System Evolution Analytics: Data Mining and Learning of Complex and Big Evolving Systems

Ph.D. Thesis

By

ANIMESH CHATURVEDI

DISCIPLINE OF COMPUTER SCIENCE & ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY INDORE

SEPTEMBER 2019

System Evolution Analytics: Data Mining and Learning of Complex and Big Evolving Systems

A THESIS

Submitted in partial fulfillment of the requirements for the award of the degree

of

DOCTOR OF PHILOSOPHY

by

ANIMESH CHATURVEDI

DISCIPLINE OF COMPUTER SCIENCE & ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY INDORE

SEPTEMBER 2019

INDIAN INSTITUTE OF TECHNOLOGY INDORE

CANDIDATE'S DECLARATION

I hereby certify that the work which is being presented in the thesis entitled **System Evolution Analytics: Data Mining and Learning of Complex and Big Evolving Systems** in the partial fulfillment of the requirements for the award of the degree of **Doctor of Philosophy** and submitted in the **Computer Science and Engineering, Indian Institute of Technology Indore**, is an authentic record of my own work carried out during the time period from May 2015 to September 2015 under the supervision of Dr. Aruna Tiwari, Associate Professor, Computer Science and Engineering, Indian Institute of Technology Indore.

The matter presented in this thesis has not been submitted by me for the award of any other degree of this or any other institute.

Signature of the student with date
Animesh Chaturvedi

This is to certify that the above statement made by the candidate is correct to the best of my/our knowledge.

Signature of Thesis Supervisor with date
Dr. Aruna Tiwari

Animesh Chaturvedi has successfully given his/her Ph.D. Oral Examination held on **<Date of PhD Oral Examination>**.

Signature of Chairperson (OEB) Date:	Signature of External Examiner Date:	Signature(s) of Thesis Supervisor(s) Date:
Signature of PSPC Member #1 Date:	Signature of PSPC Member #2 Date:	Signature of Convener, DPGC Date:
Signature of Head of Discipline Date:		

ACKNOWLEDGEMENTS

I express my sincere thanks to my parents and family members. I would like to give my sincere gratitude toward my thesis supervisor Dr. Aruna Tiwari for her continues guidance and encouragement. I am grateful to PhD Student's Progress Committee (PSPC) including, Dr. Amod Umarikar & Dr. Abhishek Srivastava. I also wish to thank three Head of the Departments (HoDs) during my PhD, namely, Dr. Abhishek Srivastava, Dr. Kapil Ahuja, and Dr. Surya Prakash. Further, I am also grateful to the Director/Prof. Pradeep Mathur and several Deans as well as Administrative Officers during the course of the program of my PhD.

I am also grateful to – Dr. Narendra S. Chaudhari, Dr. Somnath Dey, Dr. Neminath Hubballi, and Dr. Bodhisatwa Mazumdar – for their voluntary guidance and advices as a faculty member of Computer Science and Engineering. Further, I would like to thanks to office-staff, especially to CSE office-staff – Mr. Shailendra Verma, Mr. Lalit Jain, and Ms. Ujavala Gorakh Langhi – for their several helps.

I also wish to thank – Prof. Dave Binkley, Prof. Nicolas Spyratos, Dr. Srinivas Padmanabhuni, Dr. Suresh Sundaram, Dr. Hemant Mehta, and Dr Saurabh Tiwari – for their early feedbacks, and advices to do the PhD thesis research work. I want to thank IEEE SMC Japan 2018 & IEEE ICDM Singapore 2018 committee specially Dr. Yi Yu. I will always be indebted towards my family members and friends for their continuous support. I wish to express my sincere thanks to all my friends and seniors, Dr. Om Prakash Patel, Mr. Chandan Gautam, Mr. Mayank Swarnkar, Mr. Nikhil Tripathi, Mr. G. Iyyakutti Iyappan, Mr. Akhilesh Mohan Srivastava, Dr . Piyush Joshi, Mr. Syed Sadaf Ali, and Dr. Dipanjan Roy for their constant moral support and academic help. I am also thankful to other lab mates including Dr. Neha Bharill, Dr. Arpit Bharadwaj, Vikas Chauhan, Suchitra Agrawal, and Preeti Jha. I am thankful to my sister Shubhangi Chaturvedi for helping me in improvement of research writing.

Animesh Chaturvedi

Dedicated to my

Parents, Family members,

&

Teachers

ABSTRACT

Era of artificial and computational intelligence leads to various kinds of systems that evolves. Usually evolving system contains evolving inter-connected entities (or components) that makes evolving states created over time. Several evolving computing systems are stored and managed in a repository, which makes a kind of time-variant or non-stationary data. Numbers of system repositories are increasing across the globe and temporal databases are increasing in size, which makes system maintenance and evolution a challenging task. In this thesis, our objectives is to model an evolving system state as (S_i, ER_i, t_i), such that the system state S_i and the evolution representor ER_i is representing system at the ith time point t_i, where 'i' varies from 1 to N. We expressed an evolving system as a state series, SS = {S_1, S_2… S_N}, such that each state is pre-processed to make an evolution representor ER = {ER_1, ER_2… ER_N} for example evolving networks EN = {EN_1, EN_2… EN_N}. This modelling is used to do system evolution analysis. The state series represents a non-linear time-variant evolving system. A state series of an evolving system is stored and managed in a centralized repository. We introduce a ***System Evolution Analytics*** **(SysEvo-Analytics)** based on proposed *evolving system mining* and *evolving system learning*, which are techniques for an evolving system represented as a set of temporal data. The evolving (or temporal) networks represented system state series are combined to form an *evolution representor* that can be used for analysis purpose. Our goal is to analyze the evolving inter-connected entities (or features) in a state series. We present following contributory approaches:

1. To do *evolving system mining*, we applied two types of pattern mining: *network evolution rule* and *network evolution subgraph*.

- We proposed an approach for Stable Network Evolution Rule Mining and two metrics: Stability metric and Changeability metric.

- We also proposed an approach for mining Network Evolution Subgraphs such as Network Evolution Graphlets (NEGs) and Network Evolution Motifs (NEMs) from a set of evolving networks. We used frequencies of graphlets changing over graph snapshots to model four proposed metrics over a state series: the System State Complexities (SSCs), and the Evolving System Complexity (ESC), Changeability metric and Stability metric.

2. To do *evolving system learning*, we applied *deep evolution learning* techniques on the evolution representor. We proposed an approach System Evolution Recommender (SysEvoRecomd), which used our proposed Graph Evolution and Change Learning (GECL) to do network matrix reconstruction. We also proposed System Evolution Learning, which used network pattern information for training system graph structure.

For the above two contributory approaches, we used to demonstrate the application of our work on six evolving systems including: one evolving software system, two evolving natural language systems, one evolving retail market system, and two evolving sentiments in IMDB movie name-genre system.

3. To do *evolving big data mining*, we applied parallel-frequent pattern growth (i.e. a king of association rule mining) algorithm, which resulted in the generation of Big Data Evolution Rules and Big Data Evolution Concepts. We applied this approach on evolving big scholarly data. We presented experimentation results as evolving scholarly topics and the trend of evolving scholarly fields.

4. To do change mining and evolution mining of an evolving web service system. We proposed a Web-service change classifier based interface slicing algorithm that mines change information from two versions of an evolving distributed system. We also proposed four Service Evolution Metrics that capture the evolution of a system's version series. We demonstrated our approach on two well-known cloud services: Elastic Compute Cloud (EC2) from the Amazon Web Service (AWS), and Cluster Controller (CC) from Eucalyptus.

We prototyped our proposed technique as tools, which is applied to eight real-world evolving systems collected from open-internet repositories of six different domains: software system, natural language system, retail market basket system, IMDb movie genres system, evolving big scholarly data system, and evolving web-service systems. We applied our *SysEvo-Analytics* on different fields including: Software Evolution Analytics, Natural-language Evolution Analytics, Market Evolution Analytics, Movie Evolution Analytics, Big data Evolution Analytics, and Service Evolution Analytics. Based on the proposed approaches, we can generate recommendation and evolution information report about system evolution. This is demonstrated as experimentation reports containing retrieved evolution information for an evolving system.

Keywords: Systems Engineering, Artificial Intelligence, Computational Intelligence, Data Mining, Machine Learning, Data Science, Graph (Network) Theory.

Thesis project link for further information

https://sites.google.com/site/animeshchaturvedi07/research/phdresearch

LIST OF PUBLICATIONS

(A) Publications from PHD thesis work

1. Animesh Chaturvedi, Aruna Tiwari, and Nicolas Spyratos. “minStab: Stable Network Evolution Rule Mining for System Changeability Analysis”. *IEEE Transactions on Emerging Topics in Computational Intelligence* (2019). IEEE Computational Intelligence Society (Early Access). Date of Publication: 02 April 2019. DOI: 10.1109/TETCI.2019.2892734
2. Animesh Chaturvedi, and Aruna Tiwari. “System Network Complexity: Network Evolution Subgraphs of System State series”. *IEEE Transactions on Emerging Topics in Computational Intelligence* (2018). (Early Access) Date of Publication: 31 October 2018. DOI: 10.1109/TETCI.2018.2848293
3. Animesh Chaturvedi, and Aruna Tiwari. “System Evolution Analytics: Deep Evolution and Change Learning of Inter-Connected Entities”. 48th *IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, Miyazaki Japan, October 2018, pp. 3075-3080. IEEE SMC Society.
4. Animesh Chaturvedi, and Aruna Tiwari. “System Evolution Analytics Evolution and Change Pattern Mining of Inter-Connected Entities”. 48th *IEEE International Conference on Systems, Man, and Cybernetics (SCM)*, Miyazaki Japan, October 2018, pp. 3877-3882. IEEE SMC Society.
5. Animesh Chaturvedi, and Aruna Tiwari. “SysEvoRecomd: Graph Evolution and Change Learning based System Evolution Recommender”. 18th *IEEE International Conference on Data Mining Workshops (ICDMW)*, Singapore, 2018, pp. 1499-1500. IEEE Computer Society.

(A) Publications under review from PHD thesis work

6. TETCI-2019-0143: Animesh Chaturvedi, Aruna Tiwari, and Shubhangi Chaturvedi, “Network Evolution Graphlets or Motifs for System Changeability and Stability Analysis”, IEEE Transactions on Emerging Topics in Computational Intelligence, Submitted on 28-Jun-2019. (Resubmitted)
7. TETCI-2019-0076: Animesh Chaturvedi, Aruna Tiwari, and Shubhangi Chaturvedi, “System Neural Network: Evolution and Change based System Structure Learning”, IEEE Transactions on Emerging Topics in Computational Intelligence, Submitted on 05-Apr-2019.
8. TEM-19-0403 Animesh Chaturvedi, Aruna Tiwari, Shubhangi Chaturvedi, and Dave Binkley “Service Evolution Analytics: Change and Evolution Mining of a Distributed System”, IEEE Transactions on Engineering Management, Submitted on 29-May-2019.
9. ISJ-RE-19-07720 Animesh Chaturvedi, Aruna Tiwari, Yi Yu, and Shubhangi Chaturvedi, “System Evolution Analysis based on Temporal Data Representation”, IEEE Systems Journal, (Under Major Revision).

Achievements

Achievements of Animesh Chaturvedi during PhD

- Selected as top 200 young researcher in the Heidelberg Laureate Forum (HLF) 2019 to attend the event between 22-27 September 2019 at Heidelberg Germany. The event host presentations by several Turing award winners of Computer Science and Field Medal winners of Mathematics. The Turing Award and Field Medal are the most prestigious awards in Computer Science and Mathematics respectively. The HLF is one of the most prestigious annual gathering of eminent researchers with different age groups across the globe in the domain of Computer science and Mathematics.

- Received a full travel grant from Council of Scientific & Industrial Research (CSIR) to attend and present two papers on 48th IEEE *International Conference on Systems, Man, and Cybernetics* at Miyazaki Japan in October 2018.
- Received an internship offer of 90 days from NII (National Institute of Informatics) Japan.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1

Introduction

Evolving System is a kind of complex system that evolves and changes over time; this creates multiple system states. Usually, real world systems has complex environment, which makes agents to change and evolve a system with time. There are different kinds of systems (e.g. software, machine, electronic device); if such system evolves with time, this makes them evolving system [1]-[4]. Many systems evolve over time due to changing, dynamic, and evolving environment such systems are referred as *evolving systems* [1]-[8], which includes evolving software system [9] and evolving business system [10]. These systems evolve with time, and creates ample amount of information about their evolution. An evolving system analysis deals with control, prediction, classification, data processing, unpredictable environments, and adaptability.

The information about states of an evolving system can be stored and managed in a repository at a centralized place. Collection of such temporal states makes a complex data series [11]. It is good option to manage information about evolving system states in a repository (e.g. GitHub, Wikipedia, Maven, UCI repository etc.). Constant growth of evolving system - both in number of repositories and in the size of individual repository - makes maintenance of system repositories a challenging task. Usually, system data are stored and managed in a time-variant data repository that keep track/record of state history for different time instances of system data.

Data mining and learning is a process of extracting information from a database that contains interesting patterns or knowledge (implicit, non-trivial, potentially useful, and previously unknown) [12][13]. It is challenging to mine or learn evolution information in multiple states of an evolving system. The repository can be analysed using techniques like *change mining* [14], *contrast mining* [15], *resume mining* [16], *evolution mining* [17], *incremental mining* [18], *incremental learning* [19], or *non-stationary data learning* [20]. These techniques have similarities and dissimilarities that make each of the technique unique and important in their applications. Techniques to handle time-variant data or non-stationary data can be used to learn evolution information from a state series (or an evolving repository). An evolving system can be pre-processed to make non-stationary and time variant data that can be used for machine-learning purpose. A mining or learning technique can help to identify and discover the changes in an evolving system. An evolution and change analysis or a non-stationary learning extracts useful information from time-variant or non-stationary data.

Evolution and change analysis are kind of data analysis that studies evolution of time variant (or evolving or changing or incremental or non-stationary) data [14]-[20]. Analysis of multiple states can provide an overview of evolution happened in the evolving system. Such evolution and change analysis can be applied to discover hidden, non-trivial, and non-obvious information about evolution and changes happened in an evolving system. The evolution information is further helpful in decision making. An aggregated information from multiple states can make a time series that can be useful in decision-making. The evolution and change analysis is required to adapt, control, and analyze changes, which are useful in many domains. This help to study temporal states (i.e. snapshot at different time instances) of an evolving system. Some mining and learning techniques that can identify evolution and changes happened during the evolution of an evolving system. Such mining and learning process takes two or more states (or versions) of a database as input to generate evolution information. Such mining and learning techniques are applied in many domains such as source code change analysis [21], process management systems [22], and online data streams [23]. Example datasets in which evolution and change analysis is applied are: retail marketing [24], real life application [25], patent trends [26], and changes in customer behaviours [27].

In this thesis, we attempt to bridge the gap between two fields of studies: Systems Engineering and Artificial (or Computational) Intelligence. We investigate and bridge the system evolution theories of the two domains using data analysis. We propose to investigate evolving system, which are based on Artificial Intelligence (AI) and Computational Intelligence (CI) techniques for Systems Engineering. We further propose to investigate the design and development of intelligent computing algorithms and tools for *System Evolution Analytics* (SysEvo-Analytics). We are interested to do analysis of such evolving systems. The implementation of the algorithm by integrating the evolution mining and pattern (or rule) mining can retrieve the evolution patterns (or rules). The implementation of the algorithm by integrating the techniques that support both evolution learning and deep learning; this can train the machine for evolution deep evolution patterns. Basic concepts and preliminaries of this thesis are provided in Section 1.1.

1.1 Preliminaries about Evolving Systems

Era of artificial and computational intelligence leads to various kinds of systems that evolves. A complex system that keeps on changing with time is referred to as *evolving system*. An evolving system maintains changeability in the form of multiple states that we referred as *state series*. A repository stores and manages a system state series at centralize database. Suppose N states for inter-connected entities of an evolving system are represented as a state series SS = $\{S_1, S_2\ldots S_N\}$ for N time instances T = $\{t_1, t_2\ldots t_N\}$. The state series is similar to the well-known time series and data series. An evolving system can be represented as a set of Evolution Representor (ER) for temporal states. Therefore, we represent system state series as a set ER = $\{ER_1, ER_2\ldots ER_N\}$. The set of Evolution Representors can be further used for the mining and learning purpose. An advantage of this is to create a non-linear representation of an evolving system that can generate mining and learning information. This information will further help to do system evolution analysis [28][29]. We defined state series of an evolving system as given below.

Definition 1.1: *State Series* SS is a series of evolving system's states (or data points) denoted as SS = {S_1, S_2 … S_N} at various time points {t_1, t_2, t_3 … t_N}, such that each state is represented as a series of evolution representor ER = {ER_1, ER_2… ER_N} corresponding to the system states. We can define (S_i, ER_i, t_i) where the S_i stands for ith system state, the ER_i stands for Evolution Representor and the t_i stands for ith time point, where 'i' varies from 1 to N. Collection of such temporal states can be referred as a *State Series*. Example of an evolving state series are: evolving document versions and software version. A version series of an evolving software system are stored in a software repository.

Thus, an evolving system is a complex system that evolves with time and has a number of different states. Assume any system with many entities (or components) that are connected to each other with a relationship. Due to evolving environment, the system is changed from one state to another. Thus, the entity connections are also changed; studying such evolution and change would be an interesting problem. We consider such a system with two assumptions: (a) the system contains a number of distinct entities and (b) each entity might be connected to zero, one or more other entities over time. Thus, this creates an evolving network (or dynamic network) of interconnected entities. We shall refer to this network as the *system network* of entities.

A basic characteristic of an evolving system is that one or more connections present in a state S_a might not be present in another state S_b, while connections present in S_b might not present in state S_a. A system has many entities that are transformable to make a set of dynamic databases for the state series of an evolving system. These dynamic databases of a system state series can be stored and managed in a repository. The dynamic databases can be referred as nonstationary data in context of machine learning.

An example of Evolution Representor is temporal network. An evolving system contains evolving interconnected entities (or components) that make evolving networks for the system state series. We assume that a system state can be represented using a network of relationship(s) between inter-connected entities. This create networks that change over time are referred as *evolving networks* [30] or *temporal networks* [31], or *time-varying networks* [32], or *dynamic networks* [33], or *multilayer networks* [34][35]. An *evolving network* is a finite sequence of directed graphs, which describes the state S_i at ith time instance. This makes series of evolving networks {EN_1, EN_2… EN_N}, where an EN_i represents a state S_i. An evolving network $EN_i = (V, E)$ has a set of vertices V and a set of edges E.

The network science communities exhaustively exploited techniques for properties of a single network, but we are interested to exploit these properties for a set of *evolving networks* or *temporal networks* or *multilayer networks*. Examples of evolving network systems include IMDb, internet links, and social network. An evolving network has a group of nodes connected together; in our case, a node represents an evolving entity of the system. As the system evolves, its network connections also evolves, which can be used to study the system evolution.

Usually, different states are built from similar patterns of entities in an evolving system. Different states keep same functioning by keeping almost similar patterns of entity connections. All entity connections together are represented as complex network (graph). *Complex network* is a directed graph that represents entity connection relationship. In the graph, each entity is a unique vertex (node) and each entity connection is represented by an edge between the source and target entity. For example, an edge (u, v) means u (source) is connected to v (target); a special case is a loop or cycle (u, u).

An example of evolving system is an *evolving software system* under maintenance phase, which is a software system that evolves over time. It might be in different states as time passes, and its state is referred as software version. Indeed, a software system usually contains a number of procedures and each procedure might be calling zero, one, or more other procedures. Here, the entities are the procedures such that procedure P is connected to procedure P' if P calls P'; and the connections are referred as *inter-procedural calls*.

Due to continuous evolution of a system with time, its data requires a state configuration management. A system state can be represented as a network of inter-connected entities. This means, an entity is represented as a vertex (or a node) and connections between entities are represented as directed edges. Then, evolving states can be represented with several complex networks such that each state is represented as a network, which makes a set of temporal networks. Usually, temporal networks make similar connection patterns between entities in different states of an evolving system.

Suppose there are several inter-connected entities, whose *connections* are represented as edges between nodes in a network. We can represent such connections between entities using binary matrix. This means the presence of a connection between two entities can be represented as '1' and the absence of connection can be represented as '0'. Such matrix can be referred to as connection matrix or adjacency matrix or square binary matrix. A connection between two entities can be represented as a feature that can be referred to as *feature matrix*. We convert such *feature matrices* of various states to their equivalent evolution representor as a matrix or an artifact.

For example, to represent a system we use the entities, as they are the building blocks of the evolving system. These entities join together to form an evolving system, and the inter-connections between entities can be represented by a network (or a graph) N(E, C), where E is a set of entities and C is a set of connection. Alternatively, the inter-connection between entities can also be represented by a connection matrix, M(E, C). Each system state can be represented as a complex network, which contains relationship between inter-connected entities. Further, these networks can be transformed to make another evolution representor.

1.2 Motivation and Scope

This thesis is a study of System Evolution Analytics (SysEvo-Analytics), which is based on different data analysis techniques applied on complex and big evolving systems. For a given set of states of an evolving system, the motivations of this thesis are as follows.

1. To contribute in systems engineering, we propose a model for SysEvo-Analytics based on data analytics techniques over a *state series* SS = {S_1, S_2… S_N} represented as an evolution representor (like evolving network) of an evolving system. It is challenging to generate evolution information by mining a set of evolving networks {EN_1, EN_2… EN_N} that represents multiple states of an evolving system.
2. Inspired from *association rule mining* [36], *temporal/persistent rules* [37][38][39], *sequential rules* [40]-[43], and *graph evolution rules* [44]-[48]. It is interesting to detect *stable network evolution rules* (or patterns) in evolving system networks over time. It is motivating to study the stability and persistence of links in the series of evolving network databases can draw conclusions regarding system evolution. Thus, we aim to retrieve evolution and stable rules containing information about stable (or persistent) co-occurrence of source and target entities in evolving network databases.
3. Based on frequent subgraph mining [49]-[51], network motif [52][53], and graphlet mining [54][55]. It is interesting to retrieve information about *network evolution subgraphs (like graphlets* or *motifs*) over time for a set of evolving networks. It is motivating to capture *graphlets information* of a state and use it to calculate *complexity of a system state*. Another motivation is to aggregate graphlets information of all states over time to retrieve *network evolution graphlets information*, which aids to calculate *complexity of a whole evolving system.*
4. We present approaches to find change specification (like time series) of evolution patterns such as rules and subgraphs. Our approach mines and learns the changes in the connections between entities, which aids to provide recommendation about the evolving system.
5. After mining, it is interesting to compute and study system properties like: *Changeability*, *Stability*, and *Complexity*.
6. It is challenging to use a *deep learning* [56] based approach that learns evolving system states over time, which can assist SysEvo-Analytics. Thus, the objective is to introduce an intelligent methodology that can learn from the pre-evolved states of a system, and then predict/recommend about the evolving system.
7. It is challenging to generate evolution information of *evolving big data*, we target this using *association rule mining*. We present an approach to generate association rules of an evolving big data. This is followed by an interesting study of evolving *big scholarly data* [57], which depicts the evolution of scholarly fields.
8. Based on previous work on web service change management and slicing [58]-[60], a new

motivation is to study *change and evolution mining for distributed systems*, which uncover change and evolution information over time. Firstly, we use classification theory to label the changes that differentiate two versions of a distributed system. Secondly, we devise software metrics to capture the evolution over a series of versions. These two kinds of information help to characterize the changes and evolution of a cloud system. Creation of automated techniques that exploit this information is a challenging task.

The rationales behind our study over are as follows. We aim to help in improvement and future prediction while system is under development and maintenance. Our approaches aim to learn from past system states and provide system evolution information to system manager. Our approaches are applicable to analyze advantages and disadvantages for the changes occur to the evolving system.

1.3 Objectives

In an evolving system, the relationship between system entities (or components) evolves over time. Our objectives is to model an evolving system state as (S_i, ER_i, t_i), such that the system state S_i and the evolution representor ER_i is representing system at the i^{th} time point t_i, where 'i' varies from 1 to N. This modelling is used to do SysEvo-Analytics. The state series represents a non-linear time-variant evolving system. A state series of an evolving system is stored and managed in a centralized repository. The major objectives of our thesis are as follows.

1. There exist connections (or relationships) between entities [30][31], which also evolve and make a series of evolving networks EN = {EN_1, EN_2… EN_N}. We can use these evolving networks to do mining and learning over evolving system states for system evolution analysis. Our objective is to propose a SysEvo-Analytics model, which studies evolving networks over states to provide system evolution information for analysis.
2. We proposed mining and learning techniques to identify and understand the changes and evolution in an evolving system. For the proposed approaches, we aim to develop prototype tools. These tools produced results of the evolution information (in the form of time series) for an evolving system.
3. As an objective, we identified and represented the system evolution information by presenting a change specification (including time series). The time series describes the changes between system states. We aim to build such time series information, which helps to analytically quantify and understand the changes that have occurred in the evolving networks representing an evolving system. Significance of such challenging task is to predict and recommend about the evolving system.
4. The objective of the thesis is to design data mining and learning methods for SysEvo-Analytics with different proposed algorithms. We also extend these mining algorithms to calculate system properties like Changeability, Stability, and Complexity. The proposed learning algorithms are used to do prediction and recommendation about system evolutions.

5. We applied our approach on various domains of studies. Specifically, we are targeting analysis of software inter-procedural call graphs, natural-language word networks, retail market analysis, movie-genre analysis, big data analysis, and service oriented architecture analysis.
6. An aim is to present study on advance technologies like Evolving Big-data system and Evolving Web-service system. Specifically, we studied evolution of evolving big scholarly data and evolving cloud services.
7. Our proposed SysEvo-Analytics model is applied on different fields including: Software Evolution Analytics, Natural-language Evolution Analytics, Market Evolution Analytics, Movie Evolution Analytics, Big data Evolution Analytics, and Service Evolution Analytics.

1.4 Applied Case Studies

This section describes the case studies used to apply our proposed approaches of this thesis. To accomplish objectives of this thesis, we proposed different approaches based on fundamental mining and learning techniques. Using the proposed approaches, we developed prototype tools for SysEvo-Analytics. The Figure 1.1 depicts the basic flow chart to do the SysEvo-Analytics using the developed tools. To develop the tools, we majorly coded in Java technologies. Thereafter, tools are used to do exhaustive experimental analysis over various evolving systems. These experiments resulted in different kind of SysEvo-Analytics.

The source and target entities for the type of connections to create a type of network are described in the Table 1.1. The type of connection and the type of network depend upon the domain of the evolving system. The table's first column describes type of SysEvo-Analytics based on domain. Second column is the name of evolving system used for experiment. Third column describes the kind of source and target entities in an evolving system. Fourth column describes 'type of relationship' between the entities, which depends on the domain of the evolving system. Fifth column provides 'type of network' for an evolving system. Few networks has an existing well-known name (like *call graph* for software) and few networks do not have name, thus, we name them like *word network, purchase network* etc. Table 1.1 also mentions the links of the dataset that we collected before Oct 2016. Different repositories contain different kind of evolving systems, for example, Maven contains Hadoop-HDFS library jars as data, Wikipedia contains natural language as data, UCI repository contains retail market data, and IMDb contains movie genre data.

To demonstrate the application of SysEvo-Analytics, we used evolving systems belonging to the following six domains: evolving software system, evolving natural language system, evolving retail market system, evolving IMDb system, evolving web service system, and evolving big data system. This SysEvo-Analytics is described in the following six ways: software evolution analytics, natural-language evolution analytics, market evolution analytics, movie evolution analytics, service evolution

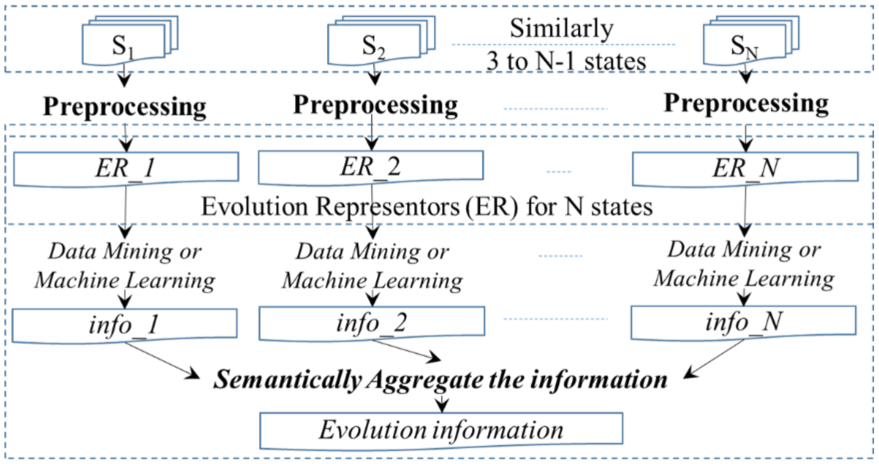

S1
S2
Similarly 3 to N-1 states
SN
Preprocessing
Preprocessing
Preprocessing
ER_1
ER_2
ER_N
Evolution Representors (ER) for N states
Data Mining or Machine Learning
Data Mining or Machine Learning
Data Mining or Machine Learning
info_1
info_2
info_N
Semantically Aggregate the information
Evolution information

Figure 1.1 A general overview for the flow of System Evolution Analytics Tools.

analytics, and big data evolution analytics. To represent a state series, we exploited the following relationships: call-graphs, word-networks, purchase-networks, sentiment networks, WSDLs, and scholarly big data, respectively. Our proposed techniques are implemented as tools that are used on the six domains of evolving systems.

In case of - evolving software system, evolving natural language systems, evolving retail market system, and evolving IMDb movie genre systems - we retrieved recommendations as well as evolution patterns information. This helped to perform - inter-procedural analysis, natural language processing, retail market analysis, and movie-genre sentiment analysis. These are helpful to the system domain expert. Similarly, we analyzed other two system domains: evolving scholarly big data and evolving web services. In our study, the six domains of evolving systems resulted in six kind of system analysis.

1.4.1 Software Evolution Analytics

We developed approaches and tools to analyse the evolving call graphs of an Evolving Software System. Our objective is to establish the SysEvo-Analytics theories, which produce outcome to develop mining software repository tools. Based on these theories and tools, we retrieve following kind of information: call graph evolution rules, call graph evolution subgraphs (e.g. motifs or graphlets patterns), and deep learning to create software neural network.

1.4.2 Natural-language Evolution Analytics

We developed approaches and tools to analyse the evolving word networks of an Evolving Natural Language System. Our objective is to establish the SysEvo-Analytics theories, which produce outcome to develop natural language text-processing tools. Based on these theories and tools, we retrieve following kind of information: word network evolution rules, word network evolution subgraphs, (e.g. motifs or graphlets patterns), and deep learning to create natural-language neural network.

Table 1.1 Experiment information about Applications of SysEvo-Analytics and Temporal networks

Applications of System Evolution Analytics	Evolving Systems	"Source" and "Target" Entities in Temporal Networks	Type of Connections	Type of temporal network
Software Evolution Analytics	Hadoop HDFS-Core[1]	"Caller" and "Callee" Procedures	Procedural calls	Call graph
Natural-language Evolution Analytics	Bible Translation[2]	Words in "Source biblical language" and "English variant" languages	Translations	Words Network
	Multi-sport Events[3]	Words in "Titles" (name) and "Scopes" (region) of events	Regional names	Words network
Market Evolution Analytics	Frequent Market Basket[4]	Words in "Product description" and Words in "Product description"	Purchases	Purchase network
Movie Evolution Analytics	Positive sentiment[6] of movie genres[5]	"Positive words in names" and "genres" of movies	Sentiments	Positive sentiment network
	Negative sentiment[6] of movie genres[5]	"Negative words in names" and "genres" of movies	Sentiments	Negative sentiment network

1. https://mvnrepository.com/artifact/org.apache.hadoop/hadoop-hdfs
2. https://en.wikipedia.org/wiki/List_of_English_Bible_translations
3. https://en.wikipedia.org/wiki/List_of_multi-sport_events
4. https://archive.ics.uci.edu/ml/datasets/Online+Retail
5. https://www.cs.uic.edu/~liub/FBS/sentiment-analysis.html
6. http://www.imdb.com/interfaces/

Table 1.2 Information of the Experiment on Application of Service Evolution Analytics

Services used to do Service Evolution Analytics	Kind of Relationship between entities	Average number of source lines per operations in WSDL	Average number of parameters per operations	Average number of message per operations	Average number of source code lines per operation
Eucalyptus Cluster Controller	Operation and procedure with their input-output parameters	60.20	3.42	2	205.34
AWS- Elastic Compute Cloud (EC2)	Operation and procedure with their input-output parameters	52.25	8.87	2	Source code is not available

1.4.3 Market Evolution Analytics

We developed approaches and tools to analyse the evolving product-purchase network of an Evolving Market System. Our objective is to establish the SysEvo-Analytics theories, which produce outcome to develop market analysis tools. Based on these theories and tools, we retrieve following kind of information: product-purchase network evolution rules, product-purchase network evolution subgraphs (or motifs or graphlets patterns), and deep learning to create product-purchase neural network.

1.4.4 Movie Evolution Analytics

We developed approaches and tools to analyse the evolving movie-genre network of an Evolving IMDb System. Our objective is to establish the SysEvo-Analytics theories, which produce outcome to develop IMDb analysis tools. Based on these theories and tools, we retrieve following kind of information: movie-genre network evolution rules, movie-genre network evolution subgraphs (e.g. motifs or graphlets patterns), and deep learning to create movie-genre neural network.

1.4.5 Big data Evolution Analytics

We developed approaches and tools to analyse the Evolving Scholarly Data containing scholarly keywords and phrases. Our objective is to develop a specialized tool, which identify the scholarly keywords, topics, and fields that are frequently occurring in the several titles of the articles published. To do this, we used the dataset provided by Microsoft Academic Graph (MAG)[1] and Spark Parallel Frequent Pattern – Growth[2] algorithm. As a result, we generated several rules of scholarly keywords, which indicate the evolution of the scholarly field over the time.

1.4.6 Service Evolution Analytics

We developed approaches and tools to analyse the Evolving Web service system using a specialized tool based on change mining and evolution mining. Our objective is to establish service evolution analysis theory, which produce outcome to develop service evolution analysis tools. Based on the theory and tool, we fulfilled following goals: WSDL Slicing, Web Service Slicing, Change Impact Analysis based Regression testing or Web Service, Service Evolution Metrics. We demonstrated our approach on two well-known cloud services (mentioned in Table 1.2): Elastic Compute Cloud (EC2) from Amazon Web Service (AWS), and Cluster Controller (CC) from Eucalyptus.

1.5 Contributions and Thesis Organisation

Rest of the thesis is organized as following.

Chapter 2: Describes related works of this thesis. Here, we provided basic and exiting state-of-the-arts to make foundation of this thesis. The contributory chapters are followed next.

Chapter 3: This chapter describes a given system *state series* SS is represented as a set of evolving networks of evolving inter-connected entities. Then this chapter describes two basic contributory models for system evolution analysis. First basic contribution is a model for SysEvo-Analytics based on rule and subgraph mining. We discuss *network evolution rule mining* to detect

1. ACM SIG-KDD-CUP-2016 https://www.kdd.org/kdd-cup/view/kdd-cup-2016

2. “Frequent Pattern Mining - RDD-based API” https://spark.apache.org/docs/latest/mllib-frequent-pattern-mining.html, September 2017.

network evolution and change rules that contain information about evolution happened to the source and target entities. Then, we discuss *network evolution subgraph mining* to detect evolution and change patterns that contain information about evolution happened to the source and target patterns of system entities. Second basic contribution is a model for SysEvo-Analytics based on proposed *System Evolution Learning*. The learning generates meaningful knowledge (in the form of a neural network) for evolution happened to source and target entity-connection patterns. Both the contributions are further elaborated and described in Chapter 4, 5, and 6.

Chapter 4: We proposed an approach for Stable Network Evolution Rule Mining and two metrics: Stability metric and Changeability metric. Based on this, we developed an intelligent tool, which is used for experiments on evolving systems. The main contributions of this chapter are as follows: (a) the introduction of stability, as a new measure for characterizing network evolution over time, (b) a new algorithm for mining Stable NERs, (c) the detailed "System Network Stability" approach for an evolving system, (d) the formula for the system stability and changeability analysis, (e) a demonstration tool, called 'SNS-Tool' that we use to do experiments with evolving systems. We also discuss experimental results of applying our approach to a number of real world evolving systems. We applied our approach to a number of real-world systems including: software system, natural language system, retail market system, and IMDb system. It results Stable NERs and Changeability Metric value for each evolving system. Our approach identifies the evolution and stability information about network rules of several inter-connected entities in a state series of an evolving system. The proposed mining approach helps to perform system evolution analysis based on source-target nodes in a set of evolving networks.

Chapter 5: We describe subgraph mining for a state and present our proposed keyword definitions, which include *Network Evolution Graphlets* (NEGs), *Network Evolution Motifs* (NEMs), *Changeability Metric* (CM), *Stability Metric* (SM), *System State Complexity* (SSC), and *Evolving System Complexity* (ESC). Thereafter, to analyze an evolving system, we proposed an approach for mining Network Evolution Subgraphs such as NEGs and NEMs from a set of evolving networks. We used frequencies of graphlets and NEGs over states to model four proposed metrics: the SSCs, and the ESC, the CM and the SM. All this is described as algorithms, which includes *mining_NEGs_NEMs_CM_SM* and *System Network Complexity* (SNC). Then, we describe our SNC-Tool based on proposed work, which is used to demonstrate the application of SNC on six evolving systems of four different domains.

Chapter 6: Using well-known deep learning concept, we introduced an approach to do *evolution and change learning*, which uses an *evolution representor* and forms a *System Neural Network* (SysNN). Additionally, we proposed two similar approaches *System Structure Learning* (SSL) and *System Evolution Recommender* (SysEvoRecomd), which uses an intelligent algorithm Deep

Evolution Learner (DEL) to learn about evolution and changes happened to a state series of system. The DEL generates a *Deep System Neural Network* (Deep SysNN) to do network (graph) reconstruction. To do this, we used network pattern information for training system graph structure. We design three variants by exploring three different deep learning techniques: Restricted Boltzmann Machine (RBM), Deep Belief Network (DBN), and denoising Autoencoder (dA). Based on SysEvoRecomd, we developed an automated tool named as SysEvoRecomd-Tool, which is used to conduct experiments on various real-world evolving systems.

Chapter 7: We introduce an approach to do *Big Data Evolution Analytics* (BDE-Analytics) for evolving big data generated through evolving system. The proposed approach pre-processes a given evolving big data, which make multiple states as a state series. Then, it mines frequent patterns (association rules) that are further processed to generate knowledge as mined *Big Data Evolution Rules* (BDERs) to generate *Big Data Evolution Concepts* (BDECs) for the given state series. These BDERs and BDECs as information are further useful to do big data evolution analysis. We demonstrated this approach on an *evolving scholarly system* of Microsoft Academic Graph (the dataset of KDD CUP 2016). We presented experimentation results as *evolving scholarly topics* and the trend of *evolving scholarly fields*.

Chapter 8: This chapter describes approaches for *change mining and evolution mining of an evolving distributed system*. The first approach proposed a *service change classifier*, and the second approach proposed four *service evolution metrics*. This chapter also describes a Service Evolution Analytics model and tool (AWSCM) that supports change and evolution mining of an evolving distributed system. We described two empirical experiments, which consider two self-made illustrative evolving web services and two real-word evolving web services.

Chapter 9: After contributory chapters from Chapter 3 to 8, this chapter concludes thesis and provides future scope of the thesis work.

Chapter 2

Literature Review

In this chapter, we discuss existing state-of-the-art contributions related to this thesis. Initially, we start discussing survey for evolving system and its graph representation, which is used to identify evolution information. Further, we presented contributions of graph (network) and temporal (evolution) mining, which is related to the evolution information retrieval. Then, we presented survey on evolving system analysis and its application. Then we presented work related to the deep learning on various fields of research. Following are the related literature survey, which we carried-out during this thesis work.

2.1 Evolving Systems and Networks

We represented an evolving system as a *state series*, which is extended and advanced form of well-known concept of *time series* or *data series*. Our tool uses *evolving networks* [30] (or *temporal networks* [31] or *time-varying networks* [32] or *dynamic networks* [33]) of an evolving system. Our tools retrieve rules and graphlets (or motifs as subgraphs) as well as learns (trains) a representation pattern of inter-connected entities in an evolving network of an evolving system. For our work, we investigated open-source codes for tools development based on the fundamental mining and learning algorithms. Thereafter, we used different algorithms and tools to build and develop different extended tools based on our proposed works. We applied our tools on different evolving systems to do evolving system analysis.

To begin with, Liu *et al.* [35] developed analytical tool to study controllability of complex self-organized systems; the tool identifies a set of driver nodes with time-dependent control that can guide the system dynamics. Thereafter, Liu *et al.* [62] developed an approach to study observability by reconstructing the system's internal state from its outputs; the approach identifies sensors that are used to monitor a selected subset of state variables. In contrast, our approach and tool finds nodes in the antecedent and consequent of a rule; further, we use system network evolution to study stability, changeability, and complexity.

Work related to the comparison of network includes following. Pržulj [63], which is a systematic measure of a network's local structure based on the degree distribution and graphlets of a large network. Yaveroğlu *et al.* [64] developed a framework to analyze and compare networks by discovering the interaction between roles played by nodes in a network. Ali *et al.* [65] described Netdis to measure topology-based distance between networks, which aids to do network reconstruction.

Recently, Wegner *et al.* [66] introduced a new network comparison methodology NetEmd that aims to identify common organizational principles in networks. Additionally, Wegner *et al.* [66] compared *NetEmd* with the works of Przulj [63], Yaveroglu *et al.* [64], and Ali *et al.* [65]. The work by Przulj [63], Yaveroglu*et al.* [64], Ali *et al.* [65], and Wegner *et al.* [66] presented approaches to analyze and compare networks. The four approaches deals with comparison of networks, whereas our approach does not intends to compare two graphs directly. Alternatively, we aim to compare two evolving network series of two evolving systems.

2.2 Temporal (or Change or Evolution or Dynamic) Mining

Data mining explores data (typically business or big data) to retrieve consistent patterns and semantic relationships between variables [12][13]. It is an analytic process designed to apply the retrieved patterns for decision-making. Alternative names or subfields of data mining such as knowledge discovery in databases (KDD), data analysis, pattern mining, data archeology, knowledge extraction, information harvesting, data dredging, business intelligence, etc. A database also has alternative name such as *repository*, *transaction database*, and *sequence database*. As we are dealing with the evolving system, thus we specifically exploit the evolution and change analysis approaches. Therefore, we provide various temporal mining techniques.

Böttcher *et al.* [14][15] presented change mining and contrast mining. They defined change mining as data mining over a volatile and evolving repository with an objective of understanding evolution. They stated change analysis as; *contrast mining* if done on two data instances, and; *change mining* if done on multiple data instances. Change mining is one of the major problem domains of data mining, which have numerous applications. Wu *et al.* [16] applied *resume mining* on fields such as social networks. Takaffoli *et al.* [17] presented a framework to model and detect community evolution in social networks; a community-matching algorithm is used to identify and track similar communities over time. Mansoureh and Tamas presented *evolution mining* [17][18] as a kind of data mining that analyses time variant data or incremental data. In contrast, we took advantage of temporal graph (network) theories. Our approach is applicable to any kind of evolving system represented as a set of evolving system networks.

2.3 Graph (Network) Mining

We can analyse the evolving system using the evolving (dynamic) network of the entities. Evolution and change(s) happened in an evolving system can be analysed by representing the relationship between entities of the system as a graph or network. In the network, nodes are linked together, where a node represent an entity of the system. If a system is evolving, then the network of its entities is also evolving. A network represents relationship between various entities of a system. There are many network systems like movies at IMDB, World Wide Web internet links, and social network. Such network systems can be analysed using techniques like finding frequent patterns, recurrent structural patterns, finding subgraphs, detecting cliques, discovering coevolving patterns, and identifying sub-sequences. To do evolution and change analysis of evolving networks, we are interested in two well-known specialized fields: *network rule mining*, and *network subgraph mining*.

2.3.1 Network Rule Mining

Network Rule Mining is based on the fundamental concepts of *association rule mining*. A *transaction database* contains itemsets in an unordered fashion. *Association rule mining* [36] identifies the occurrence of two item together in a *transaction database* according to given threshold values. A rule is denoted as A ➔ B, where 'A' and 'B' are the set of items in a *transaction database* where 'A' is antecedent and 'B' is consequent. The entities have random order in *association rules* whereas entities are ordered in *sequential rules* (retrieved from a *sequence database*) [40]. We use sequence database representing a *network database*, which contains entity-sets (i.e. itemsets) of ordered network of entities. This network database helps to retrieve *network rules* from a network database.

Böttcher *et al.* [15] demonstrated association mining and key performance indicator to detect changes at different points of time and retrieved vast number of change rules. Abonyi *et al.* [67] presented views about links between computational intelligence and data mining; this is supported with a case study to extract knowledge represented by fuzzy rule-based expert systems that uses data mining. Liu *et al.* [68] presented intelligent computation of association rules based on a fixpoint operator for computing frequent itemsets. Ting *et al.* [69] present a study of linkage discovery (a topic in Genetic Algorithms) by using association rule mining instead of applying GAs for data mining. Luna *et al.* [70] proposed a technique for mining context-aware association rule for itemsets depending upon a contextual feature. Xuan *et al.* [71] proposed graph topic model (GTM), which models the edges between nodes in a graph. Similar to contributions [70][71], we used network rule over system states to retrieve *Network Evolution Rules* (NERs), which are provides information about multiple states of an evolving system. Extensively, we propose *Stable Network Evolution Rule Mining* (SNERM) algorithm on *evolving system* represented as a set of evolving networks.

2.3.2 Temporal Rule Mining

Temporal rule mining was introduced by Ale and Rossi [37], which has following contributions. Techniques related to rule and pattern mining includes following contributions. Some database may have *temporal association rules* over time [37]. Related to this, Liu *et al.* [38] presented a statistical approach to analyse temporal databases to identify stable rules, trend rules, and removed unstable rules based on statistical calculation. We are interested to retrieve persistent (or stable) link in dynamic network databases for example and some network may have *persistent links* over time [39]. Jakkula and Cook [72] described and applied temporal relations discovery in smart home datasets. Liu *et al.*

[73] proposed a general post-filtering framework based on association mining and temporal filtering for context knowledge discovery. Recently, Qin *et al.* [74] presented interactive temporal association analytics framework named as Temporal Association Rule Analytics (TARA). Applications of temporal association rule mining are: sensor data in smart environment [72], semantic concept detection in video [73], calendar based rule [75], temporal document collections [76], Web Log Data [77], and rapport detection in peer tutoring [78].

Shokoohi *et al.* [79] proposed *time series rule* that are identified using time series motifs. Rolfsnes *et al.* [80] presented application of association rule mining to do accurate change recommendations for open-source softwares. Thereafter, Rolfsnes *et al.* [81] used random forest classification models to improve interestingness measures (such as confidence and support) by learning from previous change recommendations. Most recently, Gyorgy and Arcak [82] presented a broad dynamical model of interconnected modules to study the emergence of patterns; they also characterized the stability for the spatially non-homogeneous patterns. In comparison to these approaches, we additionally studied evolution as well as stability, changeability, and complexity properties of network (or graph or tree) rules for an evolving system.

In comparison to the work done in temporal rule mining, we presented system network evolution rule mining based on system engineering and graph theory. Extensively, we introduce an algorithm to retrieve Stable Network Evolution Rules (SNERs). Our novel content includes our proposed evolution rules, stable rules, stability metric, and changeability metric.

2.3.3 Network Subgraph Mining

In a given system state series, an evolving network EN_i represents state S_i. The graph $H = (V', E')$ is a subgraph of EN_i if $V' \subseteq V$ and $E' \subseteq E$. The frequency of subgraphs H is the number of occurrence of H in EN_i. Subgraphs are the building blocks that construct its network; thus, subgraphs are used to study network properties. One such property is *frequent subgraphs*. If the frequency of H is higher than a threshold value, then H is a *frequent subgraph* of EN_i. Few frequent-subgraph mining techniques are AGM [49], FSG [50], and gSpan [51]. A *network motif* [52], [53] is a well-known kind of frequent subgraph that are statistically obtained using random graphs. The *graphlet* [54], [55] is another well-known kind of subgraph, which is a small connected induced subgraph.

Network Subgraph Mining is another technique to analyse a network. The subgraph can be of two type. On one hand, frequent subgraphs (frequency higher than the given threshold) are termed as network motifs. On other hand, induced subgraph are termed as graphlets. *Network motif mining* and *Graphlet mining* are the well-known techniques of *network subgraph mining*. A *frequent subgraph* is a subgraph that occurs frequently in a network. *Network Motifs* are small *frequent subgraph* (or a pattern) in a network with significantly high frequency as compared to random networks. *Motif mining* gives insights on characteristics and internal working of a network. We use motif mining to capture *frequent subgraph* as a pattern.

Ahmed *et al.* [83] proposed a mining technique to capture non-redundant evolution paths in evolving networks. Sun *et al.* [84] proposed a mining technique to detect the co-evolution of multi-typed objects (such as multi-typed clusters) in a dynamic star network. Shah *et al.* [85] described patterns in a real-world dynamic graphs, identification, and ranking of those patterns; then, proposed TimeCrunch algorithm to find coherent and temporal patterns in dynamic graphs.

In contrast to the approaches, our approach deals with the evolution happened in evolving networks of an evolving system. The primary focus of these approaches is to study network evolution or dynamics. Generally, network-mining techniques do not consider multiple states of network. Similarly, most of the evolution and change analysis techniques do not consider network of inter-connected entities. Mining on such entities in multiple states of the repository can retrieve many hidden information. However herein, we integrated network and evolution information together to propose *Network Evolution Mining*. Additionally, we present an application of *mining Network Evolution Graphlets*, *Network Evolution Motifs*, *Changeability Metric*, *Stability Metric*, and *System Complexity* for six set of evolving networks representing the six evolving systems.

2.3.4 Temporal Network Motif

Kovanen *et al.* [86] proposed a technique to detect *temporal motif* based on significant, intrinsically dynamic, mesoscopic structures, and graphlets in temporal networks. Paranjape *et al.* [87] presented a framework for counting *temporal motif* using algorithms for classes of motifs to find key structural patterns as induced subgraphs on sequences of temporal edges in temporal networks. The *temporal motif* [86][87] and *temporal network analysis* [88] techniques deal with the time-variant patterns occurring as connections that are active only for restricted periods. Holme [88] presented review on the methods for *temporal network analysis* of social media data; he also discussed techniques for temporal network motifs. We used the temporal network motif are used to describe the evolving system network. Borgwardt *et al.* [89] extended subgraph mining from static graphs to the time series of graphs. After that, Wackersreuther *et al.* [90] presented a framework to perform frequent subgraph discovery in dynamic networks with edge insertions and edge deletions over time. Patel *et al.* [91] followed by Chiu *et al.* [92] (with Keogh as common co-author) coined the *time series motif* that sounds similar to our *network evolution motif*, although both are different. The first deals with motifs occurring in a time series, whereas the later deals with motifs occurring in evolving network series.

