodes are connected [65]. To efficiently grow a node (i.e., pattern) of the lattice, 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 selects a child node for further expansion if that node corresponds to the minimum DFS code.

DFS code of a CIRM

Given a CIRM c, represented as a graph $G_c = (N, E_c)$, we perform a depth-first search in G_c to construct a DFS tree T_c and assign subscripts to the vertices (N) from 0 to $n-1$ for $|N| = n$ according to their discovery time. We define a linear order $\prec_{T_c,E_c}$ based on the combination of the following partial orders to construct the edge sequence for a DFS code of a CIRM. Given two edges $e_1 = (v_{i_1}, v_{j_1})$ and $e_2 = (v_{i_2}, v_{j_2})$, the partial order relations (i.e., $\prec_{T_c,XX}$) are defined as:

a) $e_1, e_2 \in E_{T_c,fw}$, if $j_1 < j_2$, then $e_1 \prec_{T_c,fw} e_2$.

b) $e_1, e_2 \in E_{T_c,bw}$, 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) $e_1 \in E_{T_c,bw}$ and $e_2 \in E_{T_c,fw}$, if $i_1 < j_2$, then $e_1 \prec_{T_c,bw+fw} e_2$.

d) $e_1 \in E_{T_c,fw}$ and $e_2 \in E_{T_c,bw}$, if $j_1 \leq i_2$, then $e_1 \prec_{T_c,bw+fw} e_2$.

Here $E_{T_c,fw}$ and $E_{T_c,bw}$ are the forward and backward edge sets respectively.

Given a DFS tree T_c of a graph G_c, we can order all edges in G_c using $\prec_{T_c,E_c}$ to form an edge sequence that represents a DFS code of G_c. An edge of the DFS code of a CIRM (i.e., G_c) is represented similar to the CIRM 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. For example, the following is a DFS code for CIRM in Figure 7.2.

$$\begin{aligned}
&\langle(0, 1, \langle \texttt{BEL}, \texttt{BEL}, \texttt{BEL}\rangle, \langle \texttt{AE/IE/FG}, \texttt{IE/FG}, \texttt{ME/FG}\rangle, \langle \texttt{NTH}, \texttt{NTH}, \texttt{NTH}\rangle),\\
&\ (1, 2, \langle \texttt{NTH}, \texttt{NTH}, \texttt{NTH}\rangle, \langle \omega, \texttt{RM}, \texttt{PM}\rangle, \qquad\qquad \langle \omega, \texttt{GDR}, \texttt{GDR}\rangle),\\
&\ (2, 0, \langle \omega, \texttt{GDR}, \texttt{GDR}\rangle, \quad \langle \omega, \texttt{AE}, \texttt{PE}\rangle, \qquad\qquad \langle \texttt{BEL}, \texttt{BEL}, \texttt{BEL}\rangle),\\
&\ (2, 3, \langle \omega, \texttt{GDR}, \texttt{GDR}\rangle, \quad \langle \omega, \omega, \texttt{RM}\rangle, \qquad\qquad \langle \omega, \omega, \texttt{ITA}\rangle),\\
&\ (3, 0, \langle \omega, \omega, \texttt{ITA}\rangle, \qquad \langle \omega, \omega, \texttt{AE}\rangle, \qquad\qquad \langle \texttt{BEL}, \texttt{BEL}, \texttt{BEL}\rangle)\rangle
\end{aligned}$$

DFS Lexicographical Ordering of CIRMs

Let two DFS codes of a CIRM be $\alpha = \langle e_\alpha^0, e_\alpha^1, \ldots, e_\alpha^p\rangle$ and $\delta = \langle e_\delta^0, e_\delta^1, \ldots, e_\delta^q\rangle$, where p and q are the number of edges, and $0 \leq p, q$. 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 belonging to α and δ respectively. Now, $e_\alpha^x < e_\delta^y$, if any of the following is true:

i) $e_\alpha^x \in E_{\alpha,bw}$ and $e_\delta^y \in E_{\delta,fw}$, or
ii) $e_\alpha^x \in E_{\alpha,bw}$, $e_\delta^y \in E_{\delta,bw}$, and $j_\alpha^x < j_\delta^y$, or
iii) $e_\alpha^x \in E_{\alpha,bw}$, $e_\delta^y \in E_{\delta,bw}$, $j_\alpha^x = j_\delta^y$ and $\boldsymbol{l_{i_\alpha^x, j_\alpha^x}} < \boldsymbol{l_{i_\delta^y, j_\delta^y}}$, or
iv) $e_\alpha^x \in E_{\alpha,fw}$, $e_\delta^y \in E_{\delta,fw}$, and $i_\delta^y < i_\alpha^x$, or
v) $e_\alpha^x \in E_{\alpha,fw}$, $e_\delta^y \in E_{\delta,fw}$, $i_\alpha^x = i_\delta^y$ and $\boldsymbol{l_{i_\alpha^x}} < \boldsymbol{l_{i_\delta^y}}$, or
vi) $e_\alpha^x \in E_{\alpha,fw}$, $e_\delta^y \in E_{\delta,fw}$, $i_\alpha^x = i_\delta^y$, $\boldsymbol{l_{i_\alpha^x}} = \boldsymbol{l_{i_\delta^y}}$ and $\boldsymbol{l_{i_\alpha^x, j_\alpha^x}} < \boldsymbol{l_{i_\delta^y, j_\delta^y}}$, or
vii) $e_\alpha^x \in E_{\alpha,fw}$, $e_\delta^y \in E_{\delta,fw}$, $i_\alpha^x = i_\delta^y$, $\boldsymbol{l_{i_\alpha^x}} = \boldsymbol{l_{i_\delta^y}}$, $\boldsymbol{l_{i_\alpha^x, j_\alpha^x}} = \boldsymbol{l_{i_\delta^y, j_\delta^y}}$, and $\boldsymbol{l_{j_\alpha^x}} < \boldsymbol{l_{j_\delta^y}}$,

where $E_{X,fw}$ and $E_{X,bw}$ are the forward and backward edge sets. Based on the above definitions, we compare α and δ such that $\alpha \leq \delta$, iff any of the following conditions is true:

a) $F_\omega(\alpha) \leq F_\omega(\delta)$, where $F_\omega(x)$ is the number of edges $(\boldsymbol{l_{i_x^t, j_x^t}})$ that are set to ω, or
b) $\exists t, 0 \leq t \leq min(p, q)$, $e_\alpha^k = e_\delta^k$ for $k < t$, and $e_\alpha^t < e_\delta^t$, or
c) $e_\alpha^k = e_\delta^k$ for $0 \leq k \leq p$ and $p \leq q$.

Note that the DFS lexicographical ordering ranks edges with label vectors containing no ω labels higher than the edges which does.

Minimum DFS Code of a CIRMs

A CIRM can be represented by different DFS trees resulting in different DFS codes. The lexicographically smallest DFS code is called the minimum DFS code ($min\_code(c)$) and selected as the canonical label of the CIRM. Given two CIRMs c and c', c is isomorphic to c' if and only if $min\_code(c) = min\_code(c')$.

CIRM growth

The CIRM enumeration process follows the rightmost extension rule 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, we go 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, we traverse through all snapshots within each motif's span and select 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, we then represent the set of edges resulting from a particular embedding of a CIRM as a sequence of IDs.

Figure 7.4 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, we select the adjacent vertex v_3 for this embedding. Hence, we need 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), we identify 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: $\langle I_1, I_2, I_3, I_4, I_5 \rangle$.

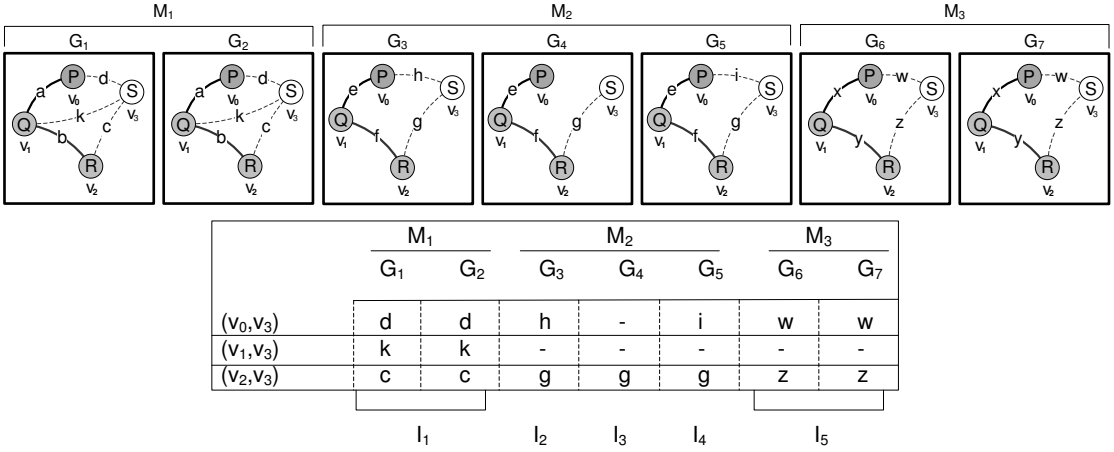

Figure 7.4: Generating ID sequences from a set of edges. Shaded vertices and solid line edges are part of the existing CIRMs. Vertex v_3 is considered as the candidate vertex. For this example, since the vertex labels remain same in all snapshots, the edges are represented using only edge labels.

We represent the collected sets of edges from all embeddings as a collection of ID sequences and apply frequent sequence mining technique [107] 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 7.4 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 7.4), both IDs I_2 and I_4 contains the span that belongs to motif M_2. To identify CIRMs according to Definition 3, we need 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, our 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., $M_j \neq M_{j+1}$). We discard 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.

Enumeration Optimizations

To efficiently perform the above operations, the challenge is to design a strategy that minimizes the additional complexity of generating frequent candidate sets of edges and eliminates redundant extensions. To achieve this, we developed an approach that performs one vertex growth of the CIRM in two phases. In the first phase, to locate an adjacent new vertex of the CIRM, we determine all frequent forward edges from the rightmost path of the CIRM. Each of the candidate edges are then added to the CIRM to form a *pseudo CIRM* that contains an additional edge than the original CIRM. We verify whether the pseudo CIRM contains a minimum DFS code and has sufficient support. If the pseudo CIRM passes both checks, we initiate the second phase to include all backward edges to transform the pseudo CIRM into a frequent CIRM. Note that by discarding all non frequent pseudo CIRMs, a significant performance gain is achieved.