Following are the application of temporal (or evolving) network motifs. Braha and Bar-Yam [93] studied denser motifs on e-mail dataset in which each snapshot network represents contacts aggregated over a day. Zhao *et al.* [94] studied the temporal annotations (e.g., timestamps and duration) of historical communications graphlet in social networks using communication motifs and their maximum flow. Jurgens and Lu [95] proposed a method to represent Wikipedia revision history as a temporal bipartite graph of editor interactions, which identifies significant author interactions as network motifs of diverse editing behaviors. Meng and Guo [96] proposed gene regulatory networks (GRNs) based self-organizing robotic to generate evolving network motifs for path detection in unknown environment autonomously. Bhattacharya *et al.* [97] applied network motifs of software system to capture, analyze, and infer software properties.

Hulovatyy *et al.* [98] presented a dynamic graphlets based analysis of temporal networks by capturing inter-snapshot relationships. Recently, Martin *et al.* [99] proposed two work: (a) REConstruction Rate (REC) to determine the rate of graphlet similarity between different states of a network and (b) REC Graphlet Degree (RGD) to identify the subset of nodes with the highest topological variation. In contrast to these approaches, we presented network evolution subgraphs that are further used to calculate the complexity of evolving system states.

2.3.5 Graph Evolution Patterns

Berlingerio *et al.* [44] proposed an approach to mine Graph Evolution Rules (GERs) using a tool GERM and shown four real world networks. Subsequently, Leung *et al.* [45] proposed mining interesting Link Formation Rules (LFRs) containing link patterns as dyadic and/or triadic structures in social networks. Afterwards, Fan *et al.* [46] proposed graph-pattern association rules (GPARs) to discover regularities between entities in social media graphs, which help in marketing by identifying influencing customers. Recently, Scharwächter *et al.* [47] proposed an EvoMine, a tool to mine frequent graph evolution rules with insertions and deletions of edges and nodes using labelling on node and edge; additionally, they [47] also compared the GERM, LFR miner, with their EvoMine. Recently, Namaki *et al.* [48] presented an approach to discover graph temporal association rules (GTARs) using support and confidence.

We performed hybrid mining by integrating *network rule mining* and *subgraph mining* with *evolution mining* to retrieve evolution information. Our findings are similar to Graph Evolution Rule (GER) [44], Link-Formation Rules (LFR) [45], and Graph-Pattern Association Rules (GAPRs) [46]. Like others [44][45][46], we also demonstrated experiments to retrieve Network Evolution Rules (NERs). In addition to these state-of-the-arts, we presented Stable NERs, Stability metric, and Changeability metric for evolving systems. The strength of our proposed Mining NERs and SNERs algorithm over the current state-of-art is as following. First, in contrast to GERs [44], LFRs [45], GPARs [46], EvoMine [47] and GTARs [48], we identify NERs and then compute stability of NERs to retrieve the SNERs. The existing mining algorithms do not use the minimum stability (minStab) measure, which is novel in our approach. Second, the existing mining algorithms are not extended to find system changeability metric. The stability of SNERs provides important statistical information about the NERs over time to compute the stability and changeability metric.

2.4 Evolving System Analysis

After retrieval of mining and learning information, we can analyse various system evolution properties of an evolving system based on change in generation (huge change in system) and state (small change in the system). We restricted our studied for three such properties: Changeability, Stability, and Complexity.

2.4.1 System Changeability and Stability

Changeability [100] is system's ability to change from one state to another such that it does not undergo for a generation change. *Evolvability* [101] is system's ability to undergo drastic change such that the generation of system will change. Both changeability and evolvability are used to measure the chaotic and dynamic nature of system. *Stability* [102] is another kind of property, which is opposite to the changeability and evolvability. Stability is system's ability to stay unchanged or unevolved from one state to another. Stability provides constant functionalities by enduring changes, which is opposite to the changeability. Our goal is to study *changeability* and *stability* of an evolving system by modelling system's state as graph (or network).

The difference between changeability and evolvability can be understood with the following example. Suppose the generations of a software are given as V8, V9, V10, and V11. If the software is updated within a generation (say V10), then it will create states (i.e., versions V10.1, V10.2 etc.) of same generation (V10). Changeability deals with analysis of states within a generation, whereas evolvability deals with analysis for a set of different generation. However, stability property remains same for analysis within a generation and between the different generations.

Contributions related to the changeability analysis includes following works. Ross and Rhodes [103] introduced Epoch-Era Analysis (EEA) as an approach to model a tradespace exploration process about the temporal system value environment. Here, an *epoch* is defined as one of the potential contexts for the system lifecycle and an *era* is defined as sequence of epochs. Afterwards, the EEA is used to identify changeability in system design [104] and sustaining lifecycle value of system [105]. The EEA was extended by Fitzgerald *et al.* [106] to propose Valuation Approach for Strategic Changeability (VASC), which assesses changeability over a system's lifecycle. Two application of changeability, Koh *et al.* [107] assess the changeability of complex engineering systems, and Fluri [108] assessed changeability of source code entities in evolving software systems.

Xuan *et al.* [109] proposed keyword association line network (KALN) to do uncertainty analysis of keyword system for web events. Recently, Avalos *et al.* [110] analyzed impact of changeability during external agent changes, and analyzed change affects the evolution on the complex adaptive system (CAS). In contrast to these contributions [103]-[110] for changeability analysis, we made following contribution and improvement. Our approach computes novel changeability metric for a state series of an evolving system. Our approach is based on knowledge discover in databases (KDD) and network (graph) theory. We also demonstrate its application on six evolving systems.

Earlier contribution based on system stability analysis includes following. Belykh *et al.* [111] determined the global stability of graphs using the Lyapunov function in addition to graph theoretical reasoning. Moreau [112] described stability analysis model based on set-valued Lyapunov theory with graph-theoretic and system-theoretic tools. Delvenne *et al.* [113] measured stability of partition for community structure using dynamic Markov process. Angulo *et al.* [114] showed most motifs emerge from interconnection patterns (as modules) can identify intrinsic stability characteristics. The work presented in [103]-[114] are fundamental theoretical stability analysis. Whereas, we calculated changeability and stability based on knowledge discovery and data mining of evolving system networks.

There are many theoretical and mathematical approaches for the stability of graph e.g. [111][112][113][114][115]. In contrast, our approach is based on automated evolution rules and motifs (i.e., graphlets) mining of evolving system's graphs (networks). There are many contributions for stability analysis on different kinds of systems: chaotic systems [111], multi-agent systems [112], communities based systems [113], and peer-to-peer systems [116]. In contrast, our approach focuses on knowledge discovery and data mining of an evolving system. We applied our approach on evolving systems modeled as a set of evolving networks. Additionally, we conducted experiments to demonstrate a study of SysEvo-Analytics.

2.4.2 System Complexity

There is one more property of systems - other than changeability, evolvability, and stability - which analyses *complexity*. *System complexity* is a measure of entropy and randomness between the interconnected entities. Specially, study of system complexity over multiple states can result in useful evolution information. The recommendation and knowledge can support the changeability, evolvability, stability, and complexity measure. Pincus [117] described approximate entropy (ApEn) as mathematical family of formulas and statistics, which quantify the concept of changing complexity. Rosen [118] described the complexity as a property to interact with systems around us that are continually changing. Different techniques are applied to measure the complexity of software system [119] and ecological system [120]. In contrast, our subgraph mining approach aids to calculate complexity based on McCabe [121] technique.

2.5 Deep Learning and Deep Neural Networks

Artificial Neural Network (ANN) is a paradigm of machine learning, which is inspired from biological systems. The ANN models are based on simplified topology and tries to mimic information processing capabilities of the human nervous systems in a computational domain. This implies ANN mimic human neural systems. The learning process of ANN adapts its neural network according to training data. This process is much like a stimuli-response model of neural network seen in human. *Deep Neural Network* (DNN) [56] is a class of ANNs, which are trained using *deep learning* algorithms. First DNN is based on the work of Paul Smolensky [122], who invented Harmoniums in 1986, which is now known as RBM [123]. The RBM gained popularity quite later in 2006, when Geoffrey Hinton *et al.* [124] described a fast learning algorithm to train DBN. The persistent contrastive divergence algorithm proposed by Tieleman [125], who introduced a fast and simple way to train such models.

In this work, we use deep learning to propose an approach, which learns useful information from *time variant data* of an *evolving system*. In our approach, we describe how the deep learning can be used to create DNNs for an *evolving system*. This technique reduces human intervention by recommending about evolution of the evolving system. The recommender system is constructed to help in decision making about evolving system.

Naturally created data often contain patterns, it is possible for humans to identify and label such patterns. These labels prove quite useful to train machine-learning models, known as '*supervised learning*' models. However, in other cases, it becomes quite tedious or even impossible to do such a labeling of patterns in data. For such cases, there exist different classes of machine learning models called '*unsupervised learning*' models. The ability of unsupervised techniques to learn directly from unlabeled data makes them very useful, especially for our purpose, to detect the history of change patterns in an evolving system. Hence, we are interested to apply DNN models on evolving systems. We discuss three well-known unsupervised DNN models: *Restricted Boltzmann Machines* (*RBM*), *Deep Belief Networks* (*DBN*), and *denoising Autoencoders* (*dA*); thus, we explicitly describe preliminaries for these three models.

2.5.1 Restricted Boltzmann Machine (RBM)

The RBM is a type of neural network, where the neurons in the network topologically form a complete bipartite graph, separating the neurons into visible units and hidden units. The weights between the nodes are denoted by a matrix W, where W_{ij} represents the weight between ith visible unit and jth hidden unit. The state of ith visible neuron is denoted by the variables v_i and that of the jth hidden neuron by h_j. Energy of the machine is defined by the formula

$$E(v, h) = -\sum_{i,j} v_i h_j W_{i,j} - \sum_i v_i b_i - \sum_j h_j b_j$$

where b stands for the biases. This energy formula is used to make a probability equation defined by

$$P(v, h) = e^{-E(v, h)/Z} / Z$$

where Z is the normalization constant $Z = \sum_{v,h} e^{-E(v, h)}$. Therefore, the probability of a data point can be defined as

$$P(v) = \sum_h P(v, h) \qquad \textbf{...Eq1}$$

Now, the network can be trained by adjusting the weights and biases, which maximizes the probability for the training set. To do this, the most common and efficient way is to use the approximate gradient of the contrastive divergence [127].

2.5.2 Deep Belief Network (DBN)

There are two phenomena *shallow neural network* and *deep neural network*, which explain the fundamental understanding of DBN. A *shallow neural network* can classify a data with the large error rate, which is good for surface level learning. *Deep neural networks* can classify a data with the small error rate, which is good for deeper level learning. Before DBN, the *shallow networks* dominated the field of neural networks, and the *deep architectures* (or *deep networks*) suffered from vanishing gradient problem [128]. However, due to the novel learning techniques developed by Hinton et. al. [124], even deep architectures can now be efficiently trained by training each layer greedily. Train the first, second, and other layers as greedy layer-wise pre-training using RBM is given by the formula

$$P(x, h^1, h^2\dots, h^L) = ({}^{L-2}\pi_{k=0}\ P(h^k \mid h^{k+1}))\ P(h^{L-1}, h^L) \qquad \textbf{…Eq2}$$

where x represents input vector, h represents hidden layer and L is the number of hidden layers. Then fine-tune the parameters of deep network using supervised gradient descent of the negative log-likelihood cost function [124]. Such neural network gives better results in different applications.

2.5.3 Autoencoders

With the rise of popularity in deep architectures a new *deep learning* technique *autoencoders* also gain popularity. The *autoencoders* learns internal patterns of the input data x by encoding it into some representation c(x) so that it can be later reconstructed. Simplest *autoencoders* consist input, hidden, and output layers.

Bengio *et al.* [129][130] suggested that stochastic gradient descent yields useful results. The network is usually trained by minimizing the negative log-likelihood of the reconstruction error. Thus, this minimizes error of reconstruction. In case of binary inputs, the error of reconstruction is calculated as the sum of Bernoulli cross-entropies, which is given by formula

$$L(x, z) = - \sum_i x_i \log[z_i + (1-x_i) \log(1- z_i)] \qquad \textbf{…Eq3}$$

where $z_i = d(c(x_i))$ represent reconstructed x_i and where, c(x) represents the coded x and d(c(x)) represents decoded c(x).

2.5.4 Denoising Autoencoders (dA)

Autoencoders have been receiving a lot of interest in the research. Some of the promising variants of *autoencoders* are the *denoising Autoencoders* (*dA*) [126], *Sparse Autoencoders* and *Variational Autoencoder*. However, we restrict ourselves only with the dA, due to its simplicity among all. Vincent *et al.* [126] introduced dA, which is modification of the original *Autoencoder*. In dA, during the training phase instead of the actual input data, a slightly corrupted version of the input is given to the neural network. This makes a neural network more robust to changes in input data comparatively to its predecessor.

2.5.5 Graph learning

In graph learning techniques, Yang, *et al.* [131] presented a framework, Concept Graph Learning (CGL), for the academic graphs of courses and concepts using course links onto the concept links. Yanardag and Vishwanathan [132] presented Deep Graph Kernels, a framework that learns latent representations of sub-structures for graphs by learning dependency information between sub-structures. Cao *et al.* [133] proposed a model for learning graph by encoding each vertex as a low dimensional vector representation for learning based on stacked denosing autoencoders. Bui *et al.* [134] proposed a framework, Neural Graph Machines, with a graph-regularized that applies neural networks on label/unlabeled data with Feed-forward NNs, CNNs, and LSTM RNNs. Somanchi *et al.* [135] presented an approach of Learning Graph Structure (LGS), which is a greedy framework for efficient learning of graph structure. Similarly, we proposed the Graph Evolution Learning to study the system evolution. Nori *et al.* [136] proposed ActionGraph, which is used to predict interests of social media users based on time-evolving and multinomial relational data. In contrast, we presented deep learning of evolving (dynamical) system over time, but our approach takes advantage of robust fundamental network (graph) theory.

2.6 Recommender system

Current state-of-art with work related to recommender or predictive systems includes following. Castro-Herrera *et al.* [157] describe a recommender system that helps in the challenges of dynamic evolving internet forums and representations of user profiles. Susto *et al.* [158] presented an adaptive machine learning and regularized regression based method to support decisions for flexible predictive maintenance. Yin *et al.* [159] proposed temporal context-aware recommender system (TCARS) for analysing user behaviours in social media systems that are influenced by intrinsic interest and the temporal context. Zeng *et al.* [160] proposed dynamical context drift model using particle learning for time varying contextual multi-armed problem. The graph learning approaches is applicable to do scenario like Collective Robot Learning (as described by Kehoe *et al.* [161]). Our SysEvoRecomd can be applicable to all such environments.

Our approach can help to construct a recommender system using matrix factorization technique along with temporal information; for further reading refer to Koren and Robert [162], which described matrix factorization techniques for recommender systems. Our SysEvo-Analytics can improve the system thinking; for further reading Whitehead *et al.* [163] described about the system thinking. Zhao *et al.* [164] proposed a framework of coordinate matrix factorization to construct entity-entity association matrix for measuring semantic relatedness between entities, categories, and words in Wikipedia. Lian *et al.* [165] proposed a deep fusion model (DFM) by improving the representation learning abilities in deep recommender systems for personalized news recommendations.

2.7 Matrix and Network Reconstruction

LeCun *et al.* [56] described Deep Learning techniques were proven successful in matrix reconstruction (as an output) e.g. while reconstructing (brain-neurons and their mutual contacts) neural circuits. Unnikrishnan *et al.*, [166] discussed about applications of network reconstruction in the systems like: social networks, bioinformatics, and chemical reaction modelling. Yue *et al.* [167] addressed and solved the problem of reconstruction of linear dynamic networks from heterogeneous datasets using regression model.

Recently, Angulo *et al.* [168] found that reconstructing any property of the interaction matrix is generically difficult; they presented fundamental limitations to design algorithms for better network reconstruction. Most recently, Porfiri and Marin [169] used finite-state ergodic Markov chain model and described an approach for entropy-based network reconstruction. Whereas, we exploited the properties of deep learning for reconstructing interaction (connection) matrix to do network reconstruction.

2.8 Possible applications of our work

In this thesis, our approach is useful for the application of evolving graph mining and temporal network analysis. Specially, where there is a need to identify stable (or persistent) rules of system entities over time. Our approach has many applications, which includes: temporal networks analysis of social media [88], analyzing sensor data using rule-based technique [137], and human interaction patterns in temporal networks [138].

We demonstrate the two major applications field of study. First, for an evolving software system, which is an active working problem. Recently, Di Nucci *et al.* [139] proposed a technique to select a set of classifiers, which are better to predict the software bug proneness. We demonstrate our technique on a software repository of Hadoop-HDFS. Second, for evolving natural language processing, which is a current working issue in computation intelligence [140]. We demonstrate our technique over repositories of natural language system.

Some possible application of our approaches and automated tools are as follows. Our approach can be useful to enhance the quality of Association Link Network (ALN as described by Luo *et al.* [141]) for organizing loose web resources over time instances. Nasraoui *et al.* [142] presented an approach to do mining of web usage patterns from web log files for discovering and tracking evolving user profiles. Analogously, we demonstrate experiments on real-world evolving systems. Possibly our approach may also be helpful in the business automation based on evolving services (as presented by Huang *et al.* [143]); our approach can help in predicting the evolution and stability of the service network.

Our approach can be applicable to analyze social network structures [144], to characterize communication network motifs [145], and to do systems thinking by aiding systems modeling language (SysML) [146]. Some other application areas of our work are as follows: biological network, computer network, economical network, attackers-victims patterns in computer networks, star patterns in sky, co-author patterns over a decade in scholarly data, and family-members patterns over few generations in social sciences studies.

Possible applications of our work is to study complexity of domains like social media [93], communication motifs [94], natural language processing [95], robot path [96], and software systems [97]. Our approach is applicable to study the fields of evolving (or temporal) networks, and specially to identify persistent patterns of entities over time [39]. For example, analysis of temporal networks in social media [88], dynamic graphlets in the biological dynamic networks [98], dynamic motifs in socio-economic networks [147], and communication motifs of dynamic networks [148].

Deep Recurrent Neural Network (Deep RNN) is a type of recurrent neural network (RNN), which is constructed using a deep learning approach [149][150]. Deep RNN is useful in many field such as speech recognition [151][152], natural language processing [153], and image recognition [154]. Our technique can also be applicable on other kind of evolving systems to do temporal analysis e.g. recognizing temporal facial action [155] and facial pain expressions [156].

Chapter 3

System Evolution Analytics (SysEvo-Analytics)

There are many evolving system having many entities (or components), which are evolving over system states. The connection (or relationship) between entities also evolves over system state, which makes series of evolving networks. It is a challenge to do evolution and change analysis over state series of an evolving system. Thus, there is need of SysEvo-Analytics model, which studies evolving networks over evolving states to provide system evolution information for analysis. This can be achieved with the help of hybrid mining and learning approaches. In this chapter, we introduce a SysEvo-Analytics model, which is divided into following two parts.

- We propose two pattern-mining techniques: *network evolution rule mining* and *network evolution subgraph mining*. The first technique retrieves *Network Evolution Rules* (NERs), and the second technique retrieves *Network Evolution Subgraphs* (NESs). The network rule information can be detected using *network rule mining*. The network subgraph information can be retrieved using *network subgraph mining*. The evolution information is detected using *evolution mining*.
- We propose *System Evolution Learning*, which uses two kinds of hybrid learning. First, the network pattern information is trained using *graph structure learning*. The evolution information is trained using *evolution and change learning*. We accomplish this by implementing a *deep evolution learning*. This technique uses an evolving matrix to generate *evolving memory* in the form of a proposed *System Neural Network* (SysNN). The SysNN is useful to predict and recommend based on system evolution learning.

We start by redefining the Definition 1.1 in context of evolving network as evolution representor. This is followed by problem definition that we target in Chapter 3, 4, 5, and 6.

Definition 3.1: A <u>*State Series*</u> SS = {S_1, S_2 … S_N} at various time points {t_1, t_2, t_3 … t_N} is pre-processed to make a series of an evolving network EN = {EN_1, EN_2… EN_N} corresponding to the system states. Thus, we can define (S_i, EN_i, t_i) such that the S_i represents system state and the EN_i represents its evolving network at the i^{th} time point t_i, where ‘i’ varies from 1 to N.

Problem definition: Suppose evolution and change happened in an evolving system can be denoted as transition of one state to another. Each state is represented as a system network (or graph) of inter-connected entities, such that each node represents an entity in a state and each edge represents a connection between two entities. System entities and their connections represent a system network (or graph), where entity-connections are prone to undergo evolution and change. The evolution and change analysis on an evolving system retrieves hidden evolving information that can be helpful in decision-making. For an evolving system represented as an evolving network series, it is challenging to study system properties like changeability, stability, complexity, and evolvability. There are following two challenges in evolving system mining and learning:

A. Usually an evolving system is complex, thus pre-processing of an evolving system is a challenging task. This process requires experience of system domain expert. We provide an overview for pre-processing an evolving system to make a series of evolving network.
B. One state of an evolving system can be processed with a data analysis algorithm. However, the processing of multiple states of an evolving system with the same algorithm is usually a challenging task. Thus, such algorithm needs customization. We present an overview of two models for mining and learning a system state series.

3.1 SysEvo-Analytics Based on Pattern Mining of Inter-Connected Entities

This section proposes SysEvo-Analytics model. We start by defining keywords.

Definition 3.2: Network Evolution Mining (NEM) is defined as mining of evolving networks, which retrieves the network evolution and change information about evolving inter-connected entities in a system state series.

Definition 3.3: Network Evolution Rule (NER) is defined as set of evolution rules, which contains the network evolution and change information about inter-connected entities in a system state series. The NER can be interpreted as "if evolution and change(s) occur to a source entity-set, then evolution and change(s) is likely to occur in its target entity-sets". For example, suppose if an entity-set $\{e_1, e_2\}$ depends on another entity-set $\{e_3\}$. Then all possible rules are $\{e_1\}\rightarrow\{e_3\}$ and $\{e_2\}\rightarrow\{e_3\}$. The rule $\{e_1\}\rightarrow\{e_3\}$ means $\{e_1\}$ *may be connected to* $\{e_3\}$, which predicts connection with probability (*minSup* and *minConf*).

Definition 3.4: Network Evolution Subgraph (NES) is defined as set of subgraphs, which contains the network evolution and change information about evolving inter-connected entities in a system state series. The subgraph information is described as follows.

- Subgraph information is a doubleton set of subgraph key and subgraph frequency, which is represented as $(H_j, freq_j)$, i.e. H_j has independently occurred $freq_j$ times in the network. The subgraph key and subgraph frequency is described as follows.

- Subgraph key is an enumeration key of *subgraph pattern* and denoted as H_j. Here, H_j denotes the jth subgraph key. A set of all possible subgraph keys with a given number of nodes in a subgraph is denoted as $\{H_1, H_2 \ldots H_M\}$, where M is the

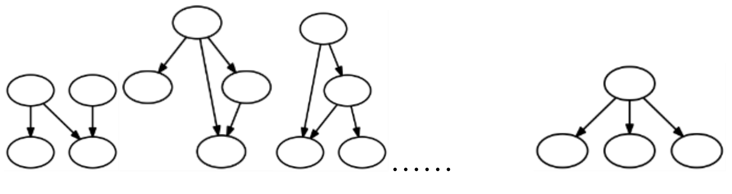


Figure 3.1 Example of subgraphs of size 4 (number of nodes) as a set of subgraph key $\{H_1, H_2, H_3, \ldots H_{198}\}$.

total number of all possible subgraph patterns (as shown in Figure 3.1).

- Subgraph frequency is the count of independent and non-overlapping occurrences of subgraph key H_j in a graph G and is denoted as $freq_j$. Here, $freq_j$ denotes the subgraph frequency corresponding to jth subgraph key H_j. A set of subgraph frequencies for all possible subgraph keys is denoted as $\{freq_1, freq_2 \ldots freq_M\}$, where, M is the total number of all possible subgraph patterns.

Solution definition: We propose a model to develop an algorithms and tools for mining of time variant data of evolving system. This model reduces human intervention by providing automated hidden evolution information about the evolving system. The information can be helpful in decision making about evolving system. The model has four steps to do evolution and change mining of an evolving system, which are shown in the Figure 3.2 and described as follows:

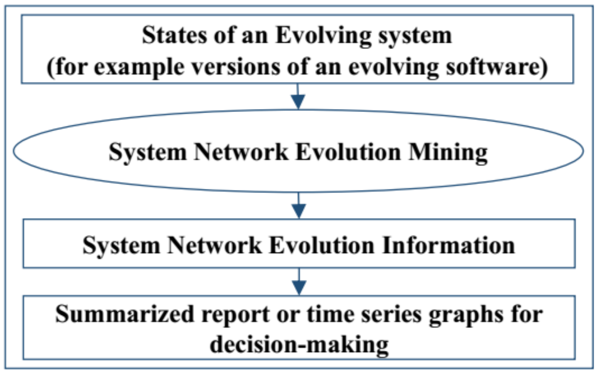

States of an Evolving system
(for example versions of an evolving software)
System Network Evolution Mining
System Network Evolution Information
Summarized report or time series graphs for decision-making

Figure 3.2 An overview of the System Evolution Analytics model.

1. Preprocess a state series of an evolving system to make a series of evolving networks $EN = \{EN_1, EN_2 \ldots EN_N\}$, where each evolving network EN_i stands for state S_i. Each evolving network consists of relationship between inter-connected entities of a system. Due to several reasons, the connections between system entities evolve with time.
2. Retrieve mining information about each evolving network using well-known network (or graph) mining techniques, such as network rule mining and network subgraph (motif or graphlet) mining.
3. Thereafter, merge the retrieved information of each state together to make an aggregated evolution information, which describe mining information about the state series of an evolving system.
4. As a summarized report, create time series graphs, charts, or tables. Then, use the report to analyze the evolution of an evolving system as per the state series.

Algorithm 3.1 ***System_Evolution_Mining**(EvolvingSystem)*

```
Initialize i ∈ integer 1 to N
For each retrieved N states (S1, S2… SN) of the EvolvingSystem and stored in a directory DirRepository
 { evolvingNetworks = Preprocess(DirRepository)   ⎫
   mining_info = NEM(evolvingNetworks)            ⎬ NEM
   network_evolution_infos = merge(mining_info)   ⎭
 }
report_time_series = info_present( network_evolution_info)
```

The SysEvo-Analytics model can be expressed as the Figure 3.2 and as the *System Evolution Mining* algorithm. *Network Evolution Mining* (NEM) finds the hidden network evolution and change information in different states of an evolving system. The output is in the form of evolution and change information, which further help in decision-making. Our mining algorithm uses different states as input to generate evolution and change information as output. The brief description of two approaches are given in the following two sub-sections:

3.2 Network Evolution Rule Mining

In this sub-section, we describe an approach of rule evolution mining, which is based on three mining techniques: *Network rule mining*, *Sequential rule mining*, and *Evolution mining*. This approach is shown in the Figure 3.3. Every system has a *sequence* of entities that are *connected* together such that entity connections evolve over time. *Sequential rule mining* can detect *sequential rules* of these inter-connected entities. Thus, these sequential rules are the network rules that are the most frequently inter-connected entities. Discovering evolution and change(s) in these *sequential rules* over several states aid to generate evolution and change information about evolving systems. For this, we propose *NER Mining* with the following four steps.

1. Collect different state of an evolving system and then pre-process each state of the evolving system to make an evolving network for the state. Thus, a system state series is pre-processed to make a series of evolving networks. We further pre-processed each evolving networks to make a system network database for each state, such that the database has two ordered sets of source and target entities. Due to ordered sets, the system network database is a type of sequence database.

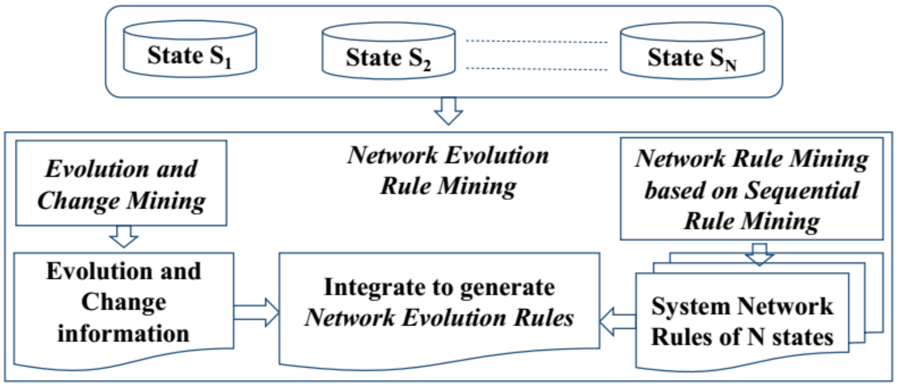

State S_1
State S_2
State S_N
Evolution and Change Mining
Network Evolution Rule Mining
Network Rule Mining based on Sequential Rule Mining
Evolution and Change information
Integrate to generate Network Evolution Rules
System Network Rules of N states

Figure 3.3 An overview of proposed Network Evolution Rule Mining.

2. The *Network Rule Mining* needs a state of an evolving system as input (Figure 3.4). The system state requires pre-processing to make a *system network database*. The approach internally uses sequential rule mining, which outputs all sequential rules for each state based on the sequential occurrence of source and target entities. The rule has support no less than some thresholds *minimum support* and confidence no less than some threshold *minimum confidence* given by the user. In this case, the sequence rules are the network rules, such that each rule contains set of source entities followed by set of target entities.

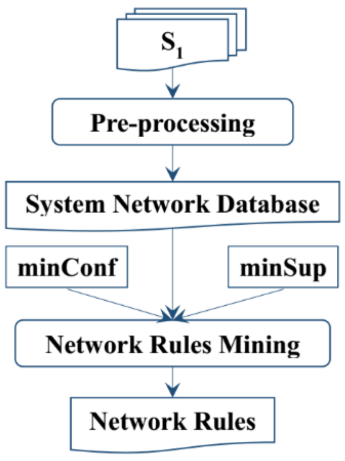


Figure 3.4 A flowchart of *Network Rule Mining* process for a state S_1 of an evolving system.

3. Thereafter, our approach merges all the network rule of

different states to make a collection of NERs. In short, our approach takes system network databases as input, with few threshold parameters. It retrieves NERs that contain antecedent and consequent according to the desire threshold. The rules look like {1, 3, 4} => {2, 5}.

4. Out of the collection of NERs, we retrieved the rules that are occurring in most number of states. These frequently occurring NERs (over the states) are the output of our approach, which we named as stable (or persistent) rules. These stable rules are helpful for the calculation of system stability over states. This aggregated system stability helps in decision-making.

3.3 Network Evolution Subgraph Mining

In this sub-section, we describe an approach of subgraph evolution mining based on two mining techniques: *Subgraph mining* and *Evolution mining*. This approach is shown in the Figure 3.5. *Network Evolution Subgraph Mining* finds the subgraph information for each states of an evolving system. The subgraph information has the frequency of subgraph occurrence in an evolving network of a state. Thereafter, the retrieved subgraph information is merged over all the states, which generates *network evolution subgraph information*. For this, we propose *NES Mining* with the following four steps.

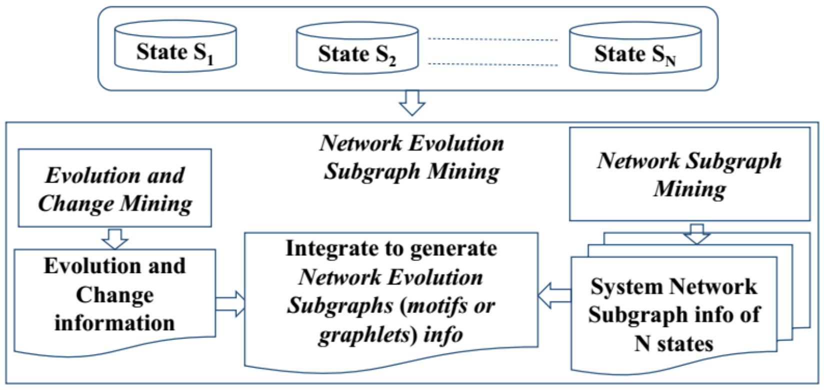

State S_1
State S_2
State S_N
Network Evolution Subgraph Mining
Evolution and Change Mining
Network Subgraph Mining
Evolution and Change information
Integrate to generate Network Evolution Subgraphs (motifs or graphlets) info
System Network Subgraph info of N states

Figure 3.5 An overview of proposed Network Evolution Subgraph Mining.

1. We pre-processed each state of the evolving system to make an evolving network for the state. Thus, a system state series is pre-processed to make a series of evolving networks. We further pre-processed each evolving networks to make it compatible for network subgraph mining algorithm.

2. Then our approach internally uses network subgraph mining, which outputs all the subgraph information for each state of an evolving system. Usually every subgraph mining approach takes network as input to retrieve *subgraph information* as output. Each subgraph information contains combination of *subgraph key* and *subgraph frequency*. The simplest *subgraph mining* steps are shown in Figure 3.6. Where, each subgraph represents inter-connected entities pattern.

3. Thereafter, our approach merges all the network subgraphs information of different states to make a collection of NESs. In short, our approach takes a set of evolving networks as input and retrieves NESs with its aggregate frequency of occurrence in the series of evolving networks. Each NES

represent subgraph of evolving inter-connected entities.

4. Out of the collection of NESs, we retrieved the subgraphs and induced subgraph. Such induced subgraph are referred as graphlets. Thus, we retrieved network evolution subgraphs (motifs and graphlets) as output of our approach over the states. These NESs and their aggregate frequencies are helpful for calculation of system complexity over states. This aggregated complexity helps for decision-making.

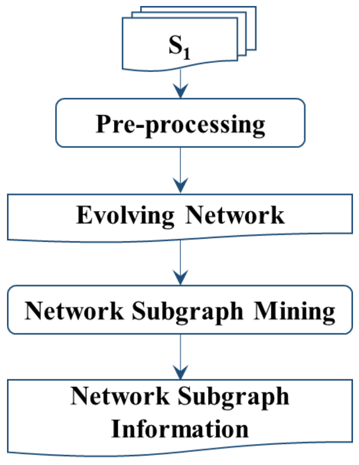


Figure 3.6 A flowchart of Network Subgraph Mining process for a state S_1 of an evolving system.

3.4 SysEvo-Analytics based on Deep learning of Evolving system

In this section, we describe our second proposed SysEvo-Analytics model for a state series. A machine learning technique can learn from an *evolution representor* (such as an unlabeled evolving matrix), which is useful to detect change patterns. The evolution learning depends upon machine learning that uses evolution representor (evolving matrix). Now, we describe the detail of proposed evolution and change learning of time-variant data. First, we describe the *evolution and change learning* technique using an evolution representor.

Definition 3.5: *Evolution and Change learning* is a machine learning of evolution and change(s) happened to a time-variant data at various states, using a representation of evolution (e.g., evolving matrix). It uses an evolution information to learn the evolution of a time-variant data of a state series. This makes a computer capable enough to understand the evolution of a time-variant data without any explicit programming.

Evolution and Change learning trains itself from an evolution representor built for a state series of evolving system. An existing machine learning technique can learn from the evolution representor stored in an evolving matrix. Such technique learns and understands the evolution and change(s) in an evolving system. *Evolution and Change Learning* technique can learn evolution and change(s) happened during the evolution of an evolving system. Such techniques can be applied to train a machine for hidden, non-trivial, and non-obvious information about evolution happened in an evolving system. Evolution and Change learning adapts, controls, and analyzes according to the evolution and change(s) automatically. Such learning process takes two or more states of a time-variant data as input to generate evolution information that helps to study temporal states of an evolving system. Next, we define the learning terminologies for an evolving system.

Definition 3.6: *System Evolution Learning* is a machine learning of evolution and change(s) happened over evolving system states at various time points. With this, a machine becomes capable to understand the system evolution without any explicit programming.

The system evolution learning can be accomplished by learning its evolving system graphs as representation of various states. Thus, next we define *graph evolution learning*, which aids to explain about *network evolution learning* and *system graph evolution learning*.

Definition 3.7: *Graph Evolution Learning* is machine learning of evolution and change(s) happened to evolving graphs from an evolution representor with the help of learning evolution and change patterns. With this, a machine becomes capable to understand the graph evolution without any explicit programming.

- The GEL may also be referred as *Network Evolution Learning* (NEL).

- *System Graph Evolution Learning* is machine learning of evolving system graphs, which makes machine capable enough to understand system graph evolution and change information about evolving inter-connected entities in an evolving system graph.

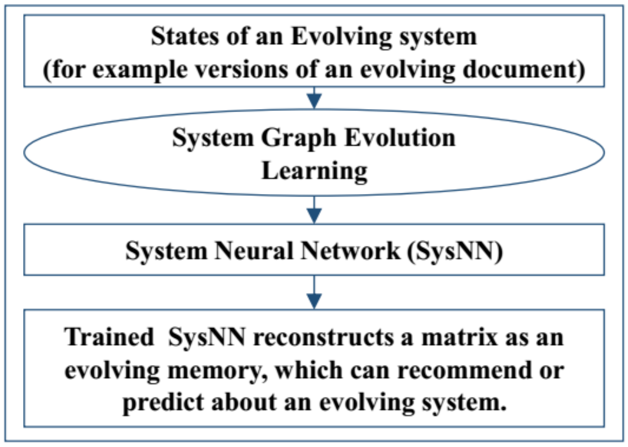
In our case, for machine learning we used deep learning that forms a deep neural network (a kind of artificial neural network). Thus, due to the use of deep learning we formed a new kind of artificial neural network that we defined as follows.

Definition 3.8: *System Neural Network* (SysNN) is an *artificial neural network* that contains information and understanding of evolution and change(s) happened to an evolving system. The weights are adjusted such that neurons provide probable connections (occurrences) of two entities (features) together. The SysNN help to recommend about the time-variant data (or non-stationary data) by learning and memorizing evolution information about system data.

Solution definition: The model has few steps to do evolution and change learning of an evolving system, which are shown in the Figure 3.7 and described as follows:

1. Preprocess multiple states of an evolving system, which makes a series of evolving networks {EN_1, EN_2… EN_{N+1}}. Where, each evolving network EN_i represents a relationship between evolving inter-connected system entities of state S_i.

2. Thereafter, represent each of an evolving network as a connection matrix for a state. This makes (N+1) number of connection matrices to do machine learning on the evolution and change patterns of connections between system entities. Out of (N+ 1) number of matrices (*N+ 1_Matrices*), only N number of matrices (*N_Matrices*) are used for training and the remaining 1 matrix is used for the testing purpose. The *N_Matrices* are used to make an evolution representor referred as evolving matrix. This


States of an Evolving system
(for example versions of an evolving document)
System Graph Evolution Learning
System Neural Network (SysNN)
Trained SysNN reconstructs a matrix as an evolving memory, which can recommend or predict about an evolving system.
Figure 3.7 An overview of the proposed System Evolution Analytics model.

evolving matrix is used for evolution and change patterns learning based on machine learning technique such as active or deep learning.

3. Thereafter, the learning of the evolving connection matrix makes evolution information for prediction. In case of deep learning, the evolution information is in the form of a System Neural Network (hidden black box realized as weight matrix) that can be used to reconstruct a matrix. This makes an output matrix as a kind of evolving memory about evolving system's evolution.

4. Then compare the binary output matrix with a testing matrix (representing a state not used for training purpose). To compare, we can use metrics like accuracy, f-measure, precision, and recall. Then, we can demonstrate the metrics result values in the form of graphs, charts, or tables. Then, we can use the report to analyze the quality of recommendation provided by the machine learning for the evolving system.

The four steps of the model can also be represented as four steps in the following algorithm.

Algorithm 3.2 ***System_Evolution_Learning*(*repository*)**

```
Retrieve N+1 states ($S_1$, $S_2$… $S_N$, $S_{N+1}$) of an evolving system at a repository
1. evolvingNetworks = preprocess(repository)
2. N+1_Matrices = convert(evolvingNetworks)
        evolving_matrix = evolution_representor(N_Matrices)
3. graph_evolution_learning(evolving_matrix)
        binary_output_matrix = deep_evolution_learning(evolving_matrix)
4. binary_classifier_metric(binary_output_matrix, testing_matrix)
```

Usually deep learning technique uses input matrix to generate a reconstructed matrix (as output). We represented each evolving system state as an evolving network, which makes N+1 evolving networks (*evolvingNetworks*) for a state series. Then each evolving network information is transformed to a matrix corresponding to each state. This makes N+1 matrices (*N+1_Matrices*). Thereafter, we merged N matrices (*N_Matrices*) to make an *evolving_matrix* that contains information about multiple states for learning purpose. After this, graph evolution learning uses deep evolution learning to make a neural network that contains system states information. Thereafter, the information is stored as an output matrix that is transformed to a *binary_output_matrix* (an evolving memory), which can be used for recommendation purpose.

The *System Evolution Learning* (*SEL*) generates *SysNN* (as a system graph evolution and change information), which needs to be accurate. Several states of an evolving system can undertake testing of *SEL* performance and accuracy based on information generation. User will classify correctness of the outputs. To improve and achieve correct output, the *SEL* algorithm is continuously adapted and fine-tuned. Fine-tuning will generate information satisfactory according to the human (user). The users can use their experience and understanding (visually and heuristically) to check correctness of the generated information. The *SEL* learns from its user 'how to use *deep learning* and *evolution learning* together', 'how to perform *deep evolution learning*', and 'how to generate *SysNN*'. This process will continue to generate the improved information.

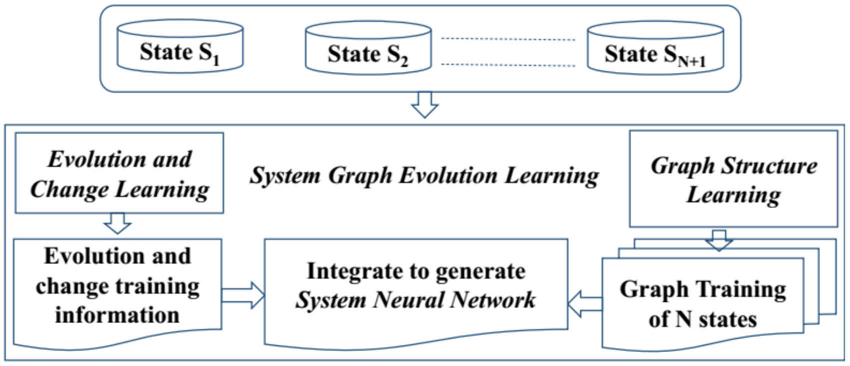

State S_1
State S_2
State S_{N+1}
Evolution and Change Learning
System Graph Evolution Learning
Graph Structure Learning
Evolution and change training information
Integrate to generate System Neural Network
Graph Training of N states

Figure 3.8 The System Evolution Learning (SEL) on N states of an evolving system.

The *SEL* underlies upon *graph structure learning* as well as *evolution and change learning* as shown in the Figure 3.8. Specifically, the *SEL* algorithm takes N+1 states of an evolving system (as input), uses *deep evolution learning* and generates the *SysNN* (as output).

3.5 System Evolution Analytics (SysEvo-Analytics) Tools

This sub-section describes the SysEvo-Analytics tools. With the SysEvo-Analytics model, our objective is to construct algorithms, which used various existing Java API to develop tools. Thus, we prototyped three different tools based on the four steps of the model discussed in Sections 3.2, 3.3, and 3.4.

First proposed approach (in Section 3.2) is prototyped as a tool based on fundamentals of Network Evolution Rule (NER) mining a kind of association rule mining. The proposed tool detects NERs from system network databases of a state series. Details about mining NERs and its applications are given in the Chapter 4.

Second proposed approach (in Section 3.3) is prototyped as a tool based on fundamentals of Network Evolution Subgraph (NES) mining. The proposed tool detects the recurring NESs (e.g. motifs or graphlets) with its aggregated frequency of occurrence in the evolving network state series. Details about mining NESs and its applications are given in the Chapter 5.

Third proposed approach (in Section 3.4) is prototyped as a tool based on fundamentals system evolution learning. Specifically, we implemented the *SEL* algorithm in the tool as java application. The tool learns evolution and change patterns from evolving matrix. The evaluation is done for the recommendation provided by the trained SysNNs, which are generated from the system evolution learning on evolving systems. The experiments are conducted to generate and report *evolving memory* as SysNN that helps to do recommendation about system. Details about SysNN and its applications are given in the Chapter 6.

All the tools follow the steps of the SysEvo-Analytics model. The first two tools retrieve network evolution patterns (in the form of rules and subgraphs information) and the third tool learns network evolution patterns happened over states. These tools are built and requires machine enabled with JRE and JDK 7th version or higher. To test the efficiency of the tool, we applied the tools on six evolving systems. The experiment on these systems rendered reports with NERs, NESs, and SysNN. Details about the experimentations of NERs, NESs, and SysNN are given in the Chapter 4, 5, and 6 respectively.

3.6 Summary

In this chapter, we proposed three System Evolution Analytics (SysEvo-Analytics) models based on three algorithms, which are prototyped as tools that are used to do experiments on six evolving systems. This chapter provided an overview of the artifacts proposed: *Network Evolution Rules* (NERs), *Network Evolution Subgraphs* (NESs), and *System Neural Network* (SysNN). Specially, we demonstrated the application of the tools for the system evolution analysis.

This chapter introduction is an abstract about the SysEvo-Analytics models. Specially, the three Sections 3.2, 3.3, and 3.4 of the Chapter 3 are elaborated in the Chapters 4, 5, and 6 respectively. The elaborations are described as algorithms, tools, and experimental empirical evaluation. The algorithms describe exact programmable steps, which help to visualize the technical details. The tools are based on the algorithms; the tools are used as proof of concept to do experiments. The experiments describe the usefulness and applications of our approaches.

Chapter 4

Stable Network Evolution Rule Mining for System Stability and Changeability analysis

Growing number of evolving systems creates demand for system evolution analysis with modern computational and artificial intelligence algorithms and tools. An evolving system has several inter-connected entities with some connections changing over time, which creates multiple states as a *state series* SS = {S_1, S_2… S_N}, where S_i is the ith state of the system. Each state can make a database, which could be mined to generate association rules. Here, we elaborate the subsection 3.2 on network evolution rule mining. In this chapter, we introduced *stability* characteristics of rules. A rule is called *interesting* in state S_i if its support and confidence exceed given thresholds. However, a rule might be interesting in one state, but not interesting in another state. A rule occurring in multiple states is called an *evolution rule* and its *stability* is defined to be the fraction of states in which the rule is interesting. An evolution rule is called *stable rule* if its stability exceeds a given threshold *minimum stability* (minStab). Additionally, we defined *stability metric* and *changeability metric*, which are quantitative measures of system evolution. All this is explained using an algorithm, *System Network Stability* (SNS), which uses minStab to mine Network Evolution Rules (NERs) and Stable NERs (SNERs).

We proposed a *Stable Network Evolution Rule Mining* (SNERM), *Stability Metric* (SM) and a *Changeability Metric* (CM) for an evolving system. For this, we use two different characteristics of Network Evolution Rules (NERs). First, given a network of a system state S_i, we call a NER *interesting* in S_i if its support and confidence exceed given thresholds (minimum support and minimum confidence). Second, given a set of networks for a set of states SS, we define the *stability* of a NER to be the fraction of states in SS in which the rule is interesting. We call a NER *stable* in SS if its stability exceeds a given threshold named as *minimum stability* (minStab). Further, we introduce new measures of *stability* and *changeability* for system evolution analysis over time. Based on this, we developed an intelligent tool, which is used for experiments on evolving systems. This work is automated as a SNS-Tool to demonstrate its application on real world systems. We applied our approach to a number of real-world systems including: software system, natural language system, retail market system, and IMDb system. It results Stable NERs, Stability Metric, and Changeability Metric value for each evolving system.

4.1 Introduction

A set of evolving networks can be pre-processed to make a database series (like a data series

[11]). Such a database series is also referred as dynamic databases or time variant data. The system state series is an unstructured (real-world) representation that needs pre-processing, whereas database series is structured representation to do data mining. Usually, such database series are stored and managed in a repository to keep the representations of the continuously evolving states. A data mining technique can be helpful to understand and analyse dynamic databases of an evolving system.

A database of entities (or items) can be mined using *Association Rule Mining*, which identifies the association rules (A$\rightarrow$B co-occurrence of entity sets A and B) with *support* and *confidence*. The sequence of entities in a database can be mined using *Sequential Rule Mining* (SRM), which generates *sequential rules* with support and confidence. The entities are unordered in association rules and ordered in sequential rules. There are other such rules e.g. prediction rules and temporal rules.

As the system evolves, its database also evolves. As the database evolves, its rules evolve as well. These rules can be used to study the system evolution. Rule mining and evolution mining can discover rules and evolution information of entities, respectively. Evolution and change analysis on a dataset over time retrieves the evolution information. We present evolution rule mining to retrieve evolution rule information in the multiple states of an evolving system. This uncovers interesting and actionable information for an evolving system.

This chapter is inspired by recent advancements in the study of system evolution based on graph evolution mining. Based on fundamentals of well-known association rule mining, we aim to create an intelligent algorithm and tool, which helps to study the system evolution and changeability. Evolving systems has properties like changeability, evolvability, and stability. We aim to compute the stability and changeability of an evolving system. Given a state series SS = $\{S_1, S_2 \ldots S_N\}$ of an evolving system, we are interested in mining rules of the form X$\rightarrow$Y, called Network Evolution Rules (NERs), with the following characteristics:

(a) the rule is “interesting”, in the sense that if the source entities in set X exist in a state, then target entities in set Y will also present in that state.

(b) the rule is “stable”, in the sense that its “stability” exceeds a given threshold, where “stability” is defined to be the fraction of states in SS in which the rule is interesting.

The first parameter (“interestingness”) characterizes the behaviour of the NER within an individual state S_i of SS, whereas the second parameter (“stability”) characterizes the behaviour of the whole set of states SS. The advantage of “stability” is to retrieve Stable NERs (SNERs) that remains stable (or persistent) in sufficient number of states over time.

We are also motivated to compute the system’s changeability metric for system evolution analysis. For a given set of states SS of an evolving system, our approach mines two kind of rules: Network Evolution Rules (NERs) and Stable NERs (SNERs) in a state series. We use NERs and SNERs counts to compute the changeability for an evolving system. The proposed approach is useful for an evolving system whose states can be represented as *system network databases*. For a real-world evolving system, the challenge is to construct and mine a set of dynamic databases over the multiple states.

The main contributions of the chapter are as follows. We introduce stability as a new measure for characterizing NERs over time, which helps to retrieve SNERs. We introduce novel formulas to compute system Stability Metric (SM) and Changeability Metric (CM), which aids in quantitative system analysis. For this, we propose a new algorithm for System Network Stability (SNS), which uses SNERs Mining algorithm. We demonstrate experimental analysis by applying our intelligent approach for six real-world evolving systems. Rest of the chapter is organized as following:

In Section 4.2, we present novel definitions concerning evolution rules and stable rules, whose types are NERs and SNERs. This is followed by another central notion of our work, namely the *stability metric* and *changeability metric* to perform system evolution analysis. This is illustrated as an example in Section 4.3. In Section 4.4, we proposed an algorithm System Network Stability (SNS) and its computation. In Section 4.5, we describe our prototype 'SNS-Tool'. In Section 4.6, we use our SNS-Tool to demonstrate experimental results on six real-world evolving systems. These contributory sections are followed by conclusions in Section 4.7.

4.2 Proposed Definitions and Concepts

In this section, we present formal definitions and notations. This includes our key definitions of Evolution rule, Stable rule, Connection pairs, System Network Database, Network rule, Network Evolution Rule (NER) and Stable Network Evolution Rule (SNER) with support, confidence, and stability. At last, we formulate system's Stability Metric (SM) and Changeability Metric (CM).

4.2.1 Evolution Rule and Stable Rule

We start with the definitions of *Evolution rule* and *Stable rule* of the form $X \rightarrow Y$, where X is an antecedent set, and Y is a consequent set for the co-occurrence of the two sets (X and Y) of entities. The symbol $\rightarrow$ means co-occurrence such that if X occurs then Y will also occur. Formal definitions are as follows.

Definition 4.1: Given a rule ($X \rightarrow Y$) in database D_i for state S_i and a database series DS = {D_1, $D_2 \ldots D_N$} for a state series SS. The rule is *interesting* in S_i, if its support and confidence exceeds given thresholds.

- A distinct rule occurring in multiple states is said to be an evolution rule that has some stability in the state series, where stability is the fraction of states in which it is interesting.

- An evolution rule is stable if its stability exceeds a given threshold named minimum stability (minStab). The stable evolution rule is said to be *stable rule*.

The *interestingness* characterizes the association rules for a database D_i of state S_i, whereas the *stability* characterizes the *evolution rules* and *stable rules* of the database series DS of SS. The interestingness and stability is the probabilistic measure to form the rule ($X \rightarrow Y$) in a given database. The stability is a new measure for characterizing evolution rules over time and constitutes one of the main conceptual contributions of this chapter. The functional meaning of stability is the measure of evolution happened in the database series. Stable rules provide set of persistently co-occurring entities over a database series.

4.2.2 Connection pair and System network database

As mentioned earlier, an evolving system contains distinct interconnected entities in an evolving system network. If entity e has connecting direction toward entity e', then we call e the *source* and e' the *target* of the connection. A convenient way for representation is directed graph to represent connections between entities in a given state S_i. We call this graph, *system network* of S_i such that nodes represent entities and edges represent connections ($e \rightarrow e'$ if and only if source entity e is connected to target entity e').

Given a state S_i of an evolving system, we shall denote the set of source entities by sS_i and the set of target entities by tS_i. Let SS = $\{S_1, S_2 \ldots S_N\}$ be a set of states of an evolving system. Let sSS be the set of all source entities in SS, that is, $sSS = sS_1 \cup sS_2 \cup \ldots \cup sS_N$; and let $tSS = tS_1 \cup tS_2 \cup \ldots \cup tS_N$ be the set of all target entities in SS. Note that sSS and tSS are not necessarily disjoint, which means an entity can be a source and a target at the same time.

To define the support and confidence of a network rule in a state S_i, we first define the concept of “connection pair” in S_i. Assume a system state series is pre-processed to make a set of evolving networks represented as EN = $\{EN_1, EN_2 \ldots EN_N\}$, such that evolving network EN_i represents state S_i. Each evolving network contains many connections between entities. For such evolving networks, we explain a concept of *connection pair*. A *connection pair* in S_i is an ordered pair of two subsets: source entities L followed by their target entities R. A source entity e in source set L has a target entity e' in target set R. The e and e' forms a directed connection from e to e' in the network of state S_i. A connection pair is denoted as (L, R), where the symbols L and R as mnemonics for Left and Right, respectively.

Definition 4.2: A <u>connection pair</u> in S_i is an ordered pair (L, R) such that L is a subset of source entities and R is a subset of target entities. For each e in L there exit e' in R that makes a connection $e \rightarrow e'$ in the network of state S_i. Where, the symbols L and R stand for “left” and “right”. Thus, we shall say that a set $X \cup Y$ of entities occurs in a connection pair (L, R) if $X \cup Y$ is a subset of $L \cup R$.

A collection of *connection pairs* makes a *system network database* as required input for Network Rule Mining.

Definition 4.3: A <u>*system network database*</u> (SysNetDb) is made-up of a set of *connection pairs* $\{CP_1, CP_2 \ldots CP_M\}$, where M is the total number of connection pairs. The SysNetDb of state S_i is denoted as *SysNetDb_i*. Dynamic databases for a state series is denoted as

SysNetDbs = {SysNetDb_1, SysNetDb_2… SysNetDb_N}.

The connection pairs of a state create a *System Network Database* (*SysNetDb*). A collection of system network databases for multiple states is referred as *SysNetDb**<u>s</u>***.

Assume X is a subset of source entities in L and Y is a subset of target entities in R. During pre-processing, each connection pair (L, R) is transformed into a numerical sequence based on the following two steps:

- The set of all entities appearing in state S_i are encoded as positive integers, that is, each entity is associated with a unique positive integer.
- For each connection pair (L, R), we replace the entities in L and in R with the sets of their encodings. The encodings of all connection pairs are stored in a database, which we represent them as SysNetDb_i.

4.2.3 Network Rule, Network Evolution Rule, and Stable Network Evolution Rule

Now, given a set of states SS, we are interested in the count of network rules that occurs in one or more states. Additionally, we consider three kinds of rules the Network Rule, the Network Evolution Rule (NER), and the Stable NER. All of them have the form $X \rightarrow Y$, but they have different characteristics. A network rule has a characteristic of interestingness, which measures support and confidence with respect to a given state. A NER and a Stable NER have an additional characteristic of stability, which is defined with respect to a given state series.

Definition 4.4: A <u>*Network Rule*</u> (NR) in state S_i is an expression of the form $X \rightarrow Y$, where X is a subset of source entities sS_i and Y is a subset of target entities tS_i.

A *network rule* can be interpreted as "if source entity (or entities) occurs in set X, then target entity (or entities) in Y are likely to occur with a given support and confidence in a connection pair". The support and confidence of a network rule in a state S_i is computed with respect to a given set of connection pairs. Such a set is defined with respect to some aspect of the evolving system. However, we shall assume that a set of connection pairs is given. Different kind of systems has different mechanism to generate their system network. Thus, mechanism to collect connection pairs depends upon pre-processing of system network. For example, if an evolving system is a software system then it is usually structured as modules and each module contains a number of procedures (as entities). In each module, each procedure calls zero, one, or more procedures in the same module or in different modules. Therefore, each module determines a connection pair between its own set of procedures, say L, and the set of target procedures, say R.

Definition 4.5: Given a set of connection pairs in S_i, let $X \rightarrow Y$ be a network rule and let $X \cup Y$ be a set of entities in $X \cup Y$.

- The *support count* of $X \cup Y$ in S_i is denoted as supCount($X \cup Y$, S_i), and defined by supCount($X \cup Y$, S_i) = m, where m is the number of connection pairs in which $X \cup Y$ occurs.

- The *support* of $X \cup Y$ in S_i is denoted as sup($X \cup Y$, S_i), and defined by sup($X \cup Y$, S_i) = supCount($X \cup Y$, S_i) ÷ card(S_i), where card(S_i) is the cardinality (number) of connection pairs in state S_i.

- The *support* of $X \rightarrow Y$ in S_i is denoted as sup($X \rightarrow Y$, S_i), and defined by: sup($X \rightarrow Y$, S_i) = sup($X \cup Y$, S_i) = sup($X \cup Y$, S_i).

- The *confidence* of $X \rightarrow Y$ in S_i is denoted as conf($X \rightarrow Y$, S_i), and defined by: conf($X \rightarrow Y$, S_i) = sup($X \cup Y$, S_i) ÷ sup(X, S_i)

- An *interesting network rule* has support (or support count) and confidence greater than thresholds minSup (or minSupCount) and minConf, respectively.

Definition 4.6: A *Network Evolution Rule* (NER) in SS is an expression of the form $X \rightarrow Y$, where X is a subset of source entities sSS and Y is a subset of target entities tSS. Each NER has a characteristic *stability* because it is distinct and interesting in one or more states.

- The *stability count* of $X \rightarrow Y$ in SS is denoted as stabCount($X \rightarrow Y$, SS), and defined by stabCount($X \rightarrow Y$, SS) = n, where n is the number of states in SS that has $X \rightarrow Y$ as interesting NER.

- The *stability* of $X \rightarrow Y$ in SS is denoted as stab($X \rightarrow Y$, SS), and defined by stab($X \rightarrow Y$, SS) = stabCount($X \rightarrow Y$, SS) ÷ card(SS), where card(SS) is the cardinality of SS i.e. number of states N in SS.

- The NER $X \rightarrow Y$ is defined to be *stable* in SS if its stability (or stability count) is greater minStab (or minStabCount); such NERs are *Stable Network Evolution Rules* (SNERs).

The stability is a new measure for characterizing NERs over time. The stability of a NER is our conceptual contribution. The stability count of a NER (stabCount) is the frequency of states in which the NER is interesting. Consequently, *stability of NER* is the fraction of states in which the NER is interesting. The SNER is a new information for system network evolution over time and constitute another conceptual contribution. The functional meaning of the discovered NER is frequent network rule of interconnected entities occurring in a SysNetDb of a state. The functional meaning of a SNER is the frequent NER of inter-connected entities occurring in SysNetDbs for a set of states.

Intuitively, network rule $X \rightarrow Y$ in a state means, "presence of the source entities (X) implies presence of the target entities (Y) in a sufficiently high fraction of connection pairs (in a SysNetDb)". In a network rule, the *antecedent* X contains frequent source entities co-occurring with the *consequent* Y containing frequent target entities.

Intuitively, stable NERs are the rules of co-occurring persistent (stable) entities over a set of graphs. The collection of network rules (from multiple state) makes NERs. A NER is stable in a state series, and referred as SNER, if it exceeds the three expert specified thresholds minSup-minConf-minStab. The minSup and minConf characterizes the behaviour of network rules in a state S_i, and the minStab characterizes the behaviour of NERs over a state series SS.

4.2.4 Stability metric and Changeability metric

We can now define another central concept of our work, namely *stability metric*. The intuitive significance of the stability metric is to measure the speed of evolution, which helps to do SysEvo-Analytics using threshold minStab, count of evolution rules, and count of stable rules. In a state series SS, we can measure the speed of evolution in an evolving system based on two obvious facts.

Fact 1: if there is high minStab i.e. fraction of minStabCount over the number of states, then the system has evolved slowly (i.e. undergone few changes). Inversely, if there is a low minStab i.e. fraction of minStabCount over the number of states, then the system has evolved fast (i.e. undergone numerous changes). This implies *system stability is directly proportional to the minStab i.e. fraction of minStabCount over the number of states.* This makes equation

$$\text{System Stability} \propto minStab = \frac{\text{minStabCount}}{\text{N}} \quad \ldots(1).$$

Fact 2: if there are many *stable rule*s, then it reflects slow evolution among states. Inversely, if there are few *stable rule*s, then it reflects fast evolution among states. This implies *system stability is directly proportional to the stable rule count and inversely proportional to the distinct evolution rule count*. This makes equation

$$\text{System Stability} \propto \frac{\text{SR\_Count}}{\text{ER\_Count}} \quad \ldots(2)$$

where, SR_Count stands for the StableRule_Count (i.e. the number of stable rules) and ER_Count stands for the EvolutionRule_Count (i.e. the number of evolution rules). These two facts (i.e., eq. (1) and (2)) are combined to define stability metric that characterizes system evolution.

Definition 4: Given a state series SS = $\{S_1, S_2 \ldots S_N\}$ of an evolving system with its evolution and stable rules, then the <u>*stability metric*</u> of the system is defined as

$$\text{Stability metric} = \text{minStab} * \frac{\text{SR\_Count}}{\text{ER\_Count}} * 100 \quad \ldots(3).$$

This equation (3) is a generalized formula for evolution and stable rule count. Now, we present formula in context of system network, named as <u>*System Network Stability metric*</u> (SNS metric) and defined as

$$\text{SNS metric} = \text{minStab} * \frac{\text{SNER\_Count}}{\text{NER\_Count}} * 100 \quad \ldots(4)$$

where, SNER_Count stands for the stable NER count, (i.e. the number of stable NERs) and NER_Count stands for the NER count (i.e. the number of NERs).

Two important remarks regarding the stability metric:

1) On one hand, low values of minStab and SR_count produces low value of the stability metric, which means that few stable rules occurring in less number of states. On the other hand, high values of minStab and SR_count produces high value of *stability metric*, which means that many stable rules occurring in many states. Low value reflects fast evolution and high value reflects slow evolution in the given state series. The low value of *stability metric* does not reflect unstable system. However, the lower bound of stability metric is 0 for a volatile system (that has no common rule between any two states). Conversely, the upper bound of stability metric is 100 for a constant system (that does not changed with time and has all states same). Both lower and upper bound are ideal conditions, which rarely occur with a normal system. In real-world system, stability metric value varies due to three reasons: the system domain, the fraction of (minStabCount ÷ N), and the fraction of (SR_count ÷ ER_count).

2) We can use the *stability metric* to know about the stability of a new state. Suppose the *stability metric* is SM_N for N states. If we add a new state to the system, then new *stability metric* is SM_{N+1} for N+1 states. Following three conditions are possible $SM_{N+1} > SM_N$ or $SM_{N+1} < SM_N$ or $SM_{N+1} = SM_N$. If $SM_{N+1} > SM_N$, then changes are significantly done to construct a new state. If $SM_{N+1} < SM_N$, then the new state is not changed much as compared to the old states. If $SM_{N+1} \cong SM_N$, then the new state is similar to the old states.

Next, we use the numerical values of input-output parameters in the SNER mining to do the system changeability analysis. The changeability is a system property, which depends upon both evolution and stability information of an evolving system. To compute the changeability (as another conceptual contribution) for an execution of SNER mining, we defined following metric.

Definition 4.8: A <u>*Changeability Metric*</u> (CM) in a set of states SS of an evolving system is defined as

$$\text{CM} = \frac{\text{N}}{\text{minStabCount}} * \frac{\text{NER\_Count}}{\text{SNER\_Count}} * 100 \quad \ldots(5)$$

where the N is the number of states and the minStabCount is the threshold value used for SNER mining. Where, the NER_Count is the number of NERs retrieved and the SNER_Count is the number of SNERs retrieved. Note, the value of (N/minStabCount) is equal to (1/minStab) i.e. inverse of stability.

4.2.5 Overview of Approach

The advantage of the new measure for *stability of NERs* is to compute *stability* and *changeability* property of an evolving system. Figure 4.1 describes the summary of all the steps to intelligently compute the SNER and the Changeability metric. These steps are according to the guideline steps of Knowledge Discovery in Databases (KDD).

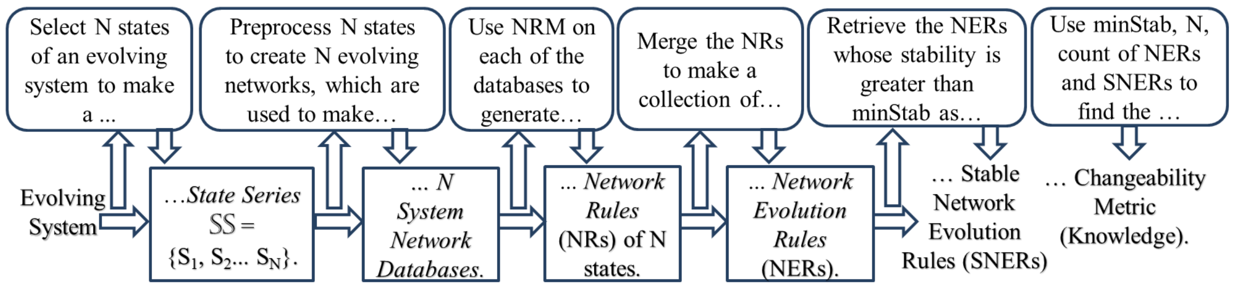

Select N states of an evolving system to make a ...
Preprocess N states to create N evolving networks, which are used to make…
Use NRM on each of the databases to generate…
Merge the NRs to make a collection of…
Retrieve the NERs whose stability is greater than minStab as…
Use minStab, N, count of NERs and SNERs to find the …
Evolving System
…State Series SS = {S1, S2... SN}.
... N System Network Databases.
... Network Rules (NRs) of N states.
... Network Evolution Rules (NERs).
… Stable Network Evolution Rules (SNERs)
… Changeability Metric (Knowledge).

Figure 4.1 The process-artifacts involved to intelligently compute the SNERs and the Changeability Metric. Where, each rounded-rectangle shows a process and rectangle shows input-output artifact as per arrows. Each sentence starts from process at top and finishes at the artifact box.

4.3 Illustrative Examples

This section discusses an illustration for pre-processing and Network Rule Mining (NRM) of a state, followed by NERs and stable NERs retrieval for a state series. We also calculate Stability and Changeability metrics for the illustrative example. We present this in the following four sub-sections.

4.3.1 Pre-processing of a State

This sub-section describes pre-processing of a state to make a dynamic database: *System Network Database* (SysNetDb). Firstly, pre-process a system state series to make a series of evolving networks, where a node represents an entity and an edge represents directed entity connection. The pre-processing of state S_i depends upon the type of system i.e. different systems have different techniques to create evolving networks.

Secondly, use the evolving networks to gather a set of connection pairs with respect to a *connection relationship* between entities of an evolving system. The connection relationship decides the connection pairs to make a set of SysNetDbs. For example, sequence of words in a sentence decides a connection pair of words (as entities). Thus, formation of SysNetDb depends upon connection relationship chosen to form connection pairs. The experimental section describes mechanism to gather connection pairs using connection relationship for making SysNetDbs. The following sections assume that SysNetDbs are given for mining.

Assume E = {e_1, e_2, e_3 … e_K} be a set of entities, where K is total number of entities and let *SysNetDb_i_ID* = {CP_1, CP_2… CP_M} be a set of connection pairs. In Figure 4.2, we assume a connection relationship providing three connection pair from the network of S_i, thus K= 5 and M = 3. The figure shows a general model for system pre-processing to make *SysNetDb_i* of S_i in the form of *entityName*. Each *entityName* based connection pair (L, R) is transformed to *entityID* based connection pair. The process of encoding and storing all connection pairs to make a *SysNetDb_i_ID* is as follows.

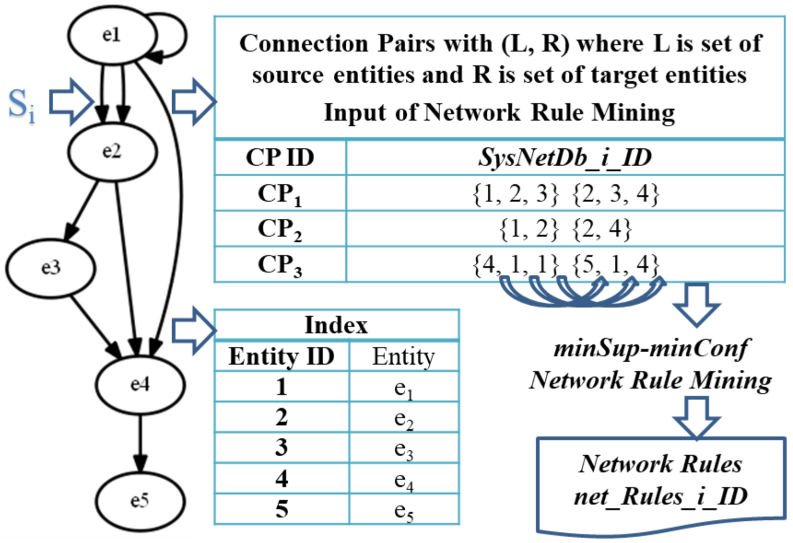
A) The set of all entity names appearing in S_i are encoded as entityID (a positive integers i(e)). Each entity e is associated with a unique positive integer.

B) In each connection pair (L, R) in *SysNetDb_i*, the entities in L and in R are replaced with the sets of their encodings. This creates numerical SysNetDb, which is denoted by *SysNetDb_i_ID*.

In the next two subsections, we illustrate mining of Network Rule, Network Evolution Rule (NER), and Stable NER (SNER) with support, confidence, and stability.

4.3.2 Network Rule Mining of a State

This sub-section describes use of a System Network Database (SysNetDb) for NRM. The NRM takes three inputs (SysNetDb, minSupCount, and minConf) to generate network rules. Figure 4.2 shows an overview of NRM, first pre-process a state S_i to make *SysNetDb_i_ID* that is used by NRM along with minSupCount and minConf to generate network rules in the form of ID (*net_Rules_i_ID*).

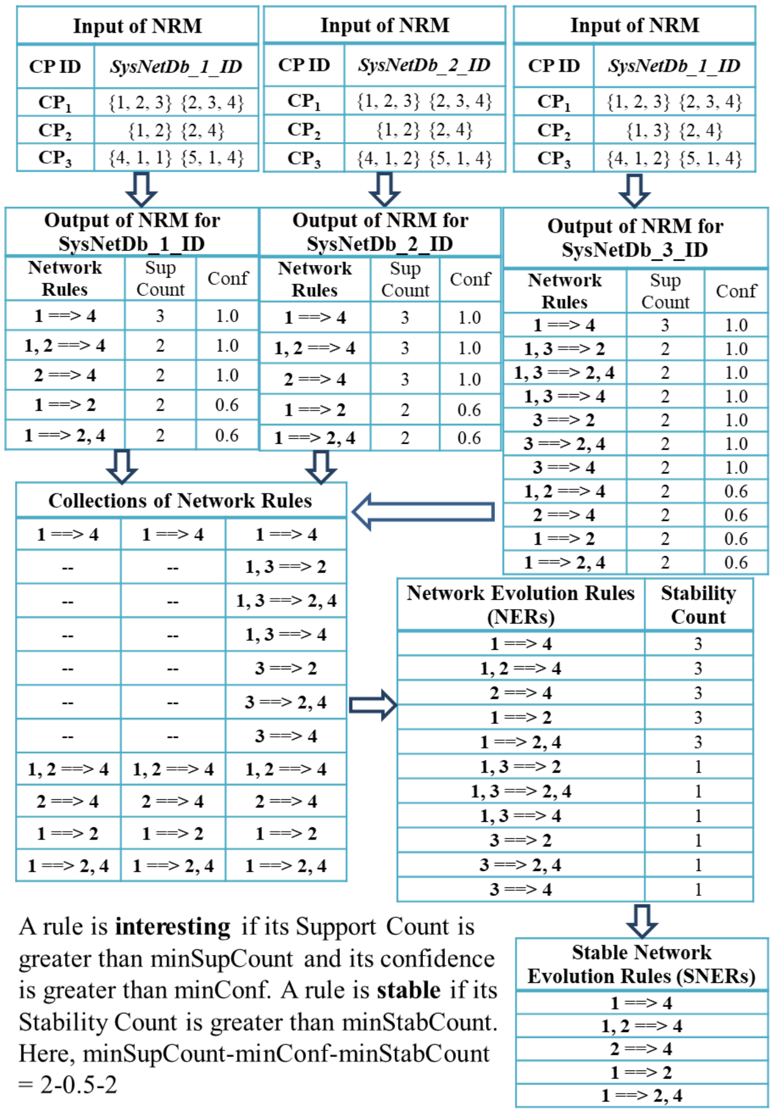


S_i
e1
e2
e3
e4
e5
Connection Pairs with (L, R) where L is set of source entities and R is set of target entities
Input of Network Rule Mining
CP ID
SysNetDb_i_ID
CP_1 {1, 2, 3} {2, 3, 4}
CP_2 {1, 2} {2, 4}
CP_3 {4, 1, 1} {5, 1, 4}
Index
Entity ID Entity
1 e_1
2 e_2
3 e_3
4 e_4
5 e_5
minSup-minConf Network Rule Mining
Network Rules net_Rules_i_ID

Figure 4.2 An overview of pre-processing on a state S_i to make SysNetDb_i_ID that contains three connection pairs. This is followed by an overview of Network Rule Mining step on the SysNetDb_i_ID.

Search space of NRM is SysNetDb of a state that contains connection pairs for entity connection relationship. The NRM computes the support and confidence of a network rule $X \rightarrow Y$ in a state S_i according to the given connection pairs in *SysNetDb_i_ID*. To do NRM, scan the SysNetDb of state S_i to calculate frequency of each entity and then identify all entities with frequencies greater than minSupCount. Then use the frequent entities to generate ordered network rules $X \rightarrow Y$. Also, calculate the support (frequency) and confidence (conditional probability) for co-occurrence of antecedent X and consequent Y. The output of NRM is network rules with support and confidence higher than or equal to user-defined threshold minSupCount and minConf.

A connection pair (in SysNetDb) has two ordered entity-sets, which resembles it with sequence (in sequence database used for sequential rule mining). Thus, SysNetDb becomes a kind of sequence database, which converges NRM to the sequential rule mining that can generate sequential rules as network rules. Hence, the NRM is a special case of sequential rule mining because the input sequence database (SysNetDb) and the resulting sequential (or network) rules contain ordered set of source and target entities.



Input of NRM	
CP ID	*SysNetDb_1_ID*
CP_1	{1, 2, 3} {2, 3, 4}
CP_2	{1, 2} {2, 4}
CP_3	{4, 1, 1} {5, 1, 4}

Input of NRM	
CP ID	*SysNetDb_2_ID*
CP_1	{1, 2, 3} {2, 3, 4}
CP_2	{1, 2} {2, 4}
CP_3	{4, 1, 2} {5, 1, 4}

Input of NRM	
CP ID	*SysNetDb_1_ID*
CP_1	{1, 2, 3} {2, 3, 4}
CP_2	{1, 3} {2, 4}
CP_3	{4, 1, 2} {5, 1, 4}

Output of NRM for SysNetDb_1_ID		
Network Rules	Sup Count	Conf
1 ==> 4	3	1.0
1, 2 ==> 4	2	1.0
2 ==> 4	2	1.0
1 ==> 2	2	0.6
1 ==> 2, 4	2	0.6

Output of NRM for SysNetDb_2_ID		
Network Rules	Sup Count	Conf
1 ==> 4	3	1.0
1, 2 ==> 4	3	1.0
2 ==> 4	3	1.0
1 ==> 2	2	0.6
1 ==> 2, 4	2	0.6

Output of NRM for SysNetDb_3_ID		
Network Rules	Sup Count	Conf
1 ==> 4	3	1.0
1, 3 ==> 2	2	1.0
1, 3 ==> 2, 4	2	1.0
1, 3 ==> 4	2	1.0
3 ==> 2	2	1.0
3 ==> 2, 4	2	1.0
3 ==> 4	2	1.0
1, 2 ==> 4	2	0.6
2 ==> 4	2	0.6
1 ==> 2	2	0.6
1 ==> 2, 4	2	0.6

Collections of Network Rules		
1 ==> 4	1 ==> 4	1 ==> 4
--	--	1, 3 ==> 2
--	--	1, 3 ==> 2, 4
--	--	1, 3 ==> 4
--	--	3 ==> 2
--	--	3 ==> 2, 4
--	--	3 ==> 4
1, 2 ==> 4	1, 2 ==> 4	1, 2 ==> 4
2 ==> 4	2 ==> 4	2 ==> 4
1 ==> 2	1 ==> 2	1 ==> 2
1 ==> 2, 4	1 ==> 2, 4	1 ==> 2, 4

Network Evolution Rules (NERs)	Stability Count
1 ==> 4	3
1, 2 ==> 4	3
2 ==> 4	3
1 ==> 2	3
1 ==> 2, 4	3
1, 3 ==> 2	1
1, 3 ==> 2, 4	1
1, 3 ==> 4	1
3 ==> 2	1
3 ==> 2, 4	1
3 ==> 4	1

Stable Network Evolution Rules (SNERs)
1 ==> 4
1, 2 ==> 4
2 ==> 4
1 ==> 2
1 ==> 2, 4

A rule is **interesting** if its Support Count is greater than minSupCount and its confidence is greater than minConf. A rule is **stable** if its Stability Count is greater than minStabCount. Here, minSupCount-minConf-minStabCount = 2-0.5-2

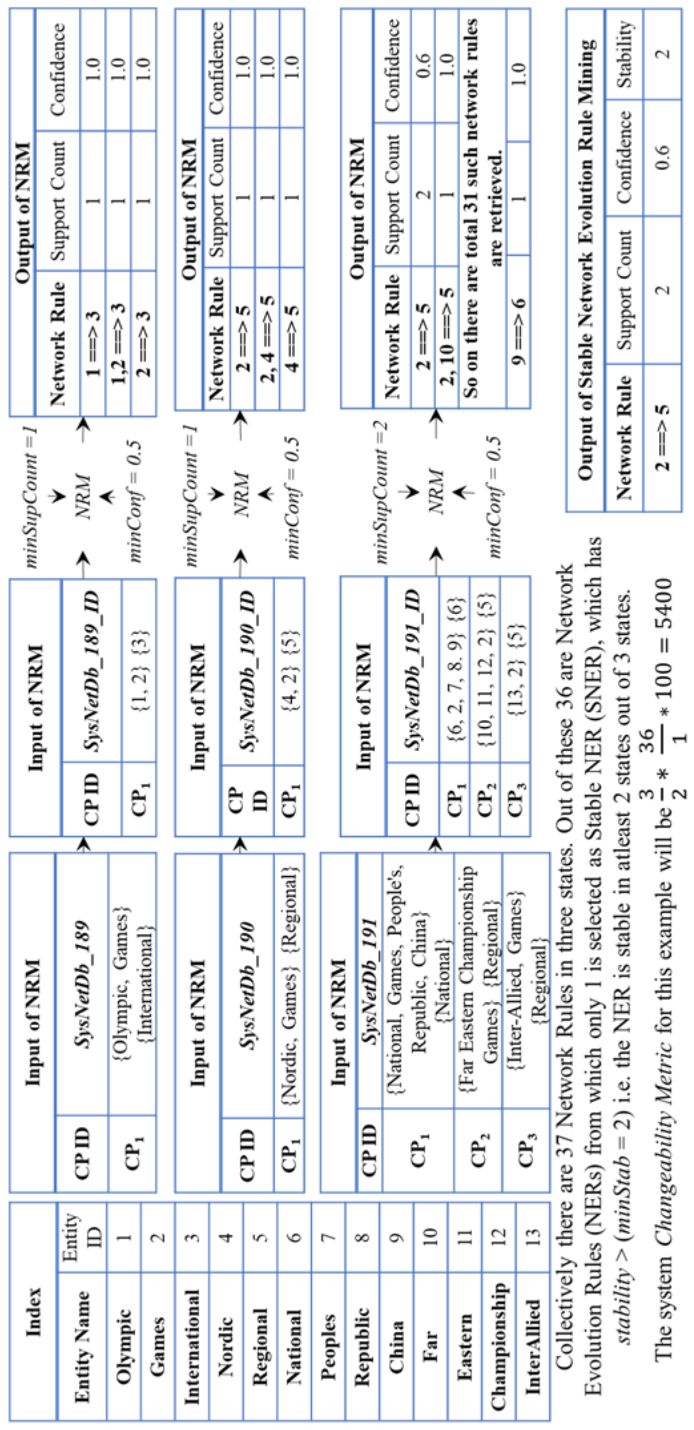
Figure 4.3 Collection of network rules to make Network Evolution Rules (NERs) from which Stable NERs (SNERs) are retrieved.

4.3.3 *Stable Network Evolution Rules of State series*

After pre-processing, we applied NRM over a state series to retrieve network rules of all the states, and then we identified NERs and SNERs. Figure 4.3 continues the example given in Figure 4.2 by assuming that *SysNetDb_i_ID* as *SysNetDb_1_ID*. Here, number of states (N) is three. In Figure 4.3, the interesting network rules are generated according to minSupCount = 2 and minConf = 0.5. Thereafter, we merged the network rules of the three states to create collection. From the collection, we identify distinct NERs with their stabilities. Then, we retrieved the SNERs from NERs whose Stability Count is greater than minStabCount = 2. For the illustration in Figure 4.3, the SNS metric as given in equation (4) is $\{(2 \div 3) \times (5 \div 11)\} \times 100 = 30.30$.

4.3.4 *Real-world Example*

An example for the list of multi-sport events database is presented in Figure 4.4, which has one connection pair in first two SysNetDbs and 3 connection pairs in third SysNetDb. By pre-processing, we eliminated stop-words and kept only keywords (as entities).



Index	
Entity Name	Entity ID
Olympic	1
Games	2
International	3
Nordic	4
Regional	5
National	6
Peoples	7
Republic	8
China	9
Far	10
Eastern	11
Championship	12
InterAllied	13

Input of NRM	
CP ID	***SysNetDb_189***
CP_1	{Olympic, Games} {International}

Input of NRM	
CP ID	***SysNetDb_189_ID***
CP_1	{1, 2} {3}

Output of NRM		
Network Rule	Support Count	Confidence
1 ==> 3	1	1.0
1,2 ==> 3	1	1.0
2 ==> 3	1	1.0

Input of NRM	
CP ID	***SysNetDb_190***
CP_1	{Nordic, Games} {Regional}

Input of NRM	
CP ID	***SysNetDb_190_ID***
CP_1	{4, 2} {5}

Output of NRM		
Network Rule	Support Count	Confidence
2 ==> 5	1	1.0
2, 4 ==> 5	1	1.0
4 ==> 5	1	1.0

Input of NRM	
CP ID	***SysNetDb_191***
CP_1	{National, Games, People's, Republic, China} {National}
CP_2	{Far Eastern Championship Games} {Regional}
CP_3	{Inter-Allied, Games} {Regional}

Input of NRM	
CP ID	***SysNetDb_191_ID***
CP_1	{6, 2, 7, 8. 9} {6}
CP_2	{10, 11, 12, 2} {5}
CP_3	{13, 2} {5}

Output of NRM		
Network Rule	Support Count	Confidence
2 ==> 5	2	0.6
2, 10 ==> 5	1	1.0
So on there are total 31 such network rules are retrieved.		
9 ==> 6	1	1.0

Output of Stable Network Evolution Rule Mining			
Network Rule	Support Count	Confidence	Stability
2 ==> 5	2	0.6	2

Collectively there are 37 Network Rules in three states. Out of these 36 are Network Evolution Rules (NERs) from which only 1 is selected as Stable NER (SNER), which has *stability* > (*minStab* = 2) i.e. the NER is stable in atleast 2 states out of 3 states.

The system *Changeability Metric* for this example will be $\frac{3}{2} * \frac{36}{1} * 100 = 5400$

Figure 4.4 An illustrative example to mine NERs, SNERs, and Changeability metric.

For example, the Index in Figure 4.4 has 13 entities, which are encoded to transform the SysNetDb. For example, in Figure 4.4 the SysNetDb_189, SysNetDb_190, and SysNetDb_191 denotes SysNetDbs for decades (as states) 1890, 1900, and 1910 respectively. Figure 4.4 also shows an illustration for Network Rule Mining (NRM). Use “input of NRM” along with threshold minSupCount

= 1 and minConf = 0.5 to generate "output of NRM" containing network rules. The support count is convertible to the support fraction by dividing supCount with the number of connection pairs (i.e. 3). Thus, minSup is equal to minSupCount (= 1) divided by cardinality (= 3) for SysNetDb_191, which is equal to 1/3. In Figure 4.4, the network rule in "output of NRM" is in the order of their interestingness i.e. top most rule (in first row) is highest interesting, rules in middle row are moderately interesting in decreasing order, and last rule is least interesting. In Figure 4.4, the SNER mining resulted in 36 NER and 1 SNER, whose execution time was 1 second. This resulted in 5400 as CM value. Next section presents an algorithm for mining SNERs from a given set of states SS.

4.4 System Network Stability

Thus far, we presented novel definitions, concepts, and an illustrative example. This section presents proposed algorithms *System Network Stability* (SNS), a theory to analyse stability of pre-evolved system. Search space of SNS is an evolving system that can be represented as a set of *evolving networks* EN = {EN_1, EN_2… EN_N} for a *state series* SS = {S_1, S_2 … S_N} at time points {t_1, t_2, t_3… t_N}. The *system network database* series is SysNetDbs = {SysNetDb_1, SysNetDb_2… SysNetDb_N}, whose i^{th} database is represented as SysNetDb_i for a state S_i. We are interested to find SNERs having "the presence of source entities (X) implies the presence of target entities (Y) in a sufficiently high fraction of states". Next, we introduce a new SNS algorithm, which has following four steps (also manifested in Figure 4.5):

1. Pre-process N states stored in a repository to make N *SysNetDbs* and an *IndexFile* with <*entityName*, *entityID*>.
2. Mining NERs and SNERs use four inputs (SysNetDbs, minSupCount, minConf, and minStab) to retrieve the *NERs_ID* and *SNERs_ID*.
3. The *NERs_ID* and *SNERs_ID* are in the form of *entityIDs*. Use the *IndexFile* to replace each *entityID* with its *entityName* in each NER and SNER. This converts *NERs_ID* into *NERs_Name* and *SNERs_ID* into *SNERs_Name*.
4. Calculate stability (SNS) metric given in Equation (4) using the minStab, number of SNERs retrieved (SNER_Count), and number of NERs retrieved (NER_Count).

```
Algorithm SNS(repository)
Retrieve N system states and store them in a repository.
1.SysNetDbs & IndexFile = Pre-process(repository)
2.NERs_ID & SNERs_ID = Mining_NERs_SNERs(SysNetDbs, minSupCount, minConf, minStab)
3.NERs_Name & SNERs_Name = Indexing(NERs_ID, SNERs_ID,  IndexFile)
4.SNS_metric = stabMetric(minStab, SNER_Count, NER_Count)
```

The following four sub-sections elaborate these steps.

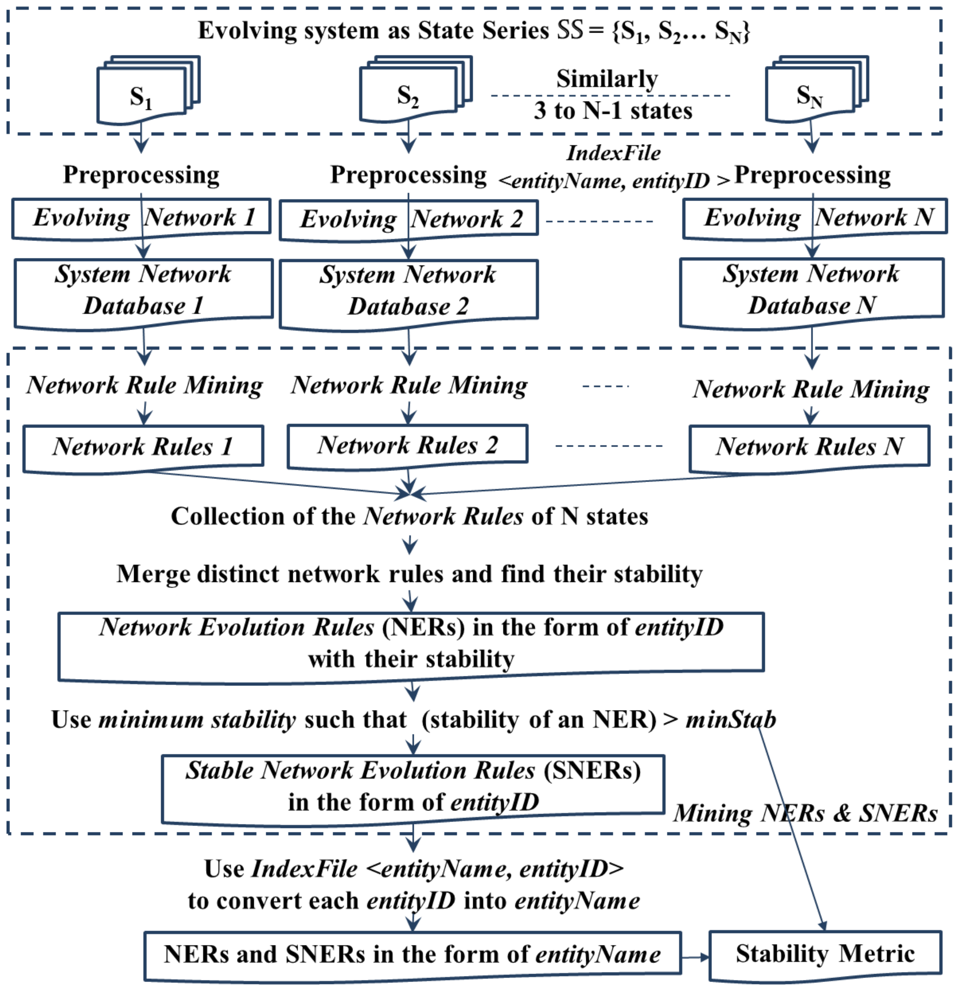

Evolving system as State Series SS = {S1, S2… SN}
S1
S2
Similarly 3 to N-1 states
SN
IndexFile <entityName, entityID >
Preprocessing
Evolving Network 1
Evolving Network 2
Evolving Network N
System Network Database 1
System Network Database 2
System Network Database N
Network Rule Mining
Network Rules 1
Network Rules 2
Network Rules N
Collection of the Network Rules of N states
Merge distinct network rules and find their stability
Network Evolution Rules (NERs) in the form of entityID with their stability
Use minimum stability such that (stability of an NER) > minStab
Stable Network Evolution Rules (SNERs) in the form of entityID
Mining NERs & SNERs
Use IndexFile <entityName, entityID> to convert each entityID into entityName
NERs and SNERs in the form of entityName
Stability Metric

Figure 4.5 The System Network Stability, a theory for N states in a state series of an evolving system.

4.4.1 Pre-processing of a state series

The Algorithm 4.1 Pre-process takes N states of an evolving system to make N *System Network Databases* (SysNetDbs) for a connection relationship. A SysNetDb of state S_i is *SysNetDb_i* that contains connection pairs of entities in the form of *entityName*. Then, transform each *SysNetDb_i* to make *SysNetDb_i_ID* in the form of *entityIDs*. An entity occurring in several states must have the same *entityID*. Thus, an *entityID* keeps track for multiple occurrences of an entity in a state series. Simultaneously, an *IndexFile* is also created, which has a list with two entries <*entityName*, *entityID*> for each entity. Output of pre-processing is N SysNetDbs & an *IndexFile*. The N SysNetDbs is a set of dynamic databases such that each SysNetDb follows the semantics, syntax, and structure for input required in NRM. For reproducibility, we present following pseudo code for pre-processing of a state series.

Algorithm 4.1 **Pre-process(***repository***)**

```
Initialize HashMap Index< entityName, entityID >
Initialize integer counter = 1
Initialize String Buffer buffer
For each state $S_i$ where i ∈ integer and i varies from 1 to N
```

```
    Depending on type of repository, extract relationship between set of inter-connected entities to
    create a SysNetDb_i for a state $S_i$
    Read SysNetDb_i and store it in buffer until end of file
    For each line of buffer, scan entityName
        If an entityName is in the Index
            In buffer, replace the entityName with its entityID
        Else
            entityID = counter
            Add the new tuple < entityName, entityID > in the Index
            In buffer, replace the entityName with its entityID
            Increment the counter by 1 i.e. counter = counter + 1
    End of Scan when end of buffer is reached
Write the buffer in SysNetDb_i_ID
Store the file SysNetDb_i_ID in directory SysNetDbs
End For when all the states are pre-processed
Make an IndexFile and store Index<entityName, entityID>.
Return SysNetDbs & IndexFile
```

4.4.2 Mining NERs and SNERs

The Algorithm 4.2 Mining_NERs_SNERs takes four inputs: SysNetDbs, minSupCount, minConf, and minStab. The algorithm has minStab a threshold for measuring stability of a NER over a state series (a set of states ordered by time).

The algorithm executes the NRM to processes SysNetDbs (N *system network databases*) and generates network rules for N states. This step processes each *SysNetDb_i_ID* to produce network rules (*net_Rules_i_ID*) for state S_i. The network rules for N states are stored in N files that are collectively stored in a directory referred as *netRules*. This collection may have some similar rules and some dissimilar rules. The algorithm uses the network rules of the N states in the following three steps.

- Firstly, merge network rules (each as X→Y) of N states to make a collection of rules as *Collect_NRs_ID*. The algorithm identifies Network Rules (NRs) using a Network Rule Mining (NRM) algorithm. The **NRM** algorithm depends upon the sequential rule mining over the set of connection pairs (in SysNetDbs). The source and target sets in a connection pair is a sequence of a sequence database. A network rule (as a sequential rule) can be identified using minimum support and mining confidence on the network database (as a sequential database). Thus, intuitively **NRM** is reducible to **SRM** because the SRM efficiently used as a subroutine to solve the NRM efficiently. This reduction of transforming NRM algorithm to the SRM algorithm is our conceptual contribution. We accomplished this by applying the **NRM** algorithm on *SysNetDb_i* to identify the network rules (*net_Rules_i*) for a state S_i. A network rule X→Y has antecedent X as a subset of source entities and consequent Y as a subset of target entities for connection pairs.

- Secondly, using **Merge** algorithm the network rules of different states are merged together to make a collection of network rules (*Collect_NRs*). From this collection, we select unique (or distinct) network rules over states as NERs. Count the occurrence of each identical rule *NER_ID* (X→Y) in

Collect_NRs_ID. The count is the *stabilityCount* of a *NER_ID*, such that *stabilityCount* is the number of states in which *NER_ID* is interesting. Calculate *stability* as the fraction of *stabilityCount* and number of states (N). Then add the *NER_ID* and its *stability* to the hash-map (*NERs_HM*). Make a file *NERs_ID* to store the retrieved *NERs* and their stability.

- Thirdly, select a *NER_ID* as *SNER_ID*, if the *NER_ID* has stability more than minStab. Make a file SNERs_ID and store each SNER (X→Y) only once. For each NER, we count its number of distinct occurrence in states - the count is the 'stability count' of the NER. An expert specifies a threshold minStabCount, which aids to retrieve SNERs from NERs whose stability count is greater than minStabCount value.

The threshold (minSupCount-minConf-minStab) is used to control the quality of output rules. A set of low values of threshold can generate exhaustive number of rules, which may be useless in decision-making. A best possible high value of threshold optimizes the number of interesting NERs and SNERs. We determine the best possible high values of threshold based on well-known theory of exploration and exploitation. The threshold values are dependent on domain and SysNetDbs. Search minSupCount-minConf values, which generate optimal number of NERs. For optimum low values of minSupCount-minConf, *explore NERs* by varying the value of minStab. After getting optimum number of NERs, start searching a range of minStab. In the range, start *exploiting SNERs* by varying the value of *minStab.* This results in best possible high values of the three thresholds. Furthermore, optimize the number of SNERs by using distinct NERs in each state. This helps in decision-making and taking-actions.

The retrieved NERs and SNERs are generated as the output. In the output, the SNERs are more meaningful as compared to NERs because a SNER is interesting as well as stable in an evolving system. Here, 'meaningful' means selected SNERs have more 'stability' than the unselected NERs, where stability is measured with the threshold minStab

Algorithm 4.2 **Mining_NERs_SNERs(***SysNetDbs*, *minSupCount*, *minConf*, *minStab***)**

```
Initialize File net_Rules_i_ID
Initialize Array Collect_NRs_ID,
Initialize HashMap NERs_HM < NER_ID, stability >
Initialize File NERs_ID, SNERs_ID
Initialize i ∈ integer

For each SysNetDb_i_ID in SysNetDbs, where i varies from 1 to N
    net_Rules_i_ID = NRM(SysNetDb_i_ID, minSupCount, minConf)
    Store net_Rules_i_ID file in directory netRules
End For

For each state net_Rules_i_ID in netRules, where i varies from 1 to N
    Collect_NRs_ID = Merge(Collect_NRs_ID, net_Rules_i_ID)
End For

For each distinct rule (as NER_ID) in Collect_NRs_ID
     Initialize int stabilityCount = 0
     For each rule_x in Collect_NRs_ID
```

```
        if(NER_ID equal to rule_x)
            then stabilityCount++
        end if
    End for
    Initialize float stability = stabilityCount ÷ N
    if(NER_ID is not in NERs_HM)
        then Add(<NER_ID, stability> to NERs_HM)
    end if
    if(stability > minStab)
        if(NER_ID is not in SNERs_ID)
            then Add(NER_ID to SNERs_ID)
        end if
    end if
End For
NERs_ID = NERs_HM
Return NERs_ID and SNERs_ID
```

Algorithm 4.2.1 **NRM(***SysNetDb_i*, *minSupCount*, *minConf***)**

```
seq_rule_i = SRM (SysNetDb_i, minSupCount, minConf)
// The NRM is reducible to the sequential rule mining (SRM) because SysNetDb is a kind of sequence database, where each connection pair is a sequence of source and target sets.
// The SRM uses minSupCount and minConf to generate sequential rules (i.e. network rules with source and target sets.
Return net_Rules_i
```

Algorithm 4.2.2 **Merge(***Collect_NRs*, *net_Rules_i***)**

```
For each distinct network rule (NR) in net_Rules_i
    append(NR) to Collect_NRs
// This makes multiple NR for multiple states.
// These NR in Collect_NRs is referred as NER.
// This multiple entries of NR helps to calculate stability of the NER in the SNERM algorithm.
Return Collect_NRs
```

The meaningful NERs depend upon the threshold minStab. If the minStab is higher than the expected range of meaningful NERs, then less number of most meaningful SNERs are retrieved. Conversely, if the minStab is lower than the expected range of meaningful NERs, then the retrieved SNERs includes both meaningful and meaningless NERs. On one hand, for a high value of minStab, there might be no SNERs. On the other hand, for a low value of minStab, there might be exhaustive number of SNERs.

The NERs are not retrieved as SNERs belongs to the set of less stable rules, which can be considered as interesting but unstable NERs for a threshold. Such NERs also have support, confidence, and stability; however such NERs has stability lesser than the minStab. The thresholds are measured by the values of minSupCount-minConf–minStabCount that are decided by performing experiments.

The retrieved NERs and SNERs are in the form of *entityIDs*. These *entityIDs* are hard to comprehend because it is in the number format. Thus, for better comprehension convert them into the form of *entityNames*.

The **Mining_NERs_SNERs** algorithm finds and compares the counts of NERs and SNERs. The computational complexity of the **Mining_NERs_SNERs** algorithm mainly depends upon O(N×λ), where λ is the complexity of NRM algorithm.

4.4.3 Indexing

The Algorithm 4.3 Indexing replaces all the *entityIDs* in the *NERs_ID* and *SNERs_ID* with their corresponding *entityNames*. The algorithm uses the *IndexFile* produced in pre-processing. Make two files *NERs_Name* and *SNERs_Name* to store the NERs and SNERs in the form of *entityNames*. For example, the format of SNERs (X➔Y) in *SNERs_ID* contains *entityID* like {1, 2}➔{3}, which are same as format of association rules with X = {1, 2} and Y = {3}. These rules can be converted to *entityName* format rules like {butter, jam}➔{milk} where 1, 2, and 3 stands for butter, jam, and milk, respectively.