Even though there can be a large number of child CIRMs from one vertex extension, the lexicographic ordering of the CIRM's DFS codes is able to order all candidates and enable us to perform pre-order search on the pattern lattice. It allows us to prune the pattern with non-minimum DFS codes and their descendants without impacting the completeness of the results. The CIRMs that contain additional motifs than the anchor contain a modified version of the original anchor. Since the additional motifs of a child CIRM splits one of the motifs of the parent CIRM's, the new anchor captures the same entity relation with two or more motifs containing the same relation (i.e., edge label) and adjacent motif span. This ensures that the new anchor cannot be a duplicate of another initially identified anchor. Since CIRMminer algorithm generates non-redundant anchored CIRMs, the CIRMs with additional motifs than includes a dynamically generated unique anchor will also be non-redundant.

7.2.4 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 [104] to calculate the minimum support of a CIRM in a dynamic network.

As defined in Chapter 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 CIRM. 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 $m \leq T$, the minimum image based support of c is defined as:

$$\sigma(c, \mathcal{N}) = \min_{v \in V_c} \mid \{\varphi_i(v) \text{: } \varphi_i \text{ is an occurrence of } c \text{ in } \mathcal{N}\} \mid. \tag{7.1}$$

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.

7.2.5 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 Chapter 2) 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:

$$\rho(c) = \frac{\min\limits_{1 \leq i \leq m} \{|V_{M_i}|\}}{|N_c|}, \tag{7.2}$$

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 [105]. 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.

7.3 Experimental Methodology

All experiments are conducted on a 64-bit Linux desktop with 8-core Intel® Core™ i7-3770 processor at 3.40GHz and 16GB of RAM. To ensure that all experiments are easily reproducible, we provided datasets and CIRMminer software at our project site[1] .

7.3.1 Datasets

We have used two different types of datasets to evaluate CIRMminer. We presented all datasets details in Section 4.2. The DBLP co-authorship network is a real world dynamic network that captures yearly co-authorship relations. The bioprocess network (GT) and the sales network (Sales) datasets are based on multivariate time-series data. To characterize the relations among different variables and understand changes over time, we represent the time-series data as a dynamic network. The CIRMs discovered from these networks can be used to characterize the overall network, as shown in Chapter 8.

7.4 Results & Discussion

The evaluation consists of three parts. The first focuses on analyzing the performance of CIRMminer by evaluating how the different parameters associated with the definitions of CIRM impact the performance of the algorithm. The second focuses on comparing the number of discovered patterns and the runtime between CIRMminer and CRMminer [106] for different values of minimum support. The third focuses on assessing the information that can be extracted from the discovered CIRMs by analyzing some of the patterns of the coevolving induced relational motifs that were identified in the three datasets.

In order to asses 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.

[1] `https://sites.google.com/site/cirm2014sup/`

Table 7.1: Support (ϕ) & Overlap (β) study for CIRMs.

Data	β	$\#T_{\text{CIRM}}$	$\#Q_{\text{CIRM}}$	$\#T_{\text{CIRM}}$	$\#Q_{\text{CIRM}}$	$\#T_{\text{CIRM}}$	$\#Q_{\text{CIRM}}$
		ϕ=100; t=268		ϕ=90; t=357		ϕ=80; t=627	
	0.60	1,115	905	1,493	1,189	2,102	1,610
DBLP	0.50	”	1,106	”	1,476	”	2,058
	0.40	”	1,115	”	1,493	”	2,102
		ϕ=35; t=35		ϕ=30; t=1024		ϕ=25; t=6868	
	0.60	6,806	2,461	74,855	4,742	288,059	10,006
GT	0.50	”	4,116	”	9,236	”	23,116
	0.40	”	6,331	”	49,536	”	161,796
		ϕ=35; t=11		ϕ=30; t=72		ϕ=25; t=199	
	0.60	2,335	935	19,567	6,075	74,908	22,227
Sales	0.50	”	1,722	”	12,195	”	42,576
	0.40	”	2,308	”	18,948	”	71,324

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

7.4.1 Performance Results

Minimum Support & Overlap analysis

Table 7.1 shows the results that we gathered by using CIRMminer for different values of the minimum support and overlap threshold using three different datasets. For all datasets, as the support threshold decreases, both the number of discovered CIRMs ($\#T_{\text{CIRM}}$) and the number of valid CIRMs ($\#Q_{\text{CIRM}}$) increase, consequently the runtime increases to discover those CIRMs. The increase in the number of CIRMs is expected due to the anti-monotonic property of the support threshold. Note that the β threshold is applied to select valid CIRMs (Q_{CIRM}) after identifying all CIRMs (T_{CIRM}). Thus, for a specific ϕ value, the mining runtimes are the same for different β thresholds. Similar to the support threshold, we observed that the number of valid CIRMs increases as the overlap threshold decreases. This is expected as the lower overlap requirement allows a greater number of different nodes in the CIRM. For most of the datasets (except GT), the runtimes scale linearly to the size of the output space.

Table 7.2: Minimum Span (m_{min}) study.

		$m_{min} = 3$				$m_{min} = 4$			
Data	ϕ	#A	$\#T_{\text{CIRM}}$	$\#Q_{\text{CIRM}}$	Time	#A	$\#T_{\text{CIRM}}$	$\#Q_{\text{CIRM}}$	Time
DBLP	90	637	1,493	1,476	357	14	15	15	1.38
	80	874	2,102	2,058	627	16	25	25	1.77
	70	1,156	3,066	2,944	1,572	20	46	46	2.44
GT	35	15	6,806	4,116	35	2	34	34	0.19
	30	20	74,855	9,236	1,024	4	74	74	0.33
	25	27	288,059	23,116	6,868	9	204	198	0.59
Sales	30	136	19,567	12,195	72	18	2	2	0.14
	25	175	74,908	42,576	199	33	44	43	0.56
	20	226	312,955	147,232	604	56	570	429	2.47

m_{min} denotes the minimum number of motifs per CIRM, #A denotes the number of anchors, and rest of the columns are described in Table 7.1. For all datasets, β is 0.50, k_{min} is 4 and k_{max} is 8.

For the *DBLP* dataset, as the support threshold decreases from 100 to 80, both the number of CIRMs ($\#T_{\text{CIRM}}$) and the number of valid CIRMs (Q_{CIRM}) increase by 1.8 times and the runtime increases by 2.3 times. For the *Sales* dataset, the relationship between runtime and output space increase is similar to the *DBLP* dataset. For the *GT* dataset, as the support threshold decreases from 35 to 25, the total number of CIRMs ($\#T_{\text{CIRM}}$) increases by 42 times and the runtime increases by 196 times. We noticed that the higher rate of increase in runtime is due to the fact that for ϕ=25 *GT* contains a large number of candidate CIRMs that contain many edges compared to the number of candidates for ϕ=35. For instance, CIRMminer processed $2,748,974$ candidate CIRMs containing 7 vertices and 13 edges on average for ϕ=25 compared to $2,883$ candidates of similar size for ϕ=35. As a result, CIRMminer spent most of its time processing these large candidates and their embedding lists, leading to its substantial increase in runtime.

Minimum Span analysis

The performance of the algorithm for different values of the minimum span (m_{min}) is shown in Table 7.2. The value of m_{min} represents the minimum number of motifs per CIRM. From the reported results in Table 7.2, we observe that as the value of m_{min}

decreases, the number of discovered CIRMs and the runtime increases. The value of m_{min} directly impacts the number of anchors and as the number of anchors increases the total number of CIRMs increases. For the $DBLP$ dataset and $\phi = 90$, the number of anchors increases from 14 to 637 for $m_{min} = 4$ to $m_{min} = 3$. As a result, both the number of discovered CIRMs ($\#T_{\text{CIRM}}$) and the number of valid CIRMs ($\#Q_{\text{CIRM}}$) increases about 99 times and the corresponding runtime increases by 258 times. For the GT dataset, the increase in the number of valid CIRMs and the runtime remains correlated with the minimum span (m_{min}). For $\phi = 30$, as the number of anchors increases from 4 to 20 for $m_{min} = 4$ to $m_{min} = 3$, the number of discovered CIRMs ($\#T_{\text{CIRM}}$) increases by 1000 times, the number of valid CIRMs increases by 125 times, and the runtime increases by 3000 times. For the $Sales$ dataset, we observe the increase in the number of CIRMs and the runtime is similar to the GT dataset.

Table 7.3: 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. $\#T_{\text{CRM}}$ denotes the total number of discovered CRMs. $\#Q_{\text{CRM}}$ denotes the number of CRMs that meet the β threshold out of $\#T_{\text{CRM}}$. Rest of the column labels are described in Table 7.1. For all datasets, β is 0.50, m_{min} is 3, k_{min} is 4 and k_{max} is 8.

7.4.2 Comparing CRMs with CIRMs

Table 7.3 presents the number of patterns discovered and the runtime comparison between CRMminer [106] and CIRMminer for different values of the minimum support using all three datasets. For both datasets, as the support threshold decreases, the number of discovered valid CRMs and CIRMs increases, consequently the runtime increases to discover those CRMs and CIRMs. 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. Though both algorithms start the enumeration process using the same number of anchors, the induced subgraph constraint of the CIRMs is able to eliminate large number of CIRM candidates and prune further expansion of those patterns. The runtime to discover CIRMs is lower by 24 to 938 times than mining CRMs as the support threshold decreases. Even though additional calculation is needed for the induced isomorphism check, CIRMminer enumerates a fraction (i.e., the induced ones) of all frequent CRMs. Similar output and runtime ratios are observed for the *DBLP* and the *Sales* datasets.

Chapter 8

Qualitative Analysis & Applications

8.1 Qualitative Analysis of EIRS

In this section we present some of the EIRSs that were discovered by our algorithm in order to illustrate the type of information that can be extracted from the dynamic networks by focusing on how stable relations changed over time.

8.1.1 EIRS Case Studies

The EIRSs that are presented in this section have been selected as the highest ranking EIRSs based on the TD interestingness measure.

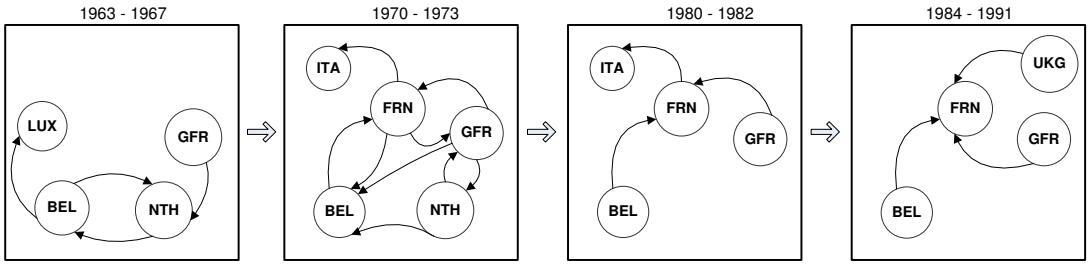

Figure 8.1: An EIRSs capturing a trade relation between EU countries. The nodes in the figure are LUX=Luxembourg, BEL=Belgium, GFR=German Federal Republic, NTH=Netherlands, ITA=Italy, and UKG=United Kingdom.

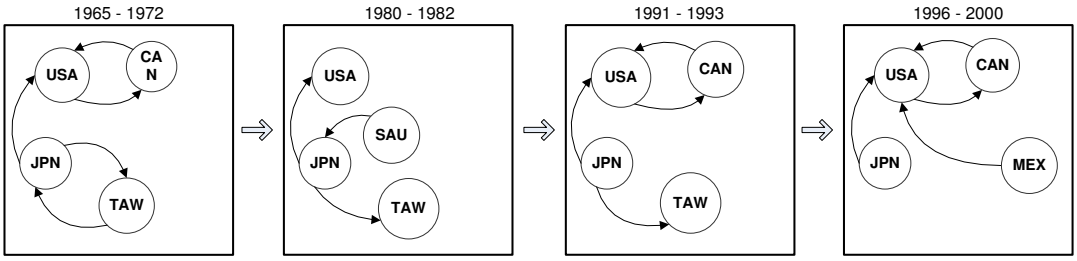

Figure 8.2: An EIRSs capturing a trade relation of USA. The nodes in the figure are USA=United States of America, CAN=Canada, JPN=Japan, TAW=Taiwan, SAU=Saudi Arabia, and MEX=Mexico.

In Figure 8.1 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 total drift for this EIRS is $12/18 = 0.67$. 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 to 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 captures 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 Figure 8.2 we present another EIRS generated from the trade network capturing a stable trade relation of USA with other countries over 35 years period. The total drift for this EIRS is $7/16 = 0.44$. 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 USA and Japan.

In Figure 8.3 we present an EIRS generated from the email communication network

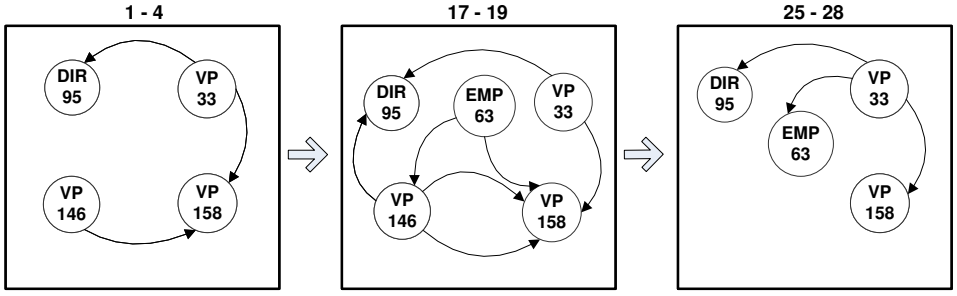

Figure 8.3: An EIRSs capturing Enron email traffic pattern.

capturing email exchange patterns among a group of employees over a period of time. The total drift for this EIRS is $7/12 = 0.58$. Although it is difficult to understand the communication of the employees without the message content, the direction of the communication can be found in this EIRS. We see that VP[id:33] had always initiated the conversation and was very active in email communication to have stable relations over all the captured periods. It is interesting to notice that VP[id:146] is not present in last period (25-28). One can investigate and confirm whether he/she was replaced or terminated and the cause for such actions.

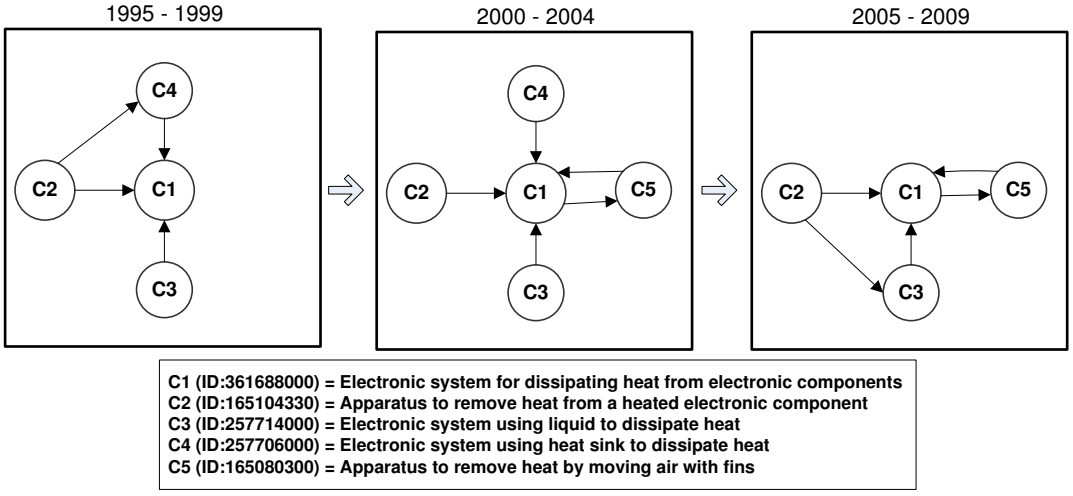

Figure 8.4: An EIRSs capturing patent class relations. The nodes in figure denote patent classes and the legend captures the USPTO definition of the classes C1 to C5.

In Figure 8.4 we present an EIRS generated from the patent citation network capturing the evolution of the relations between some patent classes over a 15 years period. The total drift for this EIRS is $7/14 = 0.5$. Based on the illustration, the patent classes C2, C3 and C4 were citing C1 during 1995-1999. We can interpret that as the patents of class C1 are earlier inventions and later the patents of class C2, C3, and C4 used the ideas found in the patents belonging to C1. Over time the relations changed as class C5 appears in later years and both C1 and C5 are citing each other. It is possible that patents of class C5 represent a newer technology that cited earlier patents of class C1 as reference and the newer patent of class C1 are using C5 as reference. This captures a complex relational dependence between entities in a dynamic network. Moreover, we also observed that class C4 disappeared in the period of 2005-2009. This could indicate that the technology introduced in the products of class C4 is no longer used.

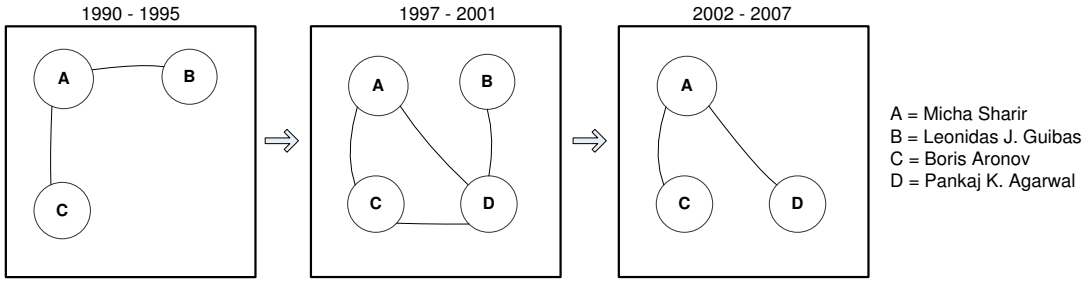

Figure 8.5: An EIRSs capturing co-authorship relations. The nodes in figure denote authors and the legend captures the name of the authors A to D.

In Figure 8.13 we present an EIRS generated from the co-authorship network capturing the collaboration among a group of authors and their progress in scientific achievements over 18 years period. The total drift for this EIRS is $5/8 = 0.62$. The EIRS captures co-authorship relations among Micha Sharir, Leonidas J. Guibas, Boris Aronov and Pankaj K. Agarwal. Based on the publications listed in DBLP dataset, the contributions from these authors were primarily in the area of Computational Geometry. Each IRS captures the collaboration patterns among the authors over 5 years or more. Note that an EIRS captures the collaboration among the significant contributors in a certain field, since most of the authors consistently contributed for 15 years of more.

Table 8.1: Interestingness Measure Comparison.

	TD		MD		MDIS	
Rank	Score	Count	Score	Count	Score	Count
1	0.62	2	0.75	2	0.80	1
2	0.58	4	0.70	1	0.79	2
3	0.57	1	0.69	1	0.73	3
4	0.56	1	0.68	3	0.71	1
5	0.55	23	0.67	8	0.70	1

Dataset DBLP is used for this experiment, the number of vertices in an IRS is 3 to 5, the maximum allowed vertex difference between two linked IRSs is 1 and ϕ=5. Rank denotes the top 5 highest ranking EIRS based on the interestingness measure. Score denotes the interestingness score based on the type of measure. Count denotes the number of EIRSs that got the specific score. TD denotes Total Drift. MD denotes Mean Drift. MDIS denotes Mean Drift from Initial State.

8.1.2 Analysis of Interestingness Measures

Based on the experimental results captured in Section 5.4.2, it is possible to detect a large number of EIRSs depending on the choice of the search parameters. Using the interesting measures TD, MD, and MDIS, we can rank the EIRSs to only focus on the EIRSs that capture highest amount of changes. We have performed some comparative analysis of the interesting measures using the DBLP dataset and captured in Table 8.1. Based on the parameters used for the EIRSs search, we collected a total of 125 EIRSs and all three interestingness measures were used to rank these EIRSs. In Table 8.1, we noticed that each measure is able to discriminate the EIRSs differently. The total number of EIRSs that scored in the top 5 rank are 31 (25%) for TD measure, 15 (12%) for MD measure, and 8 (6%) for the MDIS measure respectively. Based on this observation, we can state that MDIS measure is able to better differentiate the changes in EIRSs than TD and MD measure.