Algorithm 4.3 **Indexing(***NERs_ID, SNERs_ID*, *IndexFile***)**

```
Initialize HashMap Index<entityName, entityID>
Initialize integer counter = 1
Initialize String Buffer $buffer_1$, $buffer_2$
Make two file NERs_Name, SNERs_Name

For each line of IndexFile
    Scan line and Store <entityName, entityID> in Index
End For when IndexFile is completely scanned

For each line of file NERs_ID,
    Scan and store the line in buffer
    In buffer, replace each entityID with its entityName in Index
    Store the buffer in NERs_Name
End For when NERs_ID is completely scanned

For each line of file SNERs_ID,
    Scan and store the line in buffer
    In buffer, replace each entityID with its entityName in Index
    Store the buffer in SNERs_Name
End For when SNERs_ID is completely scanned

Return NERs_Name, SNERs_Name
```

4.4.4 SNS and Changeability Metric

After retrieving the NERs and SNERs, finding their counts is a straightforward process. Thereafter, use stability and changeability metrics formula to compute the system stability and changeability metrics. Keep note of the number of states N and minStab used to retrieve NERs and SNERs. As per the definition and its formula, this step uses: N, minStab, NER count, and SNER count. It measures the stability quantitatively for a state series of an evolving system. The calculation of stability given in equation (4) is self-explanatory. Similarly, find NERs_Count, and SNERs_Count, then use Changeability Metric formula (given in equation (5)) to compute and store in *changeability metric.*

Based on our approach, we developed a prototype tool on Java. The next section present a tool based on the described Algorithm SNS.

4.5 SNS-Tool

Based on Algorithm SNS, we developed an automated tool and named it as System Network Stability Tool (SNS-Tool) using Java technology (JRE and JDK 7th version or higher). As input, the SNS-Tool takes N *system network databases* (SysNetDbs) with minSupCount-minConf-minStabCount. As output, the tool retrieves network rules, collection of NERs, and SNERs. The SNS-Tool discovers these system evolution rules using three components '*Pre-processing*', '*Mining_NERs_SNERs*' and '*Indexing*'.

First component pre-processes each state to produce its network, which is a directed graph to describe relationships between inter-connected entities. The tool creates a *system network database* using evolving network of a state. In this way, the tool creates N SysNetDbs for N states. Each SysNetDb contains entity connection information as a connection pairs (L, R) that contains set of source entities and target entities.

Second component processes the N SysNetDbs to retrieve NERs and SNERs. Our tool internally uses fundamental Sequential Rule Mining (SRM) [40] (a kind of association rule mining [14]) to do Network Rule Mining (NRM). For NRM, the tool uses SRM because the *system network database* is the kind of sequence database. A SysNetDb contains sequence of two sets of source and target entities. Specifically, for each state, our tool used RuleGrowth [41][42][43] to generate network rules using a set of connection pairs in a SysNetDb. In our tool, we used *RuleGrowth* algorithm (and tool) proposed (and developed) by Fournier *et al.* [41][42][43] as sequential rule mining algorithm, which finds most frequently occurring partially ordered *sequential rules*.

Although any sequential rule mining algorithm can be used in the SNS-Tool, we used RuleGrowth algorithm because it mines sequential rules common to several *sequences* (i.e. *connection pairs*). In our case, a *sequence* is a *connection pair* as a special case, thus we got network rules between the inter-connected entities. Rest of the approach is same as described in the *Mining_NERs_SNERs* algorithm. We merged unique network rules to make a collection of NERs. Thereafter, the SNS-Tool finds NERs with stability count greater than minStabCount to retrieve non-redundant SNERs.

Third component converts the entity ID based rules (NERs and SNERs) to entity name based rules, which are in natural language, thus comprehensible. The tool summarizes the SNERs into a report file, which contains information about the co-occurrence of interconnected entities. The NERs and SNERs can predict missing and possible co-occurring entities.

This tool is interesting for practitioners, who deal with an evolving system that can be represented as a set of evolving networks. The evolving network represents connections between several entities in a system state. Existing approaches concentrate on either network mining or evolution mining, but we used both network and evolution mining together. We also added characteristic of stability for a system and its evolving networks. We used network rule mining approach, which is an automated technique for knowledge discovery. With the proposed SNS technique and tool, one can retrieve stable and evolution rules about system evolution. This information can be used to identify the system stability over a time-period. The SNS-Tool experiment aims to show application and usefulness to achieve automation. Further, a practitioner can apply similar steps of SNS to gain insight about an evolving system. The insightful information is useful due to evolution and stable characteristic of rule for inter-connected system entities. The next section describes the use of SNS-Tool, and we discuss experimental results by applying our tool for six real-world evolving systems.

4.6 Experimentation and Results

This section describes application of SNS using our automated SNS-Tool on six real-world evolving systems collected from open-internet repositories. We pre-processed six evolving systems to generate their six set of *evolving networks* such that each network is a directed graph, which contains relationship information about their inter-connected entities. Each line of a network file contains two entities (as nodes) to represent a directed connection (as edge) from the source entity to the target entity. For each evolving system, we transformed its set of evolving networks to produce SysNetDbs (*system network databases*) for a state series.

In Table 4.1, the first two columns describe the six evolving systems belonging to four different domains. Third column contains total number of entities used to make networks. Fourth column contains average number of neighbours (entities). Fifth column presents calculation for the number of aggregated connections using: number of states, number of entities, and average number of neighbours.

We used our SNS-Tool to show the practical implication of proposed approach and algorithm described in previous sections. We now discuss experiments on six evolving systems of following four domains. This also demonstrates how to apply the SNS technique and tool on different kinds of evolving system. We describe three experimental results on evolving systems following three sub-sections: Stable Network Evolution Rules (SNERs), Comparing NERs and SNERs, and System Network Stability Metrics.

4.6.1 Stable Network Evolution Rules (SNERs)

With the proposed tool, we conducted experimentation on six evolving systems shown in Table 4.2, where the first column is list of evolving system names. The second column listed the number of states used in the experimentation for an evolving system. The third column listed the number of entities in an evolving system. The fourth column listed the experimental thresholds of minSupCount-

Table 4.1 Domain Information of Evolving systems and their Evolving networks.

Domains of Evolving System	Evolving Systems	Number of entities	Average number of neighbours	Number of aggregated connections
(A) Evolving Software Systems	Hadoop HDFS	3129	2.166	15×3129×2.166 = 101661.21
(B) Evolving Natural-language Systems	List of Bible Translation	246	1.456	13×246×1.456 = 4656.288
	List of Multi-sport Events	141	1.786	13×141×1.786 = 3273.73
(C) Evolving Retail Market System	Retail Market	1872	7.204	13×1872×7.20 = 175219.2
(D) Evolving IMDb movie-genre System	Positive sentiment of movie genres	284	2.661	16×284×2.661 = 12091.58
	Negative sentiment of movie genres	510	3.303	16×510×3.303 = 26952.48

minConf-minStabCount. Here, minSupCount and minStabCount are in frequency, which are convertible to fraction minSup and minStab. Where, the minSupCount is a threshold for the number of entities occurring together in the connection pairs, and the minStabCount is a threshold for the number of states in which NER occurs.

As we used fixed size of individual system network database for each evolving systems' experiments, thus both support count and support fraction have same significance. Hence, we used minSupCount for thresholding the support count (since integer value is simple to interpret as compared to float). Similarly, we used fixed number of states (N) for each evolving systems' experiments, thus both stability count and stability fraction have same significance. Hence, we used minStabCount for thresholding the stability count.

Table 4.2 also demonstrates the empirical evaluation of the experiment by mentioning the experimental results in following columns. The fifth column provides the number of NERs retrieved in the experiments. The sixth column provides number of SNERs retrieved. The seventh column provides the number of distinct source and target entities in the SNERs. The eighth column provides changeability metric value computed from the given details about experiment in each row.

Table 4.2 Information about SNERM experiments conducted on the six evolving systems.

Evolving Systems	N	Number of entities	minSupCount-minConf-minStabCount	NER Count	Stable NER Count	Number of source and target entities	Changeability metric
HDFS-Core	15	3129	4-0.4-3	5	4	4 & 4	6.25
List of Bible Translation	13	246	3-0.3-2	3715	16	2 & 4	1509.21
List of Multi-sport Events	13	141	3-0.2-2	11	6	3 & 2	11.91
Retail Market	13	1872	4-0.6-3	131	12	12 & 2	46.94
Positive sentiment of movie genres	16	284	2-0.3-4	146	5	5 & 3	116.8
Negative sentiment of movie genres	16	510	2-0.3-4	258	10	9 & 5	103.2

The different values of minSupCount, minConf, and minStabCount generate different number of rules (with varying interestingness and stability) as output. The low values of minSupCount-minConf-minStabCount results in many SNERs, which include less interesting and less stable rules. For low thresholds, such exhaustive number of rules is tedious to manage. Therefore, best possible high values of minSupCount, minConf, and minStabCount produce optimized number of interesting and stable rules, which help in decision-making. Thus, in the experiments, we used the best possible high values of minSupCount-minConf-minStabCount to optimize number of interesting and stable NERs.

A system data engineer wants to report less number of manageable and meaningful rules by choosing best-possible high value of thresholds. The meaningfulness is a qualitative measure and optimized with the quantitative values of the threshold. Although we can generate large number of NERs, such exhaustive number of NERs, which have both meaningful and meaningless NERs. Thus, we used explore and exploit theory to choose of threshold values: minSupCount, minConf, and minStabCount. We explored minSupCount and minConf to generate optimized number of NERs, and then exploited the minStab to generate meaningful SNERs. Experiments are done for the best-possible high values of thresholds to generate manageable number of NERs and meaningful SNERs. Each SNER ($X \rightarrow Y$) is a kind of persistently linked co-occurring entities over a network states. Such NERs and SNERs are helpful in inferencing, decision-making, and action-taking.

Table 4.3 shows the SNERs retrieved for the experiment mentioned in Table 4.2. The experiments are done for the thresholds given in the fourth column of Table 4.2. We show only SNERs because they are interesting as well as stable. The NERs and SNERs are generated using Mining_NERs_SNERs code for a set of evolving system's states. The SNERs (in Table 4.3) represents antecedents as set of source entities and consequents as set of target entities. To do experiments, we used *set of states* as follows: *set of versions* for the Hadoop-HDFS (a software), *set of centuries* for the list of bible translation, *set of decades* for the list of multi-sport events, *set of months* for the retail market data, *set of decades* for the Internet Movie Database (IMDb).

We describe experiment on each evolving system separately by making inferences from the SNERs that are mentioned in Table 4.3. We also explain the advantages of those SNERs for their evolving systems. The SNS-Tool automatically retrieved many other such evolution rules (NERs and SNERs). The inferences and advantages of the SNERs for the six evolving systems are as follows.

a) Call graph analysis of evolving software: Hadoop-HDFS

We pre-processed 15 jars of Hadoop-HDFS[1] to make 15 *evolving call graphs* (networks) that are further pre-processed to make 15 SysNetDbs for 15 versions (states). A retrieved SNER (create ==> convert) suggest that procedure 'create' is a source entity (caller procedure) and 'convert' is a target entity (callee procedure). The rule suggests that the procedure 'create' frequently depends upon the procedure 'convert' for a given threshold. This makes us to know changes in procedure 'convert'

Table 4.3 For each evolving system, the Stable NERs are generated based on the minSupCount-minConf-minStabCount as mentioned in Table 4.2.

Hadoop-HDFS[1]	**List of Multi-sport Events**[3]	**Positive**[6] **sentiment in IMDb**[5]
create ==> convert readAll ==> readFully checkAccess ==> getDelegationToken close ==> getRemoteAddressString	World ==> International Games ==> Regional Games ==> International Games, World ==> International Games, Asian ==> Regional Asian ==> Regional	premier ==> Short fond ==> Short fine ==> Short humor ==> Comedy grand ==> Music
List of Bible Translation[2]	**Retail Market**[4]	**Neagative**[6] **sentiment in IMDb**[5]
English ==> Vulgate English ==> Masoretic English, Modern ==> Masoretic English ==> Masoretic, Text English, Modern ==> Masoretic, Text English ==> Text English, Modern ==> Text English ==> Text, Greek English, Modern ==> Text, Greek English ==> Greek English, Modern ==> Greek Modern ==> Masoretic Modern ==> Masoretic, Text Modern ==> Text Modern ==> Text, Greek Modern ==> Greek	SUKI SHOULDER BAG ==> 17841 ASSORTED MONKEY SUCTION CUP HOOK ==> 17841 CARRIAGE ==> 14911 SKULL DESIGN TV DINNER TRAY ==> 17841 ASSORTED COLOUR LIZARD SUCTION HOOK ==> 17841 SMALL YELLOW BABUSHKA NOTEBOOK ==> 17841 LIPSTICK PEN RED ==> 17841 UNION STRIPE CUSHION COVER ==> 17841 DISCO BALL CHRISTMAS DECORATION ==> 17841 BLUE/CREAM STRIPE CUSHION COVER ==> 17841 CAKE PLATE LOVEBIRD WHITE ==> 17841 KITCHEN METAL SIGN ==> 17841	rue ==> Short terrible ==> Short rue ==> Documentary pain ==> Short vent ==> Short passe ==> Short sin ==> Drama brat ==> Drama terror ==> Horror perverse ==> Adult

affects to the procedure ‘create’.

In Hadoop-HDFS, the SNER “readAll ==> readFully” means that if procedure ‘realAll’ is present in a module then most probably ‘readFully’ is also present in that module. Similarly, the other rules can also be interpreted. The advantage of these SNERs is in program analysis to do efficient software maintenance and evolution automatically.

b) ***Processing of evolving natural language: List of Bible Translation and Multi-sport Events***

We used natural language text from two Wikipedia pages: list of bible translations[2] and list of multi-sport events[3]**.** First dataset contains *list of bible translations*, which mention the translations done since 7th century until 2014, from source biblical languages (like Hebrew and Greek) to English variant languages (like Modern and Old English). We combined three tables: *incomplete Bibles, partial Bibles*, and *complete Bibles* given in the Wikipedia page. We made evolving networks using inter-connected entities (words) in columns of ‘English variant’ and ‘Source’ of bible, such that each network is for a century. We extracted 13 evolving networks that are converted to 13 SysNetDbs for 13 centuries. Two retrieved SNERs (English, Modern ==> Greek) and (English ==> Vulgate) suggest that bible in the Modern English language is mostly translated from Vulgate or Greek with a probability (i.e. minSup-minConf-minStab). In List of Bible Translation, the SNERs suggest that most of the ‘Modern English’ bibles are translated from the ‘Valgute’, ‘Masoretic Text’, and ‘Greek Text’. The advantage of these SNERs is in natural language processing to study bible evolution automatically.

Second dataset contains *list of multi-sport events*, which mention events happened since 1890’s until 2015. We pre-processed this dataset to create a set of evolving networks from connections between entities (words) in columns of ‘Title’ and ‘Scope’ (regional, international, and provinces), such that each network is for a decade. We extracted 13 evolving networks that are converted to 13 SysNetDbs for 13 decades. Two retrieved SNERs (Games, Asian ==> Regional) and (Games, World ==> International) suggests that the ‘Asian’ ‘Games’ are of type ‘Regional’ sport. Similarly, we can deduce ‘World’ ‘Games’ are of type ‘International’ sport. In the List of Multi-sport Events, the SNERs suggest that events named with ‘World Games’ are probably having ‘International’ scope (level) and events named with ‘Asian Games’ are probably having ‘Regional’ scope (level). The advantage of these SNERs is in natural language processing to study multi-sport event evolution automatically.

c) ***Market analysis of evolving retail-market***

We used a dataset for retail market [170], which contains purchasing information between December 2010 and December 2011 from online retail based on UK. To make evolving networks, we used *product descriptions* and *customer IDs* for each month in following way. A word in product description is used as a source entity only if the word appears in more than or equal to 10 months. Similarly, a customer ID is used as a target entity only if the customer did purchasing in more than or equal to 10 months. These frequent product words and customer IDs are used to create 13 evolving networks that are further converted to 13 SysNetDbs for 13 months.

A retrieved SNER “SUKI SHOULDER BAG ==> 17841” suggest that the product with description (source entity) ‘SUKI SHOULDER BAG’ is frequently purchased by the customer (target entity) ‘17841’, with a given probability (i.e. minSup-minConf-minStab). Such SNERs are useful to do target marketing on customers. There could be many such evolution rules (NERs and SNERs) for low values of minSupCount-minConf-minStabCount. In Retail Market, the SNER suggest that the customer with ID ‘17841’ frequently purchases shopping items listed as the antecedent of the rules. Similarly, the customer with ID ‘14911’ frequently purchases ‘carriage’ as the shopping item. These shopping items are useful to do target marketing on such customers. The advantage of these SNERs is in automated market analysis.

d) ***Positive and Negative Sentiment analysis on evolving IMDb system***

We used movie name and genre data of IMDB[5] for 16 decades (1870’s - Oct 2016). We made two types of evolving networks using two sentiment-list[6] of positive and negative words; Minqing and Bing [171][172] created the list.

First type of evolving networks have connections between positive words in ‘movie names’ as source entities and its ‘genres’ as target entities. A retrieved SNER “premier ==> Short” suggests that the movie name containing ‘premier’ appears in the ‘Short’ genre movies for a given threshold. Such SNERs are useful to find positive words for naming a movie according to a genre, which is suitable to target a positive kind of audience. In the positive sentiment based dataset of IMDb, a rule suggest that movie name with positive sentiment words like ‘premier’, ‘fond’, and ‘fine’ probably belongs to ‘Short’ movie genre. Similarly, positive sentiment word ‘humor’ most probably appears in movies of genre: Comedy. Similarly, positive sentiment word ‘grand’ most probably appears in movies of genre: Music. The advantage of these SNERs is in automated positive sentiment analysis of movie names-genre.

Second type of evolving network has connections between negative words in ‘movie names’ as source entities and its genre as target entities. A retrieved SNER “sin ==> Drama” suggests that the movie name containing ‘sin’ appears in ‘Drama’ genre for a given threshold. Such SNERs are useful to find negative words that are suitable for a genre to target a negative kind of audience. In the negative sentiment of IMDb result, a rule suggest that movie name with positive sentiment words like ‘rue’, ‘terrible’, ‘pain’, ‘vent’, ‘passe’ probably belongs to ‘Short’ movie genre. Similarly, negative sentiment word ‘rue’ most probably appears in movies of genre: Documentary. Similarly, negative sentiment words ‘sin’ and ‘brat’ most probably appears in movies of genre: Drama. Similarly, ‘terror’ and ‘perverse’ belongs to movie genres Horror and Adult respectively. The advantage of these SNERs is in automated negative sentiment analysis of movie names-genre.

4.6.2 Comparing NERs and SNERs

Interpreting NER and SNER count optimization in Figures 4.6, 4.7, 4.8, 4.9, 4.10, and 4.11, which describes a brief overview to demonstrate experimentation results. Each figure is a bar chart that demonstrates nine pair of bars to represent nine experiments. In each bar chart, horizontal-axis depicts the values of *minSupCount-minConf-minStabCount* for which experiments are performed to generate NERs and SNERs. In each bar chart, vertical-axis depicts the count of NERs or SNERs; such that the SNERs count is in decreasing order. Each pair of bars represents the number of NERs and SNERs retrieved for single execution of mining NERs and SNERs algorithm. Each figure shows nine pair of bars for nine such executions. Each pair of bars is for a combination of the values of minSupCount-minConf-minStabCount used in Mining_NERs_SNERs algorithm. Based on these figures (4.6 - 4.11), we can observe following three inferences for relative changeability between six evolving systems.

- In Figures 4.6 and 4.8 of the Hadoop-HDFS and the List of Multi-Sport Events, respectively - for an experiment - the difference between the counts of SNERs and NERs are less. Thus, both of the systems are less prone to changeability.

- The evolving systems of Figures 4.9, 4.10, and 4.11 have moderate difference between the counts of SNERs and NERs (for an experiment). Thus, the three evolving systems are moderately prone to changeability.

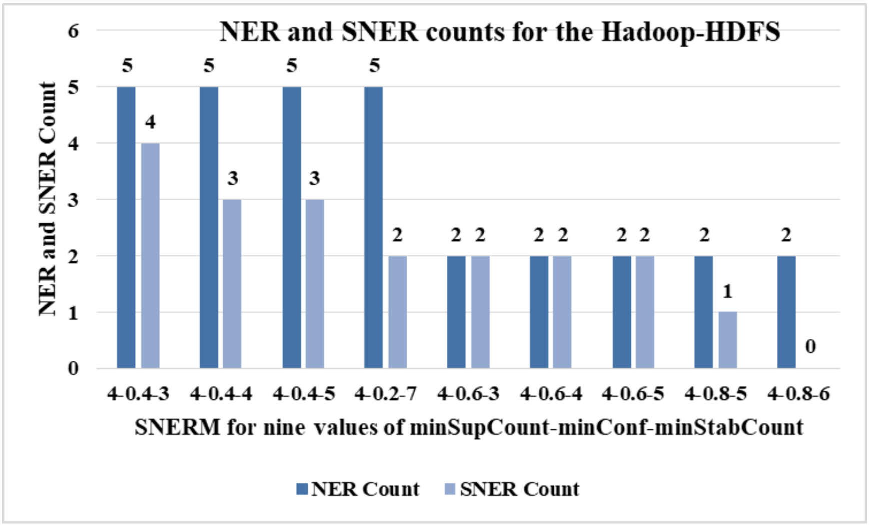
- In Figure 4.7 of the List of Bible Translation, the difference between the counts of SNERs and

NERs are high. Thus, it is highly prone to changeability.

The above three inferences are also true in case of the changeability metric value given for the six evolving systems in Table 4.2. The NERs and SNERs can also be retrievable in other fields like bioinformatics data, geological data, agricultural data etc. Such domains can use and take advantage of the Mining_NERs_SNERs algorithm. The Mining_NERs_SNERs applied on an evolving system can help a subject matter specialist (or domain expert).

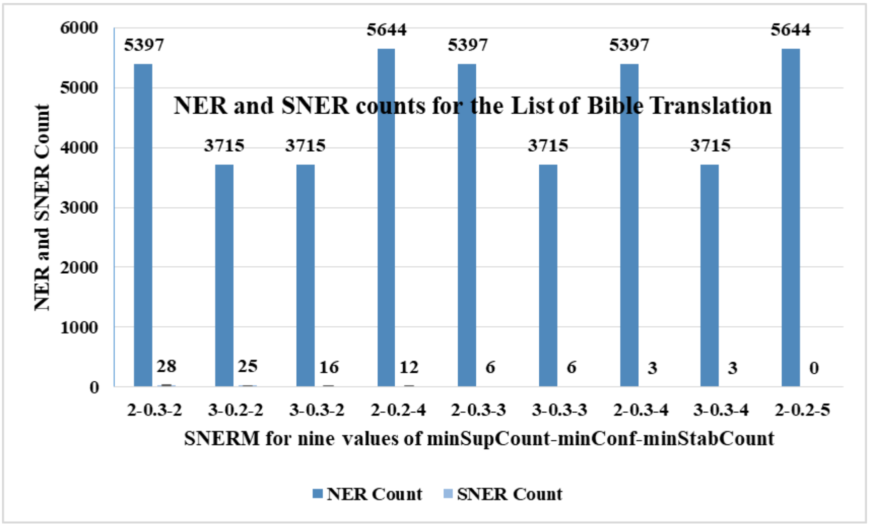
We presented novel technique to retrieve NERs, and then we used NERs as baseline for SNERs to show improvement. Both NER and SNER are "interesting" (same as an interesting association rule), but a SNER is also "stable" in the system states. If we assume the retrieved NERs as the baseline then the SNERs are the improvements. We compared this improvement in terms of the number of stable rules (SNERs) with respect to the remaining unstable rules (in NERs). The Figures 4.6, 4.7, 4.8, 4.9, 4.10, and 4.11 shows pair of bars as improved optimization such that the number of NERs is larger than number of SNERs. This provides the quantitative improvement in efficiency due to retrieved SNERs from NERs.

The accuracy of the NERs and SNERs depends upon the algorithm for Network Rule Mining (NRM), which further depends upon the Sequential Rule Mining (SRM). The execution time of Mining_NERs_SNERs algorithm mainly depends upon four factors. First factor is the chosen SRM algorithm. Second, the database size used for the SysNetDbs; larger the database size more the execution time and smaller the database size lesser the execution time. Third factor is the threshold values used for experiments; lower values need more execution time and higher values need less execution time. Fourth factor is the complex connections between the entities used to make connection pairs; complex network connections make a complex SysNetDb and simpler network connections make simpler SysNetDb. The complexity of network connections to create SysNetDb depends upon system domain of the dataset. Higher the complexity of network to create SysNetDb results in higher execution time. Lower the complexity of network to create SysNetDb results in lesser execution time.



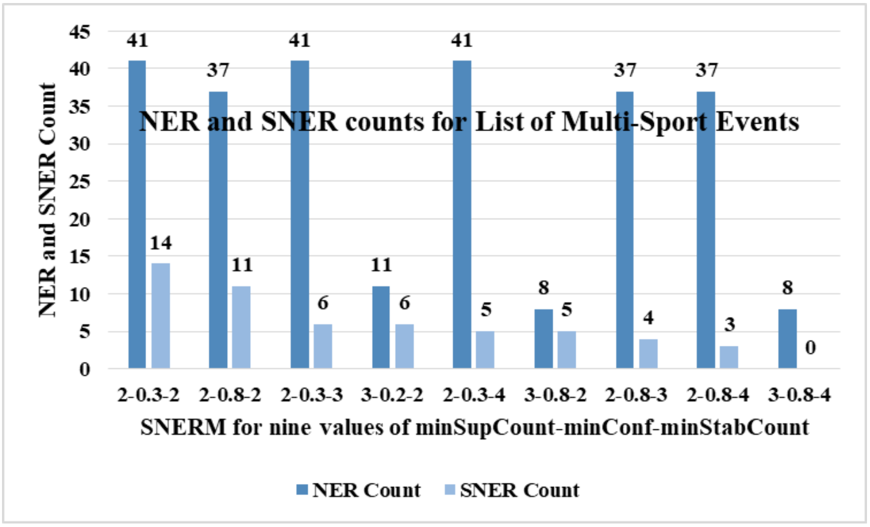
	4-0.4-3	4-0.4-4	4-0.4-5	4-0.2-7	4-0.6-3	4-0.6-4	4-0.6-5	4-0.8-5	4-0.8-6
NER Count	5	5	5	5	2	2	2	2	2
SNER Count	4	3	3	2	2	2	2	1	0

Figure 4.6. Mining_NERs_SNERs experiments for the Hadoop-HDFS.



	2-0.3-2	3-0.2-2	3-0.3-2	2-0.2-4	2-0.3-3	3-0.3-3	2-0.3-4	3-0.3-4	2-0.2-5
NER Count	5397	3715	3715	5644	5397	3715	5397	3715	5644
SNER Count	28	25	16	12	6	6	3	3	0

Figure 4.7 Mining_NERs_SNERs experiments for the list of bible translation.



	2-0.3-2	2-0.8-2	2-0.3-3	3-0.2-2	2-0.3-4	3-0.8-2	2-0.8-3	2-0.8-4	3-0.8-4
NER Count	41	37	41	11	41	8	37	37	8
SNER Count	14	11	6	6	5	5	4	3	0

Figure 4.8 Mining_NERs_SNERs experiments for the list of multi-sport events system.

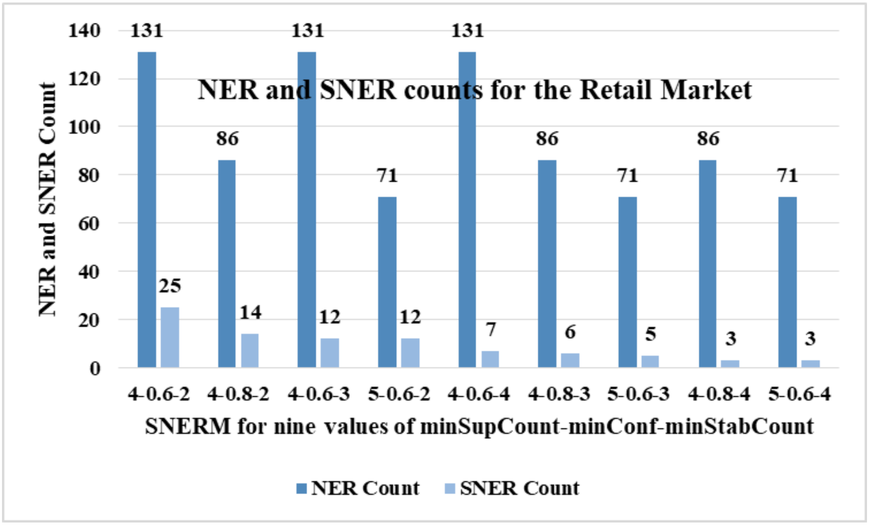


Figure 4.9 Mining_NERs_SNERs experiments for the retail market.

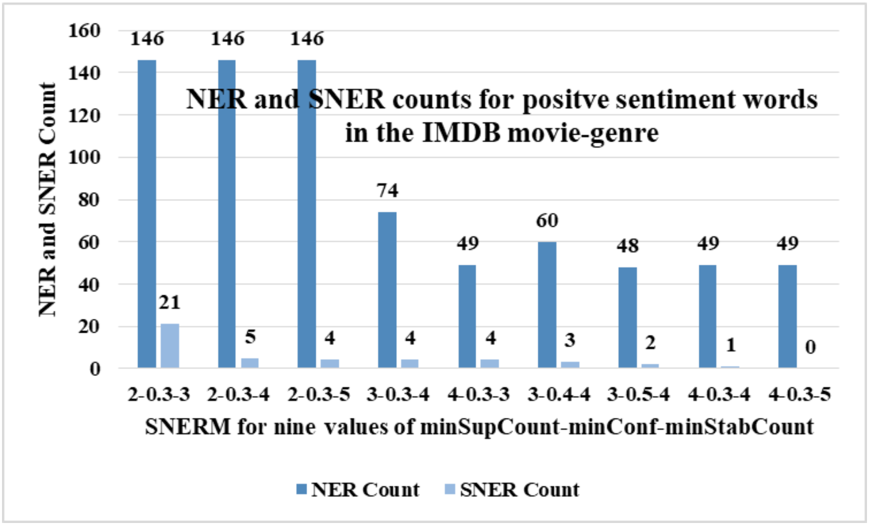


Figure 4.10 Mining_NERs_SNERs experiments for the positive sentiment in IMDb data.

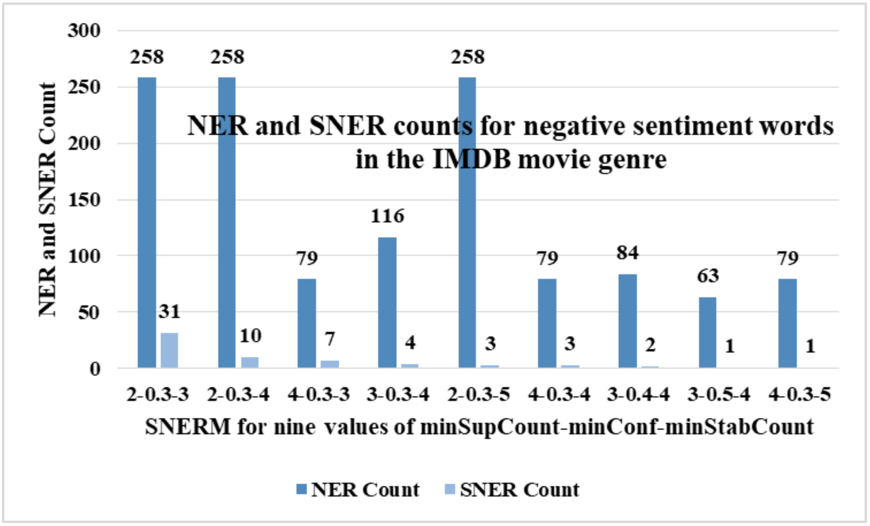


Figure 4.11 Mining_NERs_SNERs experiments for the negative sentiment in IMDb data.

4.6.3 System Network Stability Metrics

This subsection describes two experimental results for the (System Network Stability) SNS metric calculation. We demonstrate nine experiments for each evolving system, which proves usefulness of automation achieved with our SNS-Tool. For all nine experiments of an evolving system, we present details about SNS metric (equation (4)) calculation. In the end, we show two time series plots (Figure 4.12) to demonstrate stability metric values of the six evolving systems.

System Network Stability (SNS): The automated SNS-Tool retrieves information about evolution and stability of rules, which are useful to calculate system stability. Table 4.4 reports the nine experiments to calculate SNS metric for each of the six evolving systems. We divided the table into six part such that each part represents nine experiments for an evolving system, where each row describes an experiment. In each part, there are eight columns that describes following. First column mentions the experiment number (1 to 9) and the values of minSupCount-minConf-minStabCount to do the experiment. Second column contains the minStabCount value of the experiment. Third column contains the number of states (N) of an evolving system. Fourth column contains the fraction of minimum

Table 4.4 SNS metric calculations for nine experiments on each of the six evolving systems.

	minSupCount-minConf-minStabCount	min Stab Count	N	min Stab fraction	SNER Count	Total NER Count	Stable NERs fraction	SNS metric
	Hadoop-HDFS Software System							
1	4-0.6-5	5	15	0.33	2	2	1	33.33
2	4-0.6-4	4	15	0.27	2	2	1	26.67
3	4-0.4-5	5	15	0.33	3	5	0.6	20
4	4-0.6-3	3	15	0.2	2	2	1	20
5	4-0.2-7	7	15	0.47	2	5	0.4	18.67
6	4-0.8-5	5	15	0.33	1	2	0.5	16.67
7	4-0.4-3	3	15	0.2	4	5	0.8	16
8	4-0.4-4	4	15	0.27	3	5	0.6	16
9	4-0.8-6	6	15	0.4	0	2	0	0
	List of Multi-sport events a natural language system							
1	3-0.8-2	2	13	0.15	5	8	0.625	9.62
2	3-0.2-2	2	13	0.15	6	11	0.545	8.39
3	2-0.3-2	2	13	0.15	14	41	0.341	5.25
4	2-0.8-2	2	13	0.15	11	37	0.297	4.57
5	2-0.3-4	4	13	0.31	5	41	0.122	3.75
6	2-0.3-3	3	13	0.23	6	41	0.146	3.38
7	2-0.8-3	3	13	0.23	4	37	0.108	2.49
8	2-0.8-4	4	13	0.31	3	37	0.081	2.49
9	3-0.8-4	4	13	0.31	0	8	0	0
	Positive sentiment of IMDb movie genres system							
1	2-0.3-3	3	16	0.19	21	146	0.144	2.70
2	2-0.3-4	3	16	0.25	5	146	0.034	0.86
3	2-0.3-5	4	16	0.31	4	146	0.027	0.86
4	3-0.3-4	4	16	0.25	4	74	0.054	1.35
5	4-0.3-3	4	16	0.19	4	49	0.082	1.53
6	3-0.4-4	4	16	0.25	3	60	0.050	1.25
7	3-0.5-4	5	16	0.25	2	48	0.042	1.04
8	4-0.3-4	4	16	0.25	1	49	0.020	0.51
9	4-0.3-5	5	16	0.31	0	49	0.000	0.00

Table 4.4 SNS metric calculations for nine experiments on each of the six evolving systems.

	minSupCount-minConf-minStabCount	min Stab Count	N	min Stab fraction	SNER Count	Total NER Count	Stable NERs fraction	SNS metric
				List of Bible translations a natural language system				
1	3-0.2-2	2	13	0.15	25	3715	0.007	0.1
2	2-0.3-2	2	13	0.15	28	5397	0.005	0.08
3	3-0.3-2	2	13	0.15	16	3715	0.004	0.07
4	2-0.2-4	4	13	0.31	12	5644	0.002	0.07
5	3-0.3-3	3	13	0.23	6	3715	0.002	0.04
6	2-0.3-3	3	13	0.23	6	5397	0.001	0.03
7	2-0.3-4	4	13	0.31	3	5397	0.001	0.02
8	3-0.3-4	4	13	0.31	3	3715	0.001	0.02
9	2-0.2-5	5	13	0.38	0	5644	0	0
				Retail market system				
1	4-0.6-2	2	13	0.15	25	131	0.191	2.94
2	5-0.6-2	2	13	0.15	12	71	0.169	2.6
3	4-0.8-2	2	13	0.15	14	86	0.163	2.5
4	4-0.6-3	3	13	0.23	12	131	0.092	2.11
5	4-0.6-4	4	13	0.31	7	131	0.053	1.64
6	5-0.6-3	3	13	0.23	5	71	0.07	1.63
7	4-0.8-3	3	13	0.23	6	86	0.07	1.61
8	5-0.6-4	4	13	0.31	3	71	0.042	1.3
9	4-0.8-4	4	13	0.31	3	86	0.035	1.07
				Negative sentiment of IMDb movie genres system				
1	2-0.3-3	3	16	0.19	31	258	0.12	2.25
2	4-0.3-3	3	16	0.19	7	79	0.089	1.66
3	2-0.3-4	4	16	0.25	10	258	0.039	0.97
4	4-0.3-4	4	16	0.25	3	79	0.038	0.95
5	3-0.3-4	4	16	0.25	4	116	0.034	0.86
6	3-0.4-4	4	16	0.25	2	84	0.024	0.6
7	3-0.5-4	4	16	0.25	1	63	0.016	0.4
8	4-0.3-5	5	16	0.31	1	79	0.013	0.4
9	2-0.3-5	5	16	0.31	3	258	0.012	0.36

Stability {minStabCount ÷ N}. Fifth column contains the number of SNERs retrieved in the experiment. Sixth column contains the distinct NER count retrieved in the experiment. Seventh column contains the fraction of stable NERs {(SNER count) ÷ (total NER count)}. Eighth (the last) column contains the value of the SNS metric for the experiment. Each part of Table 4.4 is sorted in increasing order of SNS metrics for an evolving system.

Figure 4.12 presents six time series for six evolving systems. Each time series has nine co-ordinates that represent nine SNS metric values for the nine experiments done on an evolving system. The X-axis in the figure has same sequence as mentioned in Table 4.4 for each evolving system. For the time series of Hadoop-HDFS, in X-axis the '1' represents the first experiment '1' mentioned in first column of Table 4.4 for Hadoop-HDFS with minSupCount-minConf-minStabCount = 4-0.6-5.

Figure 4.12 helps to measure speed of evolution (0 to 100 in Y-axis) for the evolving systems. We inferred all the six systems are evolving in nature with different speeds. The List of Bible translation system has evolved fastest, while Retail market system and IMDB movie genre system has evolved moderately, whereas List of Multi-sport event and Hadoop-HDFS has evolved slowly. The positive and negative sentiment system has similar evolution rate.

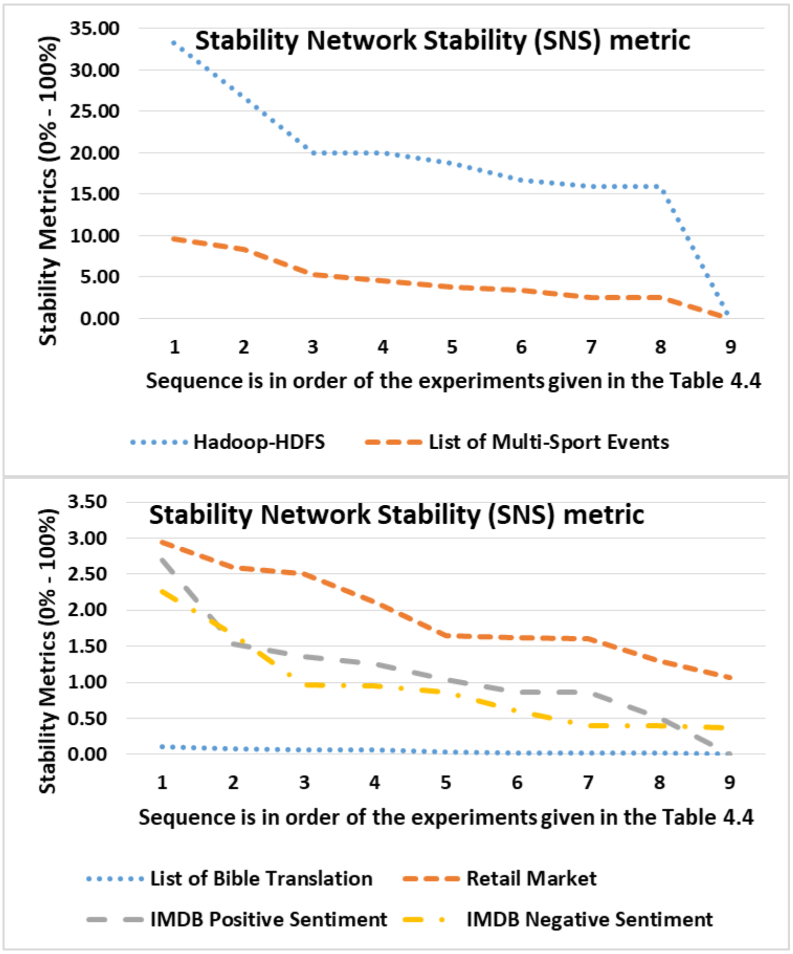


Figure 4.12 SNS metrics using the nine experiments for all six evolving systems.

We cannot infer the following two deductions from the SNS metric values. First, the two evolving systems of different domains are not comparable based on the SNS metric values because an experiment also depends upon factors like: type of entities and their connections. Second, low SNS metric value does not reflects unstable system, but it reflects fast evolution.

4.6.4 Advantage over existing works

In addition to the advantages of SNERs as compared to the baseline NERs, we also compared our approach with the existing state-of-the-arts (in Chapter 2). Despite unavailable exact comparable experiments; nevertheless, in the Chapter 2 we compared of our approach with some similar existing approaches, and described applications of our approach. In contrast to existing state-of-the-art, our proposed SNS algorithm is applicable to an evolving system that can be represented as a set of evolving networks. On one hand, usually rule-mining techniques are independent of state series. On the other hand, evolution (or temporal) mining techniques are independent of inter-connected entities. However, in this chapter, we presented an approach to mine evolution and stable rules about inter-connected entities of an evolving system. We demonstrated the use of prototype SNS-Tool for real-world applications that includes four domain based six evolving systems. In each experiment, we used optimize number of NERs and SNERs to calculate the SNS metric. The Algorithm SNS depends upon rule mining, thus pros-cons of rule mining are also pros-cons for Algorithm 4.2 Mining_NERs_SNERs.

4.7 Summary

In this chapter, we proposed System Network Stability (SNS) algorithm that uses an algorithm for mining Network Evolution Rules (NERs) and Stable Network Evolution Rules (SNERs). The algorithm analyses a pre-evolved system using novel parameter minimum Stability (minStab) to generate Stable NERs (SNERs). We introduced stability as a new measure for characterizing Network Evolution Rules (NERs) over time. We did this using a proposed algorithm Mining_NERs_SNERs, which retrieves NERs and Stable NERs (SNERs) in a set of states SS = $\{S_1, S_2 \ldots S_N\}$ of an evolving system. The purpose of SNERs is to study and compute the stability and changeability property of an evolving system.

Based on SNS algorithm, we developed an intelligent tool, which is used to do experiments on evolving systems. We prototyped the SNS algorithm as a tool named as SNS-Tool. We used SNS-Tool to demonstrate experiments to mine NERs and SNERs from six state series of six different evolving systems. We also demonstrated applications of the tool on six evolving systems. We summarized experimental results to show various SNERs for some chosen value of minimum support count (minSupCount), minimum confidence (minConf), and minimum stability count (minStabCount). We also demonstrated the optimization in the number of SNERs as compared to NERs.

Using the mining information, we presented a formula for *stability metric* and its type *SNS metric*, which quantifies the speed of evolution in the evolving system. Additionally, we also calculated nine SNS metric values for nine experiments of each evolving system. To the best of our knowledge, we found our approach is novel in context of system evolution analysis based on SNERs, Stability Metric, and Changeability Metric.

Our SNS-Tool automatically retrieves Stable NERs (rules) that can help to deduce evolution information. We quantified the speed of evolution by measuring slow or fast evolution of an evolving system, which is interesting and non-obvious information. The information may be further helpful to study system evolution analysis. We presented one of the novel approach and tool to do the stability and changeability analysis of an evolving system based on knowledge discovery and data mining.

Chapter 5

Network Evolution Subgraph (Graphlet or Motif) Mining for System Stability, Changeability, and Complexity analysis

We consider a complex system that evolves with time, which creates a state series SS = {S_1, S_2… S_N} such that each state corresponds to a time point. In this chapter, we introduce two kinds of *Network Evolution Subgraphs* such as *Network Evolution Graphlets* (NEGs) and *Network Evolution Motifs* (NEMs) retrieved from a set of evolving networks {EN_1, EN_2… EN_N} that represents multiple states of an evolving system. Such evolving subgraph patterns are interesting information about dynamic networks. Here, we elaborate the subsection 3.3 on network evolution subgraph mining.

We used frequencies of graphlets changing over graph snapshots to model a *changeability metric*, which is an evolution information about a set of evolving networks. Based on the changeability metric, we defined *stability metric* that aids to do system stability analysis. Further, we proposed an algorithm for mining NEGs and NEMs information followed by calculating changeability metric and stability metric over a state series. This technique is prototyped as a tool that we used to demonstrate the application of our work on six evolving systems including: one evolving software system, two evolving natural language systems, one evolving retail market system, and two evolving sentiments in IMDB movie name-genre system.

We also used graphlets information of a state to calculate *System State Complexity* (SSC). The *System State Complexities* (SSCs) represent time-varying complexities of multiple states. Additionally, we also used the NEGs information to calculate *Evolving System Complexity* (ESC) for a state series over time. We proposed an algorithm named *System Network Complexity* (SNC) for mining NEGs, SSCs, and ESC, which analyzes a pre-evolved state series of an evolving system. Again, we prototyped this technique as a tool named *SNC-Tool*, which is applied to six real-world evolving systems collected from open-internet repositories of four different domains: software system, natural language system, retail market basket system, and IMDb movie genres system. This is demonstrated as experimentation reports containing retrieved — NEGs, NEMs, SSCs, and ESC.

5.1 Introduction

Graph (or Network) can be used as a data structure to represent system entities (as nodes) and their connections (as relationships). Frequent subgraph mining is a popular data mining technique. A *frequent subgraph* is a graph whose support is greater than minimum support threshold as presented in AGM [49], FSG [50], and gSpan [51].

Network Motif is one of the well-known kind of *frequent subgraph*. Network motifs are patterns in the form of subgraphs in a network, which represents a statistically recurrent (or significant re-occurring) subgraph *H* in the network EN_i. A network motif is an interconnection occurring with significantly high frequency in a network. Milo *et al.* [52][53] has defined and detected "network motifs" for protein-protein interaction, food webs, and electronic circuits. This resulted in the development of many tools with the objective to retrieve network motif. These motifs are the building blocks that make up the network and useful to analyse the underlying network.

Graphlets [54][55] are the small connected non-isomorphic induced subgraphs. The motifs are statistically significant subgraphs, and the graphlets are induced subgraphs. Induced subgraphs means when they are aggregated according to their frequency then they represent its network structure. Frequency of induced subgraph is referred as graphlet frequency [173][174], which is the graphlet count to quantify the occurrence of local structural properties of a network. Mining of graphlets and motifs information is computationally expensive for large graphs, but they provide useful information. There are few more graph properties like centrality, degree distribution, and connectedness. We aim to calculate aggregate frequencies of subgraphs over time to generate network evolution information.

Since the inception of network motifs and graphlets, the following subgraph mining tool came into existence: mFinder [52][53], FANMOD [175], Kavosh [176], NetMode [177], ORCA [173], acc-Motif [178], and Co-evolving relational motifs [83]. Such tools find subgraphs occurring in the input graph, and then group the isomorphic subgraphs in classes. As output, the tools retrieve a class of subgraph with its frequency. Our work is independent of how subgraph-mining techniques work. We are interested to study their outputs over time. Our primary objective is to study the network evolution subgraphs, and our secondary objective is to study of system changeability and stability analysis. Our third objective is to study system network complexity.

For this, an algorithm is proposed, which retrieves information of *Network Evolution Graphlets* (NEGs) with their frequencies over states. Then, *Network Evolution Motifs* (NEMs) are retrieved from the set of NEGs. Further, the graphlet's evolving frequencies are used to calculate system's *Changeability Metric* (CM), *Stability Metric* (SM), *System State Complexity* (SSC), and *Evolving System Complexity* (ESC). Our approach is simple and easy to understand; therefore, it can be included into existing evolving systems. To show such application, we conducted experiments on multiple real-world evolving systems of different kinds and characteristics.

Rest of the chapter is organized as follows. Section 5.2 describes the preliminaries about subgraph mining for a state. Section 5.3 contributes novel definitions, and equations for NEG, NEM, CM, SM, SSC, and ESC. Section 5.4 presents illustrative examples. Section 5.5 contributes a proposed algorithm to compute NEGs, NEMs, CM, and SM for an evolving system. Section 5.6 contributes an approach *System Network Complexity* (SNC), which calculates SSCs and ESC to analyze an evolving system. Section 5.7 describes SNC-Tool based on proposed work. Section 5.8 demonstrates the experiments for System Stability and Changeability analysis. Section 5.9 demonstrates the experiments for System Network Complexity analysis. Lastly, Section 5.10 gives concluding remarks.

5.2 Preliminaries for Subgraph Mining

This section presents subgraph mining for a system state. Before defining terminologies, we start with basic notations and concepts. For a fixed number of nodes (subgraph-size), subgraphs are grouped into classes (i.e. motifs or graphlets). In the strict manner, the motif and graphlet represent two different properties of subgraph in a given network. For better understanding, we introduce some auxiliary notations and search space for subgraph mining.

Initially, choose a *subgraph size* to fix the number of vertices in a subgraph. Suppose there are M numbers of subgraph classes. We discriminated various subgraphs (or motifs or graphlets) by enumerating subgraphs from 0 to M. A subgraph class of induced subgraphs (graphlets) can be given as $\{G_0, G_1, G_2 \ldots G_M\}$, where G_j denotes jth graphlet. The $\{G_0, G_1, G_2 \ldots G_M\}$ denotes a set of all possible graphlets for a given *graphlet size*, where M is the total possible graphlets. *Graphlet frequency* is the count of a subgraph occurring in a network and denoted as *freqj* corresponding to the G_j. The $\{freq_0, freq_1, freq_2 \ldots freq_M\}$ denotes a set of *graphlet frequencies* for all possible graphlets. Similarly, $\{M_0, M_1, M_2 \ldots M_M\}$ denotes a subgraph class of frequent subgraph (network motifs), where M_j denotes jth motif. Where G_0 is same as motif M_0, G_1 is same as motif M_1 and so on. The graphlet or motif frequency is denoted as $freq_j$ corresponding to the jth graphlet or motif (G_j or M_j). Thus, the $\{freq_0, freq_1, freq_2 \ldots freq_M\}$ also denotes a set of frequencies for the set of motifs (i.e. frequent graphlets).

These graphlets can be retrieved with a subgraph mining technique. Search space of subgraph mining in our case is a system network containing connections between entities. Local subgraph mining on a network retrieves *graphlets information* that is denoted as *graphlets_info*. Subgraph mining retrieves induced subgraph information i.e., *graphlets information,* which is a doubleton set of graphlet and its frequency. The graphlets information is denoted as

$$graphlets\_info = < G_j, freq_j > \ \mid 0 \leq j, m \leq M \qquad \text{-- (1)}$$

where, G_j is a graphlet occurring $freq_j$ number of times in the network, and j is in enumeration of graphlet. Such that *j* and *m* ranges from 0 to M. Where, G_M is the highest enumerated graphlet.

After retrieving graphlets information, we retrieve motifs from the set of graphlets whose frequency is significantly high. However, the number of retrieved motifs is less than the number of retrieved graphlets. The definition for the graphlet of state S_i is simple "a subgraph which is an induced subgraph in an evolving network EN_i of state S_i". Similarly, the definition for the network motif of state S_i is simple "a subgraph which is statistically significant (or frequent) in evolving network EN_i of state S_i".

Different pre-processing techniques make different networks (i.e., relationship between entities as source and target nodes). The pre-processing depends upon the kind of the evolving system, i.e., different system has different pre-processing to make a series of evolving networks. This means a system pre-processing requires system domain expert, who can identify appropriate entities and relationship between those entities. The selection of relationship between inter-connected entities depends upon the required result (as final output). Even a system itself has different kinds of complex pre-processing to generate its evolving network. There two example of pre-processing: first, procedure-calls are entity-connections relationships in a software system; second, word-sequences are entity-connections relationships in a natural language system.

Subgraph mining can solve crucial problems by retrieving insight about characteristics and internal working of system network. In Figure 5.1, a state S_i is pre-processed to make an evolving network EN_i using intrinsic property of inter-connected entities in the evolving system. Further, the subgraph mining tool takes EN_i as input to retrieve graphlets information as output. Counting of subgraphs (e.g., motifs or graphlets) in a given network is the primary task of subgraph mining [173][180][181]. In subgraph mining, counting a class of subgraph requires different approaches of graph theory. Usually a subgraph mining tools and techniques involve (following) three steps. Firstly, it finds isomorphic subgraphs in a network. Secondly, it groups them into classes. Thirdly, determine frequencies of the subgraph classes. We are interested to enhance such existing subgraph mining technique that can study system network evolution over time.

Subgraph mining for a network EN_i of a state S_i can retrieve graphlets information. Sequentially, we can retrieve graphlets information for all states one by one (locally on each state). Our aim is to process a state series to identify the network evolution graphlets and complexities of an evolving system. Afterwards, we explain a novel technique for mining evolving networks, which discovers the hidden information for an evolving system.

5.3 Proposed Definitions and Concepts

This section describes proposed terminologies and definitions that we need to present this

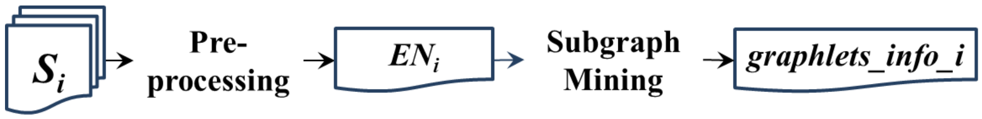


Figure 5.1 Subgraph mining of an evolving network ENi for a state Si.

chapter. To start with, we express a system state series SS as a set $\{S_1, S_2 \ldots S_N\}$ at various time points such that S_i belongs to time t_i.

Suppose we used a subgraph mining technique. Assume we already used a local subgraph mining technique (or tool) on each evolving network for all states of a system. This retrieves graphlets information for each state. Then, we aggregate this information over set of states, which retrieves *network evolution subgraphs* with their frequencies. Based on this assumption, our key conceptual contribution is definitions in the following subsections.

5.3.1 Network Evolution Subgraphs

Definition 5.1: A *network evolution subgraph* is a subgraph that has multiple occurrences in a state series SS, such that it appears in multiple states of a system with changing frequencies. Depending upon graph properties, the *network evolution subgraph* can be of three types:

- *network evolution frequent subgraph*: if subgraph is frequently occurring (means more than minimum support and minimum confidence) in a set of evolving networks;
- *network evolution motif*: if subgraph is statistically recurrent in a set of evolving networks; and
- *network evolution graphlet*: if subgraph is induced subgraph in a set of evolving networks.

For a *network evolution subgraph*, its frequencies keep on changing with states. These frequencies are aggregated to generate accumulated knowledge for a *network evolution subgraph* of a state series. On one hand, *frequent subgraphs* and *network motifs* are frequently (or statistically) occurring (or recurrent) as compared to a given threshold. On the other hand, graphlets are induced subgraphs, which provide frequencies that can be combined to form and represent its network. Thus, we are interested to accumulate knowledge of graphlets over a state series. We are interested to retrieve *network evolution graphlets* with its frequency in given set of evolving networks to make an aggregated information. This knowledge leads to define *aggregate frequency* for each retrieved *network evolution graphlets*. Thus, network evolution graph is defined as follows.

Definition 5.2: A *Network Evolution Graphlet* (NEG) is a graphlet G_j (induced subgraph) that appears in multiple evolving networks with evolving frequencies $freq_{ji}$ over a state series SS, where 'i' stands for state S_i.

Next, we aggregate frequencies $freq_{ji}$ to generate accumulated knowledge for a state series. To do this, we present a formula to calculate the *aggregate frequency* for each retrieved graphlets and it is defined as follows.

Definition 5.3 *Aggregate frequency:* Let G_j is a *network evolution graphlet* (NEG) appearing in multiple states. The $Aggregate\_freq_j$ denotes *aggregate frequency* of a NEG G_j. The aggregate could be calculated in many ways, e.g. mean, median, mode, sum, count, max, min etc. The arithmetic mean of frequency over N states for a NEG Gj is given by

$$Aggregate\_freq_j = \frac{\sum_{i=1}^{N} \boldsymbol{freq_{ji}}}{\boldsymbol{N}} \qquad \text{-- (2)}$$

where, $freq_{ji}$ is the *frequency* of NEG G_j in state S_i such that *i* vary from integer 1 to N and *j* is constant. This means *i* is a variable for different states and *j* is constant for G_j.

Definition 5.4: Network Evolution Graphlets information is a doubleton set that contains pair of retrieved NEGs (as a subgraphs) and their *aggregate frequencies*. The *NEGs information* is denoted as

$$NEGs\_info = < G_j, Aggregate\_freq_j > \mid 0 \leq j, m' \leq M \quad \text{-- (3)}$$

where, G_j is jth NEG with aggregate frequency as $Aggregate\_freq_j$ and j is the enumeration of the NEG; such that m' is the number of retrieved NEGs (distinct graphlets) over all the states in SS. Such that j and m' ranges from 0 to M. This makes pair of retrieved NEGs with their aggregate frequencies.

Definition 5.5: A Network Evolution Motif (NEM) is a NEG G_j that occurs statistically significant (or frequent) with aggregate frequency ($Aggregate\_freq_j$) above a threshold given by an expert.

5.3.2 System Changeability Metric and Stability Metric

As we assumed, we already computed graphlets information of multiple states using an existing technique (or tool). We use this graphlets information over a state series to make a function F_j representing a function of jth graphlet frequency

$$F_j = F\left(i,\ freq_{ji}\right) \quad \text{-- (4)}$$

where i represents state and j represents enumeration of graphlet G_j. The function F_j is used to measure the evolving system's changeability and stability, which is our second key conceptual contribution. The changeability and stability are the important measuring criteria that measures evolution over states. The equation (4) aids to define changeability metric. We can use derivative over the curve of a graphlet's frequencies (in multiple states) to compute the system changeability.

Definition 5.6: Changeability Metric (CM) is defined as summation of differential gradient of a graphlet's frequencies over states, which is further summed over all the retrieved graphlets

$$Changeability\ Metric = \sum_{j=1}^{m'} \sum_{i=1}^{N} \left| \frac{d\,\mathrm{F}\left(i, freq_{ji}\right)}{d\,i} \right| \quad \text{-- (5).}$$

Physical significance of changeability metric is that it measures randomness (or entropy) of inter-connected entities over a state series. The term $d\mathrm{F}\left(i,\ freq_{ji}\right)$ is computed by finding difference between frequencies of jth graphlet retrieved for two consecutive states (i.e. {$freq_{j(i+1)}$ and $freq_{ji}$}), and then divide this by epoch between the two consecutive states (epoch is the time-lag between two states {$(i+1)$th state to ith state}). We took its modulus in equation (5) because we are interested only in magnitude for change and evolution analysis. The value of the derivation depends upon the change in the graphlet's frequency between two states. A system remains stable over a long period, means it is not changed frequently. If system network dynamics is changing slowly over states, then the NEGs' frequencies will be stable. However, if system network dynamics are fluctuating, then NEGs' frequencies will also vary.

Less change in a graphlet's frequencies between states (e.g. from $freq_{ji}$ to $freq_{j(i+1)}$ of G_j) suggest less change between the two consecutive states. Note, i and (i+1) represents two consecutive states. Less change between states depicts stability in the evolving system, whereas high changes depict high instability due to high evolution-rate. From the changeability metric, we made following remarks. On the one hand, low change in the values of NEGs produces low value of the changeability metric, which means that only few NEGs are stable. On the other hand, high change in the values of NEGs produces high value of changeability metric, which means that many NEGs are unstable. However, the range of changeability metric is [0, ∞). The lower bound is zero (0) for a constant system, which is never underwent any change. The upper bound is infinity (∞) for an unstable system, which is underwent infinite changes.

High value of changeability metric depicts less stability, whereas low value of changeability metric depicts high stability. Based on this, we defined *stability metric* as follows.

Definition 5.7: *Stability Metric (SM)* is defined as the inverse of the changeability metric

$$Stability\ Metric = \frac{1}{Changeability\ Metric} \qquad \text{-- (6).}$$

Physical significance of stability metric is that it measures stability (or persistence) of inter-connected entities over a system's state series. However, the range of stability metric is [0, ∞), such that lower bound is zero (0) for an unstable system and upper bound is infinity (∞) for a constant system.

We can use the *stability metric* to know about the relative stability of a new state. Suppose the stability for N^th state is SM_N. If we add a new state to the system, then new stability for (N+1)^th state is SM_{N+1}. There are three conditions possible. If $SM_{N+1} > SM_N$, then changes are significantly done to construct a new state. If $SM_{N+1} < SM_N$, then the new state is not changed much as compared to the old states. If $SM_{N+1} = SM_N$, then the new state is similar to the old states.

5.3.2 System State Complexity and Evolving System Complexity

This subsection describes proposed system network complexity analysis definitions and concepts. A graph can be useful to calculate system complexity as a measure of interaction between various system entities as nodes. To calculate system network complexities, we use the well-known and widely appreciated metric, *cyclomatic complexity* by McCabe's [121]. It is a metric to calculate the complexity from a graph of system (especially software). The cyclomatic complexity is a graph metric that is proven useful for determining the complexity of computer programs when applied to their control and decision flow graphs. Mathematically, cyclomatic complexity is a number of regions or a number of linearly independent paths in the graph and calculated by the formula, $C = (e - n + 2)$, where ‘e’ stands for the number of edges and ‘n’ stands for the number of nodes in the graph. In this chapter, we demonstrate that “cyclomatic complexity is also applicable to systems that are represented as a network (or graph)”. Such result is non-obvious and non-trivial information about an evolving system.

We represent an evolving system as a set of evolving networks to retrieve graphlets. Collectively these graphlets are helpful to calculate the complexity of the evolving networks. We enhance the existing concept of cyclomatic complexity to calculate the system network complexity over time. To do this, we calculate and store the cyclomatic complexity (number of independent paths) of all possible graphlets. Usually, the cyclomatic complexity also considers the number of connected components, but in our case, cyclomatic complexity is calculated only for small-connected subgraphs (e.g., graphlets or motifs). The frequencies of graphlets are meaningful quantity that can be useful to calculate complexity for the network. Using cyclomatic complexity and subgraphs, we can quantify the complexities of an evolving system states. Every subgraph possesses a certain cyclomatic complexity. Thus, our technique is applicable to any system that can be represented as network.

Firstly, use the *graphlets information* of a state to calculate its System State Complexity.

Definition 5.8 System State Complexity (SSC): The SSC$_i$ is the complexity of a state Si in an evolving system. Numerically, it is the weighted arithmetic mean of cyclomatic complexities over the frequencies of all retrieved graphlets, such that a frequency represents the weight. Its formula is given by the following equation

$$\text{System State Complexity of } S_i\ (SSC_i) = \frac{\sum_{j=0}^{m}(freq_{ji} \times C_j)}{\sum_{j=0}^{m} freq_{ji}} \quad \text{-- (7)}$$

where C$_j$ denotes cyclomatic complexity and *freq*$_{ji}$ represents frequency for graphlet G$_j$ of state S$_i$ and *m* is count of retrieved graphlets. System State Complexities (SSCs) information is given by

$$\text{SSCs\_info} = < S_i, SSC_i > \mid 0 \leq i \leq N \quad \text{-- (8)}$$

Secondly, use the *NEGs information* to calculate the Evolving System Complexity.

Definition 5.9: *Evolving System Complexity* (ESC) is defined as the aggregated complexity of a state series SS for an evolving system. Numerically, it is the weighted arithmetic mean of cyclomatic complexities over the aggregate frequencies for all NEGs, such that an aggregate frequency represents the weight. Its formula is given by the following equation

$$\text{Evolving System Complexity} = \frac{\sum_{j=0}^{m'}(Aggregate\_freq_j \times C_j)}{\sum_{j=0}^{m'} Aggregate\_freq_j} \quad \text{-- (9)}$$

where, *Aggregate_freq*$_j$ denotes the *aggregate frequency* of a graphlet G$_j$ over time, C_j denotes the *cyclomatic complexity* of the graphlet G$_j$ and m is count of retrieved NEGs.

Mathematically, we can deduce the *average of non-zero SSCs* is equal to the ESC. Note the ESC is not equal to the average of SSCs. The SSC$_i$ calculates complexity of an individual state, whereas

ESC calculates complexity of a state series. We need both quantities because they express their own physical significance independent of each other.

An evolving system with less ESC has fewer interactions between its entities, whereas an evolving system with high ESC has many complex interactions between its entities. If an evolving system has many frequent subgraphs (i.e. NEMs) that have less complexity, then probably it has less ESC. If an evolving system has many frequent subgraphs (i.e. NEMs) that have high complexity, then probably it has high ESC. Using the ESC, we can quantitatively compare the complexity of two evolving systems of the same domain. Additionally, using the SSCs, we can quantitatively measure the changes between states.

5.4 Illustrative Example

In this section, we describe an illustrative example to describe definition and overall approach. To do local subgraph mining for a single state S_i in Figure 5.2, which consider an evolving network $EN_1 = (E_1, C_1)$ of state S_1, where E_1 denotes a set of evolving entities and C_1 denotes a set of directed evolving connections. Further, the output of pre-processing is adjusted automatically such that it can be acceptable (as input) by a subgraph mining algorithm. This means, we need coding to format each network according to the required input of the mining algorithm.

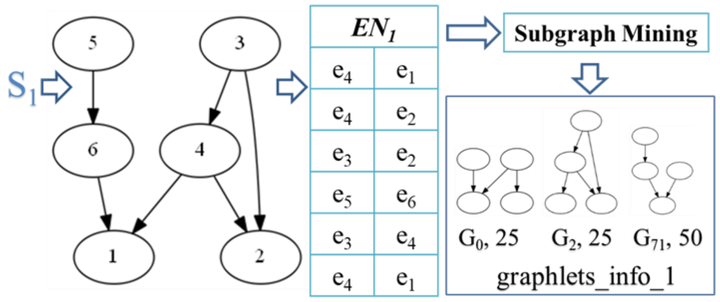


Figure 5.2 An overview of local subgraph mining for an evolving network EN_1 of state S_1.

Usually, subgraph (motif or graphlet) mining tool takes adjacency list (of network) shown as EN_1 in the Figure 5.2. In the figure, the local subgraph mining is applied on the evolving network EN_1 to retrieve graphlets (G_0, G_2, and G_{71}) with frequencies (25, 25, and 50) respectively. The frequency is given in percent occurrence of a graphlet in the formation of its network. For example, both the graphlets G_0 and G_2 are occurring 25% of times, additionally a graphlet G_{71} is occurring 50% of times. This means 'G_0' occurs 25% in network formation of state S_1 with node (6, 1, 4, 2). Similarly, the graphlets G_0 and G_{71} has 25% and 50% of occurrence in the network formation.

For a Network Evolution Graphlet (NEG), its *frequency* keeps on changing over states. Suppose three states $\{S_1, S_2, S_3\}$ where S_1 is already presented in the Figure 5.2. The Figure 5.3 highlights time series of 'G_{71}' for state S_1 with 50% frequency. In the Figure 5.3, for state S_2 and S_3 the graphlets information is [<G_{20}, 50>, <G_{30}, 25>, <G_{135}, 25>] and [<G_{20}, 20>, <G_{71}, 20>, <G_{30}, 20>, <G_{100}, 20>,

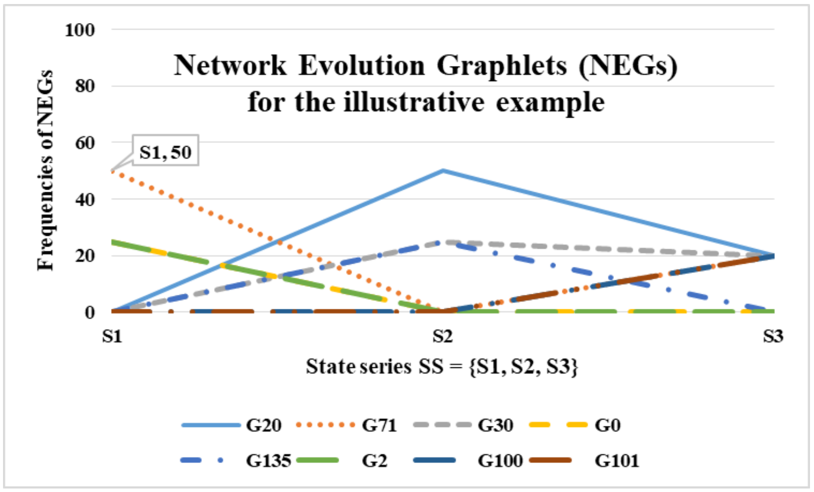


Figure 5.3 Time series graph describing *frequencies of NEGs* for a state series SS = {S_1, S_2, S_3}.

<G_{101}, 20>] respectively. Such graphlets information over states is useful to do system changeability and stability analysis because evolving frequencies are depicting the evolution happened in the system.

For the illustrative example in the Figure 5.2 and 5.3, the graphlets information of a state series also aids to calculate the *NEGs_info*. The *NEGs_info* is [<G_{20}, 23.33>, <G_{71}, 23.33>, <G_{30}, 15>, <G_0, 8.33>, <G_{135}, 8.33>, <G_2, 8.33>, <G_{100}, 6.66>, <G_{101}, 6.66>]. The NEMs is [<G_{20}, 23.33>, <G_{71}, 23.33>] if threshold frequency is 20%. The changeability and stability are 320 and 0.003125 respectively.

Now, we are extending the illustrative example (in Figure 5.4) for subgraph mining of three network states. In Figure 5.4, suppose a state S_1 is pre-processed to make an evolving network EN_1 =

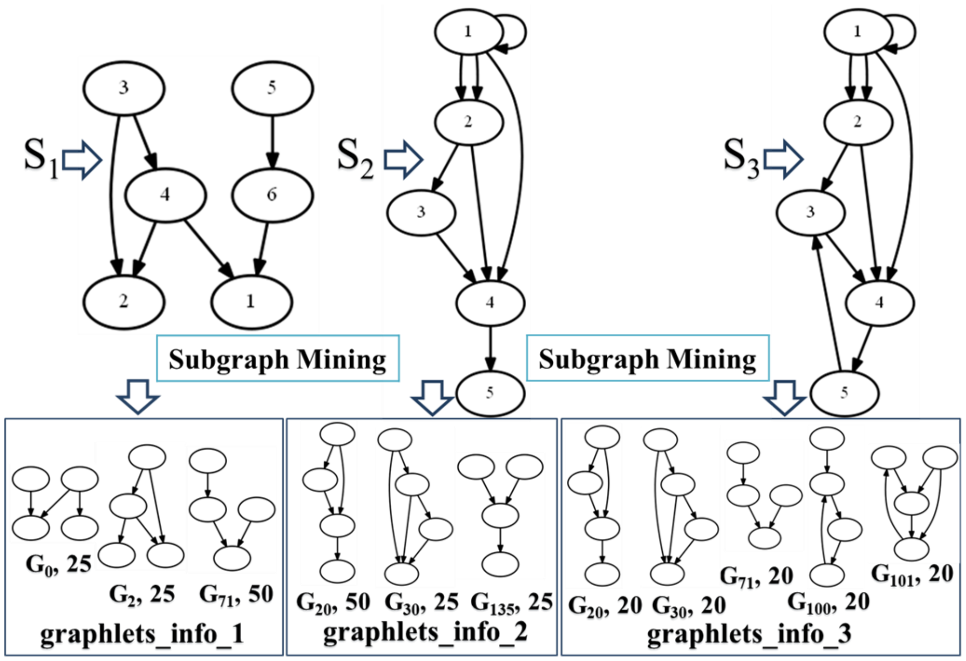


Figure 5.4 Illustrative example of subgraph mining for a state series SS = {S_1, S_2, S_3} represented as three evolving networks.

(E_1, C_1), where E_1 = {e_1, e_2, e_3, e_4, e_5, e_6} denotes a set of entities and C_1 = {(e_4, e_1), (e_4, e_2), (e_3, e_2), (e_5, e_6), (e_3, e_4), (e_6, e_1)} denotes a set of directed evolving connections. Here, (e_4, e_1) represent a directed edge from entity e_4 to e_1. Similarly, S_2 and S_3 are also pre-processed to make EN_2 and EN_3 respectively. Thereafter, we do subgraph mining on all three evolving networks locally.

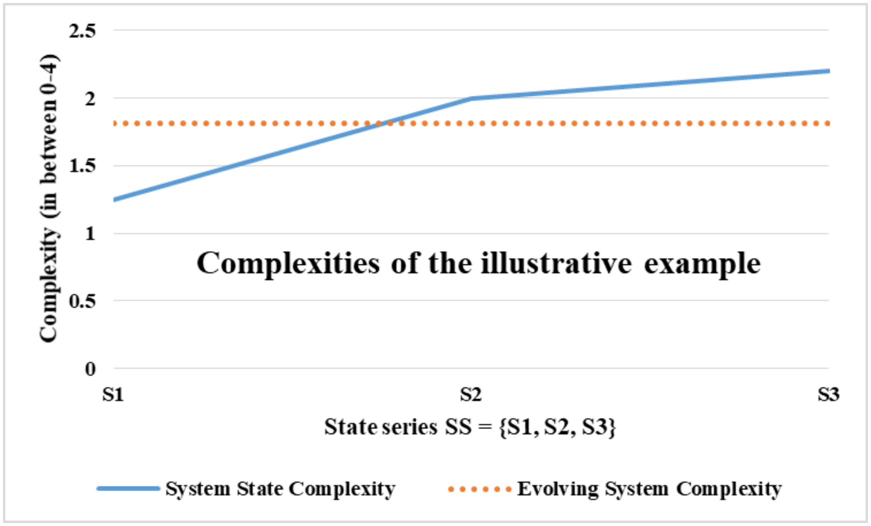


Figure 5.5 Illustrative example of complexity calculation for three states given in Figure 5.4.

In Figure 5.4, for state S_1 graphlets (G_0, G_2, and G_{71}) has cyclomatic complexity (C_0 = 1, C_2 = 2, C_{71} = 1) and frequency ($freq_{0,1}$ = 25, $freq_{2,1}$ = 25, $freq_{71,1}$ = 50). The resulting *system state complexity* for S_1 is calculated as (25 × 1 + 25 × 2 + 50 × 1) ÷ (25 + 25 + 50) = 1.25 also inferable in the time-varying series of Figure 5.5. The resulting *evolving system complexity* for state series SS = {S_1, S_2, S_3} is 1.816 also shown in the constant time-series of Figure 5.5.

In the following sections, we use the definitions and equations to describe the proposed approach and its applications on real-world evolving systems. Next two sections (5.5 and 5.6) describe algorithms to mine evolving networks to do SysEvo-Analytics.

5.5 Proposed: Mining Changeability Metric, Stability Metric, Network Evolution Graphlets and Network Evolution Motifs

In this section, we present an algorithm to mine changeability metric (CM), stability metric (SM), NEGs, and NEMs information for a given state series SS of an evolving system stored in a repository. Search space of our algorithm is a set of evolving networks {EN_1, EN_2… EN_N} for a state series SS = {S_1, S_2… S_N}, where N is the number of states for an evolving system.

The steps of our approach are based on the fundamental steps for knowledge discovery and data mining theory; these steps are shown in the Figure 5.6. First, pre-process a system state series to create evolving networks. Then, mine each network of a state to retrieve graphlets information. Thereafter, aggregate the evolving frequencies for the evolving networks to discover the information about NEGs as knowledge. Again, evolving frequencies is also useful to make a time-series graph over multiple states (as described in Figure 5.3). The evolving frequencies of graphlets can be used for the system's changeability and stability analysis.

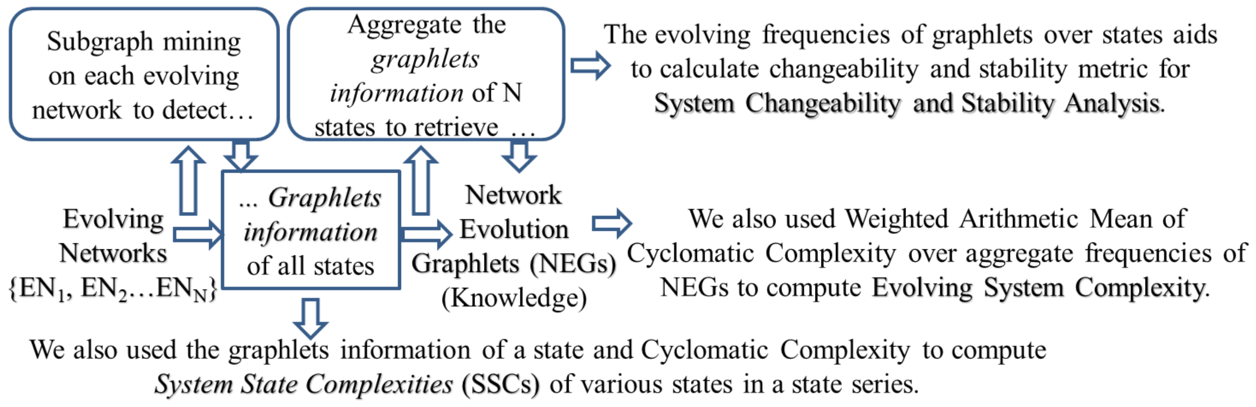

Subgraph mining on each evolving network to detect…
Aggregate the graphlets information of N states to retrieve …
The evolving frequencies of graphlets over states aids to calculate changeability and stability metric for System Changeability and Stability Analysis.
Evolving Networks {EN1, EN2…ENN}
… Graphlets information of all states
Network Evolution Graphlets (NEGs) (Knowledge)
We also used Weighted Arithmetic Mean of Cyclomatic Complexity over aggregate frequencies of NEGs to compute Evolving System Complexity.
We also used the graphlets information of a state and Cyclomatic Complexity to compute System State Complexities (SSCs) of various states in a state series.

Figure 5.6 An overview of framework presented showing combined steps of the *Algorithm mining_NEGs_NEMs_CM_SM*, the *Algorithm 2: mining_NEGs_SSCs* and the *Algorithm 3: Calculate_ESC.*

The *mining_NEGs_NEMs_CM_SM* algorithm takes N *evolving networks* (as input) corresponding to N states of an evolving system. Thereafter, the algorithm retrieves *changeability metric* (CM), *stability metric* (SM), *NEGs*, and *NEMs* (as output) in three sub-steps.

- First, the algorithm uses an existing subgraph-mining technique to retrieve the subgraph information for each state. The subgraph-mining technique processes each EN_i to detect induced subgraphs (or graphlets) information. The local subgraph mining retrieves graphlets with their frequencies as given in equation (1). We refer graphlets information for state S_i as *graphlets_info_i*.

- Second, the algorithm aggregates the *graphlets_info* of all N states by calculating the *aggregate frequency* of each NEG, according to the equation (2). The algorithm output the retrieve NEGs information according to equation (3). The NEGs with their *Aggregate_freq* greater than a user defined threshold (ϴ) can be selected as NEMs. Thus, the algorithm is useful to identify the NEGs and NEMs of interconnected evolving entities during system evolution.

Algorithm ***mining_NEGs_NEMs_CM_SM*** **(***evolvingNetworks***)**

```
Initialize $j \in$ integer 0 to M graphlets
Initialize $i \in$ integer 1 to N states
Initialize HashMap graphlets_info $< G_j, freq_{ji} >$
Initialize HashMap NEGs_info $< G_j, Aggregate\_freq_j >$
Initialize CM = 0 as the changeability metric value

For each state $EN_i$ in evolvingNetworks
    graphlets_info $< G_j, freq_{ji} >$ = subgraphMining($EN_i$)
End For
Suppose $m'$ is the count of retrieved graphlets over all states
```

```
For each graphlet $G_j$ out of $m'$ graphlets, where $1 \leq j, m' \leq M$
    Initialize float frequencySum = 0
    Initialize integer $Aggregate\_freq_j = 0$
    For each graphlets_info_i of $S_i$ where i varies from 1 to N
        frequencySum = frequencySum + $freq_{ji}$
    End For
    $Aggregate\_freq_j = frequencySum \div N$
    Add tuple < $G_j$, $Aggregate\_freq_j$ > to NEGs_info
    NEGs with their $Aggregate\_freq \geq$ threshold $\Theta$ are NEMs.
End For
For each of graphlet $G_j$ of $m'$ graphlets, where $1 \leq j, m' \leq M$
    Initialize $\frac{\Delta F_j}{\Delta i} = 0$, $\frac{\Delta F}{\Delta i} = 0$
    For each epoch between state i and (i+1), where i varies form 1 to N
        $\frac{\Delta F}{\Delta i} = \left|\frac{d\,F(i, freq_{ji})}{d\,i}\right| = \left|\frac{freq_{j(i+1)} - freq_{ji}}{(i+1) - i}\right|$
        $\frac{\Delta F_j}{\Delta i} = \frac{\Delta F_j}{\Delta i} + \frac{\Delta F}{\Delta i}$
    End For
    CM = CM + $\frac{\Delta F_j}{\Delta i}$
End For
SM = (1/CM)
Return NEGs_info, NEMs, CM, and SM
```

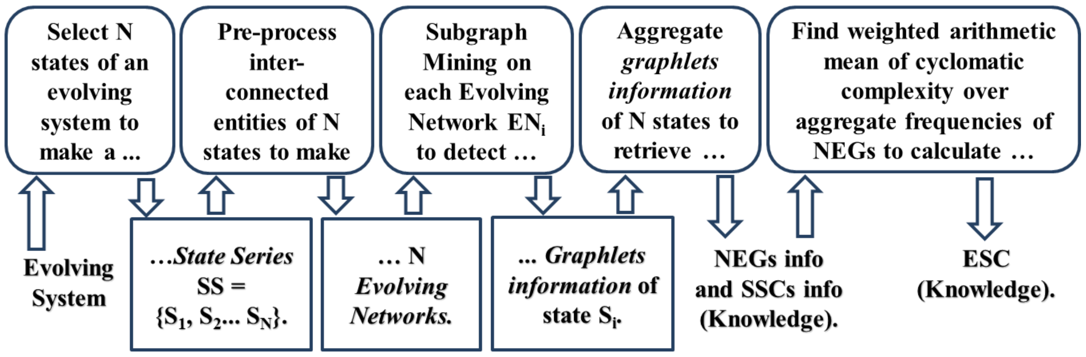
- Third, the algorithm calculates the changeability metric (CM) and stability metric (SM) based on the equation (5) and equation (6) respectively. Initially, the algorithm calculates the gradient for *j*th graphlet frequencies over the state series. Then, these gradients are added over all the retrieved graphlets, this result in the changeability metric (CM). In the last, stability metric (SM) is simply inverse of the CM.

Based on the approach presented in this section, we developed a prototype tool that we used to do experiments. Section 5.8 discusses experiments using this approach.

5.6 Proposed: System Network Complexity (SNC)

This section presents *System Network Complexity* (SNC) that analyses pre-evolved state series SS of an evolving system. The SNC aims to analyze the evolving graphlets and complexities between inter-connected entities of an evolving system. The SNC is a novel approach, which retrieves *Network Evolution Graphlets information*, *System State Complexities information*, and *Evolving System Complexity*. Functionally, the discovered *Network Evolution Graphlet* (NEG) is a subgraph pattern of inter-connected entities occurring in the evolving networks over time. Functionally, a *System State Complexity* (SSC) represents complexity of a single state.

Functionally, an *Evolving System Complexity* (ESC) represents aggregate complexity for a state series SS of an evolving system. The NEGs, SSCs, and ESC can be retrieved with our proposed SNC algorithm. Search space of SNC is an evolving system that has N evolving states as a state series, SS = {S_1, S_2… S_N}, stored in a repository. All the steps in the SNC are according to the guidelines of knowledge discovery and data mining – as shown in the Figure 5.7. The Figure 5.8 shows the flow chart of the *Algorithm SNC* that integrates the subgraph mining and evolution mining to discover subgraph evolution information. The SNC outputs evolution information about the network evolution subgraphs in the form of NEGs and SSCs information. Further, the SNC uses the NEGs information to calculate ESC.



Select N states of an evolving system to make a ...
Pre-process inter-connected entities of N states to make
Subgraph Mining on each Evolving Network EN_i to detect …
Aggregate graphlets information of N states to retrieve …
Find weighted arithmetic mean of cyclomatic complexity over aggregate frequencies of NEGs to calculate …
Evolving System
…State Series SS = {S1, S2... SN}.
… N Evolving Networks.
... Graphlets information of state Si.
NEGs info and SSCs info (Knowledge).
ESC (Knowledge).

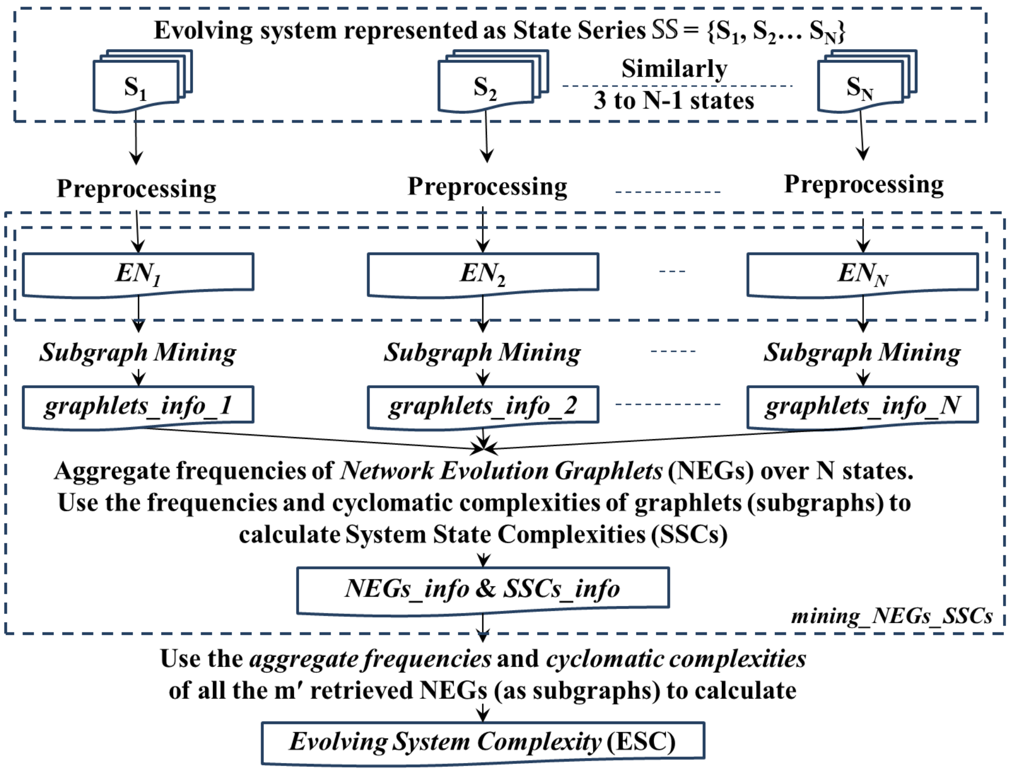
Figure 5.7 An overview of the process-artifact interaction, where each rounded-rectangle shows a process and rectangle shows artifact created after the process. Each sentence starts from top and finishes at the artifact box. Where, the upward arrow denotes artifact goes to a process and downward arrow denotes artifact is generated from a process.

Next, we will present the three steps of SNC in detail to ensure their reproducibility. *Algorithm 5.1* ***Preprocess*** uses the N states of an evolving system stored in a repository such that each state S_i is pre-processed to make an EN_i, where *i* represent a state number varying from 1 to N. The pre-processing creates N evolving networks for N states.

The *Algorithm 5.2* ***mining_NEGs_SSCs*** takes N *evolving networks* (as input) corresponding to the N states of an evolving system. The algorithm processes N *evolving networks* one by one i.e., each EN_i is processed locally, where ‘i’ varies from 1 to N. Thereafter, the algorithm retrieves *NEGs information* (as output) in two sub-steps.

- First, the subgraph mining algorithm processes each EN_i to detect graphlets information (*graphlets_info_i*) according to equation (1). To do so, we used HashMap that store pairs of (key, value) such that a set contains unique key and its value. A subgraph mining technique can be used as ***subgraphMining***, which we described in Section 5.2.

- Second, the algorithm aggregates the *graphlets_info* of a state series. Let there are m retrieved distinct graphlets over all the states. Thereafter, calculate the *aggregate frequency* of each graphlet over states using equation (2). This results in the *NEGs_info* as presented in equation (3).



Evolving system represented as State Series SS = {S1, S2… SN}
S1
S2
Similarly 3 to N-1 states
SN
Preprocessing
Preprocessing
Preprocessing
EN1
EN2
ENN
Subgraph Mining
Subgraph Mining
Subgraph Mining
graphlets_info_1
graphlets_info_2
graphlets_info_N
Aggregate frequencies of Network Evolution Graphlets (NEGs) over N states.
Use the frequencies and cyclomatic complexities of graphlets (subgraphs) to calculate System State Complexities (SSCs)
NEGs_info & SSCs_info
mining_NEGs_SSCs
Use the aggregate frequencies and cyclomatic complexities of all the m′ retrieved NEGs (as subgraphs) to calculate
Evolving System Complexity (ESC)

Figure 5.8 An overview of flow-chart for the system network complexity.

Thereafter, the algorithm calculates N *system state complexities* for all N states. The frequencies of retrieved graphlets (in *graphlets_info_i*) and the cyclomatic complexities of all M graphlets (in array C[M]) are used to calculate *system state complexity* for each state (S_i). To do this, the array with pre-calculated cyclomatic complexity (C_j) is used with frequency ($freq_{ji}$) of graphlet (G_j) according to the equation (7). The SSCs of all the N states are stored in a hash-map *SSCs_info* according to the equation (8).

The *Algorithm 5.3* ***Calculate_ESC*** uses - the aggregate frequency (Aggregate_freq*j*) and cyclomatic complexity (C*j*) of *j*th NEG (G*j*) to calculate the ESC - over all m retrieved NEGs. We assign pre-calculated cyclomatic complexity (C*j*) for all graphlets (G*j*) in an array (C[M]). Firstly, for all the m retrieved NEGs calculate "the *sum of products* between all cyclomatic complexities and aggregate frequencies. Second, for all the NEGs, also calculate "the *sum of frequencies* of all the m retrieved NEGs". Divide the *sum of products* by the *sum of frequencies* to calculate the ESC, according to the equation (9).

```
Algorithm SNC(repository)
Initialize i ∈ integer 1 to N
Retrieve N states of a state series SS = {S1, S2… SN} stored in repository
1. evolvingNetworks = Preprocess(repository)
2. NEGs_info & SSCs_info = mining_NEGs_SSCs(evolvingNetworks)
3. ESC = Calculate_ESC(NEGs_info)
```

Algorithm 5.1 ***Preprocess**(repository)*

```
For each state $S_i$ where $i \in$ integer 1 to N
    Detect an evolving network of inter-connected entities in $S_i$
    //Evolving network stores <sourceEntityIDs, targetEntityIDs>
    Denote the evolving network as $EN_i$
End For
Return evolvingNetworks
```

Algorithm 5.2 ***mining_NEGs_SSCs**(evolvingNetworks)*

```
Initialize $j \in$ integer for graphlet $G_j$ such that $0 \leq j \leq M$
Initialize $i \in$ integer varying from 1 to N states
Initialize HashMap graphlets_info< $G_j$, $freq_{ji}$ >
Initialize HashMap NEGs_info< $G_j$, $Aggregate\_freq_j$ >
Initialize HashMap SSCs_info< $S_i$, $SSC_i$ >
Where, $G_j$ is $j^{th}$ graphlet, $freq_{ji}$ is frequency of $G_j$ of state $S_i$, and $Aggregate\_freq_j$ is aggregate
frequency of NEG $G_j$ over a state series.
For each $EN_i$ in evolvingNetworks where $i$ varies from 1 to N
    graphlets_info_i < $G_j$, $freq_{ji}$ >= networkSubgraphMining($EN_i$)
End For

Suppose $m'$ is count of retrieved graphlets over all states

For each graphlets $G_j$ out of $m'$ graphlets, where $1 \leq j, m' \leq M$
    Initialize float frequencySum = 0
    Initialize integer $Aggregate\_freq_j$ = 0
    For each graphlets_info_i where $i$ varies from 1 to N
        frequencySum = frequencySum + $freq_{ji}$
    End For
    $Aggregate\_freq_j$ = frequencySum ÷ N
    Add tuple < $G_j$, $Aggregate\_freq_j$ > to NEGs_info
    End For
    Initialize cyclomatic complexity array C[M] for all graphlets
    For each graphlets_info_i of $S_i$ where $i$ varies from 1 to N
        Initialize float frequencySum = 0
        Initialize float sumOfProducts = 0
        For each $G_j$ in graphlets_info_i
            frequencySum = frequencySum + $freq_{ji}$
            sumOfProducts = sumOfProducts + { $freq_{ji} \times C_j$ }
            // where Cyclomatic complexity $C_j$ for graphlet $G_j$ at C[j]
        End For
    Float $SSC_i$ = sumOfProducts ÷ frequencySum
    Add tuple < $S_i$, $SSC_i$ > to SSCs_info
```

```
End For
Return NEGs_info & SSCs_info
```

Algorithm 5.3 **Calculate_ESC(***NEGs_info***)**

```
Let $m'$ is the number of retrieved graphlets in NEGs_info
Initialize $j \in$ integer for graphlets such that $0 \leq j \leq M$
Initialize float sumOfProducts = 0
Initialize cyclomatic complexity array C[M] for all graphlets
$G_j \in j^{th}$ NEG in NEGs_info
Initialize $Aggregate\_freq_j = 0$
$Aggregate\_freq_j \in$ aggregate frequency of $G_j$ in NEGs_info
For each NEGs $G_j$ over all states
  sumOfProducts = sumOfProducts + { $Aggregate\_freq_j \times C_j$ }
  //where Cyclomatic complexity $C_j$ for graphlet $G_j$ at C[j]
  frequencySum = frequencySum + $Aggregate\_freq_j$
End For
Float ESC = sumOfProducts ÷ frequencySum
Return ESC
```

The computational complexity of the SNC algorithm can be given in the following way. Firstly, the computational complexity of *Algorithm 5.1 Preprocess* is dependent upon the system domain and technique for pre-processing system state to make evolving networks. Secondly, the computational complexity of the *Algorithm 5.2 mining_NEGs_SSCs* is $O(N \times \lambda) + 2O(N \times m)$, where N is number of states, m is the number of retrieved NEGs over all states, and λ is the complexity of subgraph mining algorithm. Thirdly, the computational complexity of *Algorithm 5.3 Calculate_ESC* is O(m). Thus, the SNC complexity mainly depends upon *Algorithm 5.2 mining_NEGs_SSCs*, which majorly depends on $O(N \times \lambda)$ i.e., number of states (N) times complexity of subgraph mining (λ). Based on the proposed SNC algorithm we developed a prototype tool, which is described in the next section.

5.7 SNC-TOOL

Based on the SNC algorithm, we developed a Java-based tool named as SNC-Tool, which is executable on machine enabled with JRE and JDK 7th version or higher. There are many existing subgraph mining algorithms and open source tools. In *SNC-Tool,* we used the *acc-Motif* algorithm (and tool) [179] as subgraph mining algorithm developed by Meira *et al.* [178], which detects accelerated motif (acc-Motif) using combinatorial techniques. We opted for the acc-Motif because it is an efficient open-source tool, which is also implemented in Java (the language opted for our SNC-Tool). Additionally, the acc-Motif builds upon old tools and techniques. The acc-Motif has following three computational time complexities $\lambda = O(a(G)e)$, $O(e_2)$, and O(ne) for the detection of subgraph size 3, 4 and 5 in directed graphs, where a(G) is the arboricity of graph G(V, E), n is |V| vertices, and e is |E| edges. The *SNC-Tool* has three components '*PreProcessing*', '*Mining_NEGs_SSCs',* and

‘*EvolvingSystemComplexity*’. First component is based on *Algorithm 5.1 Preprocess*(*repository*). Second component is based on the proposed *Algorithm 5.2 mining_NEGs_SSCs*(*evolvingNetworks*). Third component is based on *Algorithm 5.3 Calculate_ESC*(*NEGs_info*). The next section describes the use of SNC-Tool by doing experimentation on six different evolving systems.

5.8 Experimentations

This section is divided into two subsections. First subsection demonstrates expertimenations for stability and changeability analysis. Second subsection demonstrates experimentations for system network complexity analysis.

5.8.1 Experimentations for System Stability and Changeability analysis

This subsection demonstrates six experiments using our tool on six open-internet based real-world evolving systems. For each evolving system, we discuss a detailed application of the *mining_NEGs_NEMs_CM_SM* algorithm over the state series. The evolving system details are mentioned in the first four columns of the Table 5.1. We pre-processed each evolving system’s state series of N states into a set of N evolving networks. Each evolving network is a file such that each line contains two entities (as nodes) to represent a directed connection from the source entity to the target entity of a system state. The total number of entities in all the states is mentioned in the third column. The average number of neighbours (entities) is a commonly used graph property. We find average number of entities for each network of a state series, and then calculated its average over state series (as mentioned in the fourth column). The statistic of average number of entities is useful to understand density of entity connections in a set of evolving networks.

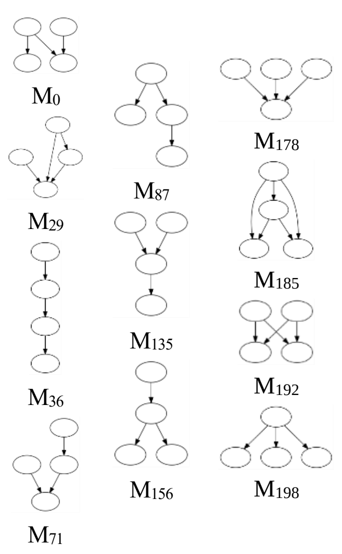
We used our implemented Java-based tool for *mining_CM_ SM_NEG*. The tool internally uses *acc-Motif* as a *subgraphMining* algorithm developed by Meira *et al.* [178], which detects induced subgraphs (i.e. graphlets). While local subgraph mining (using *acc-Motif* [179]), we used following four parameters to fine-tune the subgraph information retrieval. We used following values of the parameters: *subgraph-size* = 4, the number of random networks = 1000, the number of exchanges per edge = 3, and the number of exchange attempts = 3. Although the experiments are also possible with *subgraph-size* 3, 5, and 6, but to keep brevity we present results solely for *subgraph-size* = 4. Because the *subgraph-size* = 3 yields almost constant frequencies, and the *subgraph-size* = 5 yields many graphlets, which are hard to manage (i.e. tabulate and visualize). There are 199 graphlets {G_0, G_1 … G_{198}} for motif-4 (*subgraph-size*) in the *acc-Motif*, which are manageable number of graphlets.

We retrieved NEGs with their aggregate frequencies, which are mentioned in the fifth column of Table 5.1. We show few frequent evolution subgraphs as NEMs with high frequencies as shown in the last column of Table 5.1. For each evolving system, we kept simple threshold to choose NEMs from the NEGs. The threshold frequency for NEMs selection is the frequency mentioned in the last pair of system's NEGs information. Here, G_0 corresponds to M_0, G_1 corresponds to M_1, and so on.

The NEMs mentioned in the Table 5.1 have following inferences for their domains. For software system, four NEMs (M_{178}, M_{135}, M_0, and M_{71}) have a collective inference that mostly source node reaches to a common target node. This also happens when a program terminates at common end (example return, exception, or graphical output). For natural language systems: the Bible Translation has five NEMs (M_{198}, M_{192}, and M_{185}); and the Multi-sport Events has four NEMs (M_{178}, M_{192}, and M_{185}). All these NEMs are similar in structure, which infers either root or leaf node(s) are connected to multiple nodes. For frequent market basket, there are many kinds of NEMs (M_{178}, M_0, M_{71}, M_{87}, M_{29}, M_{135}, M_{36}, and M_{156}). This means, usually retail market system remains highly dynamic (or chaotic) as compared to other systems. For both - Positive and Negative sentiment in IMDb movie name-genre system - there are four NEMs (M_{178}, M_0, M_{198}, and M_{192}). Having same NEMs means both systems have almost similar structure, which infers root or leaf node(s) are connected to multiple nodes.

We show the result of the six experiments as a time series graph in Figure 5.9, 5.10, 5.11, 5.12, 5.13, and 5.14; where the horizontal axis represents the N states (each instance as a state) and vertical axis represents the percentage NEG frequency. This helps to study stability for the changing frequencies of NEGs over a state series SS. Each figure contains many time series that represents retrieved NEGs with their frequencies over time. In the figures, the NEGs time series with higher frequencies are visible, but some are invisible due to overlapping. Next, we describe six experiments

Table 5.1 Experimental results of Mining NEGs and NEMs for six Evolving systems.