We have included top 5 ranking EIRSs from all three measures in Figure 8.6, Figure 8.7, and Figure 8.8. In this section, we will refer each EIRS captured in these figures according to its ranking. For example, the first EIRS of Figure 8.6 is referred as TD (1). Based on the discussion in Section 5.3, TD measure is able to detect the overall changes of an EIRS. In Figure 8.6, we can see how the patterns are capturing the changes in each EIRS. The EIRS TD (1) contains 5 unique edges and a total of 8 edges among

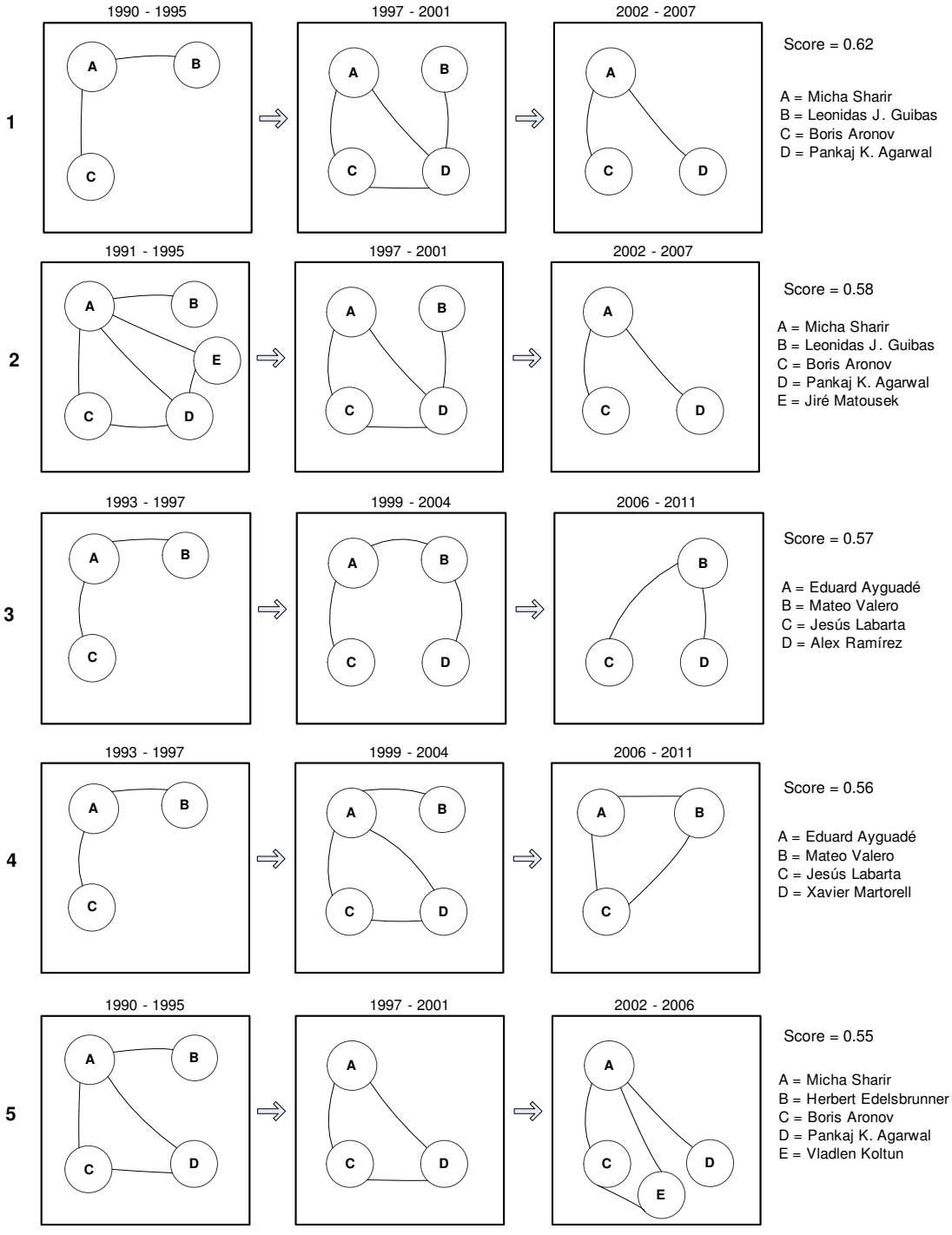

Figure 8.6: Top 5 EIRSs based on TD measure.

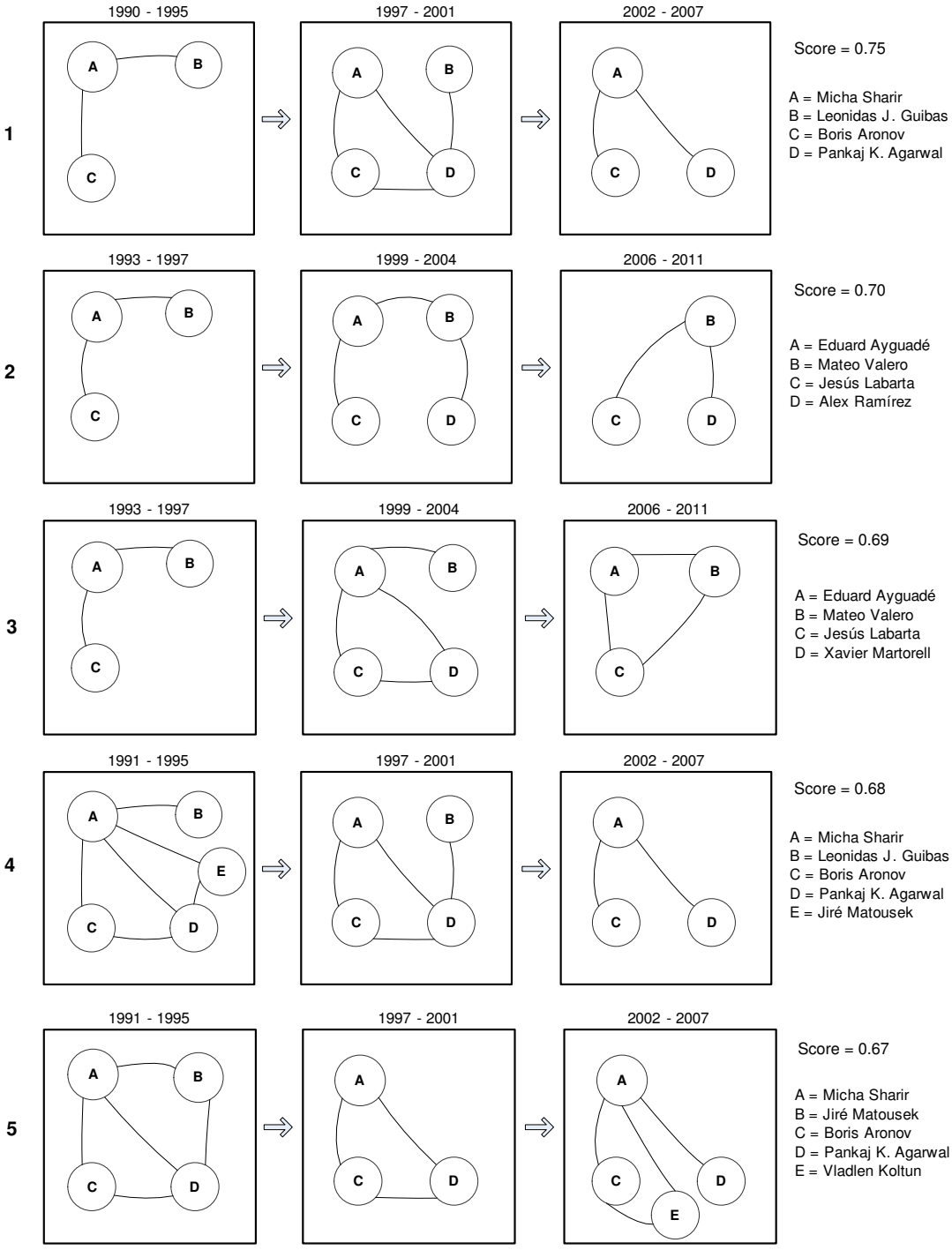

Figure 8.7: Top 5 EIRSs based on MD measure.

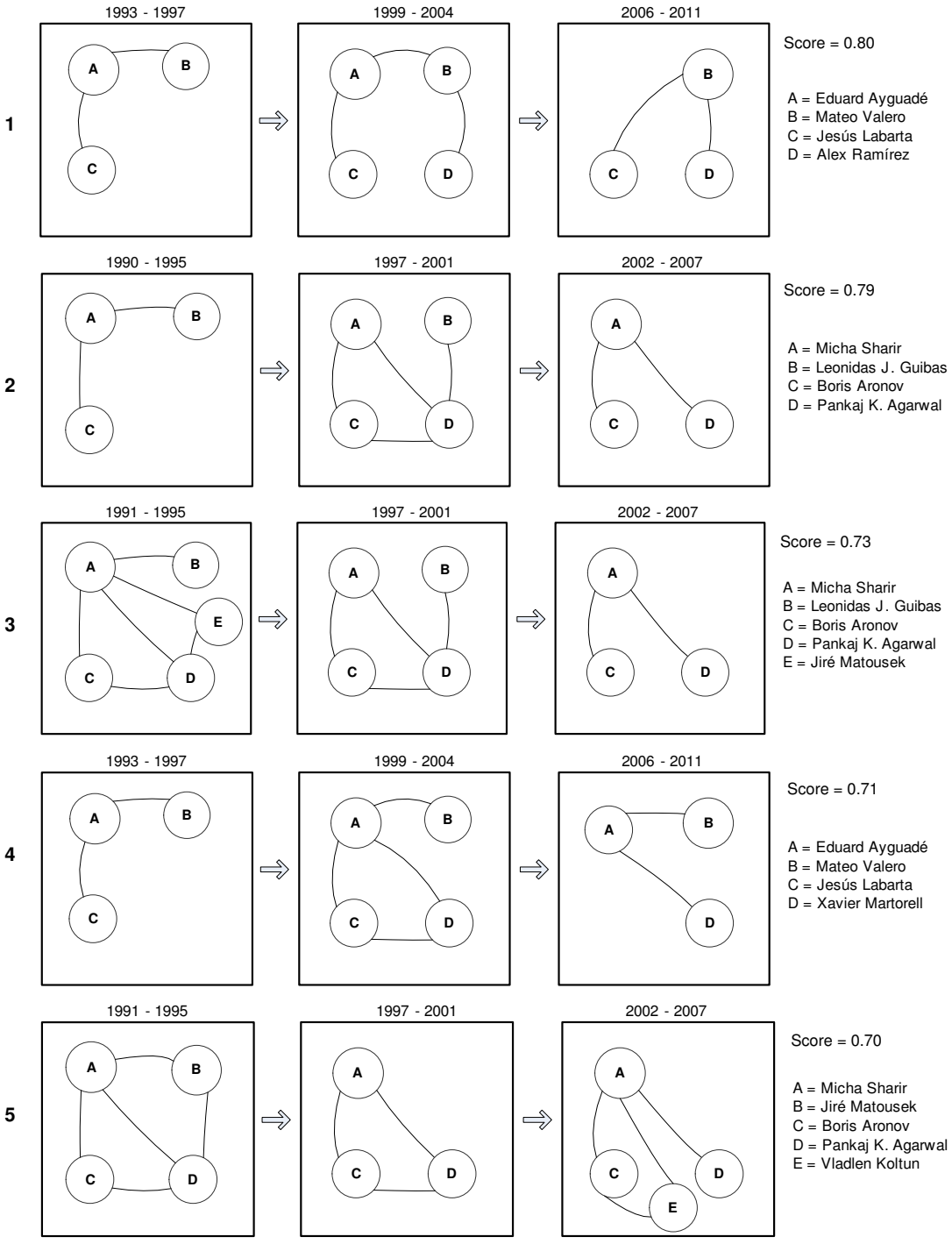

Figure 8.8: Top 5 EIRSs based on MDIS measure.