Evolving Systems	**N**	**Number of entities**	**Average number of neighbours (entities)**	**Network Evolution Graphlets Information (in decreasing frequencies)**	**Network Evolution Motifs (or frequent evolution subgraphs)**
Hadoop HDFS-Core[1]	15	3129	2.166	<G_{178}, 81.59>, <G_{135}, 11.02>, <G_0, 4.46>, <G_{71}, 0.83> and other 20 NEGs.	 M0 M87 M178 M29 M185 M135 M36 M192 M156 M198 M71
Bible Translation[2]	13	246	1.456	<G_{198}, 44.83>, <G_{192}, 15.59> <G_{185}, 7.92> and other 14 NEGs	
Multi-sport Events[3]	13	141	1.786	<G_{178}, 70.59>, <G_{192}, 9.89>, <G_{185}, 2.94> and other 7 NEGs.	
Frequent Market Basket[4]	13	118	8.002	<G_{178}, 17.79>, <G_0, 15.92>, <G_{71}, 11.83>, <G_{87}, 8.47>, <G_{29}, 5.87>, <G_{135}, 5.61>, <G_{36}, 4.89>, <G_{156}, 4.94> and other 26 NEGs.	
Positive sentiment[6] in IMDb movie names-genre[5]	16	284	2.661	<G_{178}, 70.59>, <G_0, 9.89>, <G_{198}, 2.88> and <G_{192}, 2.47>	
Negative sentiment[6] in IMDb movie names-genre[5]	16	510	3.303	<G_{178}, 70.59>, <G_0, 9.89> <G_{198}, 4.72>, <G_{192}, 1.62> and other 16 NEGs.	

1. https://mvnrepository.com/artifact/org.apache.hadoop/hadoop-hdfs 2016.
2. https://en.wikipedia.org/wiki/List_of_English_Bible_translations Oct 2016.
3. https://en.wikipedia.org/wiki/List_of_multi-sport_events Oct 2016.
4. https://archive.ics.uci.edu/ml/datasets/Online+Retail Oct 2016.
5. http://www.imdb.com/interfaces/ Oct 2016.
6. https://www.cs.uic.edu/~liub/FBS/sentiment-analysis.html Oct 2016

on the six evolving systems.

Evolving system 1, we pre-processed 15 jars of Hadoop-HDFS-Core to produce 15 *call graphs* (networks) for 15 versions (states). In a *call graph*, caller procedure (as source entity) calls callee procedure (as target entity). Thereafter, we mined NEGs on the 15 call graphs of 15 versions. The NEGs are the procedure call graphlets over versions. As a state of software is referred as a software version, thus a state series of software can be referred as a version series. The time series graph in Figure 5.9 shows 24 time series representing 24 NEGs for 15 versions as a version series.

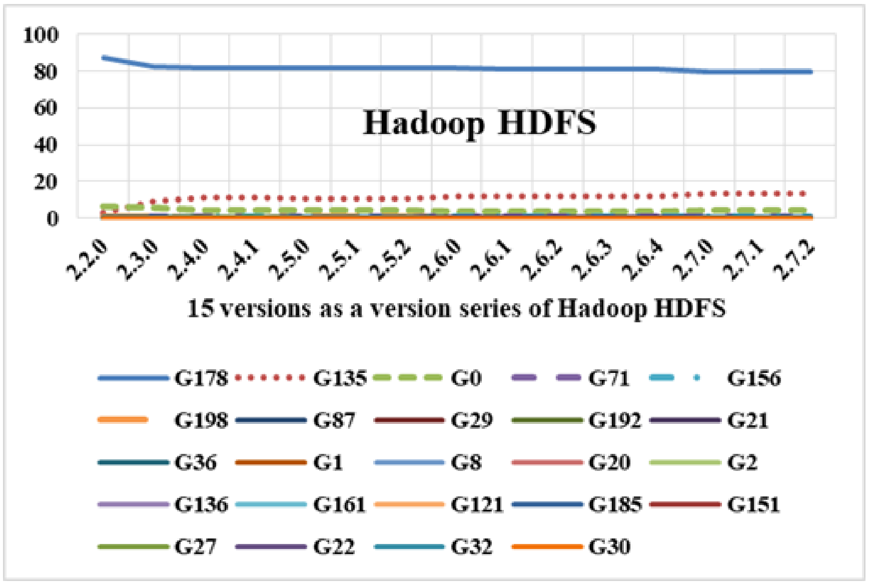


Figure 5.9 The time series for frequencies of NEGs over 15 versions (states) as a version series of the Hadoop HDFS-core system.

Evolving system 2, list of Bible translations, contains short information about all the bible translation done since 7th century until 2014. The translations are from "source biblical languages" like

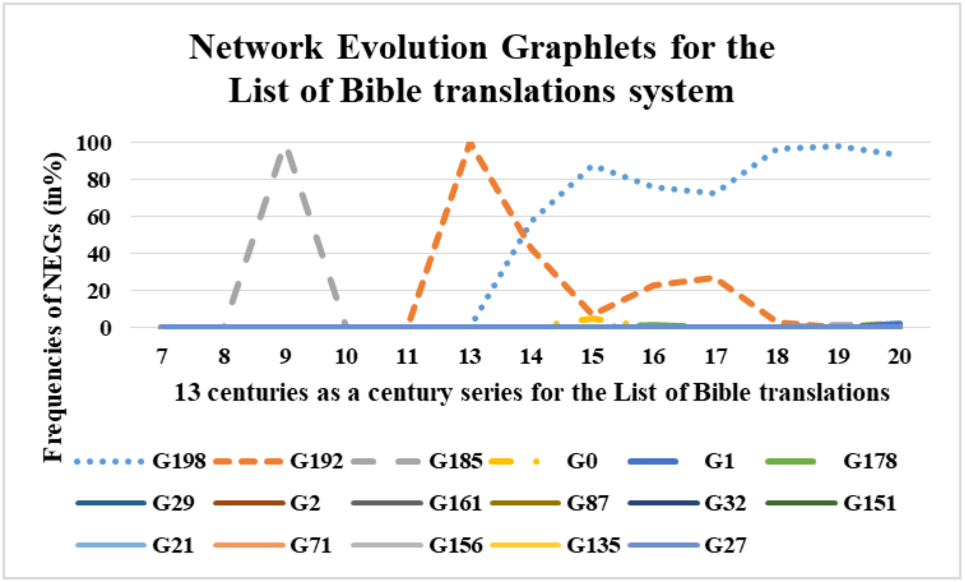


Figure 5.10 The time series for frequencies of NEGs over 13 centuries (states) of the List of Bible translation system.

Hebrew, Aramaic, and Greek etc. to the various "English variant languages" like Modern English, Old English etc. We pre-processed the dataset to make evolving networks from connections between words (as entities) in columns of 'English variant' and 'Source biblical language'. The words in the 'English variant' are chosen as source entities and the words in the 'Source biblical language' are chosen as target entities. This aids us to make 13 evolving networks for 13 centuries. Thereafter, we mined NEGs on such word networks of 13 centuries. As single state is for a century, thus a state series of this system can be referred as a century series. The time series graph in Figure 5.10 shows 17 time series representing 17 NEGs for 13 centuries as 13 states.

Evolving system 3, list of multi-sport events, contains short information about all the multi-sports events happened since 1890's until 2015. We pre-processed the dataset to make evolving networks from connections between words (as entities) in two columns: 'Title' (name) and 'Scope' (regional, international, and provinces) of an event. The words in the 'Title' are chosen as source entities and the words in the 'Scope' are chosen as target entities. This aids us to make 13 evolving networks for 13 decades. Thereafter, we mined NEGs on such word networks of 13 decades. As a single state is for a decade, thus a state series of this system can be referred as a decade series. The time series graph in Figure 5.11 shows 10 time series representing 10 NEGs for 13 decades as 13 states.

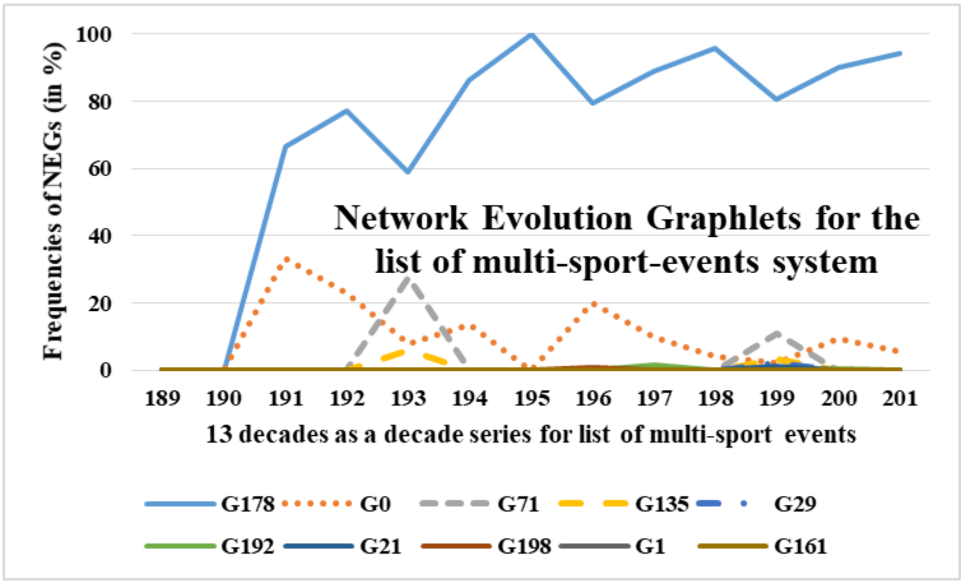


Figure 5.11 The time series for frequencies of NEGs over 13 decades (states) as decade series of the List of multi-sport events system.

Evolving system 4, evolving retail market system, a dataset of retail market [170] is available on UCI Repository. The dataset contains all the transactions occurring between 01/12/2010 and 09/12/2011 for a UK-based online retail (non-store). Initially, we made a frequent market basket from column 'Description' of the product, such that we set threshold to select a product sold atleast 10 months. Thereafter, in each line of the 'Description', we used the sequence (as a connection) between two words (as entities) to make an evolving network (for each month). This aids us to make 13 evolving networks for 13 months. Thereafter, we mined NEGs on such product description networks of 13 months. As a single state is for a month, thus a state series of this system can be referred as a month series. The time series graph in Figure 5.12 shows 8 time series representing 8 NEGs for 13 months as 13 states. There are total 34 NEGs retrieved; few significant NEGs (i.e. NEMs) are shown in Figure 5.12 and Table 5.1.

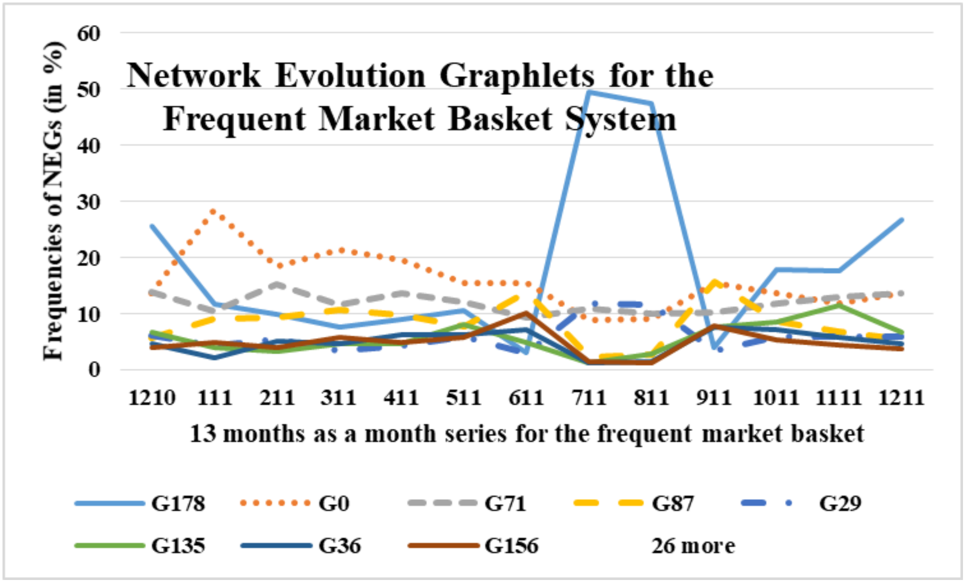


Figure 5.12 The time series for frequencies of NEGs over 13 months (states) as month series of the frequent market basket state series.

Evolving system 5 and 6 is based on evolving IMDb movie name-genre system. We collected the movie name and genre data of IMDb for 16 decades since 1870's until Oct 2016. The subset of the IMDb dataset is available online. We used two list of positive and negative sentiment words, which was created and used by Minqing and Bing *et al.* [171][172]. As a single state is for a decade, thus a state series of this system can be referred as a decade series.

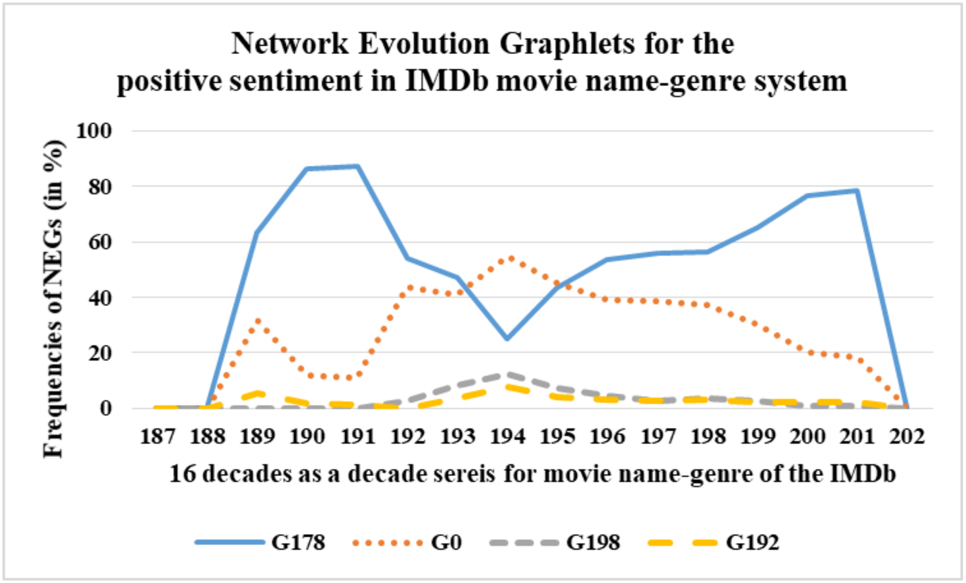


Figure 5.13 The time series for frequencies of NEGs over 16 decades (states) of the positive sentiment in IMDb movie names-genre system.

Evolving system 5: We used the positive sentiment words in movie names (as source entities) and their genres (as target entities) to generate evolving networks. While using movie names, we specifically used only positive word(s) in each name. The Figure 5.13 shows experimental results for connections between entities: 'positive words in movie name' and 'genres' in the IMDb dataset. The figure shows 4 time series representing 4 NEGs for 16 decades as 16 states.

Evolving system 6: We used the negative sentiment words in movie names (as source entities) and their genres (as target entities) to generate the evolving networks. While using movie names, we specifically used only negative word(s) in each name. The Figure 5.14 shows experimental results for connections between entities: 'negative words in movie name' and 'genres' in the IMDb dataset. The figure shows 20 time series representing 20 NEGs for 16 decades (states).

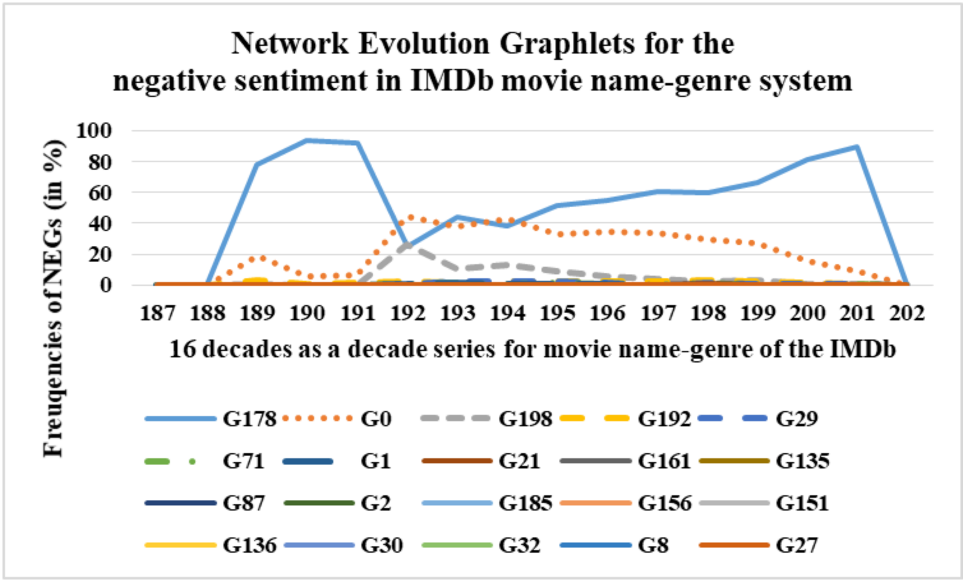


Figure 5.14 The time series for frequencies of NEGs over 16 decades (states) of the negative sentiment in IMDb movie names-genre system.

Next, we calculated the equation 4 and 5 for evolving frequencies of graphlets retrieved over a state series. The results are mentioned in the Table 5.2. We derived following inferences from the time series graphs and the Table 5.2.

1. System evolution can be analysed by measuring changeability and stability between two states.

2. The changeability and stability observations over a state series are as follows.

– From Figure 5.9, we observed that time series are almost constant (or stable) due to low variation in the frequencies of NEGs. Thus, we inferred that the Hadoop HDFS-Core system is relatively stable as compared to other five evolving systems.

– From Figure 5.10, we observed that time series are highly varying due to high variation in the frequencies of NEGs. Thus, we inferred that the List of Bible Translation system has relatively underwent more changes as compared to other five evolving systems.

– From Figure 5.11, 5.12, 5.13, and 5.14, we observed that time series are moderately varying due to moderate variation in the frequencies of NEGs. Thus, we inferred that these four systems are moderately change (or moderately stable) as compared to other system.

The above three observations and inferences are also true in case of changeability and stability values given in Table 5.2. From Table 5.2, we directly infer that software system (HDFS-Core) is stable and less changeable as compared to other five systems. Whereas, we directly infer that the List of Bible Translation system is less stable and more changeable.

Table 5.2 Experimental results of Changeability and Stability Metrics for the six Evolving systems.

Evolving Systems	Changeability metric	Stability Metric
Hadoop HDFS-Core	25.66	0.03896
List of Bible Translation	1695.20	0.000589
List of Multi-sport Events	468.46	0.00213
Frequent Market Basket	538.11	0.00185
Positive sentiment in IMDb movie name-genre	571.06	0.00175
Negative sentiment in IMDb movie name-genre	552.59	0.00180

3. Using the proposed approach, we can compare two evolving systems, which have same pre-processing technique to make evolving network series. Thus, the two evolving network series - negative and positive sentiment systems - are comparable to each other based on the time-series plots for *frequencies of NEGs* in the Figure 5.13 and 5.14. In these figures, the first four graphlets are same with slightly different evolving frequencies. In Table 5.2, we can see negative sentiment system has slightly less value of changeability metric as compared to positive sentiment system. Thus, the stability metric value of the negative sentiment system is slightly more than the positive sentiment system. Thus, the negative sentiment system is slightly more stable as compared to the positive sentiment system.

5.8.2 Experimentations for System Network Complexity analysis

This subsection describes an empirical evaluation of SNC-Tool for evolving systems available on open internet repositories of four domains: software, natural language, retail market, and IMDb. We also discuss a detailed application of SNC for each domain and demonstrate experiments (using SNC-Tool) on six evolving systems. Description of the Table 5.3 is as follows. First column contains name of the system used in the experiment. Second column contains number of states used in the experimentation. Third column contains the number of entities. Fourth column contains *average number of neighbours* (entities), it is a commonly used property of network or graph. Fifth column contains number of NEGs. Sixth column contains calculated value of Evolving System Complexity (ESC).

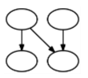
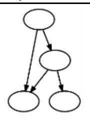
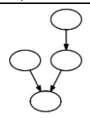
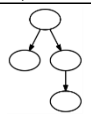
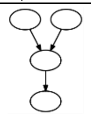
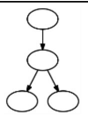
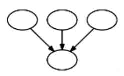
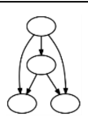
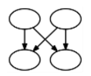
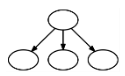
Evaluation of NEGs: We retrieved NEGs information for each evolving system using the proposed mining algorithm that uses the set of evolving networks along with input parameters. We used parameters to fine-tune the retrieval of subgraph according to the desired accuracy. Although in each experiment we retrieved many NEGs, but we present significant NEMs (network evolution motifs i th high frequencies) in the Table 5.4, which also includes complexities of those subgraphs (size 4).

Table 5.3 Experimental results on the Evolving systems.

Evolving Systems	N	# entities	Average # neighbour	# NEGs	ESC*
Hadoop-HDFS[1]	15	3129	2.166	24	1.0054
Bible Translation[2]	13	246	1.456	17	1.456
Multi-sport Events[3]	13	141	1.786	10	1.00487
Frequent Market Basket[4]	13	118	8.002	34	1.33888
Positive sentiment[6] **of movie genres**[5]	16	284	2.661	4	1.03050
Negative sentiment[6] **of movie genres**[5]	16	510	3.303	20	1.03683

# stands for "number of". *Range of ESC is in between 1 to 4 because cyclomatic complexity of subgraph-size (subgraph-4) is in between 1 to 4.

Table 5.4 Most significant retrieved NEMs (subgraphs).

M_0, $C_0 = 1$	M_2, $C_2 = 2$	M_{71}, $C_{71} = 1$	M_{87}, $C_{87} = 1$	M_{135}, $C_{135} = 1$
				
M_{156}, $C_{156} = 1$	M_{178}, $C_{178}=1$	M_{185}, $C_{185}=3$	M_{192}, $C_{192}=2$	M_{198}, $C_{198} = 1$
				

Evaluation of SSCs and ESC: We used the retrieved information about subgraphs of an evolving system to calculate its complexity. Using *frequencies* and *cyclomatic complexities* of the graphlets (in *graphlets_info*), we computed SSC for each state according to the equation (7). Using *aggregate frequencies* and *cyclomatic complexities* of the *NEGs*, we computed ESC for a state series according to the equation (9). We calculated ESC for all evolving systems, which are given in Table 5.3. Since every *NEG* (subgraph) has exactly 4 nodes and all such subgraphs have complexity in between 1 to 4. Therefore, SSCs and ESC values will range from 1 to 4. Low complexities of retrieved frequent subgraphs results in low SSCs and ESC. High complexities of retrieved frequent subgraphs results in high SSCs and ESC.

The six experimental results are shown as six time series plots. In each plot, the x-axis represents state series SS of N states where each instance is a state (S_i) and y-axis represents the complexity of each state. In the time series plots (of Figures 5.15, 5.16, 5.17, 5.18, 5.19, and 5.20), there are two time series for various states. First, the time-varying series shows the *system state complexity* (SSC) of each states. Second, the constant time series shows the ESC of the whole system state series. For example, to read the time series in Figure 5.15, the Hadoop-HDFS version 2.2.0 (a state) has SSC = 1.00625 and for whole state series it has ESC = 1.0054.

Next, for all the six evolving systems, we describe pre-processing of each evolving system to make evolving networks that are used to calculate the SSCs and ESC.

A) *Evolving software systems*: Evolving software systems [9] are stored and maintained as a state series SS at some software repository. A state series of software can be referred as a version series because a software state is referred as a software version. We collected 15 Hadoop-HDFS[1] versions (states) of jars. Many open-source tools are available to construct *call graph* of a software. Software developed in different programming languages has different tools to make a call graph. The call graph contains procedure call information stored in the form of a network. It is a directed graph to describe procedure-call (or dependency) relationships. Each line of a *call graph* file contains two nodes (as procedures) to represent a directed edge (as call) from the first node to the second node. Where, the first node represents caller procedure and the second node represents callee procedure. A HDFS version series of 15 versions are pre-processed to make a series of 15 *call graphs* (networks). During pre-processing, a single *procedureID* keeps track for a procedure appearing in different modules of a repository. We analyzed the call graphs by identifying the graphlets information that helps in the calculation of complexities for the evolving software versions. The time series plot in Figure 5.15 shows the time-varying complexity of each version (state) and the ESC (1.0054). We got "M_{178}" as the most significant *NEM*, where M_{178} suggests a subgraph where three procedures are calling another procedure, which depicts less complex subgraph. This high frequency of low complex subgraph leads to the low ESC of the Hadoop-HDFS.

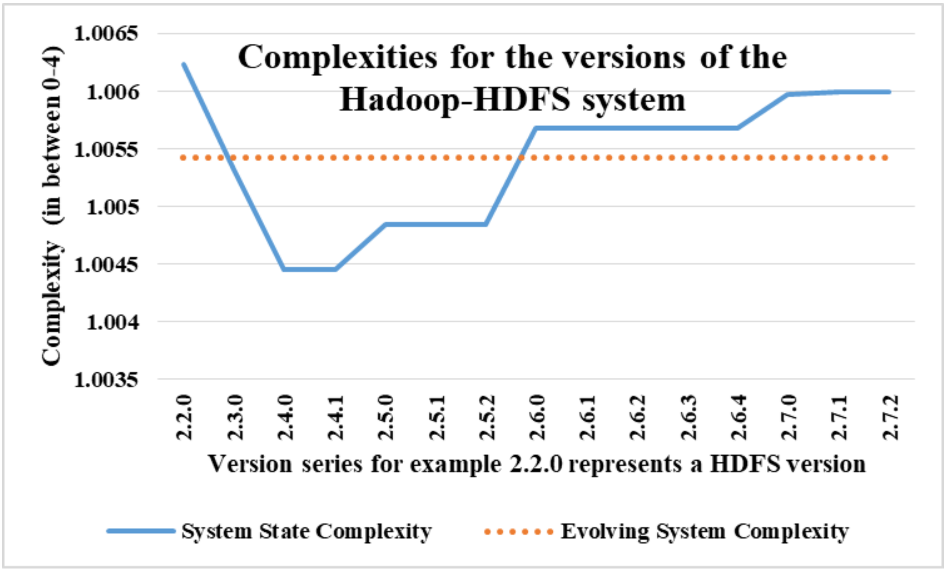


Figure 5.15 The time series representation of SSCs and ESC for Hadoop-HDFS over 15 versions as a version series.

B) *Evolving natural language systems:* We used text of two web pages available on Wikipedia.

i) List of Bible translations: This dataset contains table that includes translation information from "source biblical languages" (like Hebrew, Aramaic, and Greek) to the various "English variant languages" (like Modern English and Old English). To generate the evolving state series SS, we combined the tables of "incomplete Bibles", "partial Bibles" and "complete Bibles". We made evolving networks from connections between entities (words) in columns of 'Source biblical language' and

‘English variant’, such that each network is for a century. There are total 13 centuries that are pre-processed to make 13 evolving networks. The time series plot in Figure 5.16 shows the time-varying complexity of each state (century) and the ESC (1.45). We can observe three points from the Figure 5.16. First, the figure shows spike in between 8th to 10th centuries because there are many translations happened in this period. Second, the figure shows increase in between 11th to 14th century this is due to increase in number of translation in this period. Third, the figure shows, after 14th century the bible translation complexity is almost constant due to stable number of translations in this period.

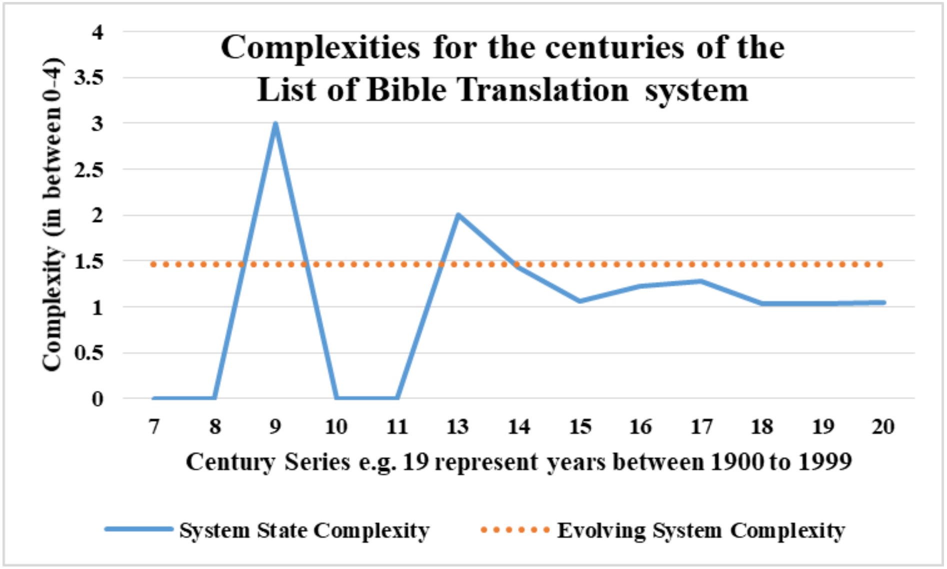


Figure 5.16 The time series representation of SSCs and ESC for list of bible translation over 13 centuries as a century series.

ii) List of multi-sport events: We made evolving networks from connections between entities (words) in two columns: ‘Title’ (name) and ‘Scope’ (regional, international, and provinces) of an event, such that a network is for a decade (10 years). There are total 13 decades that are pre-processed to make 13 evolving networks. The time series plot in Figure 5.17 shows complexity for connections between entities ‘Title’ and ‘Scope’ of multi-sport events. In the figure, first time series is for the time-varying complexity of each state (decade) and second time series is for ESC (1.004). We can observe from the Figure 5.17 that the system state complexities are almost constant between 191 to 196’s decade and 506 to 201’s decade because there are few target nodes of scopes.

C) *Evolving retail market system:* This is a dataset of retail market [170] for UK based registered non-store online retail transaction between 01/12/2010 and 09/12/2011. To make evolving network we followed following steps. First, we make a collection of market basket from products bought by customers with their IDs during each month. Second, to make frequent market basket of products, we kept a product as node in the network only if it is sold atleast 10 months. We used these frequent products to generate the evolving networks for products bought by customers, such that connections between words in column ‘Description’ of products. We made 13 such evolving networks for 13 months such that each network is for a month.

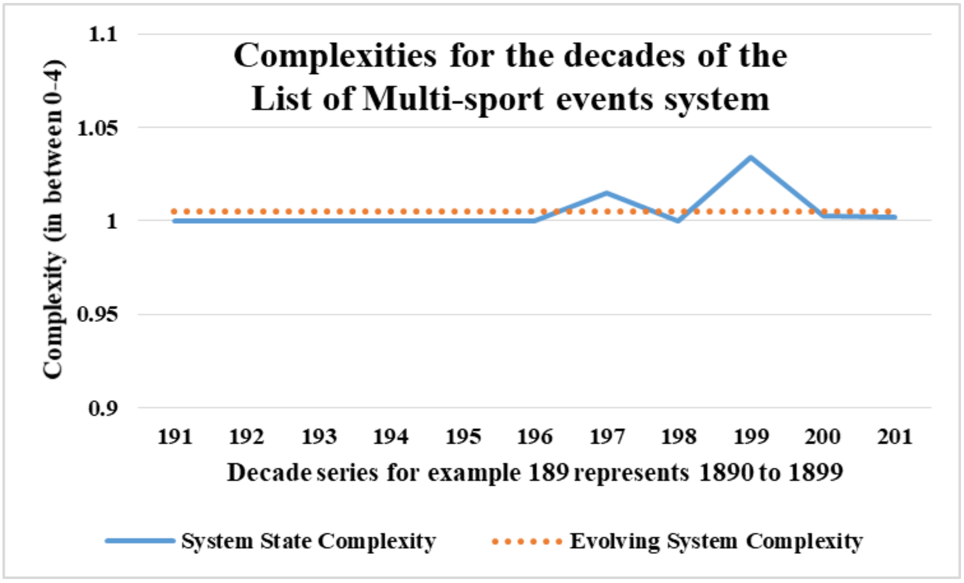


Figure 5.17 The time series representation of SSCs and ESC for multi-sport events over 11 decades as a decade series. Note, out of 13 decades, we omitted two decades (189's and 190's) from the x-axis because their resulting complexity is undefined due to insufficient data.

The time series plot in Figure 5.18 shows the time-varying complexity of each state (month) and ESC (1.33). We can observe two points from the Figure 5.18. First, the figure shows high complexity in between 5th to 6th month of 2011 (i.e., 511 to 611) because of many transactions happened in this period. This is may be due to "summer season in Europe promote more shopping". Second, the figure shows two spikes, firstly sudden decrease in month January 2011 (111) this is due to "sale decreases in December just after the Christmas", and secondly sudden increase in November 2011 (1111) month this is contrarily due to "sale increases in November just before the Christmas".

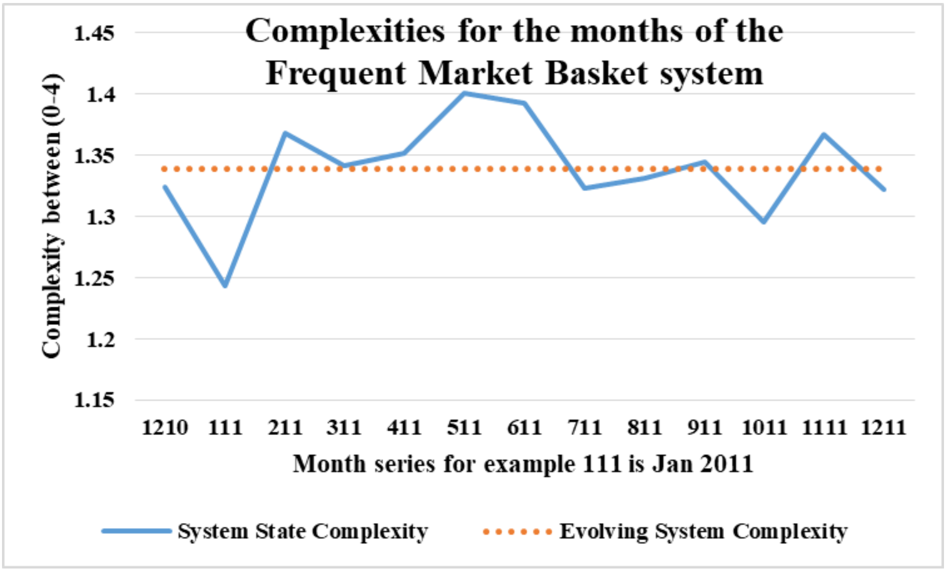


Figure 5.18 The time series representation of SSCs and ESC for frequent market basket over 13 months as a month series.

D) ***Evolving IMDb movie genre system***[5]***:*** We collected the movie and genre data of IMDb for 16 decades since 1870's until Oct 2016. In Figures 5.19 and 5.20, out of 16 decades we omitted three decades (187's, 188's, and 202's) from the x-axis because their resulting complexity is undefined due to insufficient data. We used list[6] of positive and negative words, which was created and used by Minqing and Bing *et al.* [171][172].

i) Evolving system 5: In this experiment, we used positive sentiment words in movie names and all genres of IMDb. While generating the evolving networks, we specifically used only positive words in movie names. The Figure 5.19 shows experimental results for entities (words) connections between 'positive words in movie names' and 'genres' in the IMDb dataset. The time series plot in figure shows the time-varying complexity of each state (month) and the ESC (1.03).

ii) Evolving system 6: In this experiment, we used negative sentiment words in movie names

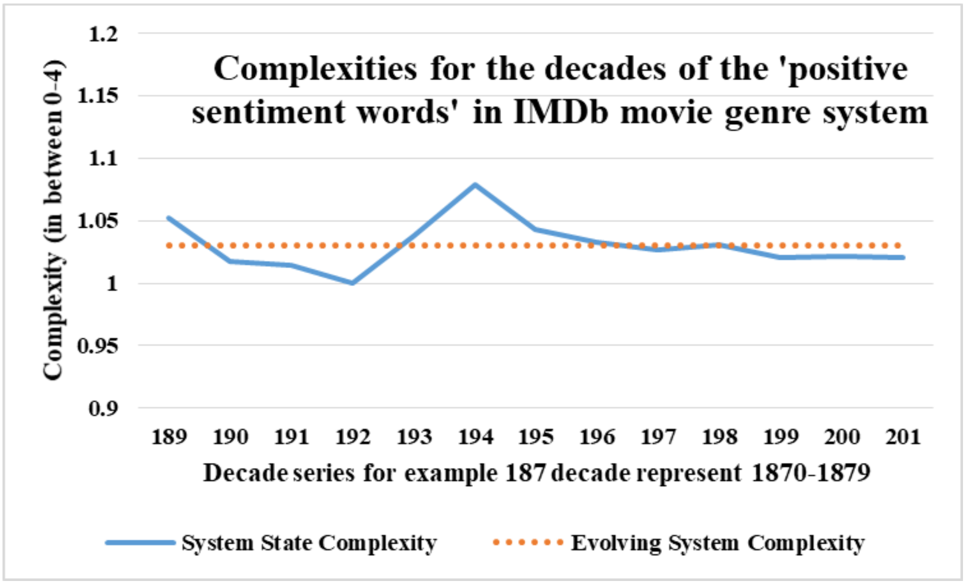


Figure 5.19 The time series representation of SSCs and ESC for positive sentiment IMDb system over 13 decades as a decade series.

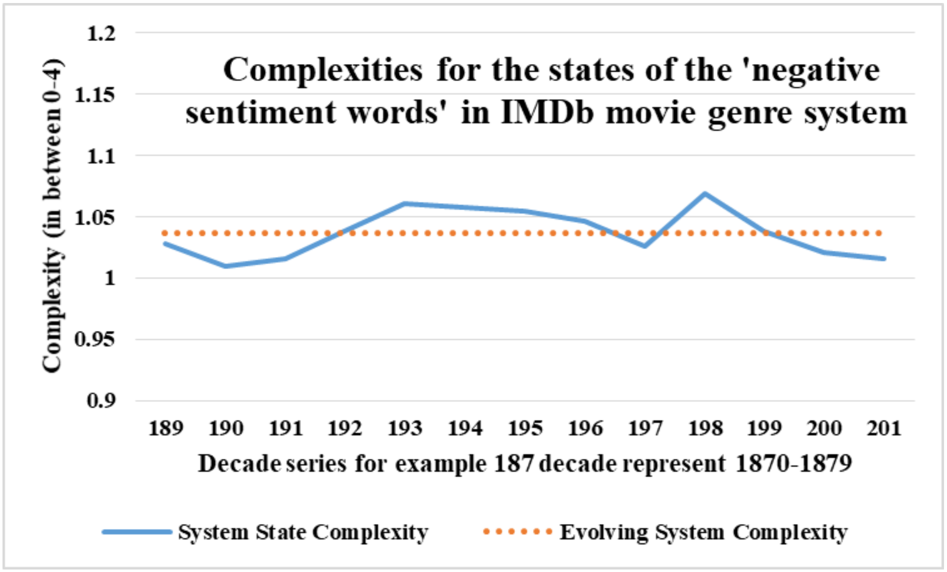


Figure 5.20 The time series representation of SSCs and ESC for negative sentiment IMDb system over 13 decades as a decade series.

and all genres of IMDb. While generating the evolving networks, we specifically used only negative words in movie names. The Figure 5.20 shows experimental results for connections between entities 'negative words in movie names' and 'genres' in the IMDb dataset. The time series plot in figure shows the time-varying complexity of each state (month) and the ESC (1.03).

We made two observations from the Figure 5.19. First, the SSCs are virtually uniform in between 198's to 201's decades because there is large number of similar genre movies in this period. Second, the SSCs vary for decades between 189's to 195's because there are fewer interactions in less number of different genres. In Figure 5.20, the SSCs vary for all decades because of fewer interactions due to less number of different genres. From Figures 5.19 and 5.20, we infer values of both SSCs and ESC are approximately 1.04 for both the systems.

Observations:-

– Using time series plots, we can measure the change in the complexities between two states in an evolving system.

– We got M_0, M_{178}, and M_{198} as the most common and significant NEMs in the most of the evolving systems. The M_{178} and M_{198} suggest a subgraph where a single entity connects to three different entities. All three depicts less complex subgraphs.

– When two evolving systems have different pre-processing to generate their evolving networks, then they are not comparable to each other based on their SSCs and ESC.

– When different evolving systems have same pre-processing semantics to generate their evolving networks, then they are comparable to each other based on their SSCs and ESC. For example, we can compare the positive and negative sentiment in the IMDb movie genre system because of the same pre-processing semantics to generate evolving networks. The Table 5.3 quantitatively shows that "the movies promote negativity more than positivity" because use of negative words (i.e., entities) is more than the positive words (i.e., entities) in movie names. The Table 5.3 also depicts - the average number of neighbours, the NEGs (subgraphs), and the ESC of negativity are more than positivity in the IMDb movie genre system.

– The SNC depends upon subgraph mining, thus pros-cons of subgraph mining are pros-cons of SNC algorithm. As a limitation, our algorithm depends on the efficiency of subgraph mining technique. Although our SNC algorithm can work with any subgraph-mining technique, but the choice of the subgraph-mining technique influences the performance of the SNC. Thus, the choice of subgraph mining tool should be prudent, which depends on the required output.

Contributions from experiments:

– We demonstrated the integration of graphlet and evolution information while applying SNC on six state series of six evolving systems, which are collected from open- internet repositories of four domains. The SNC-Tool generated SSCs and ESC information about each evolving system.

– Using SSCs, we inferred complexity changes happened in an evolving system. Using SSCs and ESC, we compared two evolving systems (if they are of same domain with same pre-processing).

5.9 Summary

This chapter introduced two kind of *network evolution subgraph*: *Network Evolution Graphlet* (NEG) and *Network Evolution Motif* (NEM). We did SysEvo-Analytics based on network evolution subgraph mining. We proposed four metrics: *Changeability Metric* (CM), *Stability Metric* (SM), *System State Complexity* (SSC), and *Evolving System Complexity* (ESC). This provides an insightful, interesting, and non-obvious information, which helps to do SysEvo-Analytics. With these metrics, we can compare two similar evolving systems. Application on state series of an evolving system makes our approach novel and useful. Thereafter, we described two main proposed approaches.

First approach, we proposed an algorithm for mining NEGs, NEMs, CM, and SM, which analyses and assesses a state series SS of an evolving system. The algorithm finds *NEGs information* that contains collective pairs of NEGs with their *aggregate frequencies* over all the states. The NEGs are induced subgraphs (or graphlets) of evolving inter-connected entities over network states. We selected NEMs from the NEGs having significantly high frequencies. The NEMs are significantly re-occurring subgraphs of evolving inter-connected entities over network states. Additionally, we create a time series for evolving frequencies of graphlets over states to measure changeability and stability of an evolving system. This aids to compare two similar evolving systems. We implemented a prototype tool and used it to conduct experiments on six open-internet based evolving systems. For each evolving system, the tool identifies NEGs, NEMs, CM, and SM information for the inter-connected entities in evolving networks of system state series SS. This helps to take decisions and actions about an evolving system.

Second approach, the *graphlets information* is used to calculate the *System State Complexity* (SSC) for each state S_i. The NEGs information is used to calculate the *Evolving System Complexity* (ESC) for a state series SS. Both SSC (of a state) and ESC (of a state series) are interesting and non-obvious information. We implemented the proposed SNC algorithm as a prototype SNC-Tool, which is used to conduct experiments on six open-internet based evolving systems. The tool calculated the complexity of individual states (as SSCs) and complexity of a state series (as ESC). The SSCs are useful to compare between two states of an evolving system. The ESC is useful to compare between two different evolving systems of the same domain. This provides insightful and actionable information about an evolving system. The merit of our work is that it is an extension of the popular and well-used cyclomatic complexity technique of McCabe [121]. Our technique to measure complexity is significantly different from existing techniques. Our SNC algorithm computation is novel with respect to study of system complexities. To the best of our knowledge, our SNC algorithm - that includes SSCs and ESC - is a novel concept for system evolution analysis.

Chapter 6

Deep Graph Evolution and Change Learning to make System Neural Network for System Evolution Recommendation

Emerging computational intelligence leads to increasing demand of computing techniques for system that evolves over time. An *evolving system* has several entities (or features) that evolve over time to make a time-variant data (or non-stationary data) {D_1, D_2 … D_N, D_{N+1}} of a state series SS = {S_1, S_2… S_N, S_{N+1}}. In this chapter, we described two similar and complementary approaches.

First, we introduce an approach to do *evolution and change learning*, which uses an *evolution representor* and forms a *System Neural Network* (SysNN). Additionally, we proposed an algorithm *System Structure Learning* (SSL) that uses evolution representor as an Evolving Design Structure Matrix (EDSM) of a system state series. The EDSM is a kind of time-variant data or non-stationary data. The SSL internally uses a *Deep Evolution Learner* (DEL) that learns from evolution and change patterns of an evolving matrix (EDSM) to generate Deep SysNN.

Second, we present our approach *System Evolution Recommender* (SysEvoRecomd), which uses a novel intelligent algorithm *Graph Evolution and Change Learning* (GECL) that does *matrix reconstruction* leading to *network reconstruction*. Internally, GECL uses Deep Evolution Learner (DEL) to learn about evolution and changes happened to a state series of system. The DEL is an extension of the intelligent deep learning algorithm, which uses an Evolving Connection Matrix (ECM) for training. The DEL generates a *Deep System Neural Network* (Deep SysNN) to do network (graph) reconstruction. The SysEvoRecomd considers the evolving characteristic of graph with deep neural network techniques. It aims to learn the evolution and changes of the system states series and reconstruct the network.

Based on SysEvoRecomd, we developed an automated tool named as SysEvoRecomd-Tool, which is used to conduct experiments on various real-world evolving systems. We apply the SysEvoRecomd-Tool to analyze six real-world evolving systems that we collected from four kind of repositories: software (Maven), natural language (Wikipedia), retail market (UCI), and movies genres (IMDb). We design three variants by exploring three different deep learning techniques: Restricted Boltzmann Machine (RBM), Deep Belief Network (DBN), and denoising Autoencoder (dA). Then, we demonstrate the usefulness of intelligent recommendations using three variants of GECL based on *RBM*, *DBN*, and *dA.*

6.1 Introduction

An *evolving system* is a system which is continuously changing with time. Improper utilization of entities may lead to inefficient maintenance. Further, there is also a possibility of logical errors and faults in newer component due to many developers. The study of the system evolution can help to tackle these problems. An evolving system can be represented as a graph of entity-connections (a set of entities are vertices and directional connections among entities are edges). An *evolving system* makes *time variant* (or *non-stationary*) *data* that has many inter-connected entities (or components). Such system evolves to different states over time. A state (or instance or version) can be represented as an evolving network (or graph). There are several applications for evolving systems to perform SysEvo-Analytics. Entity connections of such a system can be modelled as an adjacency matrix.

We can represent complex data of a system state in the form of *Design Structure Matrix* (*DSM*) [182][183]. The DSM is a simple, compact, and visual representation of a system (or project) in the form of a square matrix. In 1981, Steward [183] coined the term "design structure matrix" to solve mathematical systems of equations. Thereafter, DSM gained many industrial, engineering, and project applications [184][185][186]. The DSM represents complex structure of a system in the form of a matrix, which helps in systems engineering, project management, performance analysis, planning/designing the system (or project). The DSM has various purpose with multiple names like: *Dependency structure matrix*, *Dependency source matrix*, *Problem Solving Matrix* (PSM), *Design precedence matrix*, and *Dependency matrix*. The DSM can be: *Adjacency matrix*, *Binary matrix*, *Logical matrix*, *Relation matrix*, *Boolean matrix*, *Incidence matrix*, *N^2 matrix*, *Interaction matrix*, or *Dependency map*. To study a system evolution using a DSM, the machine learning and data mining techniques are the favourable options that aim automation.

By learning patterns of entity connections in various system states, it is possible to recommend relevant and existing *entity connections* during system development. Learning changes in patterns of entity connections improves components (or entities) reusability; this can saves both time and effort for system development by teams. Suppose a system becomes large and its patterns of entity connections became more complex. This is due to evolution and changes in the entity connections of the evolving system. Tracking changes in patterns of entity connections over time is a challenging problem for humans. This implies, these change information can predict connections, which further makes future system recommendations. Hence, there is a need to apply machine-learning techniques, which can become an appropriate option that can learn the patterns of entity connections.

Deep learning is a machine learning technique that forms a Deep Neural Network (DNN) (a type of Artificial Neural Networks (ANN) inspired by biological systems. The *DNN* models have been receiving a lot of research interest recently, especially because of promising results in image or speech or natural language processing.

In this chapter, we aim to do SysEvo-Analytics by extracting useful information (knowledge) from time-variant data of evolving states. For this, we proposed an approach System Structure Learning, which has the following two steps. First, an evolving system is represented in the form of an evolving matrix (EDSM a time-variant or non-stationary data). Second, we used a deep evolution learner (an extension of the deep learning), which learns evolution and change information from the EDSM. This learning forms a *System Neural Network* (SysNN), which helps to construct an output matrix (as a *disk memory*) that forecast about an evolving system. We also aim to reconstruct matrix of a system network (graph), which gives recommendation about system that can further help in system development and maintenance. Our approach presented is applicable to assist in the process of network reconstruction for connections between nodes (entities). Hence, this helps to do prediction of system graph structure (e.g. connection prediction) by considering the graph structure evolution over time. The details of our proposed approach are given in Section 6.2, 6.3, and 6.5.

6.2 Evolution and Change Learning based System Neural Network

This section describes our key idea to do evolution and change learning that forms a *System Neural Network* (SysNN), which is a type of Artificial Neural Network (ANN). Suppose each state (S_i) of the state series can be represented as a data (D_i). This makes a time-variant dataset {D_1, D_2 … D_N, D_{N+1}} of a state series SS = {S_1, S_2 ... S_N, S_{N+1}} at (N+1) time points {t_1, t_2 … t_N, t_{N+1}}. Next, we describe the proposed two definitions in the context of machine learning of time-variant or non-stationary data.

Definition 6.1: Evolution and Change Learning is a kind of machine learning that learns and memorizes the evolution and changes happened to a time-variant data of a state series, using an evolution representor. Such learning extracts useful information from an evolution representor constructed from a state series of temporally changing system. An existing learning technique can learn from such evolution representor. Evolution and change learning identifies, discovers, and understands the changes in an evolving system. This makes a computer capable enough to understand the evolution and changes of a system without any explicit programming. It is perceivable that evolution and change learning is a kind of non-stationary learning and memorization, which focuses on the evolution and change of a system state series.

Definition 6.2: System Neural Network (SysNN) is the type of *artificial neural network*, which contains information and understanding of evolution happened in a state series. The hidden layers contain evolution information in the form of adjustment between weights and neurons such that it gives probable connections (occurrences) of two entities (features) together. A machine with SysNN can recommend about the time-variant data (or non-stationary data) of an evolving system. The SysNN learns evolution information that constructs an output matrix (as a *disk memory*). The evolution information in the memory can be useful to forecast about possible future of the system data. It is perceivable that the SysNN is a unique and novel kind of cybernetics. The advantage of a SysNN over an ANN is to model non-linear relationships between entities (or features) in time-variant data of an *evolving system*.

Our approach has two steps to generate a SysNN.