its constituent IRSs. The relations among the nodes A, B and C in the starting IRS evolved as node D joined in the second IRS and finally the changed further in the third IRS as node B disappeared. The MD measure captures the incremental changes among the IRSs of an EIRS. In Figure 8.7, we can see that the EIRS MD (1) (same as TD (1)) received a high score, since the changes between the first and the second IRSs and the changes between the second and the third IRSs are significant. The MD measure's sensitivity towards incremental change is clearly noticeable in the EIRS MD (2) (same as TD (3)). The MD measure could detect the changes between each successive IRSs of MD (2) and ranked higher than the TD measure. In case of the MDIS measure, it captures the changes among the IRSs of an EIRS comparing to its starting IRS. In Figure 8.8, the EIRS MDIS (1) received the highest score, since the changes between the first and the second and the changes between the first and the third IRSs are significant. Note that none of the existing relations in the first IRS is present in the third IRS and such changes are successfully noticed by the MDIS measure and enabled it to receive the maximum score. This EIRS is also present as MD (2) and TD (3).

8.2 Qualitative Analysis of CRM

In this section we present some of the CRMs that were discovered by our algorithm in order to illustrate the information that can be gathered from the dynamic networks by focusing on coevolution of the relational entities.

8.2.1 DBLP Case Studies

For the $DBLP$ dataset, the yearly co-authorship relations among the authors are divided into 50 clusters based on the title of the papers (Section 4.2). To rank the discovered CRMs, we use the cosine similarity between the centroids of the clusters, referred 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. Note that the clusters are based on the publication titles and we use the most frequent words that belong to a

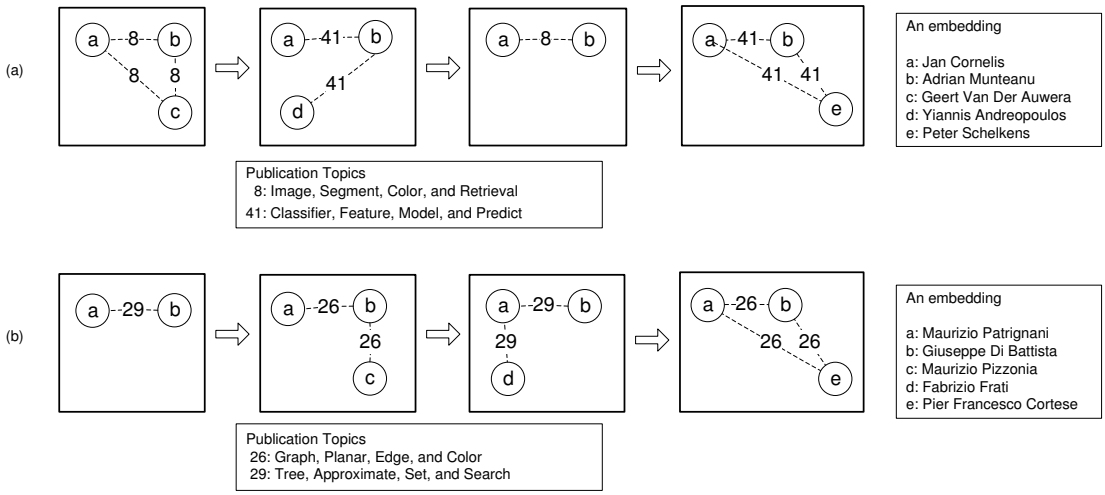

Figure 8.9: Two CRMs capturing co-authorship patterns. The edge labels represent the domain or subject of the publications the authors were involved together. The vertices are labeled to show the changes in relations between the nodes. These CRMs are collected using $\phi = 90$ and $\beta = 0.60$.

cluster to describe the topic it represents.

Two of the high-ranked CRMs are shown in Figure 8.13. 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.

8.2.2 Sales Case Studies

For the *Sales* dataset, we ranked the discovered CRMs according to their size (i.e., the number of edges) and larger CRMs are ranked higher. We want to capture the sales pattern where a large group of products either gain or lose their sales correlation (i.e., tendency of being sold together).

In Figure 8.10(a), a CRM of size 8 is presented from the Sales network capturing

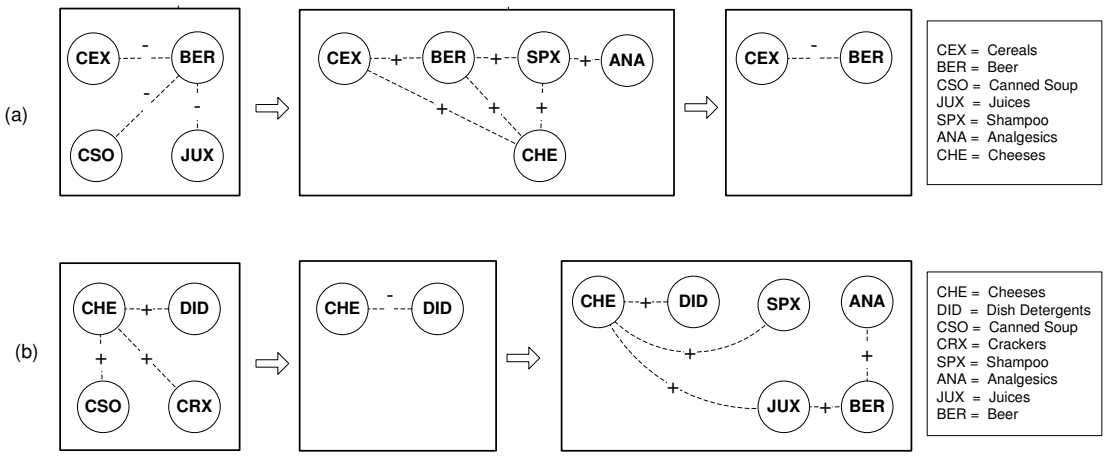

Figure 8.10: Two CRMs capturing store sales patterns. The edge labels + and − correspond to positive and negative correlation between entities. These CRMs are collected using $\phi = 40$ and $\beta = 0.40$.

correlated product groups at a certain period and the changes in their relations. At first, the sale of CEX, CSO, and JUX product groups seem to be negatively correlated with BER. In the next motif, the sale of five product groups becomes positively correlated. Based on the relations among the nodes, the itemsets (CEX, BER, CHE) and (BER, SPX, CHE) are strongly correlated. In the last motif, the relation between the items changed as (CEX, BER) became negatively correlated. Based on the details of the different embeddings and motif occurrence period, it seems that during the holiday periods (i.e., thanksgiving, Christmas) all the product groups are positively correlated.

In Figure 8.10(b), another CRM of size 7 is presented. In this case, product groups are all positively correlated at the beginning. (DID, CSO, CRX) are positively correlated with CHE. In the next motif, the sale of (CHE, DID) becomes negatively. However, at the last period, (DID, JUX, SPX) product groups become positively correlated with CHE. The sale of (JUX, ANA) are also positively correlated with BER. Note that one can focus on understanding why the relation between CHE and DID changed and try to plan for some actions that stores can take to change the sales of those products.

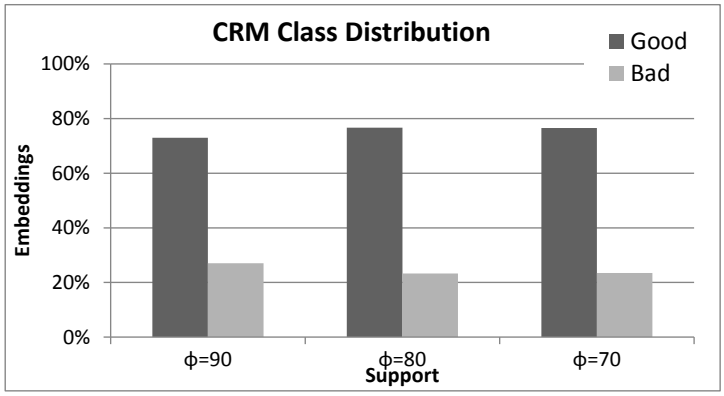

Figure 8.11: A distribution of the CRM embeddings. The Good class represents the production runs with high yield and the Bad class represents with poor yield. CRMs were collected using ϕ = (90, 80 and 70), $\beta = 0.60$, $m_{min} = 3$, $k_{min} = 4$, and $k_{max} = 10$.

8.2.3 Genentech Case Studies

For the *GT* dataset, the discovered CRMs display the dynamics of various process parameters during the cell culture process. 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 discovered CRMs by using them as features to discriminate the production runs. 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, we analyzed the embeddings of the discovered CRMs from three different experiments using ϕ of 90, 80 and 70. Since we have 96 labeled runs, we do not count the embeddings that belong to unlabeled runs. Figure 8.14 shows the class distribution of the embeddings for the discovered CRMs. It shows that the CRMs were present mostly as part of the high yield runs, since more than 70% 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.

Figure 8.12 shows a CRM of size 10 from the *GT* network capturing the changes in

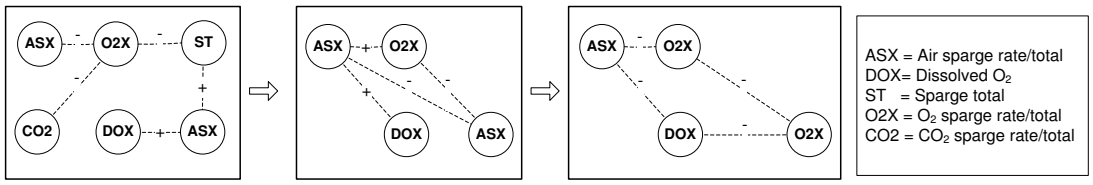

Figure 8.12: A CRM capturing a cell culture bioprocess pattern. The edge labels + and − correspond to positive and negative correlation between entities. This CRM is collected using $\phi = 90$ and $\beta = 0.50$.

entity correlation based on their recorded measurements at a particular stage of the cell culture process. Based on the embedding details, we noticed that the motif spans are mostly non overlapping. This indicates that the changes in the entity relations occur at different stages of the cell culture process based the experimental/process environment setup.

8.3 Qualitative Analysis of CIRM

DBLP Case Studies

For the *DBLP* dataset, the yearly co-authorship relations among the authors are divided into 50 clusters based on the title of the papers (Section 4.2). Note that the clusters are based on the publication titles and we use the most frequent words that belong to a cluster to describe the topic it represents. Two of the CIRMs are shown in Figure 8.13 capture the frequent co-authorship 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 shows the periodic changes in research topics represented as 2 and 20 and the topic similarity is 0.15.

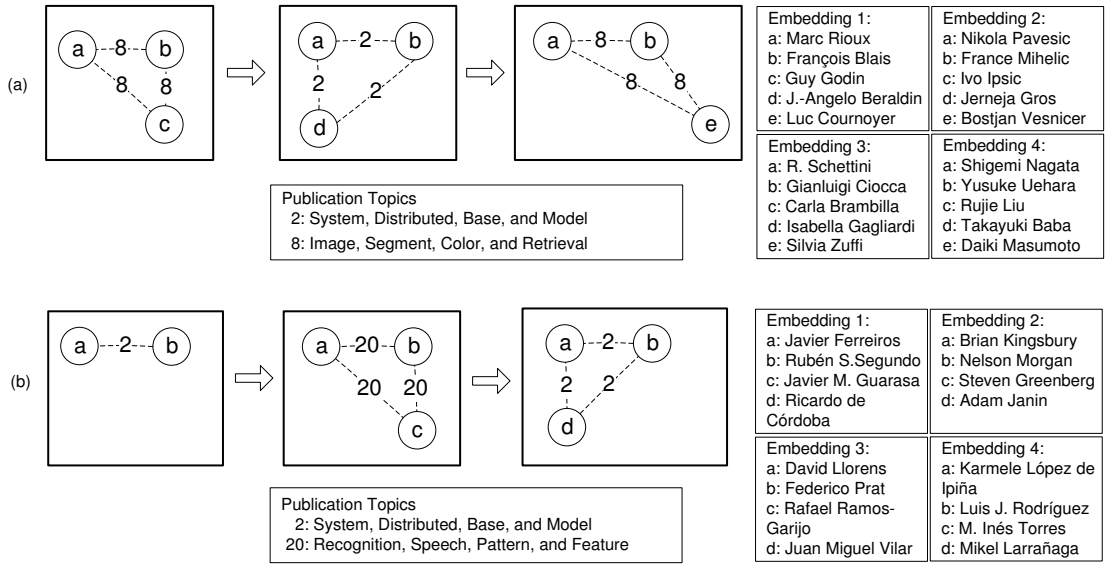

Figure 8.13: Two CIRMs capturing co-authorship patterns. The edge labels represent the domain of the publications. The vertices are labeled to identify the authors in an embedding. These CIRMs are collected using $\phi = 100$ and $\beta = 0.50$.

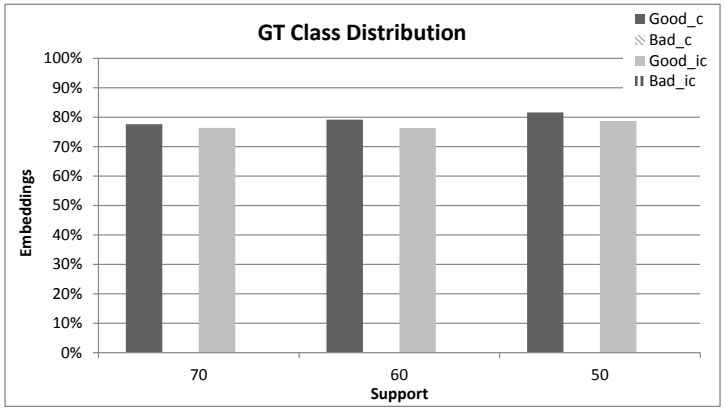

Figure 8.14: A distribution of the CIRM embeddings. The Good class represents the production runs with high yield and the Bad class represents with poor yield. CIRMs were collected using $\phi = (70, 60 \text{ and } 50)$, $\beta = 0.50$, $m_{min} = 3$, $k_{min} = 3$, and $k_{max} = 8$.

Genentech Case Studies

The authors of [106] showed that the CRMs can be used as features for building a predictive model. 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. They analyzed the embeddings of the discovered CRMs and showed that the CRMs are present mostly as part of the Good runs. We wanted to ensure that the underlying network characteristics captured by CRMs are still captured by CIRMs.

Figure 8.14 shows the class distribution of the embeddings for the discovered CIRMs and CRMs. It shows that the CIRMs are present mostly as part of 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). 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.

Chapter 9

Conclusion

In this dissertation 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 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.

9.1 Thesis Summary

The objective of this dissertation has been identifying different evolving relational patterns from dynamic relational networks that capture valuable information characterizing the underlying network. In this section we summarize the contributions and the results.

Mining the Evolution of Conserved Relational States We presented an algorithm for finding all maximal non-redundant evolution paths of the induced relational states in a dynamic network. This can be used to discover the transitions of the conserved relational states over time and to better understand the cause of such changes in the stable patterns in a dynamic network. Our experimental evaluation on multiple real world datasets show that the algorithm is able to discover interesting evolution paths

from all datasets and can scale well to large and dense dynamic networks.

Mining the Evolution of Conserved Relational States We introduced coevolving relational motifs to represent patterns that change in a consistent way over time in a dynamic network and presented an algorithm to efficiently find all frequent coevolving relational motifs. The algorithm can be used to discover unknown coordination mechanisms in a system by identifying the patterns that evolve and move in a similar and highly conserved fashion in the dynamic networks. The experimental evaluation using multiple real world datasets show that CRMminer is able to discover CRMs from all datasets and $\text{CRMminer}_{\text{x}}$ scales better than CRMminer for large and dense dynamic networks. Further, the qualitative analysis shows that the discovered patterns capture important information and can be used as differentiating features for other mining problems.

Mining the Evolution of Conserved Induced Relational States 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 handles the additional complexity of ensuring induced isomorphism and allows the anchored CIRMs to grow beyond the initial size. Using multiple 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.

9.2 Future Research Directions

Evolving Induced Relational States (EIRS) Extensions There are a number of ways that the EIRS definition can be modified to handle special classes of EIRSs. First, the definition can be extended to incorporate minimum and/or maximum constraints on the length of the gap (i.e., number of snapshots) between successive IRSs. These constraints will allow for identification of EIRSs that can capture relational changes due

to some periodic activity/influence in the dynamic network and can be imposed while mining the maximal evolution paths of the IRSs Section 5.2.2. Second, we can relax the requirement that an EIRS's successive relational states are supported by disjoint snapshot subsequences (i.e., constraint (ii) of Definition 1). In this case, the suffix of S_i's supporting snapshot subsequence can overlap with the prefix of S_{i+1}'s supporting snapshot subsequence, as long as the non-overlapping portion of a relational state's supporting snapshot subsequence contains at least ϕ' snapshots, where $\phi' \leq \phi$. This will allow for the identification of EIRSs whose transition between relational states have temporal overlaps. This constraint can also be imposed while mining the maximal evolution paths of the IRSs Section 5.2.2. Finally, we can impose constraints based on relational similarity between successive IRSs. This can be used to identify EIRSs in which both the entities as well as their relations change gradually over time. Depending on the characteristics of the underlying application domain and the analysis requirements, different approaches can be used for measuring this conservation.

Conserved Relational Motifs (CRMs) Extensions There are a number of ways the CRM definition can be modified to address important special classes of CRMs. First, require that the set of identified CRMs is non-redundant in the sense that no CRM is a subsequence of another CRM. This can substantially reduce the number of identified CRMs without loss of information. Second, the occurrences of the motifs are temporally synchronized, i.e., the snapshot in which each CRM s motif occurs is the same. Third, mine the CRMs that do not contain an anchor. Finally, require that the set of identified CRMs to be closed; i.e., at least one of its extensions occurs fewer times. Note that all above mentioned extensions are applicable in case of CIRMs too.

Common Extensions There are 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 [108]. There are several algorithms, such as LEAP [109], CORK [110], GraphSig [111], and LTS [112] 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 the discriminative potential function during dynamic pattern enumeration for pruning the exponential search space.

References

[1] Karsten M. Borgwardt, Hans-Peter Kriegel, and Peter Wackersreuther. Pattern mining in frequent dynamic subgraphs. In *IEEE ICDM*, pages 818–822, 2006.

[2] Deepayan Chakrabarti, Ravi Kumar, and Andrew Tomkins. Evolutionary clustering. In *ACM KDD*, pages 554–560, 2006.

[3] M. Berlingerio, F. Bonchi, B. Bringmann, and A. Gionis. Mining graph evolution rules. *Machine Learning and Knowledge Discovery in Databases*, pages 115–130, 2009.

[4] L. Cerf, T. Nguyen, and J.F. Boulicaut. Discovering relevant cross-graph cliques in dynamic networks. *Foundations of Intelligent Systems*, pages 513–522, 2009.

[5] C. Robardet. Constraint-based pattern mining in dynamic graphs. In *IEEE ICDM*, pages 950–955, 2009.

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

[7] Ruoming Jin, Scott McCallen, and Eivind Almaas. Trend motif: A graph mining approach for analysis of dynamic complex networks. In *IEEE ICDM*, pages 541–546, 2007.

[8] Elise Desmier, Marc Plantevit, Céline Robardet, and Jean-François Boulicaut. Cohesive co-evolution patterns in dynamic attributed graphs. In *Discovery Science*, pages 110–124. Springer, 2012.

[9] Danah Boyd and Nicole Ellison. Social network sites: Definition, history, and scholarship. *Journal of Computer-Mediated Comm.*, 13(11), October 2007.

[10] Anurat Chapanond, Mukkai S. Krishnamoorthy, and Bülent Yener. Graph theoretic and spectral analysis of enron email data. *Comput. Math. Organ. Theory*, 11(3):265–281, 2005.

[11] Jana Diesner, Terrill L. Frantz, and Kathleen M. Carley. Communication networks from the enron email corpus ït's always about the people. enron is no different*:* *Comput. Math. Organ. Theory*, 11(3):201–228, 2005.

[12] B.W. Bader, R.A. Harshman, and T.G. Kolda. Temporal analysis of semantic graphs using asalsan. In *ICDM '07. Seventh IEEE Int. Conf. on Data Mining*, pages 33–42, October 2007.

[13] W. W. Cohen. Enron email dataset. Website, 2005. `http://www.cs.cmu.edu/~enron/`.