- In first step, the state series of an evolving system is represented into an evolution representor. We denote evolution representor as **ER**, which represent state series of an evolving system. The **ER** is calculated by a function f(**SS**) named as ***EvolutionRepresentor*** given as equation

$$\mathbf{ER} = \mathrm{f}(\mathbf{SS}) \qquad \ldots(1)$$

$$\mathbf{ER} = \mathrm{f}(\{a_{11}, a_{12} \ldots a_{mm}\}, \{b_{11}, b_{12} \ldots b_{mm}\} \ldots \{x_{11}, x_{12} \ldots x_{mm}\}).$$

Where, a_{jk}, $b_{jk}\ldots$ x_{jk} are the elements of adjacency matrix, which are either 1 or 0 such that, if there exist a connection between entity j and k, then a_{jk} is 1 otherwise 0, similarly for $b_{jk}\ldots$ x_{jk}.

- In the second step, the algorithm uses this hidden vector **ER** to calculate an output vector **Y**. The calculation is done by a hidden function g(**ER**), named as ***EvolutionLearner***, which is in equation (2). The purpose of the function is to learn evolution and changes happened to the system entities. While learning, the input layer depends on the size of elements in the **ER**, a user provides the number of hidden layers, and the output layer depends on the size of output vector. An expert initializes these three parameters before neural network learning. This learning forms a SysNN that recommends about system evolution using memorized output vector **Y.**

$$\mathbf{Y} = \mathrm{g}(\mathbf{ER}) \qquad \ldots(2)$$

$$\mathbf{Y} = \mathrm{g}(\mathrm{f}(\mathbf{SS}))$$

$$\mathbf{Y} = (\{y_{11}, y_{12} \ldots y_{1m}\}, \{y_{21}, y_{22} \ldots y_{2m}\} \ldots \{y_{m1}, y_{m2} \ldots y_{mm}\})$$

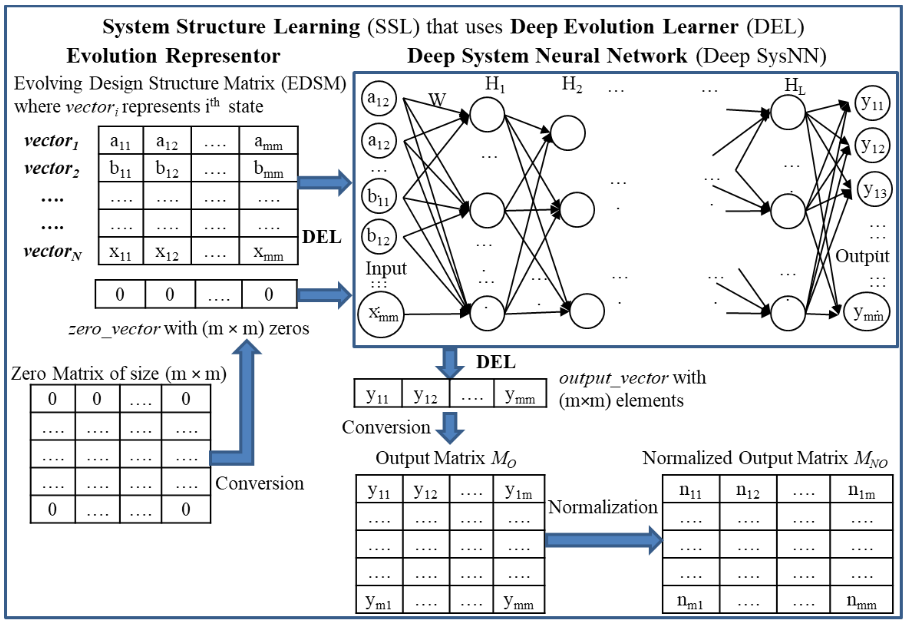
Here, **Y** is a desired *output_vector*. The variable y_{jk} is in between 0 to 1 such that, the y_{jk} gives probability for existence of a connection between entity j and k thus $1 \leq y_{jk} < 0$, where $y_{jk} = 0$ means missing connection.

The System Neural Network learns system evolution information by adjusting its neurons and weights. The neuron weight adjustments are according to the probability for existence of connections between system entities. The SysNN reconstructs a *zero vector* into an *output vector*, both contains m×m elements. After this, convert the *output vector* into *output matrix* M_O of size m×m such that this M_O is an evolution and change learning information. The M_O is a disk memory that stores information about the evolving states of a system. Subsequently, this memory will help to do recommendation about the evolving system.

Next section describes algorithmic form of the proposed System Structure Learning to realize our approach for evolution and change learning that makes a System Neural Network.

6.3 System Structure Learning using Design Structure Matrix

This section describes contributory algorithm *System Structure Learning* (SSL). It uses N Design Structure Matrices (*N_DSMs*) to represent N states of an evolving system. The Figure 6.1 gives an overview of SSL that internally uses evolution representor (as Evolving DSM i.e., *EDSM*) and Deep Evolution Learner (DEL). In the SSL algorithm, the DEL uses evolution representor to learn the evolution of a time-variant data of a state series. The DEL reconstructs a zero vector (*zero_vector*) to an output vector (*output_vector*), which is transformed to an output matrix M_o that is normalized as M_{NO}. Details about the Algorithm 6.1 SSL are given in the rest of this section, which elaborates the evolution and change learning of time-variant data.


System Structure Learning (SSL) that uses Deep Evolution Learner (DEL)
Evolution Representor
Deep System Neural Network (Deep SysNN)
Evolving Design Structure Matrix (EDSM) where vector_i represents i^th state
vector_1
vector_2
vector_N
DEL
zero_vector with (m × m) zeros
Zero Matrix of size (m × m)
Conversion
Input
Output
W
H_1
H_2
H_L
output_vector with (m×m) elements
Output Matrix M_O
Normalization
Normalized Output Matrix M_NO

Here, a_{jk} , b_{jk} , x_{jk} , n_{jk} are either 1 or 0 depending upon connections exist or not, and $0 \leq y_{jk} \leq 1$ where 'jk' represents connection between two entities 'j' and 'k'. Here, a, b, x denotes inputs, y denotes outputs, and n denotes normalized values. Here, L is the number of hidden layers H with weight set W.

Figure 6.1 An overview of System Structure Learning using Deep Evolution Learner to construct a Deep SysNN.

Algorithm 6.1 ***SSL***(*N_DSMs*)

```
Initialize a zero matrix as $M_Z$
zero_vector = matrixToRowVector($M_Z$)
EDSM = evolutionRepresentor(N_DSMs)
output_vector = deep_evolution_learner (EDSM, zero_vector)
$M_O$ = rowVectorToMatrix(output_vector)
$M_{NO}$ = normalize($M_O$)
Return $M_{NO}$
```

To begin with, the ***evolutionRepresentor*** transforms N DSMs (*N_DSMs*) into N vectors, which represents a system's state series **SS** in an Evolving DSM (EDSM). To do this, the algorithm transforms a DSM of state S_i into *vector_i*. For a state, its DSM has size m×m and each vector (*vector_i*) has m×m elements. After that, each *vector_i* combines to form an EDSM. The EDSM given in equation (3) is a type of evolution representor **ER** as described in the Section 6.2 equation (1).

$$\text{EDSM} = \boldsymbol{evolutionRepresentor}(N\_DSMs) = \text{f}(\mathbf{SS}) \quad \ldots (3)$$

Algorithm 6.2 ***evolutionRepresentor***(*N_DSMs*)

```
For each $DSM\_i$ in $N\_DSMs$ where i ∈ state number
$vector\_i = \boldsymbol{matrixToRowVector}(DSM\_i)$
    $EDSM = \left\{ \begin{matrix} vector\_i \\ + \\ EDSM \end{matrix} \right\}$
End for
Return $EDSM$
```

Now, we describe two more definitions.

Definition 6.3: *Evolving Design Structure Matrix* (EDSM) is a type of DSM that represents an ordered collection of N vectors for time-variant data of N states in a state series SS of an evolving system. Fundamentally, three types of changes are possible: addition (or insertion), modification (or alteration), and deletion (or removal). We represented these changes as the collection of N vectors of an EDSM. This EDSM can work as an input for unsupervised learning. Example of EDSM is in Figure 6.2. An unsupervised learning technique can learn from an EDSM as an unlabeled data, which is useful to detect evolution and change patterns. Thus, evolution and change learning depends upon the EDSM and the unsupervised learning.

$$\left\{ \begin{matrix} [\{a_{11}, a_{12} \ldots a_{1m}\}, & \{a_{21}, a_{22} \ldots a_{2m}\} \ldots & \{a_{m1}, a_{m2} \ldots a_{mm}\}] \\ [\{b_{11}, b_{12} \ldots b_{1m}), & (b_{21}, b_{22} \ldots b_{2m}) \ldots & (b_{m1}, b_{m2} \ldots b_{mm})] \\ & & \\ [(x_{11}, x_{12} \ldots x_{1m}), & (x_{21}, x_{22} \ldots x_{2m}) \ldots & (x_{m1}, x_{m2} \ldots x_{mm})] \end{matrix} \right\}_{(m\times m) \times N}$$

where a_{jk}, b_{jk}, x_{jk} is 1 if connection exist otherwise 0
Series of 'a', 'b' and 'x' represent 1st, 2nd and Nth state.

Figure 6.2 Evolving Design Structure Matrix (EDSM) as a time-variant data (or non-stationary data) consisting N data vector of a state series.

Definition 6.4: *Deep Evolution Learner* (DEL) elaborates the evolution learner g(**ER**) mentioned as equation (2) in the Section 6.2. The DEL learns evolution and change patterns in the time-variant data (e.g., EDSM) of a state series. It makes the machine capable enough to understand the evolution and changes happened between states without explicit programming. The DEL is an extension of deep learning that learns from an EDSM and outputs a vector (*output_vector*). The DEL internally uses deep learning to make the machine intelligent by generating *Deep System Neural*

Network (Deep SysNN).

Next, we describe the use of ***deep_evolution_learner*** algorithm in ***SSL***. The ***deep_evolution_learner*** takes the EDSM of size N × (m×m) for training purpose, where N is the number of states and m × m is the size of square DSM. The DEL makes a Deep SysNN (in the form of weight matrices) based on the training by ***delTrain*** (means deep evolution learner training). The three main training parameters (*trainParameters*) control deep learning: learning rate (LR), epoch (Ep), and number of hidden units of neurons (i.e. size and number of hidden layers L). The deep learning can use anykind of matrix, but in DEL we specifically used evolution representor (i.e., EDSM), which is the crux of our approach. The Deep SysNN consists of information about patterns of entity connections in the form of weight matrix of neural network. In DEL algorithm, the ***delReconstruct*** uses Deep SysNN to reconstructs a *zero_vector* into an *output_vector* (both the vectors has m×m elements).

```
Algorithm 6.3   deep_evolution_learner (EDSM, zero_vector)
Initialize List trainParameters< LR, Ep, L >
Initialize a weight matrix Deep_SysNN
Deep_SysNN = delTrain(EDSM, trainParameters)
output_vector = delReconstruct(Deep_SysNN, zero_vector)
Return output_vector
```

Next, we describe how ***delTrain*** and ***delReconstruct*** works in the DEL using extended objective functions of deep learning techniques. We demonstrate reconstruction property of ANN using three well-known unsupervised deep learning techniques [56]: Restricted Boltzmann Machines (RBM) [122][123], Deep Belief Networks (DBN) [124], and denoising Autoencoders (dA) [126][130]. The three deep learning structures are shown in the Figure 6.3.

The DEL reformulates the Restricted Boltzmann Machine's energy model as equation (4) with $E(\boldsymbol{EDSM},\boldsymbol{h}) =$

$$\sum_{i}\left(a_i\, ec_{jk}\right) + \sum_{l}\left(b_l\, h_l\right) + \sum_{i}\sum_{l}\left(ec_{jk}\, w_{il}\, h_l\right) \qquad \ldots (4)$$

where i represent i^{th} state, $\boldsymbol{EDSM}$ represents an evolution representor that contains ec_{jk} as the entity connection between two entities $j \; and \; k$. In the weight matrix of size (m×m) × L, the w_{il} represents the weight of i and l position. The weight matrix represent connections between SysNN such that a visible unit is a DSM of size m×m and hidden unit $\boldsymbol{h}$. As per the standard RBM equation, the two bias weights (offsets) are a_i for the visible units and b_l for the hidden units. Reformulating the energy model $E(\boldsymbol{EDSM},\boldsymbol{h})$ into a probabilistic model makes equation (5).

$$P(\boldsymbol{EDSM},\boldsymbol{h}) = \frac{1}{Z} e^{-E(\boldsymbol{EDSM},\ \boldsymbol{h})} \qquad \ldots (5)$$

Then find conditional probability of hidden layer *h* when *EDSM* is given as input, which makes

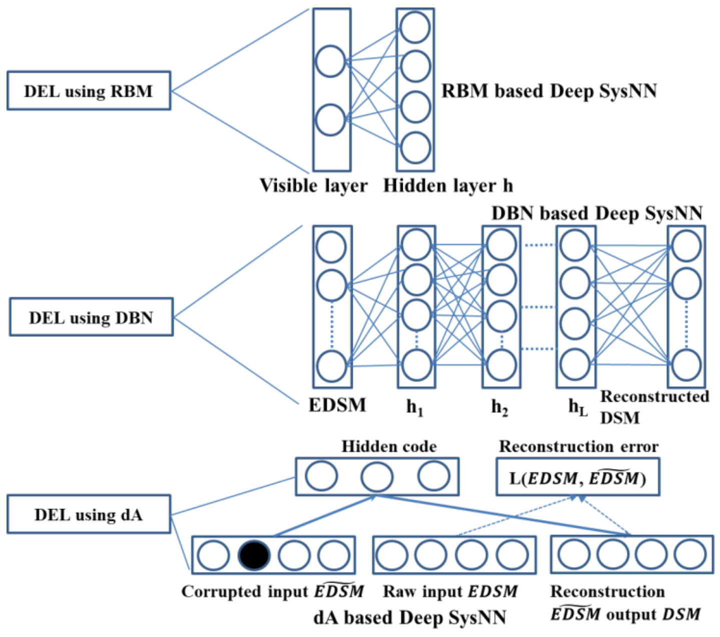


Figure 6.3 Three kinds of Deep SysNN constructed by three DEL variants based on the three deep learning techniques: RBM, DBN, and dA.

equation (6).

$$\mathrm{g}(\mathbf{ER}) = P(\boldsymbol{h}|\boldsymbol{EDSM}) = \prod_{l=1}^{L} P(h_l|\boldsymbol{EDSM}) \qquad \ldots (6)$$

For best reconstruction using equation (6), the reconstruction error needs to be minimize objective function as equation (7).

$$\frac{dlogP(\boldsymbol{h}|\boldsymbol{EDSM})}{d\boldsymbol{W}_{jkl}} \approx < ec_{jk}h_l >_{data} - < ec_{jk}h_l >_{reconstruct} \quad \ldots (7)$$

The DEL reformulates the Deep Belief Network as a stack of RBMs working together for training on EDSM. This greedy learning forms Deep Belief Network. Therefore, the reformulation makes equation (8) with

$$\mathrm{g}(\mathbf{ER}) = P(\boldsymbol{EDSM}, h^1, h^2 \ldots h^L) = \left(\prod_{l=0}^{L-2} P(h^l|h^{l+1})\right) P(h^{L-1}, h^L) \qquad \ldots (8)$$

where L denotes a user-defined number of hidden layer and h^l denotes the l^{th} hidden layer.

The DEL reformulates the auto-encoder to make a deep neural network by *encoding* the input $\boldsymbol{EDSM}$ into $c(\boldsymbol{EDSM})$. The objective is to reconstruct $c(\boldsymbol{EDSM})$ using a *decoder* function with least error. The DEL reformulates minimization of the reconstruction error by minimizing the objective function of negative log-likelihood given as equation (9)

$$RE = -\log P\big(\boldsymbol{EDSM}\big|c(\boldsymbol{EDSM})\big) \qquad \ldots (9)$$

$$RE = -\sum_{i} ec_{jk}\, log f_i\big(c(\boldsymbol{EDSM})\big) + \big(1 - ec_{jk}\big)\log\Big(1 - f_i\big(c(\boldsymbol{EDSM})\big)\Big)$$

where $f_i(x)$ is the function to *decode* x_i, and $f_i\big(c(\boldsymbol{EDSM})\big)$ is the function to reconstruct *i*th state of $\boldsymbol{EDSM}$ using the Deep SysNN.

The denoising Autoencoder (dA) is a stochastic version of the auto-encoder. The input of the dA is stochastically corrupted, and uncorrupted input is used to reconstruct target. The dA recommends the entity connections pattern using randomly selected subsets of entity connection patterns. The reformulated training equation (10) for dA to reconstruct negative log-likelihood

$$RE = -\log\big(\mathrm{g}(\mathbf{ER})\big) = -\log P\left(\boldsymbol{EDSM}\middle|c\big(\widetilde{\boldsymbol{EDSM}}\big)\right) \qquad \ldots(10)$$

where $\boldsymbol{EDSM}$ is the uncorrupted input, $\widetilde{\boldsymbol{EDSM}}$ is the stochastically corrupted input, and c($\widetilde{\boldsymbol{EDSM}}$) is encode form of $\widetilde{\boldsymbol{EDSM}}$. To obtain best reconstruction, minimize the equation (10) as objective function.

In the SSL algorithm, the *output_vector* is converted to form a matrix M_O of size (m×m). The elements in M_O are between 0 and 1, which gives probability of connections between two entities. In last step, the M_O is transformed to a normalized matrix output M_{NO}. The normalization means the element in the M_O is converted to '0' or '1' according to the normalized distribution, thus M_{NO} is a binary matrix. The M_{NO} is a kind of memorized information about an evolving system, which is useful for decision-making and taking-action. This helps to deal with adaptation, control, and analysis of evolution and changes in a system over evolving states.

6.4 SysEvoRecomd Approach

This section describes the proposed *System Evolution Recommender* (SysEvoRecomd), which uses *Graph Evolution and Change Learning* (GECL). The SysEvoRecomd learns the evolution based on changes happened among evolving system states. Suppose an evolving system has more than N+1 continuously evolving states, such that we can use N states for training purpose and 1 remaining state for testing purpose. Retrieve N+1 states $\{S_1, S_2 \ldots S_{N+1}\}$ for various time points $T = \{t_1, t_2, t_3 \ldots t_N, t_{N+1}\}$ and store them in a local directory. The S_i stands for the *i*th state in a repository, and *i* varies from 1 to N+1. The overview of the SysEvoRecomd is shown in Figure 6.4.

The SysEvoRecomd preprocess a set of states to create a collection of N+1 graphs in the form of connection lists with a *mapping* file. The SysEvoRecomd converts the collection of graphs into N+1 connection matrices. A deep learning technique can help in efficient learning of evolution in a state series. To do this, we introduced *Graph Evolution and Change Learning* (GECL), which processes the N *connection matrices* and produces an *output matrix*. The SysEvoRecomd compares the output matrix

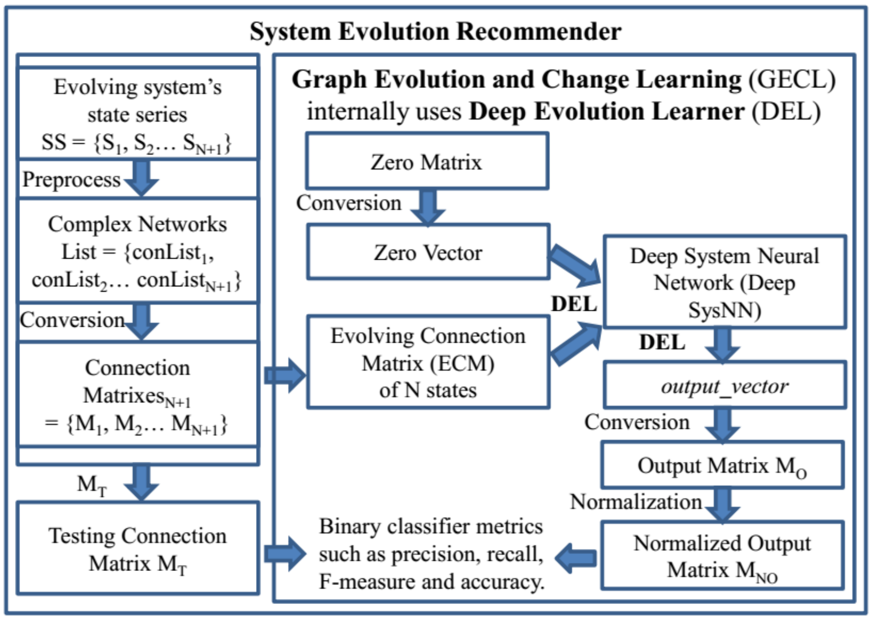

System Evolution Recommender
Evolving system's state series SS = {S1, S2… SN+1}
Preprocess
Complex Networks List = {conList1, conList2… conListN+1}
Conversion
Connection MatrixesN+1 = {M1, M2… MN+1}
MT
Testing Connection Matrix MT
Graph Evolution and Change Learning (GECL) internally uses Deep Evolution Learner (DEL)
Zero Matrix
Conversion
Zero Vector
Evolving Connection Matrix (ECM) of N states
DEL
Deep System Neural Network (Deep SysNN)
DEL
output_vector
Conversion
Output Matrix MO
Normalization
Normalized Output Matrix MNO
Binary classifier metrics such as precision, recall, F-measure and accuracy.

Figure 6.4 An overview to make a System Evolution Recommender, which uses ECM to generate a Deep System Neural Network.

with *testing matrix* (a matrix not used for training).

Definition 6.5: *System Evolution Recommender* learns evolution happened over evolving states of a system at various time-points. It makes the machine capable enough to understand the evolution in a state series without any explicit programming. The learned information can recommend about the system evolution.

Definition 6.6: *Graph Evolution and Change Learning* (GECL) is the process of learning evolution happened over evolving graphs from an intermediate representation that contains changing patterns of entity connections. It makes a machine capable enough to understand the evolution in evolving graphs without any explicit programming. The GECL can do network reconstruction in the form of a connection matrix. The GECL may also be referred as *Network Evolution and Change Learning* (NECL).

Algorithm 6.4 SysEvoRecomd

```
Input: repository
Retrieve N+1 states ($S_1, S_2… S_N, S_{N+1}$) of a repository
1: $Lists_{N+1}$ and mapping = preprocess(repository)
2: $Matrices_{N+1}$ = conversion($Lists_{N+1}$)
3: $M_{NO}$ = GECL($Matrices_N$)
4: time_series = testing($M_{NO}$, $Matrices_N$, $M_T$)
```

The SysEvoRecomd algorithm preprocesses a set of states to create a *Lists*$_{N+1}$ with a *mapping* file. Then it converts the collection of graphs into *Matrices*$_{N+1}$. An intelligent deep learning is done by *Graph Evolution and Change Learning* (GECL) algorithm, which uses the *Matrices*$_N$ and produces a M_{NO}. The SysEvoRecomd compares the M_{NO} with M_T. To facilitate the automation of *Algorithm 6.4*

SysEvoRecomd over a broad of applications, we describe an intelligent tool, which has four steps in the following four sub-sections.

6.5.1 Preprocessing of a State Series

The SysEvoRecomd theory and approach creates basis for us to develop an automation tool named as SysEvoRecomd–Tool. The tool helps in recommendations about system states, which reduces human intervention and efforts. We developed the tool in Java, which pre-processes *evolving entity connections* in *evolving networks* (graphs) of a system state series. After pre-processing, SysEvoRecomd-Tool converts each *evolving graph* into a *connection matrix*. The tool has GECL, which uses *Deep Evolution Learner* (DEL) to generate *Deep System Neural Network* (Deep SysNN). The tool uses the GECL to reconstruct a *zero matrix* into an *output matrix*, which helps to study the system evolution.

An evolving system contains entities that can be chosen to make a graph such that it represents a relationship between entities. Although there are several evolving systems having different pre-processing techniques (depending upon domain), here we describe a general way of preprocessing. Each state is pre-processed to make a graph represented as connection list (i.e. an adjacency list). Each graph contains a set of connections (as edges) between entities (as nodes). The pre-processing algorithm takes N+1 states as input from a directory (i.e. repository). Process each state (say S_i) to make a connection list for connections of entities (say *connList_i*). The *connList_i* represents a graph for a state S_i. Store all the N+1 graphs as a connection list in a directory (say $Lists_{N+1}$). Simultaneously, the algorithm also built a mapping file (say *mapping*) that contains two entries: *entityName* and *entityID* for each *entity*. Each entity reappearing in a state series must have the same *entityID* at all the appearances. This mapping file helps to map *entityID* to its *entityName* in post-processing. The pre-processing outputs are N+1 connection lists and one *mapping*.

Algorithm 6.5.1 preprocess

```
Input: repository
Output: $Lists_{N+1}$ and mapping
1:Initialize HashMap HM<entityName, entityID>
2:Initialize integer counter = 1
3:Initialize String buffer = null
4:For each state $S_i$ where $i \in$ integer 1 to N +1
5:Detect a graph of relationship between entities in the state $S_i$. The graph is in the form of <entityName, entityName> Store the graph in a file say connList_i
6:End For
7:For each connList_i where $i \in$ integer 1 to N
8:  Scan connList_i
9:  until end of file and Store it in buffer
```

```
9:  Scan buffer
10:     If an entityName is in the HM
11:     In buffer replace the entityName with its entityID
12:     Else
13:     entityID = counter
14:     Add the new tuple <entityName, entityID> in HM
15:     In buffer replace the entityName with its entityID
16:     Increment counter by 1 i.e. counter = counter + 1
17:   until End of buffer
18:     Write buffer in connList_i and store it in $Lists_{N+1}$
19: End For
20: Store the HM in a file named as mapping
21: return $Lists_{N+1}$ and mapping
```

6.5.2 Conversion

After pre-processing step, each graph (connection list) is converted into its *connection matrix*. The conversion algorithm converts N+1 connection lists ($Lists_{N+1}$ as input) to N+1 connection matrix (stored in a directory (say $Matrices_{N+1}$)). Process each *connList_i* to make a *connection matrices* (say *conn_matrix_i*) and store them in a directory (say $Matrices_{N+1}$), where i represents state such that $1 \leq i \leq N+1$.

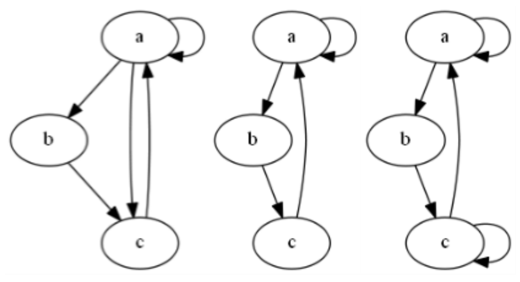
Algorithm 6.5.2 conversion

```
Input: $Lists_{N+1}$
Output: $Matrices_{N+1}$
1:  Initialize integer counter = 1
2:  Initialize String buffer = null
3:  For each connList_i where $i \in$ integer 1 to N
4:     Initialize matrix[u][v] = {(0, 0… 0),... (0, 0… 0)}
5:     For each u ∈ connList_i | u is a source node
6:       do while each v ∈ connList_i | v is target node of u
7:         matrix[u][v] = 1
8:     end do whlie
9:     End For
10:    Write matrix[u][v] as conn_matrix_i in $Matrices_{N+1}$
11:  End For
12:  return $Matrices_{N+1}$
```

6.5.3 Graph Evolution and Change Learning based on Deep Evolution Learner

This subsection presents our key contributory algorithm, *Graph Evolution and Change Learning* (GECL). After encoding a state series as connection matrices, the GECL is used. The *GECL* algorithm uses a time-variant data *Matrices*$_N$ (N *connection matrices*) of a state series as input. The GECL algorithm combines multiple connection matrices of a state series to form an *Evolving Connection Matrix* (ECM) as an unlabelled data to learn system evolution.

Suppose a system state is represented as a graph (first graph of Figure 6.5) with edges (or connections) {(a, a), (a, b), (a, c), (b, c), (c, a)}. Such that if entity '*a*' is connected to entity '*b*' then intersection between '*a*' (row) and '*b*' (column) has entry '1' else has entry '0'. The first graph is represented as the *first matrix*, such that each row and column represents connection of an entity (as node). This matrix can be converted to its equivalent vector [1, 1, 1, 0, 0, 1, 1, 0, 0]. Let the system state is evolved to make two more states, which are represented as second and third graphs (represented as second and third matrix). Firstly, we provide two formal definitions for the matrices.



$$\begin{array}{c|ccc} & a & b & c \\ \hline a & 1 & 1 & 1 \\ b & 0 & 0 & 1 \\ c & 1 & 0 & 0 \end{array}$$

$$\begin{array}{c|ccc} & a & b & c \\ \hline a & 1 & 1 & 0 \\ b & 0 & 0 & 1 \\ c & 1 & 0 & 0 \end{array}$$

$$\begin{array}{c|ccc} & a & b & c \\ \hline a & 1 & 1 & 0 \\ b & 0 & 0 & 1 \\ c & 1 & 0 & 1 \end{array}$$

Figure 6.5 An overview to make a System Evolution Recommender, which uses ECM to generate a Deep System Neural Network.

Definition 6.7: A <u>*connection matrix*</u> stores information about connections between entities in the form of a *square binary matrix*. It is a type of *adjacency matrix*, such that presence of connection is represented as '1', whereas '0' represents absence of connection. The size of connection matrix is (m×m), where m is the number of entities in a state. The connection matrix can be converted to its equivalent row vector. An appropriate name for this vector is <u>*connection vector*</u>. The number of elements in a connection vector is (m×m), where m is the number of entities in a state.

Definition 6.8: <u>*Evolving Connection Matrix*</u> (ECM) represents an ordered collection of connection vectors for different states of a system state series SS = {S_1, S_2... S_N}. The ECM also represents evolving connections between entities in a state series of an evolving system. The size of ECM is equal to N × (m × m), where N is the number of states, and m × m is the size of a square connection matrix for a state.

Secondly, the GECL uses the *Deep Evolution Learner* (DEL) that takes ECM as input to learn evolution and changes. Inside *GECL* algorithm, the ***deep_evolution_learner*** identifies evolution and changes happened over states of an evolving system. An ECM is a kind of evolution representor that consists of evolution and change information happened over several states. This ECM can be useful to learn information using deep learning algorithm. For this purpose, we extended deep learning to make the GECL intelligent.

During training, the DEL learns the evolution and change patterns of the *evolving entity connections* in ECM. Although, the ECM is a matrix but it can also be visualized as a column matrix of - N *connection vectors*, where each connection vector has m × m elements. Internally, the DEL forms a Deep System Neural Network (Deep SysNN). The training in the algorithm makes Deep SysNN learned about the evolution of a system. The Deep SysNN is in the form of weight matrices based on the training (means deep evolution learner training). The main parameters during training are:

Algorithm 6.5.3 GECL

```
Input: $Matrices_N$
Output: $M_{NO}$
1: Initialize a zero matrix as $M_Z$
2: zero_vector = matrixToRowVector($M_Z$)
3: For each conn_matrix_i in $Matrices_N$ | $i \in$ 1 to N
4: conn_vector_i = matrixToRowVector(conn_matrix_i)
5: End for
6: output_vector =
                                  conn_vector_i
                      ECM   =     +
                                  ECM

         deep_evolution_learner(ECM, zero_vector)
7: $M_O$ = rowVectorToMatrix(output_vector)
8: $M_{NO}$ = normalize($M_O$)
9: return $M_{NO}$
```

learning rate (LR), epoch, and number of hidden units of neurons.

Although the deep learning can use any kind of matrix, the DEL uses evolution represented information (as ECM), which is the key idea of our approach. The Deep SysNN contains information about evolution and change patterns of entity connections in the form of weight matrix of neural network. Thereafter, the DEL reconstructs a *zero_vector* into an *output_vector* (both the vectors has m×m elements). Thus, the Deep SysNN helps to reconstruct a *zero_vector* (made of a zero matrix M_Z of size m×m) to make an *output_vector*.

Key idea of DEL is to use reconstruction property of deep learning. The objective of DEL is to learn each connection vector of a state given in ECM (as primary input). This learning helps to reconstruct the entity connections of a zero vector (as secondary input). The pre-training happens on several connection vectors sequentially containing entity connections as feature, this initializes *weight matrix* of a *Deep_SysNN* with sensible values. Thereafter, last layer of output units is added on the top of the *Deep_SysNN* and then the whole Deep SysNN is fine-tuned using backpropagation.

The objective of *deep evolution learner* model is to minimize the connection-vector reconstruction error. To achieve this goal, the DEL uses existing and successful reconstruction theory of deep learning. We restrict the description of the DEL model to three well-appreciated deep learning techniques: Restricted Boltzmann Machine (RBM), Deep Belief Networks (DBN), and denoising Autoencoder (dA). RBM model can be re-written using ECM as equation

$$P(\boldsymbol{h}|\boldsymbol{ECM}) = \prod_{l=1}^{L} P(h_l|\boldsymbol{ECM})$$

Similarly, Deep Belief Network (DBN) model using $P(\boldsymbol{h}|\boldsymbol{ECM})$ can remodel as equation

$$P(\boldsymbol{ECM}, h^1, h^2 \ldots h^L) = \left(\prod_{l=0}^{L-2} P(h^l|h^{l+1})\right) P(h^{L-1}, h^L)$$

where, L is the total number of hidden layers and h^l is the l^{th} hidden layer.

Our remodeled training equation for dA is expressed as a reconstruction negative log-likelihood

$$RE = -\log P\left(\boldsymbol{ECM}\middle|c(\widetilde{\boldsymbol{ECM}})\right)$$

where $\boldsymbol{ECM}$ is the uncorrupted input, $\widetilde{\boldsymbol{ECM}}$ is the stochastically corrupted input, and c($\widetilde{\boldsymbol{ECM}}$) is encoded form of $\widetilde{\boldsymbol{ECM}}$. The objective is to minimize the negative log-likelihood of the equations.

After training phase, the DEL minimizes *zero_vector* reconstruction error while making an *output_vector*. Thus, this is how the DEL makes a connection vector named as an *output_vector*. In GECL, the *output_vector* is then converted to its equivalent matrix, which is an output matrix M_O. The M_O consist information about probable existence of a connection between two entities in the ECM. The DEL makes M_O (elements are in between 0 to 1) that further makes *normalized output matrix* M_{NO} (elements are binary). This M_{NO} is our desired objective, which represents a reconstructed network.

In short, the GECL algorithm helps to analyze an evolving system. The GECL algorithm uses N states to make an ECM. The GECL further depends upon DEL algorithm that learns evolving states. The DEL takes an ECM to generate a Deep SysNN, which helps to reconstruct a *zero vector* into an *output vector*. The vector reconstruction done by the DEL leads to network reconstruction matrix M_{NO} that might have some inaccuracy. Thus, we need to validate the network reconstruction result (M_{NO}) against an actual system network state (not used for training).

6.5.4 Recommendation Performance Evaluation

This section explains formal performance evaluation of our recommender system tool by testing the given recommendations. After core learning, testing is performed for the recommendations provided by M_{NO} about the entity connections of the evolving systems. The **testing** algorithm measures similarity based on four evaluation metrics: *accuracy, precision, recall,* and *f-measure*. The algorithm uses the four metrics to compare between the *normalized output matrix* M_{NO} and *connection matrices*.

For each state, the values of these four metrics are stored in a file named as *time_series*. For each *i* vary from 1 to N, the algorithm calculates the four metrics values between M_{NO} and *conn_matrix_i*. The state number '*i*' followed by the metrics values are appended as a line in the *time_series* file. Thereafter, the algorithm calculates the four metrics values between M_{NO} and *testing*

Algorithm 6.5.4 testing

```
Input: $M_{NO}$, $Matrices_N$, $M_T$
Output: time_series
1: Initialize accuracy, precision, recall, f-measure, time_series
2: For each conn_matrix_i in $Matrices_N$ where $i \in$ integer 1 to N
3:     accuracy = accuracy($M_{NO}$, conn_matrix_i)
4:     precision = precision($M_{NO}$, conn_matrix_i)
5:     recall = recall($M_{NO}$, conn_matrix_i)
6:     f-measure = fmeasure(precision, recall)
7:     time_series = addLine(i, accuracy, precision, f-measure, recall)
8: End for
9:     accuracy = accuracy($M_{NO}$, $M_T$)
10:    precision = precision($M_{NO}$, $M_T$)
11:    recall = recall($M_{NO}$, $M_T$)
12:    f-measure = fmeasure(precision, recall)
13:    time_series = addLine(N+1, accuracy, precision, f-measure, recall)
10:    return time_series
```

matrix M_T. The state number 'N+1' of M_T and the four metrics values are appended as a line in the *time_series* file. The *time_series* file aids to produce a (time series) graph, which shows similarity of M_{NO} with the system states and highlights comparison between M_{NO} and M_T. After learning and testing phases of our *SysEvoRecomd*, we can make and show time series graphs. Based on the SysEvoRecomd algorithm, we made three variants of DEL using three deep learning techniques: *RBM*, *DBN*, and *dA*. This makes three variant of GECL. For this, we used Java source code (https://github.com/yusugomori/DeepLearning).

6.5 SysEvoRecomd Tool

To show application, we developed a prototype tool (in Java) based on the proposed approach and named it SysEvoRecomd-Tool (System Evolution Recommender – Tool), which is a recommender system. Our tool learns and recommends the patterns of *evolving entity connections* over a state series. This helps to study and recommend about system's evolution. A practitioner dealing with an evolving system (represented as a series of graphs with inter-connected entities) can find our tool interesting. An evolving graph represents connections between system entities. Our SysEvoRecomd approach considers graph and evolution learning together, and it helps to do graph (network) matrix reconstruction. Our approach is an automated technique to build a recommender system. With the proposed SysEvoRecomd approach and tool, one can learn evolution happened to a system over time. This learning can be used to do recommendation (or prediction) about system. Further, our tool would be helpful in making recommendations about system evolution in following ways.

- It can predict unknown connections between entities.
- It is helpful while upgrading system to a new state.
- It is helpful during intermediate phase of the system development.
- It can assist a system developer.
- It can speed-up the system development.
- It can do automatic correction of some errors during system development.
- It can determine possible future of the system.

These advantages are analysed with the experiments (in the next section), which enables to use such recommender system for an evolving system.

6.6 Evaluation using K-Fold and Forward Chaining

This section describes evaluation of our SSL algorithm using two techniques for experimental cross validation: kFold and forward chaining (also known as time series cross validation). Our model has four parts to do system structure learning: training data, testing data, target output, and classifier metrics. The **training data** of DEL is an EDSM. The DEL learns evolution patterns from EDSM of a state series in an 'unsupervised manner'. The training data is the connection patterns between a set of entities (features) in the form of EDSM for a state series. After training, the DEL creates Deep SysNN. Learning and recommendation of changes rely on frequency of occurrence of connection between two entities (features) in an EDSM. The Deep SysNN helps to reconstruct a *zero matrix* M_Z to an *output matrix* M_O.

The **testing data** is a testing matrix M_T, which represents entity connections of a state for testing. The testing data is a DSM of a state that is unused during training phase. The **target output** is the output matrix M_O that can reflect evolution information about evolving system. Using Deep SysNN, we can automatically produce an *output matrix* M_O (containing patterns of entity connections) that recommends about evolving entity connections of an evolving system. There are four well-known **binary classifier metrics** [187] (*accuracy*, *precision*, *recall*, and *f-measure*), which checks whether the output is correct or incorrect. The four metrics can analyze the output M_{NO} according to the desired correctness.

Next, we discuss two cross validation techniques using the four binary classifier metrics. The kFold is a well-known technique [188][189], thus we skip its basic explanation. The forward chaining [190] is used for time (state) series based data where learning from earlier states has importance. To do the two types of cross validation, we made two types of directory (folder) that contains training and testing data.

- We made a *kFold folder* such that it contains N+1 sub-folder, where each sub-folder contains N training matrix and 1 testing matrix M_{T}_i of state S_i. The name of the sub-folder is the testing state name (or number), and we denote it as 'i'. Here, k stands for the N states used for training purpose.

- We made a *forwardChaining folder* such that it contains N sub-folder. Each sub-folder contains training matrices of *previous states* and 1 *testing matrix* M_{T}_i of current (i.e. testing) state S_i. The name of the sub-folder is the current state name (or number), and we denoted it as 'i'. Here, training data varies from 1 to N states.

kFold folder where k = 4:	*forwardChaining folder*
folder 1: train [2 3 4] and test [1]	folder 2: train [1] and test [2]
folder 2: train [1 3 4] and test [2]	folder 3: train [1 2] and test [3]
folder 3: train [1 2 4] and test [3]	folder 4: train [1 2 3] and test [4]
folder 4: train [1 2 3] and test [4]	

The ***kFold_forwardChaining*** algorithm performs two types of cross validation: kFold and forward chaining. The first 'for-loop' identifies the four binary classifier metrics over the *kFold folder* and stores the values in *test_metric*. The algorithm uses the *test_metric* to calculate the average of accuracy, precision, f-measure, and recall over the number of sub-folders (or results) in *kFold folder*. The second 'for-loop' identifies the four binary classifier metrics over the *forwardChaining folder* and stores the values in *test_metric*. The algorithm uses the *test_metric* to calculate the average of accuracy, precision, f-measure, and recall over the number of sub-folders (or results) in *forwardChaining folder*.

Algorithm 6.5 kFold_forwardChaining

```
Initialize accuracy, precision, recall, f-measure, test_metric
For each i^th sub-folder in kFold where $i \in$ state number
    $M_{NO}$ = SSL($Matrix_N$)
    accuracy = accuracy($M_{NO}$, $M_T\_i$)
    precision = precision($M_{NO}$, $M_T\_i$)
    recall = recall($M_{NO}$, $M_T\_i$)
    f-measure = f-measure(precision, recall)
    test_metric = addline(i, accuracy, precision, f-measure, recall)
End for
Calculate four averages of all four metrics (accuracy, precision, f-measure, recall) over the number of
states in test_metric. The four averages are: kFold<Acc., Pre., FMe., Rec.>
For each i^th sub-folder in forwardChaning where $i \in$ state number
    $M_{NO}$ = SSL($Matrix_{previous\_states}$)
    accuracy = accuracy($M_{NO}$, $M_T\_i$)
    precision = precision($M_{NO}$, $M_T\_i$)
    recall = recall($M_{NO}$, $M_T\_i$)
    f-measure = f-measure(precision, recall)
    test_metric = addline(i, accuracy, precision, f-measure, recall)
End for
Calculate four averages of all four metrics (accuracy, precision, f-measure, recall) over the number of
states in test_metric. The four averages are: forwardChaning<Acc., Pre. FMe., Rec.>
```

Return *kFold<Acc., Pre., FMe., Rec.>* and *forwardChaning<Acc., Pre., FMe., Rec.>*

Our approach is applicable to an evolving system represented as a state series of DSM, thus we present analysis of six different evolving systems in the next section.

6.7 Experimentation on Evolving Systems

This section describes application of our approach on the six real-world evolving systems of four domains as mentioned in Table 6.1. We demonstrate performance evaluation of our SysEvoRecomd-Tool on six evolving systems. Usually, some recommender systems are inherently better at studying specific input patterns, and hence can work accurately in a specific domain. The evolving systems have both regular and irregular time intervals between states. Only HDFS software versions have irregular time interval. All other systems have regular time interval e.g. bible translations have intervals of centuries, multi-sport have intervals of decades, retail market have intervals of months, movie-genres have intervals of decades.

6.7.1 Experiments for System Structure Learning (SSL)

Each experiment evaluates the recommendation of SSL algorithm for an evolving system. To do experiments, suppose there are N+1 states of a system. From these states only N states are used in training phase to create a Deep SysNN by SSL. We evaluate the SSL algorithm based SysEvoRecomd-Tool for the six evolving systems. The size of each DSM is m×m given in forth column, where m is the number of procedures (entities) in a code version (state) series of evolving system. The ***evolutionRepresentor*** algorithm transforms N DSMs into N vectors such that each vector contains m×m elements. The algorithm combines the N vectors of N training states to form an EDSM (N × m×m) as training input to the DEL. Internally, the ***deep_evolution_learner*** generates Deep SysNN that creates an *output_vector* with m×m elements. The ***evolutionRepresentor*** uses *output_vector* to make a binary matrix M_{NO} of size m×m. The testing algorithm compares the M_{NO} with the testing matrix M_T, which is a binary matrix containing few 1s and many 0s. To do evaluation, we performed the following three steps on each of the evolving system.

- First, for each experiment, we partition system data into two parts: training data and testing data. The training data contains DSM of N states, and the testing data contains *testing matrix* M_T. For each experiments, the ***kFold_forwardChaining*** algorithm uses these two partitions.

- Second, for each experiment, the SSL combines N DSMs of the training data to make an EDSM. Then, in the SSL, the DEL uses EDSM to generate a Deep SysNN that reconstructs a *zero matrix* M_Z into an *output matrix* M_O. The *output matrix* M_O is transformed to a *normalized output matrix* M_{NO} that mimics the pattern of the old states. The SysEvoRecomd-Tool models an evolving system to do recommendation based on deep evolution learner. The tool uses deep learning source code (written in java) for (RBM, DBN, and dA), written by Sugomori Yusuke [191], and available on

Table 6.1 Information about imbalanced dataset Used.

Evolving Systems	Testing matrix (M_T)	Number of 1s	Number of 0s
Hadoop HDFS	Version 2.7.2	2938	9787703
List of Bible Translation	20th Century	46	579
List of Multi-sport Events	201 i.e. 2010-2017 decade	8	188
Frequent Market Basket	1211 i.e. December 2011	617	13307
Positive sentiment of movie genres	201 i.e. 2010-2019 decade	47	578
Negative sentiment of movie genres	201 i.e. 2010-2019 decade	177	1848

GitHub [192]. Sugomori followed the same approach as suggested in techniques: RBM, DBN, and dA. We used three variants of *deep evolution learner* using three different deep learning techniques: RBM, DBN, and dA. The DEL variants have its own advantages and disadvantages to analyze the system evolution. We used following three training parameters for deep learning: learning rate (LR), Epoch (Ep), and number of hidden units/layers of neurons (L). We did many (hundreds of) experiments to determine the model training parameters. The training parameter values that resulted in high accuracy and f-measure are presented in fifth column of the Table 6.1 with a sequence of LearningRate-Epoch-HiddenLayers. The DEL works like feed-forward neural network learning algorithm, which uses many iterations (i.e. epochs) during learning phase. Learning rate controls the learning of an algorithm. For all the dA experiments, we used 0.3 for the data corruption rate. To reduce the complication, in DBN we used same learning rate for pre-training and for fine-tuning.

- Third, for each experiment, the ***kFold_forwardChaining*** algorithm compares the *normalized output matrix* M_{NO} with the *testing matrix* M_T. The algorithm calculates the similarity between the two matrices. For each evolving system, the table contains group of three rows for three DEL variants: RBM, DBN, and dA.

The use of standard statistical techniques [187]: such as *precision*, *recall*, and *accuracy* provide a better assessment of the tool's performance. The precision is the measures of "how many of all connections have been recommended correctly". The recall is the measure of "how many of all missing connections have been recommended correctly". The accuracy is the measure of "how many of all connections and missing connections have been recommended correctly". Thus, precision, recall, and accuracy measure the percentage of correct recommendation. Mathematically, these metrics are re-formulated for our goal in the following way

$$precision = \frac{\#\ correct\ recommendation\ of\ connections}{\#\ all\ connections\ recommended\ by\ the\ tool}$$

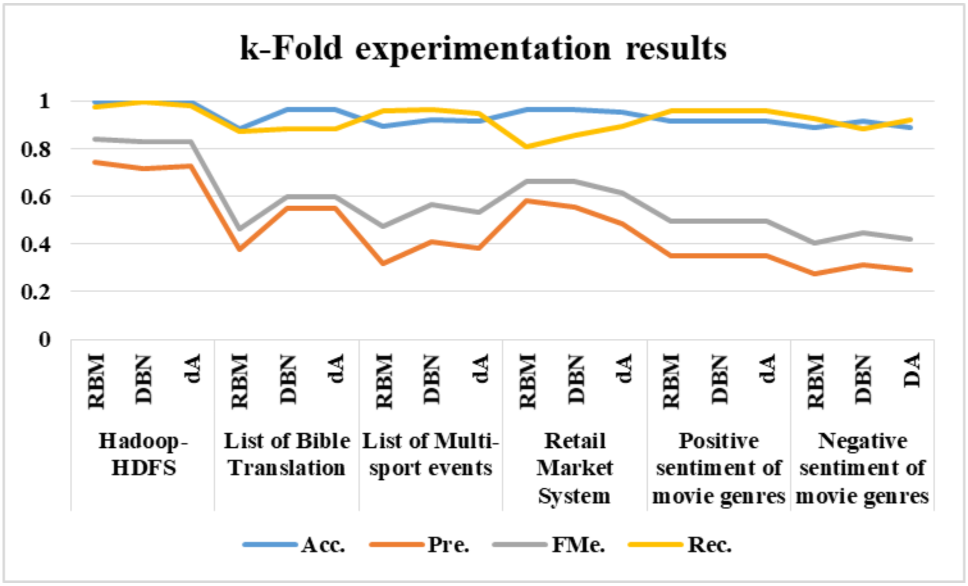
$$recall = \frac{\#\ correct\ recommendation\ of\ connections}{\left(\#\ \begin{matrix} correct\ recommendation\ of\ connections\ + \\ incorrectly\ reccomended\ connections\ as\ misconnections \end{matrix}\right)}$$

$$accuracy = \frac{\left(\#\begin{matrix} correct\ recommendation\ of\ connections + \\ correct\ recommendation\ of\ misconnections \end{matrix}\right)}{\#\ all\ possible\ entity\ connections\ i.e\ matrix\ size}$$

$$Fmeasure = \frac{2\ .\ precision\ .\ recall}{precision\ +\ recall}$$

where, # denotes "count of" and *misconnections* means missing connections.

Using these four metrics, we evaluate our recommender system. These metrics calculate similarity between the two binary matrices: *normalized output matrix* M_{NO} and *testing matrix* M_T. The similarity results are shown in Table 6.1, which are generated using the SysEvoRecomd-Tool. Higher value of a metric represents high similarity (means good result) and low value of a metric represents dissimilarity (means bad result). The accuracy metric represents the similarity between all the values



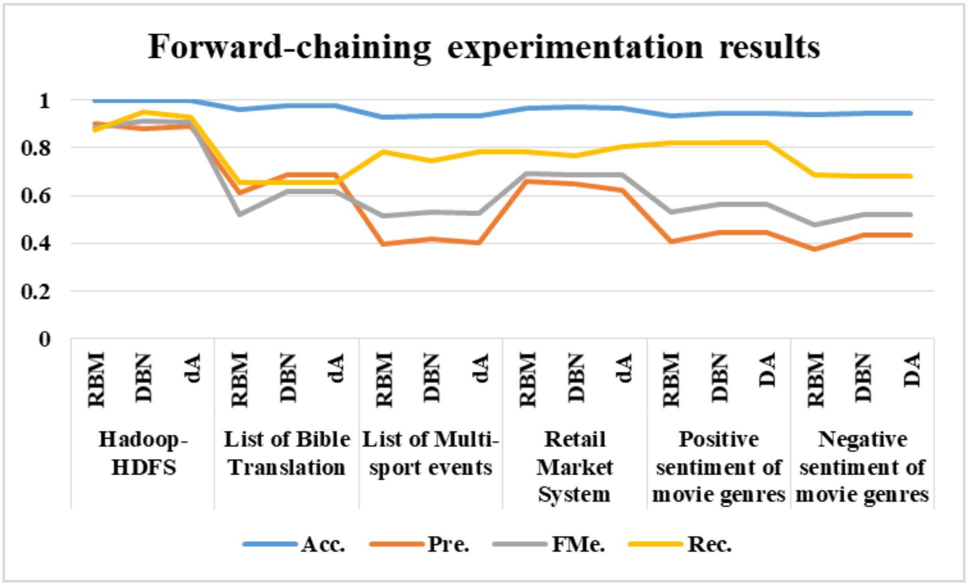


Figure 6.6 Graph plots to show experimental results for four metrics values in the vertical axis. The experiments done for six evolving systems are highlighted in the horizontal axis. The sequence of the values has same sequence mentioned for six evolving systems in the Table 6.1.