[14] Corinna Cortes and Daryl Pregibon. Giga-mining. In *Knowledge Discovery and Data Mining*, pages 174–178, 1998.

[15] Jukka-Pekka Onnela, Jari Saramäki, Jörkki Hyvönen, Gábor Szabó, M Argollo de Menezes, Kimmo Kaski, Albert-László Barabási, and János Kertész. Analysis of a large-scale weighted network of one-to-one human communication. *New Journal of Physics*, 9(6):179–204, 2007.

[16] Yehuda Koren, Stephen C. North, and Chris Volinsky. Measuring and extracting proximity graphs in networks. *ACM Trans. Knowl. Discov. Data*, 1(3):12, 2007.

[17] David Liben-Nowell and Jon Kleinberg. The link prediction problem for social networks. *Journal of the American Society for Information Science and Technology*, pages 1019–1031, 2007.

[18] Xiaoming Liu, Johan Bollen, Michael L. Nelson, and Herbert Van de Sompel. Co-authorship networks in the digital library research community. *Information Processing and Management*, 41(6):1462–1480, 2005.

[19] Steve Lawrence, C. Lee Giles, and Kurt Bollacker. Digital libraries and autonomous citation indexing. *Computer*, 32(6):67–71, 1999.

[20] Paul Zhang and Lavanya Koppaka. Semantics-based legal citation network. In *ICAIL '07: Proc. of the 11th Int. Conf. on Artificial intelligence and law*, pages 123–130, New York, NY, USA, 2007. ACM.

[21] Sergey Brin and Lawrence Page. The anatomy of a large-scale hypertextual web search engine. *Computer Networks and ISDN Systems*, 30(1-7):107–117, 1998.

[22] Bing Liu. *Web Data Mining: Exploring Hyperlinks, Contents, and Usage Data*. Springer, 2007.

[23] Jon M. Kleinberg. Authoritative sources in a hyperlinked environment. *Journal of the ACM*, 46(5):604–632, 1999.

[24] Xiaohua Hu. Mining and analysing scale-free protein protein interaction network. *Int. Journal of Bioinformatics Research and Applications*, 1(1):81–101, 2005.

[25] Chengbang Huang, F. Morcos, S.P. Kanaan, S. Wuchty, D.Z. Chen, and J.A. Izaguirre. Predicting protein-protein interactions from protein domains using a set cover approach. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 4(1):78–87, 2007.

[26] B. Schwikowski, P. Uetz, and S. Fields. A network of protein-protein interactions in yeast. *Nature Biotechnology*, 18(12):1257–1261, December 2000.

[27] Tie Wang, Jeffrey W. Touchman, Weiyi Zhang, Edward B. Suh, and Guoliang Xue. A parallel algorithm for extracting transcription regulatory network motifs. In *BIBE '05: Proc. of the Fifth IEEE Symposium on Bioinformatics and Bioengineering*, pages 193–200, Washington, DC, USA, 2005. IEEE Computer Society.

[28] Hong-Chu Chen, Hsiao-Ching Lee, Tsai-Yun Lin, Wen-Hsiung Li, and Bor-Sen Chen. Quantitative characterization of the transcriptional regulatory network in the yeast cell cycle. *Bioinformatics*, 20(12):1914–1927, 2004.

[29] M. Madan Babu, Sarah A. Teichmann, and L. Aravind. Evolutionary dynamics of prokaryotic transcriptional regulatory networks. *Journal of Molecular Biology*, 358(2):614–633, April 2006.

[30] Huilei Xu, Christoph Schaniel, Ihor R Lemischka, and Avi Ma'ayan. Toward a complete in silico, multi-layered embryonic stem cell regulatory network. *Wiley Interdisciplinary Reviews: Systems Biology and Medicine*, 2(6):708–733, 2010.

[31] Seung-Taek Park and David M. Pennock. Applying collaborative filtering techniques to movie search for better ranking and browsing. In *KDD '07: Proc. of the 13th ACM SIGKDD Int. Conf. on Knowledge discovery and data mining*, pages 550–559, New York, NY, USA, 2007. ACM.

[32] Shlomo Berkovsky, Tsvi Kuflik, and Francesco Ricci. Distributed collaborative filtering with domain specialization. In *RecSys '07: Proc. of the 2007 ACM Conf. on Recommender systems*, pages 33–40, New York, NY, USA, 2007. ACM.

[33] Yun Chi, Xiaodan Song, Dengyong Zhou, Koji Hino, and Belle L. Tseng. Evolutionary spectral clustering by incorporating temporal smoothness. In *KDD '07: Proc. of the 13th ACM SIGKDD Int. Conf. on Knowledge discovery and data mining*, pages 153–162, New York, NY, USA, 2007. ACM.

[34] Lei Tang, Huan Liu, Jianping Zhang, and Zohreh Nazeri. Community evolution in dynamic multi-mode networks. In *ACM KDD*, pages 677–685, 2008.

[35] Hanghang Tong, Spiros Papadimitriou, Philip S. Yu, and Christos Faloutsos. Proximity tracking on time-evolving bipartite graphs. In *Proc. of the SIAM Int. Conf. on Data Mining, SDM 2008*, pages 704–715. SIAM, 2008.

[36] Sitaram Asur, Srinivasan Parthasarathy, and Duygu Ucar. An event-based framework for characterizing the evolutionary behavior of interaction graphs. In *KDD '07: Proc. of the 13th ACM SIGKDD Int. Conf. on Knowledge discovery and data mining*, pages 913–921, New York, NY, USA, 2007. ACM.

[37] Yi Wang, Bin Wu, and Nan Du. Community evolution of social network: Feature, algorithm and model, 2008. Forthcoming.

[38] Tanya Y. Berger-Wolf and Jared Saia. A framework for analysis of dynamic social networks. In *KDD '06: Proc. of the 12th ACM SIGKDD Int. Conf. on Knowledge discovery and data mining*, pages 523–528, New York, NY, USA, 2006. ACM.

[39] Chayant Tantipathananandh, Tanya Berger-Wolf, and David Kempe. A framework for community identification in dynamic social networks. In *KDD '07: Proc. of the 13th ACM SIGKDD Int. Conf. on Knowledge discovery and data mining*, pages 717–726, New York, NY, USA, 2007. ACM.

[40] M. Krishnamoorthy Evrim Acar, S. A. Camtepe and B. Yener. Modeling and multiway analysis of chatroom tensors. In *In Proc. of IEEE Int. Conf. on Intelligence and Security Informatics*, volume 3495, pages 256–268. Springer Berlin, 2005.

[41] Thomas L. Friedman. *The world is flat: A brief history of the twenty-first century.* Farrar, Straus & Giroux, 2005.

[42] Stefan Kramer, Luc De Raedt, and Christoph Helma. Molecular feature mining in HIV data. In *Proc. of the seventh ACM SIGKDD Int. conf. on Knowledge discovery and data mining (KDD '01)*, pages 136–43, New York, NY, USA, 2001. ACM.

[43] Jian Pei, Jiawei Han, Behzad Mortazavi-Asl, Helen Pinto, Qiming Chen, Umeshwar Dayal, and Meichun Hsu. Prefixspan: Mining sequential patterns by prefix-projected growth. In *ICDE*, pages 215–224, 2001.

[44] Mohammed J Zaki. Efficiently mining frequent trees in a forest. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 71–80. ACM, 2002.

[45] Tatsuya Asai, Kenji Abe, Shinji Kawasoe, Hiroki Arimura, Hiroshi Sakamoto, and Setsuo Arikawa. Efficient substructure discovery from large semi-structured data. In *Proc. of the 2nd SIAM Symposium on Data Mining*, pages 158–74, 2002.

[46] Guoli Wang and Roland L Dunbrack. Scoring profile-to-profile sequence alignments. *Protein Science*, 13(6):1612–1626, 2004.

[47] Akihiro Inokuchi, Takashi Washio, and Hiroshi Motoda. An apriori-based algorithm for mining frequent substructures from graph data. In *Proc. of the 4th European Conf. on Principles of Data Mining and Knowledge Discovery*, pages 13–23. Springer-Verlag, 2000.

[48] Chen Wang, Wei Wang, Jian Pei, Yongtai Zhu, and Baile Shi. Scalable mining of large disk-based graph databases. In *Proc. of the tenth ACM SIGKDD Int. conf. on Knowledge discovery and data mining (KDD '04)*, pages 316–325, New York, NY, USA, 2004. ACM.

[49] Z. Yan, J. Han, and R. Afshar. CloSpan: Mining closed sequential patterns in large datasets. In *SIAM Data Mining Conference*, 2003.

[50] Jun Huan, Wei Wang, and Jan Prins. Efficient mining of frequent subgraph in the presence of isomophism. In *IEEE ICDM*, 2003.

[51] Siegfried Nijssen and Joost N. Kok. A quickstart in frequent structure mining can make a difference. In *Proc. of the 10th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD-2004)*, pages 647—652, August 2004.

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

[53] H. Hu, X. Yan, H. Yu, J. Han, and X.J. Zhou. Mining coherent dense subgraphs across massive biological networks for functional discovery. In *ISMB*, pages 213–221, Ann Arbor, MI, 2005.

[54] Jian Pei, Daxin Jiang, and Aidong Zhang. On mining cross-graph quasi-cliques. In *ACM KDD*, pages 228–238, 2005.

[55] Xifeng Yan, X. Jasmine Zhou, and Jiawei Han. Mining closed relational graphs with connectivity constraints. In *ACM KDD*, pages 324–333, 2005.

[56] Zhiping Zeng, Jianyong Wang, Lizhu Zhou, and George Karypis. Out-of-core coherent closed quasi-clique mining from large dense graph databases. *ACM Transactions on Database Systems*, 32(2):13–es, 2007.

[57] Michihiro Kuramochi and George Karypis. Grew—a scalable frequent subgraph discovery algorithm. In *Proc. of the 4th IEEE International Conference on Data Mining*, pages 439–442, 2004.

[58] Michihiro Kuramochi and George Karypis. Finding frequent patterns in a large sparse graph. *Data Mining and Knowledge Discovery*, 11(3):243–271, 2005.

[59] Shayan Ghazizadeh and Sudarshan S. Chawathe. Seus: Structure extraction using summaries. In *In Proc. of the 5th Int. Conf. on Discovery Science*, pages 71–85. Springer, 2002.

[60] Lei Zou, Lei Chen, and Yansheng Lu. Top-k subgraph matching query in a large graph. In *PIKM '07: Proc. of the ACM first workshop in CIKM*, pages 139–146, New York, NY, USA, 2007. ACM.

[61] K. Yoshida and H. Motoda. CLIP: concept learning from inference patterns. *Artificial Intelligence*, 75(1):63–92, 1995.

[62] Diane J. Cook and Lawrence B. Holder. Substructure discovery using minimum description length and background knowledge. *Journal of Artificial Intelligence Research*, 1:231–255, 1994.

[63] Akihiro Inokuchi, Takashi Washio, Kunio Nishimura, and Hiroshi Motoda. A fast algorithm for mining frequent connected subgraphs. *IBM Research, Tokyo Research Laboratory, Technical Report*, 2002.

[64] Michihiro Kuramochi and George Karypis. Frequent subgraph discovery. In *Data Mining, 2001. ICDM 2001, Proceedings IEEE International Conference on*, pages 313–320. IEEE, 2001.

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

[66] Xifeng Yan and Jiawei Han. gspan: Graph-based substructure pattern mining. Technical Report UIUCDCS-R-2002-2296, Dept. of Computer Science, University of Illinois at Urbana-Champaign, 2002.

[67] Christian Borgelt and Michael R Berthold. Mining molecular fragments: Finding relevant substructures of molecules. In *Data Mining, 2002. ICDM 2003. Proceedings. 2002 IEEE International Conference on*, pages 51–58. IEEE, 2002.

[68] Jun Huan, Wei Wang, and Jan Prins. Efficient mining of frequent subgraphs in the presence of isomorphism. In *Data Mining, 2003. ICDM 2003. Third IEEE International Conference on*, pages 549–552. IEEE, 2003.

[69] Jun Huan, Wei Wang, Jan Prins, and Jiong Yang. Spin: mining maximal frequent subgraphs from graph databases. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 581–586. ACM, 2004.

[70] Siegfried Nijssen and Joost N Kok. A quickstart in frequent structure mining can make a difference. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 647–652. ACM, 2004.

[71] Jian Pei, Jiawei Han, Behzad Mortazavi-Asl, Helen Ping, Qiming Chen, Umeshwar Dayal, and Mei-Chun Hsu. Prefixspan: Mining sequential patterns efficiently by prefix-projected pattern growth. In *Proceedings 2001 International Conference on Data Engineering*, 2001.

[72] L. Holder, D. Cook, and S. Djoko. Substructure discovery in the subdue system. In *Proceedings of the AAAI Workshop on Knowledge Discovery in Databases*, pages 169–180, 1994.

[73] Diane J Cook, Lawrence B Holder, and Surnjani Djoko. Knowledge discovery from structural data. *Journal of Intelligent Information Systems*, 5(3):229–248, 1995.

[74] Michihiro Kuramochi and George Karypis. Finding frequent patterns in a large sparse graph. In *SIAM International Conference on Data Mining*, 2004.

[75] P. Desikan and J. Srivastava. Mining temporally evolving graphs. In *WEBKDD Workshop*, volume 22. Citeseer, 2004.

[76] M. Lahiri and T.Y. Berger-Wolf. Mining periodic behavior in dynamic social networks. In *IEEE ICDM*, pages 373–382, 2008.

[77] B. Wackersreuther, P. Wackersreuther, A. Oswald, C. B
"ohm, and K.M. Borgwardt. Frequent subgraph discovery in dynamic networks. In *Proc. of the 8th Workshop on Mining and Learning with Graphs*, pages 155–162. ACM, 2010.

[78] J. Sun, C. Faloutsos, S. Papadimitriou, and P.S. Yu. Graphscope: parameter-free mining of large time-evolving graphs. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 687–696. ACM, 2007.

[79] C.h. You, L.B. Holder, and D.J. Cook. Learning patterns in the dynamics of biological networks. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '09, pages 977–986. ACM, 2009.

[80] A. Inokuchi and T. Washio. A fast method to mine frequent subsequences from graph sequence data. In *IEEE ICDM*, pages 303–312, 2008.

[81] Z. Liu, J.X. Yu, Y. Ke, X. Lin, and L. Chen. Spotting significant changing subgraphs in evolving graphs. In *2008 Eighth IEEE International Conference on Data Mining*, pages 917–922. IEEE, 2008.

[82] P. Shoubridge, M. Kraetzl, W. Wallis, and H. Bunke. Detection of abnormal change in a time series of graphs. *Journal of Interconnection Networks*, 3(1&2):85–101, 2002.

[83] J. Chan, J. Bailey, and C. Leckie. Discovering correlated spatio-temporal changes in evolving graphs. *Knowledge and Information Systems*, 16(1):53–96, 2008.

[84] D. Duan, Y. Li, Y. Jin, and Z. Lu. Community mining on dynamic weighted directed graphs. In *Proceeding of the 1st ACM international workshop on Complex networks meet information & knowledge management*, pages 11–18. ACM, 2009.

[85] Petko Bogdanov, Misael Mongiovì, and Ambuj K Singh. Mining heavy subgraphs in time-evolving networks. In *Data Mining (ICDM), 2011 IEEE 11th International Conference on*, pages 81–90. IEEE, 2011.

[86] Elise Desmier, Marc Plantevit, Céline Robardet, and Jean-François Boulicaut. Trend mining in dynamic attributed graphs. In *Machine Learning and Knowledge Discovery in Databases*, pages 654–669. Springer, 2013.

[87] Misael Mongiovi, Petko Bogdanov, and Ambuj K Singh. Mining evolving network processes. In *Data Mining (ICDM), 2013 IEEE 13th International Conference on*, pages 537–546. IEEE, 2013.

[88] Stephen Eubank, Hasan Guclu, Anil, Madhav V. Marathe, Aravind Srinivasan, Zoltan Toroczkai, and Nan Wang. Modelling disease outbreaks in realistic urban social networks. *Nature*, 429(6988):180–184, May 2004.

[89] M. Kretzschmar and M. Morris. Measures of concurrency in networks and the spread of infectious disease. *Mathematical Biosciences*, 133:165–195, 1996.

[90] Albert L. Barabási. The origin of bursts and heavy tails in human dynamics. *Nature*, 435:207–211, 2005.

[91] Kathleen M. Carley. Communicating new ideas: The potential impact of information and telecommunication technology. *Technology in Society*, 18:219–230, 1996.

[92] David Kempe, Jon Kleinberg, and Éva Tardos. Maximizing the spread of influence through a social network. In *KDD '03: Proc. of the ninth ACM SIGKDD Int. Conf. on Knowledge discovery and data mining*, pages 137–146, New York, NY, USA, 2003. ACM.

[93] S. Wasserman and K. Faust. *Social Network Analysis*. Cambridge University Press, Cambridge, MA, 1994.

[94] J. Baumes, M. Goldberg, M. Magdon-ismail, and W. Wallace. Discovering hidden groups in communication networks. In *Handbooks in Information Systems National Security, 2005*, pages 378–389, 2004.

[95] Malik Magdon-Ismail, Mark K. Goldberg, William A. Wallace, and David Siebecker. Locating hidden groups in communication networks using hidden markov models. In *First NSF/NIJ Symposium Intelligence and Security Informatics, ISI 2003*, volume 2665 of *Lecture Notes in Computer Science*, pages 126–137. Springer, 2003.

[96] A. Bernstein, S. Clearwater, S. Hill, C. Perlich, and F. Provost. Discovering knowledge from relational data extracted from business news. In *Proc. of the KDD-2002 Workshop on Multi-Relational Data Mining (MRDM-2002)*, pages 7–20, 2002.

[97] C. Papadimitriou. Computational aspects of organization theory. In *Lecture Notes in Computer Science*, 1997.

[98] D. Croft, J. Krause, and R. James. Social networks in the guppy (poecilia reticulata). *Proc. of the Royal Society of London Biology Letters*, 271:516–519, 2004.

[99] David Lusseau and M. E. J. Newman. Identifying the role that individual animals play in their social network. *Proc. of the Royal Society of London Biology Letters (suppl.)*, 271:477–481, 2004.

[100] Kristian Skrede Gleditsch. Expanded Trade and GDP Data. *Journal of Conflict Resolution*, 46(5):712–724, 2002.

[101] Michael Ley. Dblp, computer science bibliography. Website, 2008. `http://www.informatik.uni-trier.de/~ley/`.

[102] Huong Le, Santosh Kabbur, Luciano Pollastrini, Ziran Sun, Keri Mills, Kevin Johnson, George Karypis, and Wei-Shou Hu. multivariate analysis of cell culture bioprocess data–lactate consumption as process indicator. *Journal of Biotechnology*, 2012.

[103] Douglas B. West. *Introduction to Graph Theory.* Prentice Hall., 2001.

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

[105] Feida Zhu, Xifeng Yan, Jiawei Han, and S Yu Philip. gprune: a constraint pushing framework for graph pattern mining. In *Advances in Knowledge Discovery and Data Mining*, pages 388–400. Springer, 2007.

[106] Rezwan Ahmed and George Karypis. Algorithms for mining the coevolving relational motifs in dynamic networks. Technical Report 14-008, Department of Computer Science and Engineering, University of Minnesota, 2014.

[107] Jian Pei, Jiawei Han, Behzad Mortazavi-Asl, Helen Pinto, Qiming Chen, Umeshwar Dayal, and Meichun Hsu. PrefixSpan: Mining sequential patterns efficiently by prefix-projected pattern growth. In *ICDE*, pages 215–224, 2001.

[108] Hong Cheng, Xifeng Yan, Jiawei Han, and Chih-Wei Hsu. Discriminative frequent pattern analysis for effective classification. In *Data Engineering, 2007. ICDE 2007. IEEE 23rd International Conference on*, pages 716–725. IEEE, 2007.

[109] Xifeng Yan, Hong Cheng, Jiawei Han, and Philip S Yu. Mining significant graph patterns by leap search. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 433–444. ACM, 2008.

[110] Marisa Thoma, Hong Cheng, Arthur Gretton, Jiawei Han, Hans-Peter Kriegel, Alexander J Smola, Le Song, S Yu Philip, Xifeng Yan, and Karsten M Borgwardt. Near-optimal supervised feature selection among frequent subgraphs. In *SDM*, pages 1076–1087. SIAM, 2009.

[111] Sayan Ranu and Ambuj K Singh. Graphsig: A scalable approach to mining significant subgraphs in large graph databases. In *Data Engineering, 2009. ICDE'09. IEEE 25th International Conference on*, pages 844–855. IEEE, 2009.

[112] Ning Jin and Wei Wang. Lts: Discriminative subgraph mining by learning from search history. In *Data Engineering (ICDE), 2011 IEEE 27th International Conference on*, pages 207–218. IEEE, 2011.