(0’s and 1’s) of two binary matrices, whereas precision, recall, and f-measure represent the similarity between all the values of 1’s. In all matrices at j th row and k th column, each ‘1’ represents there exist an *entity connection* and each ‘0’ represents connection do not exist between two entities (or features).

In Figure 6.6, we presented the two kind of cross-validation results in the two graphical plots. We plotted all the values of the four binary-metrics for the four domains mentioned in the Table 6.1. It shows domain B and D are harder to learn as compared to domain A and C.

<u>Observations during experimentations</u>:

Finally, we discuss our experience while conducting experiments. Our experiments are novel for studying system evolution. Nevertheless, we present following seven outcomes of our observations.

1. Learning and Memorization both are useful in recommendations: Like human being, a machine also needs both logical and memory oriented learning. Logical learning (or generalization) provides analytical power to the machine, whereas memorized learning (or specialization) provides recommendation power to the machine. Thus, memorization of old memory helps machine to recommend an upcoming and non-existing data. Generally, the *entity connection* patterns are almost constant in every system states, which can be learned and strored as a memory in the form of a reconstructed information (*normalized output matrix* M_{NO}). Thus, we did pattern learning and then stored that information in the form of an output matrix as a disk memory. This makes our results accurate and precise. *Hence, learning and memorization both are useful for a machine to mimic patterns like a human.*

2. We made two inferences from the Table 6.1, while comparing between natural language based systems and software systems. First, natural language based system data (list of bible translation and multi-sport events) are hard to learn for recommendation. Second, inter-procedural calls of software are easier to learn for recommendation. The medium of communications (e.g. natural languages) used by humans are significantly complex in nature, whereas medium of communications (e.g. inter-procedural calls) used by the computers are relatively less complex in nature. *Hence, learning of software systems (used by machines) are relatively easier as compared to learning of natural language systems (used by humans).*

3. Large size matrices are hard problem for deep learning. When we processed Hadoop-HDFS data system, it took large time to learn because of the EDSM size 14 × (3129 × 3129). Whereas, learning for the same environment, all the other dataset (with smaller EDSM) took lesser time as compared to the large EDSM of HDFS.

4. When an entity (or feature) is present in some states of training data but missing in other states of training data, it is hard to recommend about such entity. Generally, in machine learning algorithm the size of features are fixed. However, in our case number of entities (as features) is different in the different states. For example, some entities (software procedures or natural languge words) exists in some state but do not exist in other state. Thus, in the DSM of a state, we put zeros at all the places corresponding to the entities that do not exist in the state. We find an interesting observation that it is hardest to recommend about the entities (or nodes or features) that appeared in few training states. For example, when many entities are added to a new state of an evolving software corresponding to new functionalities. We find that it is hard for an algorithm to recommend about these new entities that never existed in old state used for training purpose. *Hence, it is hard to recommend the future connections for the entities (or features) that are absent in some states used in training.*

5. For learning purpose, our SSL solves the problem for pre-processing of an evolving system. Further, our DEL generates Deep SysNN that has information about evolution and changes happened in a system state series. *The SysNN consist of information about the relationship between system entities in various evolving system states.*

6. As our approach depends upon the deep learning, which also has some drawbacks e.g. it is complex and take time to learn. Relatively some algorithm (e.g. Support Vector Machine, Decision trees, and Regression) are simpler, faster, easier to learn from a dataset. *Thus, the drawbacks of deep learning are the drawbacks of our approach too.*

7. Accuracy metrics alone cannot give a justified measure for the correctness of a model, thus we used precision, recall and f-measure. The datasets used in the experiments (presented in this section) are imbalanced. Generally, imbalanced data means classes are not distributed equally and learning form an imbalanced dataset is a challenging task [193]. Specifically, in our case (see Table 6.2), distribution of zeroes (‘0’s) and ones (‘1’s) are uneven i.e. the number of ‘0’s are significantly higher than the number of ‘1’s. For entity connection recommendation, true positive TP (connection recommended correctly) has more significance as compared to TN (true negative). Similarly, false positive FP (connection recommended incorrectly) has more significance as compared to FN (false negative). Due to imbalanced dataset, we are getting high value of accuracy metric as compared to f-measure. Because accuracy metric compare the two matrices bit-by-bit assuming each recommendation has the same significance, whereas this is not our case. Thus, we used precision, recall and f-measure metric. *Hence, for imbalanced data, in addition to accuracy metrics it is required to use some other evaluation metrics like precision, recall, and f-measure.*

6.7.2 Experiments for System Evolution Recommender

We benchmark SysEvoRecomd-Tool for six evolving systems collected from six open-internet repositories. We collected data from four types of repositories: Wikipedia, Software repository, UCI repository, and IMDb. These system data have rich state or time instance history due to active contributions on them. We used these datasets because they provide many states to evaluate our system evolution recommender tool. For each evolving system, we made its combined matrix for the training phase of GECL. To study system evolution, we used GECL by using models of deep evolution learner based on the fundamental models of deep learning.

For each evolving system, we formed a set of evolving networks, which are further converted to an ECM (evolving connection matrix) for training purpose. The GECL (graph evolution and change learning) algorithm internally uses DEL (deep evolution learner) algorithm. The DEL algorithm used the ECM to generate a reconstructed connection matrix (as output), which does network reconstruction. For network reconstruction, we used deep learning approach of matrix reconstruction.

To study system evolution, we used three variants of GECL by using three models of DEL based on: RBM, DBN, and dA. The normalized output matrix (M_{NO}) represents the required reconstructed network, thus, there are three such network reconstructions by three variants of the GECL. To compare the correctness of recommendation by network reconstruction, we compared M_{NO} and M_T (*testing matrix*). For comparison, we used four metrics: *precision*, *recall*, *accuracy*, and *f-measure*. Next, we discuss experiments on six evolving systems using Figure 6.7, which helps to study correctness of recommendations done by SysEvoRecomd-Tool.

The Figure 6.7, helps to study correctness of automatic recommendations done by SysEvoRecomd-Tool. Each (time series) graph has the horizontal axis representing the state series of the evolving system and the vertical axis represents value of metrics between (0-1). The state series are: version series for HDFS; century series for Bible translation; decade series for Multi-sport events; month series for Frequent market basket; and decade series for Positive and Negative sentiment of IMDB movie genres. We made following inferences from the time series.

<u>Hadoop-HDFS:</u> The reconstructed matrix is almost similar to all the trained versions (except slightly dissimilarities with some earlier versions: 2.2.0, 2.3.0 etc.). The testing metrics values are slightly less as compared to the last two training versions (2.7.0 and 2.7.1). In the graph, recommendation done by the reconstructed network matrix for testing version (2.7.2M_T) is significantly satisfactory.

<u>List of Bible translation:</u> The reconstructed matrix is almost similar to last two training data of centuries (15-16-17-18 and 19) and dissimilar to the earlier two training data of centuries (7-8-9-10 and 11-12-13-14). The testing metrics values are slightly less as compared to the last two training data of centuries (15-16-17-18 and 19). The recommendation done for the testing century (20M_T) by the reconstructed network matrix is significantly satisfactory.

<u>List of multi-sport events:</u> The reconstructed matrix is 50% (approx.) similar to all the trained decades. The testing metrics values are slightly less than the 50% (approx.) training decade. The recommendation done for the testing decade (201M_T) by the reconstructed network matrix is

satisfactory for the randomness (entropy) between states used for training.

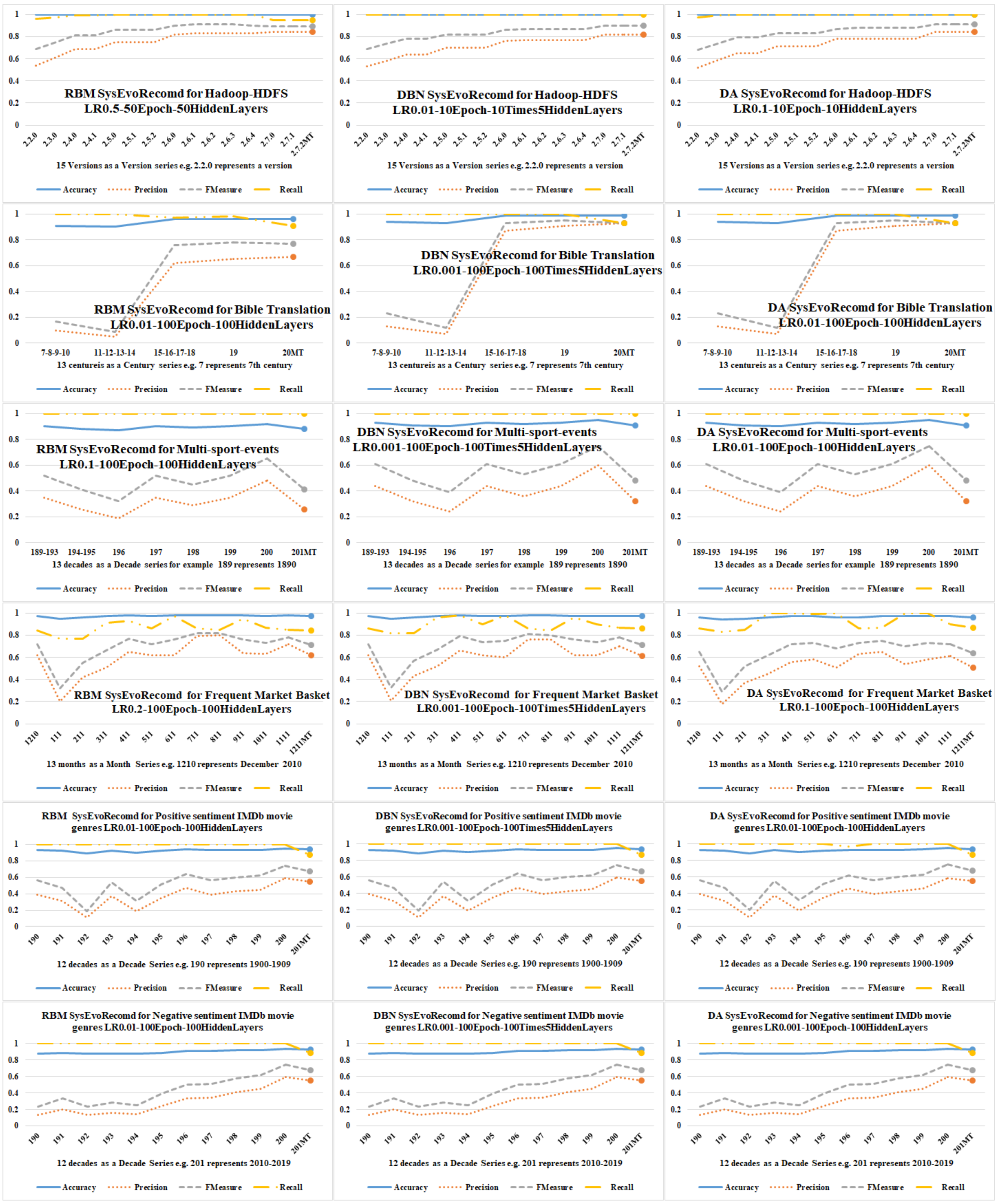


Figure 6.7 The three graphs for three variants of GECL applied on an evolving system. Each graph contains four time series with a metric value for similarity between M_{NO} and connection matrix of a state (in horizontal axis). Last value of each time series is a metric value for similarity between M_{NO} and M_T.

Frequent market basket: The reconstructed matrix is almost similar to last three training data of months (911, 1011, and 1111) and dissimilar to the training data of 111 i.e. January month of 2011. The testing metrics values are slightly less as compared to the training data of last eight months. The recommendation done for the testing month ($1211M_T$) by the reconstructed network matrix is satisfactory.

Positive sentiment of movie genres: The reconstructed matrix is almost similar to last two training decades (199 and 200) and slightly dissimilar with training decades (191, 192, 194, and 195). The testing metrics values are slightly less as compared to the last training decade 200 i.e. 2000-2009. The recommendation done for the testing decade ($201M_T$) by the reconstructed network matrix is satisfactory even for the randomness (entropy) between consecutive states in the training. Thus, the recommendations are useful.

Negative sentiment[5] of movie genres[6]: The reconstructed matrix is almost similar to last two trained decades (199 and 200) and dissimilar with earlier decades (190, 191, 192, 193, 194, 195, and 196). The similarity of the reconstructed matrix is progressive with respect to the states (time points). The testing metrics values are slightly less as compared to the last trained decade 200 i.e. 2000-2009. The recommendation done for the testing decade ($201M_T$) by the reconstructed network matrix is satisfactory even for the randomness (entropy) between consecutive states in the training dataset. Thus, the recommendations are useful.

Managerial Applications of SysEvoRecomd: The SysEvoRecomd-Tool helps to automate the process of system development, evolution, and maintenance for both existing and upcoming states. The time variant (or nonstationary) data of four domain are modeled as evolving systems, then we used SysEvoRecomd-Tool leading to following advantages.

- First, it can help to predict connection between entities: call between two procedures, link between two words, purchasing of items by a customer, and relation between two movies.

- Second, it can also be helpful while upgrading software system, correcting natural language errors, improving retail market distribution, and targeting audience for a genre.

- Third, it is helpful during software development, rephrasing a text, doing target marketing, and while naming a movie.

- Fourth, it can assist system employees e.g., software programmer, writers (of book, article, and novel), retail market (sales, finance, and marketing team), and film production team.

- Fifth, it can help to speed-up the software development, predictive text, customer billing, and selecting movie name.

- Sixth, it can do automatic correction of some errors during software debugging, autocorrecting while writing text, customer satisfaction, and re-making a movie.

- Seventh, it can determine possible future of the software, text usage trend, market analysis, and videos naming.

6.7.3 Comparison with other similar techniques

Karniel and Reich [194] proposed a use of DSM for process planning and conversion of a DSM based plan to a DSM-net process. Recently, Kosari [195] studied Work Transformation Matrix (WTM) at the level of Coupled Blocks of Activities (CBAs); they also provided equivalency between properties of dynamic systems modeled in state space and numerical process DSMs. Whereas, we presented an approach to use DSM for neural network training, which is a novel approach. We described a learning technique (using existing deep learning algorithms) that can use an evolving matrix (like an EDSM) to learn about evolution and changes. The EDSM extends existing theory of DSM.

Recurrent Neural Network (*RNN*) [196] is a type of ANN that has directed cycle to capture dynamic behavior of time-variant or sequence data. The RNN can work with the dynamic systems such as state series (like evolving software). A RNN captures the dynamic behavior of the temporal data. Whereas, we represent system data into an evolving matrix, thereafter, we used system structure learning models for learning purpose that creates SysNN (a novel type of ANN). The SysNN is more specifically representing a system's structure, whereas RNN is a generalized group of artificial neural network.

Recently a class of learning model named Memory Networks [197] is popular because it learns from a given knowledge for answering questions correctly. Thereafter, Sukhbaatar *et al.* [198] demonstrated a neural network with an explicit memory and recurrent attention mechanism for reading the memory; they trained neural network using backpropagation on diverse tasks from question answering to language modeling. Ankit *et al.* [199] introduced dynamic memory network (DMN) to process input sequence and questions, which forms episodic memories to generate relevant answers. They used recurrent neural network (RNN) and long-short term memory (LSTM) for learning purpose. In addition to such ANNs, we proposed Deep SysNN, which is based on a feed-forward deep neural network technique for learning the system state series.

Long-term memory enhances capability to remember and understand relationships between two separate entities, objects, and features. This is proven in case of dolphins [200], chimpanzees and orangutans [201], and hyenas [202]. Similarly, we did learning and memorization, which is useful in repeating (or mimicking) patterns of system entity connections.

Gulli *et al.* [203] invented a method and a system to store, extract, and analyze entity from the temporal content to present temporal trends according to user search query. Recently, Rishwaraj *et al.* [204] presented an approach of temporal difference learning using Markov games and heuristics to estimate trust in multi-agent systems. Instead of Markov's theory, we used deep learning based approach to present evolution and change learning of an evolving system.

Li *et al.* [205] proposed a RBM model for representation learning on linked data. Poria *et al.* [206] proposed a method to extract and learn features from visual and textual modalities based on deep convolutional neural networks for emotion recognition and sentiment analysis. For three variants of DEL, we observed an obvious result that recommendation done by DBN and dA has outperformed RBM. Our proposed work has applications to realize a hardware device using theory of Spiking Neural Network (SNN) [207] based on proposed System Neural Network (SysNN).

6.8 Summary

This chapter introduced evolution and change learning that forms *System Neural Network* (SysNN) a type of ANN. We exploited this concept to propose a *System Structure Learning* (SSL) algorithm, which internally uses *evolution representor* and *deep evolution learner* (DEL). The DEL uses an *Evolving Design Structure Matrix* (EDSM as *evolution representor*) to construct a feed-forward neural network i.e. Deep SysNN that contains information of evolving system states.

We have also proposed a *System Evolution Recommender* (SysEvoRecomd) algorithm that internally uses *Graph Evolution and Change Learning* (GECL) and *Deep Evolution Learner* (DEL). The GECL studies a given sequence (or series) of directed graphs using deep evolution learner algorithm. The DEL algorithm uses an *Evolving Connection Matrix* (ECM) and outputs a *Deep System Neural Network*. Based on the proposed algorithm, we constructed an automated tool named as SysEvoRecomd-Tool. We found that our tool done promising network reconstruction. Our SysEvoRecomd is novel, useful, and sensible as it uses inherent matrix reconstruction characteristics of deep learning.

Based on the proposed approach, we developed a prototype tool named as SysEvoRecomd-Tool to analyze time variant data (or non-stationary data) of an evolving system. We conducted experiments on six evolving systems collected from open internet repositories of four different domains. We demonstrated our approach using three variants of DEL based on three deep learning techniques namely, RBM, DBN, and dA. For an evolving system, each of the variant uses an EDSM as input to generate three Deep SysNNs. As an observation, we found that the Deep SysNN is a feed-forward neural network that has ability to learn and memorize information of evolving system's states by reconstructing vector of entity connections. We found our SysEvoRecomd-Tool is useful for system evolution analysis by doing system evolution recommendations.

To demonstrate automation capability of SysEvoRecomd-Tool, the experiments on six evolving systems demonstrated useful application of System Structure Learning. The experimental results are satisfactory according to the four metrics: accuracy, precision, fmeasure, and recall. This contributes to an emerging field of managing an evolving system represented as a state series. The six evolving connection matrix used as the input are huge and computational expensive for real-world applications.

However, our efficient Java codes based on deep learning solved such real-world applications in feasible-time. In future, our technique can be applicable on other kinds of evolving systems to do temporal analysis. Instead of using deep learning, another machine learning technique can be used.

Chapter 7

Big Data Evolution Analytics for Evolving Scholarly Systems

Era of developing world leads to the growth of several systems that evolve with time. Such evolving system may also create big data, which also evolves with time. Analysis of big data from such evolving system needs big data analytics techniques. We introduce an approach to do *Big Data Evolution Analytics* (BDE-Analytics) for evolving big data generated through evolving system. We proposed BDE-Analytics in two steps. Firstly, we pre-processed a given evolving big data to make multiple states as a state series SS = $\{S_1, S_2,.. S_N\}$ of an *Evolving Big Data System* (EBD-System), where S_i is a state of i^{th} instance. Secondly, we mine *Big Data Evolution Rules* (BDERs) that are further post-processed to generate *Big Data Evolution Concepts* (BDECs) for the given state series of EBD-System. These BDERs and BDECs as information are further useful to do EBD-System evolution analysis. We demonstrate this approach on an *evolving scholarly system* of Microsoft Academic Graph (the dataset of KDD CUP 2016). We presented experimentation results as *evolving scholarly topics* and graphical scholarly evolution information of past 2 centuries i.e. 217 years (1800-2017). Additionally, we presented the trend of *evolving scholarly fields* (i.e. area of studies) for the past 217 years.

7.1 Introduction

Big Data is a term that describes large volumes of high velocity, complex, and variable data that require advanced techniques and technologies to enable the storage and distribution, management, and analysis of the information [208]. Big Data pre-processing is an essential step for unstructured data. Therefore, pre-processing is required to prepare big data for analysis [209]. Even structured data requires some amount of pre-processing techniques that includes numeration, discretization, principal component analysis, feature reduction etc. After structuring a given big data, various big data analytic techniques [210] are used to analyse the given big data for identification of undiscovered correlations, patterns, and other useful information.

An evolving system may create evolving big data such that its entities keep on evolving (or incrementing) with time. The evolving big data can be pre-processed and represented as a state series. This kind of state series is similar to a data series but because system data is changing depending on the time instance, thus it is better to refer it as state series instead of data series. Data mining can be performed on each state separately and then merge the retrieved information of each state to retrieve the accumulated information. Thus, in this way we do Big Data Evolution Analytics (BDE-Analytics) on state series of an evolving big data generated by an evolving system. To do BDE-Analytics, we use text mining and association rule mining to retrieve knowledge and information from the state series of evolving big data.

The text mining is helpful to recognize the pattern in the evolving big data, thereafter we feed the pattern to a data mining algorithm [211][212] like association rule mining [213][214]. Association rule mining is concerned with identification of correlations within items of a dataset. These correlations can prove to be very useful as they can help in revealing novel, significant, and insightful information. Both text and association mining techniques are widely accepted and used to discover interesting and meaningful relationships between associated entities (or items) in a large dataset. We used them on evolving system entities in order to gain interesting insights about relationship between entities.

In this chapter, we propose an approach to do BDE-Analytics by breaking a big data into a state series $\{S_{1,}\ S_2\ldots\ S_N\}$, such that S_i represents i^{th} state of the Evolving Big Data System (EBD-System). Then we do mining on each state of big data separately to retrieve *big data evolution rules* as information of each state. Thereafter, we cleaned and arranged such information to retrieve our proposed *big data evolution concepts*. The proposed work is applied on real-world big scholarly data to retrieve *evolving scholarly topics* (as big data evolution concepts) in last 2 centuries (1800-2017).

The rest of chapter is organized as follows. Section 2 describes our approach to do Big Data Evolution Analytics. Section 3 describes experimentation done on evolving big data of evolving scholarly system over last 217 years. Section 4 and 5 presents related works and concluding remarks respectively.

7.2 Big Data Evolution Analytics

This section describes about the proposed approach to do BDE-Analytics. To begin with, we present an overview of our approach, which is shown in Figure 7.1. Firstly, unstructured big data is transformed into a representation that exhibits a higher degree of structuring as a state series. Secondly, we did *big data evolution rule mining* on each state to identify Big Data Evolution Rules (BDERs) information between associated entities. Further, we aggregate the retrieved BDERs to retrieve big data evolution concepts (BDECs), which are further used to do system evolution analysis of state series using system domain knowledge. In the proposed algorithms, we used following two notations:

- $a \leftarrow b$: This means b is assigned to a.
- $a \rightarrow b$: This means access element b inside a.

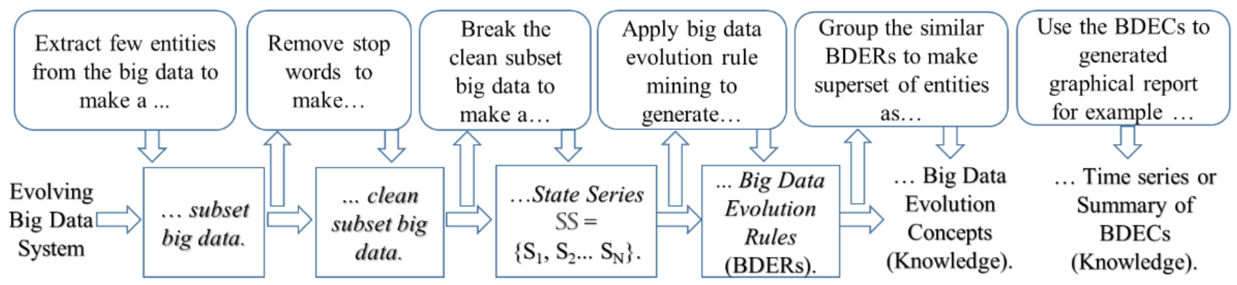

Extract few entities from the big data to make a ...
Remove stop words to make…
Break the clean subset big data to make a…
Apply big data evolution rule mining to generate…
Group the similar BDERs to make superset of entities as…
Use the BDECs to generated graphical report for example …
Evolving Big Data System
… subset big data.
... clean subset big data.
…State Series SS = {S1, S2... SN}.
... Big Data Evolution Rules (BDERs).
… Big Data Evolution Concepts (Knowledge).
… Time series or Summary of BDECs (Knowledge).

Figure 7.1 An overview of Big Data Evolution Analytics (BDE-Analytics). This figure shows Process-Artifacts of the BDE-Analytics that intelligently computes BDERs and BDECs. Where, each rounded-rectangle shows a process and rectangle shows artifact (i.e. input-output denoted by horizontal arrow). Here, a vertical arrow denotes each sentence starts from process at top and finishes at the artifact box.

Now, we present details about steps involved in our pre-processing technique.

7.2.1 Pre-processing

Parallel pre-processing of big data is often performed using Map-Reduce frameworks. Even in the specific case of evolving big data, preprocessing can be performed using these frameworks in order to produce a state series. To do the pre-processing, we extract required group of entities from a given big data. Then within the extracted dataset, we remove unwanted-entities (e.g. stop-words in case of natural language), which are the noisy-entities in the dataset. Thereafter, we prepare a state series containing evolving entities of an evolving big data, thus the data in the state series is structured over time. We explain this in following three sub-steps, which is our first contribution.

A) Extraction of the required entities. The initial dataset consists of many kinds of entities. However, only a few of them may actually be needed for data analysis. Therefore, based on the application, we select the kinds of entities that are required. We do this using a map operation where each transaction in given data is mapped to a corresponding transaction having only the entities that are required. he result of this step is referred to as *subset big data* in later sections. Here, the *normalized data* $name_{normalized}$ is data without punctuation and/or capitalization. It was one of the columns in the dataset. We have expressed this with formal manner given in Algorithm 7.1.

```
$Algorithm\ 7.1: Extract\ entites$
$\boldsymbol{map}(data_{initial})$:
$\boldsymbol{for\ each}\ line_{initialData} \in data_{initial}$:
$subset\ big\ data$
                    $= \boldsymbol{output}\big(line_{initialData} \rightarrow name_{normalized}, line_{initialData}$
                    $\rightarrow ID_{entity}, line_{initialData} \rightarrow S_i\big)$
```

```
$Algorithm\ 7.2: Remove\ stop\ words$
$List\ unwantedEntities \leftarrow \boldsymbol{getUnwantedEntitesOfData}()$
$\boldsymbol{map}(subset\ big\ data)$:
$\boldsymbol{for\ each}\ line_{selectedData} \in\ subset\ big\ data$:
        $\boldsymbol{for\ each}\ noisyEntity \in line_{selectedData} \rightarrow name_{normalized}$:
            $\boldsymbol{if}(noisyEntity \notin unwantedEntities)$:
            $unwantedEntities \leftarrow unwantedEntities + noisyEntity$
$\boldsymbol{assign}(line_{selectedData} \rightarrow name_{normalized}) \leftarrow unwantedEntities$
$clean\ subset\ big\ data$
            $= \boldsymbol{output}\big(line_{selectedData} \rightarrow name_{normalized}, line_{initialData}$
            $\rightarrow ID_{entity}, line_{selectedData} \rightarrow S_i\big)$
```

B) Removal of unwanted-entities. In big data, there might be some unwanted-entities and noise. The unwanted-entities are inconsistent and irrelevant entities that may increase the noise in the result without adding any considerable value to the meaning of the result. In order to gain insightful and relevant results, it is always necessary to reduce the noise as much as possible. Thus, the extracted data (*subset big data*) needs data cleaning i.e. pre-processing for removal of unwanted-entities. This cleaning process makes a *clean subset big data*. This is again performed using a map operation where each transaction is mapped to a corresponding transaction with noise removed. The method to remove the noise depends upon the domain of application. Thus, the map expression for this step can be written as in Algorithm 7.2.

C) Breaking big data into state series. To perform a time variant analysis, we need to make separate datasets for different states of time instances. In this step, we use Algorithm 7.3 to make state

```
$Algorithm\ 7.3: Make\ State\ Series$
$\boldsymbol{map}(clean\ data)$:
$\boldsymbol{for\ each}\ line_{filteredData} \in clean\ data$: // key-value pair
$output\ \Big(line_{filteredData} \rightarrow S_i, \big(line_{filteredData} \rightarrow name_{normalized}, line_{filteredData} \rightarrow ID_{entity}\big)\Big)$
$reduceWrite\big(data_{(key,\ value)}\big)$:
$\boldsymbol{for\ each}\ (key, value)\ as\ \Big(S_i, \big(name_{normalized}, ID_{entity}\big)\Big) \in data_{(key,\ value)}$:
$openFile(S_i)$    //Open file for writing
$stateFile.append\big(name_{normalized}, ID_{entity}\big)$
                $StateSeries_N = output\ collection\ of stateFiles\ of\ N\ state$
```

series from the clean data. We map the data into pairs, where a state is the first element in the pair and the other Entities is the second element. Thereafter, combine the resulting "paired" data in the form of Key-Value pairs of a State and Entity, by modifying the way outputs Key-Value Pairs. In Algorithm 7.3, which describes state series construction, where S_i represents a data of ith state.

7.2.2 Big Data Evolution Rules and Concepts

After data pre-processing, the big data state series can be used to mine evolution rules. This process is of three steps. Firstly, generate BDERs for a state. Secondly, we transform a set of similar BDERs to a BDEC for better understanding. Because the BDERs are large in number due to the size of big data and hence are less comprehensive. Moreover, as a final step, we clean up the set of BDECs obtained for unwanted-entities.

A) Big Data Evolution Rules of Entities. In the BDER mining algorithm, we retrieved list of unwanted entities from a file stored in a local directory. We defined a parameter, *window size*, which is the number of continuous states. This help to create a *window* of states to select fixed number of continuous states from a state series. The algorithm uses each window for BDER mining purpose. The states in a window are combined sequentially, which makes a dataset (*windowSizeData*) for rule mining.

B) Big Data Evolution Concepts of Entities. We now group a set of similar BDERs together to make the largest superset of contained entities. It is possible to analyse the rules when there are only 2 or 3 entities present in the rules. However, it is hard to comprehend when the number of entities is larger than 4 or 5 or more entities; this makes many similar BDERs. Steps of grouping similar BDERs are given next.

Suppose there are k entities $e_1, e_2 ... e_k$, such that there is a BDER R_k: $[e_1, e_2, \dots e_{k-1}] \rightarrow [e_k]$.

```
$Algorithm\ 7.4: Big\ Data\ Evolution\ Rule\ Mining$
$List\ unwantedEntities \leftarrow \mathbf{getUnwantedEntitesOfData}()$
$Integer\ \ windowBase = x, i = 0$
$\mathbf{for\ each}\ S_{i\,+\,base} \in StateSeries_N\ ;\ \mathrm{i + base \leq windowSize\ ; i++}$
    $windowSizeData = \mathbf{map}(S_{i+base}):$
    $\mathbf{for\ each}\ line_{selectedData} \in S_{i\,+\,windowBase}:$
    $\mathbf{for\ each}\ noisyEntity \in line_{selectedData} \rightarrow name_{normalized}:$
        $\mathbf{if}(noisyEntity \notin unwantedEntities)$
            $unwantedEntities \leftarrow unwantedEntities + noisyEntity$
            $\mathbf{assign}(line_{selectedData} \rightarrow name_{normalized}) \leftarrow unwantedEntities$
$BDERs = \mathbf{RuleMining}(windowSizeData)$
```

Since, R_k is a one of the generated BDERs, using the *downward-closure property* (every subset of a frequent item set is also frequent), we can successfully conclude that all such rules containing subset of entities in R_k would also exist in the generated BDERs.

The number of subset is $2^{j-1}-1$ where j is the total number of entities in the BDERs. This number would grow further to 3, 7, 15, 31, 63 or more as the number of entities in the rules would increase. Hence, it would only lead to redundancy if we wish to study the BDERs.

In the case of concept generation, both the order of the BDERs and the order of entities in BDERs do not matter. Swapping the entity on the LHS and the entity of the RHS keeps the BDER with same significance. It is now safe to conclude that we can group similar BDERs into a single large superset named as Big Data Evolution Concept (BDEC):

$$BDEC = \{e_1, e_2, e_3, \dots e_m\} \quad \dots \text{(Equation I)}$$

where, BDEC is a set that consists of all entities (on LHS and RHS) of given rules that are grouped, provided there exist rules with k number of entities (in the generated rules) of the form:

$$[e_1, e_2, \dots e_{k-1}] \rightarrow [e_k] \quad \dots \text{(Equation II)}$$

$$\text{where } \left((i_j, j) \in N\right) \wedge (k \leq m) \wedge \left(i_j \leq m\right)$$

Note that number of entities in the rule k is less than (or equal to) the number of entities in the BDEC(m). Thus, we take all rules of the form in Equation II, and form BDEC in Equation I. Also, note that the e_k in BDEC should exist in at least one of the Equation II using which the BDEC is formed.

```
$Algorithm\ 7.5: Big\ Data\ Evolution\ Concepts$
$maxSearch(BDER)$:
$maxLength \leftarrow 0$
  $\textbf{for each}\ BDEC \in BDERS$:
    $\textbf{if}\ BDER \subseteq BDEC\ \textbf{and}\ BDEC.length \geq maxLength$:
      $mappedSuperSet \leftarrow BDEC$
      $maxLength \leftarrow BDEC.length$
  $return\ mappedSuperSet$
     $map(BDECL)$:
  $\textbf{for each}\ BDER \in BDECL$:
    $output(maxSearch(BDER))$
     $reduceWrite(BDECs)$:
  $write(distinct(BDECs))as\ BDECL_{final}$
$\textbf{Output}\ BDECL_{final}$
```

Additionally, each entity in BDEC occurs in at least one of the rules that are grouped. Alternatively, we did this in three steps. Firstly, convert each rule into a corresponding set containing all the entities on its LHS and RHS, and create a list BDECL of all such sets. Secondly, map the largest superset for all the BDECs in the BDECL. Thirdly, reduce the list of largest supersets into distinct supersets and call it $\text{BDECL}_{\text{final}}$. Thus, $\text{BDECL}_{\text{final}}$ consists of all the desired sets of entities that form a *big data evolution concept*. We present the Algorithm 7.5 to make BDECs.

As an illustration, let us consider a rule R_4 with four entities: A, B, C, and D such that this forms a list of similar rules: {A, B, C → D}; {A, B → D}; {A, C → D}; {B, C → D}; {A → D}; {B → D}; {C → D}. Here, for j = 4, number of rules are, 2^{4-1} - 1 = 8 - 1 = 7, such that this forms a superset rule as $[A, B, C] \rightarrow [D]$. This forms a superset of entities (A, B, C, D) which removes redundancy in rules and makes it easier to analyse.

7.2.3 Post-processing based on BDERs and BDECs

It is tricky to remove unwanted-entities from concepts. The primary intention of using concepts made up of entities is to look for trends associated with such concepts. An entity that is misconceived, irrelevant, and inconsistent in a concept is an unwanted-entity, which is a noisy-entity causing hassle in analyzing a BDEC. Noise may be due to several issues depending on the domain of the EBD-System. Thus, a system domain expert decides, whether an entity is a noisy-entity or not. Noise can also be due to their vagueness in concept formation. If there is a noisy-entity in a BDEC, then it is better to exclude the noisy-entity.

Although, domain expert tries to remove noise by keeping all the unnecessary entities in the unwanted-entities list (see sub-section 2.1.2). However, it is possible to get unnecessary entities in the concepts. Thus, to retrieve the BDECs, there are two ways to remove such noisy-entities.

- First way is simple; remove noisy-entities (automatically or manually) from the generated BDECs.
- Second way is better; before constructing the big data state series (see sub-section 2.1.2), repeatedly add the noisy-entities to the unwanted-entities (see sub-section 2.1.1). To do this, add the noisy-entities (appearing in the BDERs) to the list of unwanted-entities, and then re-construct the big data state series of the given EBD-System. Further, re-generate the BDERs for all states from the new big data state series. The re-generated BDERs may again contain some other noisy-entities. Thus, it is required to repeat this step until the generated BDERs are satisfactory or containing noise-free entities.

We use the updated unwanted-entities list to recreate the big data state series with noise-free entities. Thereafter regenerate the BDERs, which are re-group the BDERs into largest superset of entities. Such entities' supersets are the BDECs that are the most frequently occurring associated entities of a state. In the end, a domain expert can analyse the retrieved BDECs. Usually reports or graphical representations of time series are generated for analysis. Such analysis varies from domain to domain. We present detailed statistical analysis of BDECs in the next section for the evolving scholarly system.

7.3 Experiments on Evolving Scholarly Big Data

This section describes experiment of our proposed approach on publication dataset fetched from KDD Cup 2016, “Microsoft Academic Graph”. The dataset contains “Publication Title” and “Year of Publications” in past 2 centuries. We used these two kind of entities to make a big data, which is further pre-processed to eliminate noise (irrelevant words in titles). Thereafter, we created state series with each state representing normalized data for a year. Thereafter, we did state series analysis of *clean subset big data* on publication titles. Our analysis reports the impact and trends of the various scholarly topics retrieved for the past 2 centuries (1800-2017).

We used experimental environment that has Spark and Hadoop Distributed File System (HDFS) because they combine to provide distributed storage along with scalable and fault tolerant processing of big data. In such environment, we did java based coding on Spark set-up that has the capability of performing faster distributed computing by using in-memory primitives [215]. Such coding style is well suited for fast and efficient processing of big data. In addition to this, we also used Python for web crawling and further statistical analysis.

We carried out the analysis on a machine with following configuration 32 GB RAM, i7-4770 CPU 3.40 GHz × 8 cores, 1000 GB Hard-Disk. The machine has operating system Ubuntu 14.04 along with HDFS, Spark, and Java for parallel processing. The objective of the experimentation is to verify usefulness of BDECs. It is clear that if number of node increases, then this improves efficiency of the computing. We are interested in showing significance of the EDB-Analytics experiment in generating BDERs and BDECs. The steps of data pre-processing for dataset of KDD Cup 2016 are given next.

7.3.1 Pre-processing

Initially, we used the Spark based parallel Java codes to pre-process the publication list that we stored on HDFS. Once we uploaded the data on HDFS, we used Java-Spark based parallel codes for the three algorithms on parallel-computing environment of HDFS. We implemented the algorithms 1, 2, and 3 on distributed programming models of Java-Spark-Hadoop. Using the codes, we broke the big data into datasets of state series, with one state for every year. Thereafter, this big data state series of yearly publication list is stored in HDFS. Next, the steps to construct a big data state series from the given data of publication.

7.3.1.1 Uploading data to HDFS. As a starting step, the dataset was uploaded to HDFS. The HDFS automatically splits the large input data set. This was because we intended to run Apache Spark to utilize efficient big data handling features of HDFS like splitting a big file (of the scholarly publication data) into smaller block files that are easy to read and process.

7.3.1.2 Data preparation. We prepared the big data state series based on the parallel pre-processing techniques (as in Algorithms 1, 2 and 3) using Spark. To validate our parallel implementation, we also generated state series using a serial (without using spark) java based pre-processing on some small block files taken from the HDFS. We compared the state series data generated from the serial pre-processing code and the parallel pre-processing code. The details of the parallel pre-processing steps for the scholarly publication data are provided in the following steps.

1. Entity extraction: Clearly, out of the many entities, we selected only two kind of entities – Paper Title and Year. Specifically, we were interested in using only the normalized name of publication and year of publication. The normalized name consisted of text in lowercase without any punctuation and symbols.

2. The name of publication may contain stop-words in the unwanted-entities list and the publication title needed a data cleaning pre-processing in which we removed the stop-words from the titles of the publication. Additionally, we removed noise like unwanted symbols and commonly used words in the publication that do not make up any keyword (or jargon) (e.g. result, study, scientist). We also removed the noise due to - formatting issues and vagueness of entities - while forming a scholarly topic. As we restricted our analysis to the English based scholarly data, hence we removed words and symbols of other languages. We also removed Roman numerals, place names, other language words, abbreviations that made no sense. This process leads us to make a list of stop-words (i.e. unwanted-enetities) for the scholarly publication titles in English.

3. Creating year wise datasets: We made separate dataset (state) for each year. This resulted in the formation of over 217 separate datasets as a state series (each state corresponding to a year in between 1800-2017). We mapped each line of the data into a pair, with Year as the first element and Publication Title Keywords (entities) as the second element. Now, we combined the resulting pairs in the form of Key-Value pairs of Year and Publication Title Keywords, by modifying the way Hadoop outputs Key-Value pairs. Then, by modifying Hadoop's MultipleTextOutputFormat, we were able to write data for a period of one year.

Next, we illustrate the evolving scholarly topics by retrieving the BDERs and BDECs for the constructed big data state series.

7.3.2 Evolving Scholarly Topic

Initially, we made the state series such that the data for each year is represented by a separate state. However, presenting rules for each year (state) separately is tedious because it gives 217 results (from 1800 to 2017). Thus, we combined 10 years data and considered a decade as a window. This means that the window size = 10 in the Algorithm 4 Big Data Evolution Rule Mining. This made 22 decades as windows for (217 years as 217 states), and on each we applied association rule mining. Then, we mined Big Data Evolution Rules using the Algorithm 4.

A popular association rule mining approach for big data mining is named as *Parallel Frequent Pattern Growth* (PFP-Growth), which is proposed by Li *et al.* [216]. For each state of the state series, we used PFP-Growth algorithm that is pre-implemented in spark machine learning library [217]. It is based on the Frequent Pattern Growth (FP Growth) algorithm [218][219], which is one of the most widely used association rule mining technique. Pattern growth algorithms unlike the candidate generation algorithm, do not generate all the possible candidate frequent patterns and verify, but instead complex hyper-structures are created and all the frequent patterns are extracted from such structures. Specifically, FP Growth algorithm uses a tree-based structure called FP-Tree and an associated link structure called FP-Link to extract frequent patterns. The PFP-Growth is a distributed and scalable approach to make growing FP-trees based on the suffixes of transactions. During the rule-generation phase, for each decade we used the PFP-Growth, and generated the BDERs.

A) Generating BDERs and BDECs. We generated BDERs from scholarly data of each decade (as a state). This gives BDERs for 22 decades as a decade series. To comprehend a set of similar BDERs we generated their BDECs (as evolving scholarly topics). To do this, we group a set of similar BDERs together in largest supersets. For example, a set of similar BDERs are as follows:
$[elements, effect, crops, trace, growth] \rightarrow \ [seeds]$
$[elements, effect, crops, trace, growth] \rightarrow [germination]$
$[elements, trace, growth] \rightarrow [crops]$
$[elements, trace, growth] \rightarrow [effect]$
$[germination, trace] \rightarrow [crops]$
$[germination, trace] \rightarrow [effect]$
$\ldots and\ 435\ more \ldots$

A BDEC equivalent to their largest superset is

$(elements, trace, crops, effect, growth, seeds, germination)$

With this, we removed redundancy from the BDERs as we can see from the above example the BDEC is more comprehensible than their BDERs. Here, we combined 441 rules into one largest superset. In experimentation, we observe that the BDEC is easier to analyse as compared to the number of BDERs. In an observation for a decade, we found that number of BDERs were reduced to a significantly lower number of BDECs.

B) Removing Noisy-Words from the rules. Even after removing stop-words during the pre-processing stage, we observe that there were many noisy words in rules generated for example $[xiestwa, poznanskiego] \rightarrow [wielkiego]$. These three noisy-words belong neither to English language nor to a field jargon. We realized problems with words (as entities) appearing in the BDERs and concepts. We primarily intended to use keywords of scholarly topics in English language to look for trends associated with various scholarly topics. Realizing the inconsistencies in the BDERs and BDECs made us exclude few noisy-words. The noise in the original data may be due to formatting issues, symbols, and other language words. However, due to any issue, we dropped noisy-words (as noisy-entities) from the dataset.

To remove the noisy-words from the generated BDERs and BDECs, we did the following steps. We analyzed the BDERs and BDECs for a decade and then added noisy-words (as noisy-entities) to the list of stop-words (as unwanted-entities in Algorithm 7.3). Therefore, such noisy-words needed to be added manually to the list of stop-words. We did this in an ordered fashion by adding noisy-words from each decade (chronologically) into the list of stop-words. Thereafter, using the final list of stop-words, we re-constructed the big data state series. For the new big data state series, we re-generated the BDERs and BDECs for a decade. We repeated this re-generation until proper BDECs for that decade. We did this process of re-generation of BDECs for all the decades. In last, we re-grouped the proper BDERs into their supersets as proper BDECs of every decade. These BDECs represent the most frequently occurring evolving scholarly topics in a decade.

C) Scholarly BDERs and BDECs. The rules generated were mostly in the form of:

$$[nervous, central] \rightarrow [system]$$
$$[central, system] \rightarrow [nervous]$$

This tells us that the words *central*, *nervous*, and *system* occur most frequently together. Thus, they occur prominently in many publication titles for a decade. We expressed above rule in the form of scholarly concept: *nervous central system*. We did this because there are many BDERs of same scholarly keywords, thus we merged the rule with same keywords to make a superset of keywords as a BDEC.

These keywords are the jargons (as entities) that describe the kind of scholarly field the paper titles. It makes easy to comprehend if we merge multiple similar rules representing one scholarly topic. A scholarly topic may belong to one or more scholarly fields. For example, the sample data given above tells that the words are related to Neurology, Neurosciences, and Biology. Hence, this gives inference that one of the prominent scholarly topic for the given decade is Neurology, Neurosciences, and Biology. Hence, we need an automated field detection method.

7.3.2.4 Analyzing Scholarly Fields. After the rule and concept generation, we did post-processing where we used concepts to identify respective scholarly field(s). We counted the number of available scholarly concepts for each scholarly field. This gave us information to compare different scholarly fields for a decade. In figure 7.2, 7.3 and 7.4, the horizontal axis represents decade series such that each ordinate is for a decade (e.g. 200 represents decade in between 2000 to 2009) and vertical axis represents the number of the scholarly concepts contributing to a scholarly field. Here, a BDEC represents an evolving scholarly topic, which means a collection of keywords occurring in a publication title.

Figure 7.2, 7.3, and 7.4 shows the number of frequent scholarly topics for some scholarly fields in 22 decades. For example, the number of frequent scholarly topics is highest for "Computational Intelligence" followed by "Theory of Computation" for the two decades of 21st century. Further, we can observe that the "Biochemistry" has highest in the number of frequent scholarly topics in many decades.

Manually determining the scholarly fields of the given concepts are infeasible due to many decades and the many number of BDECs per decade. Thus, this leads us to analyse many scholarly topics and fields. Additionally, since manually detecting the scholarly fields may lead to human error. For example, one might argue that *ascorbic acid* is a scholarly topic in both biology and chemistry. Thus, we need a threshold number of scholarly fields for each topic to normalise any errors in field identification. We realize that even for a number as small as 5, the number of fields might go up to 1000 or more inclusive of any errors. Hence, we implemented a python script to do automatic detection of scholarly fields for each BDEC.

To find the number of scholarly topics for a field in a decade we wrote and used a python script that does following three steps. Firstly, it sends a search query for a scholarly topic to the Springer website's search. Then, it downloads the resulting web page returned as the query result. Finally, it parses the field information given in the page.

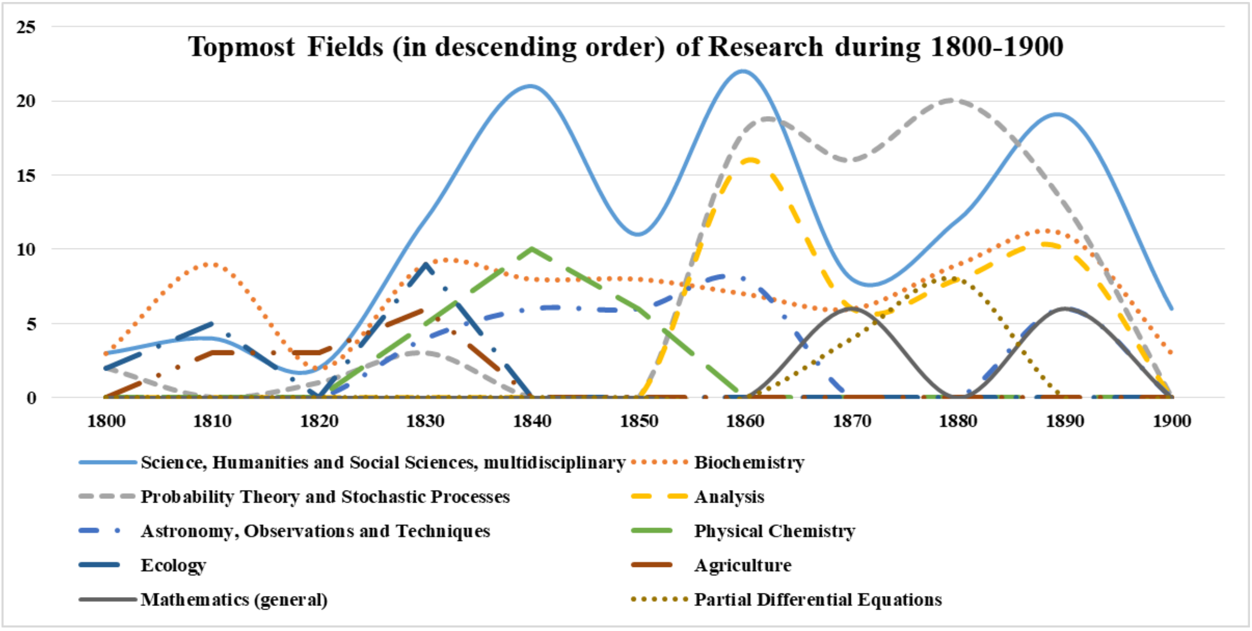


Figure 7.2 Comparison between the top-most scholarly fields of research based on their number of most frequent scholarly topics during 1800 to 1900 as a state series where each state is for a year and window is for a decade.

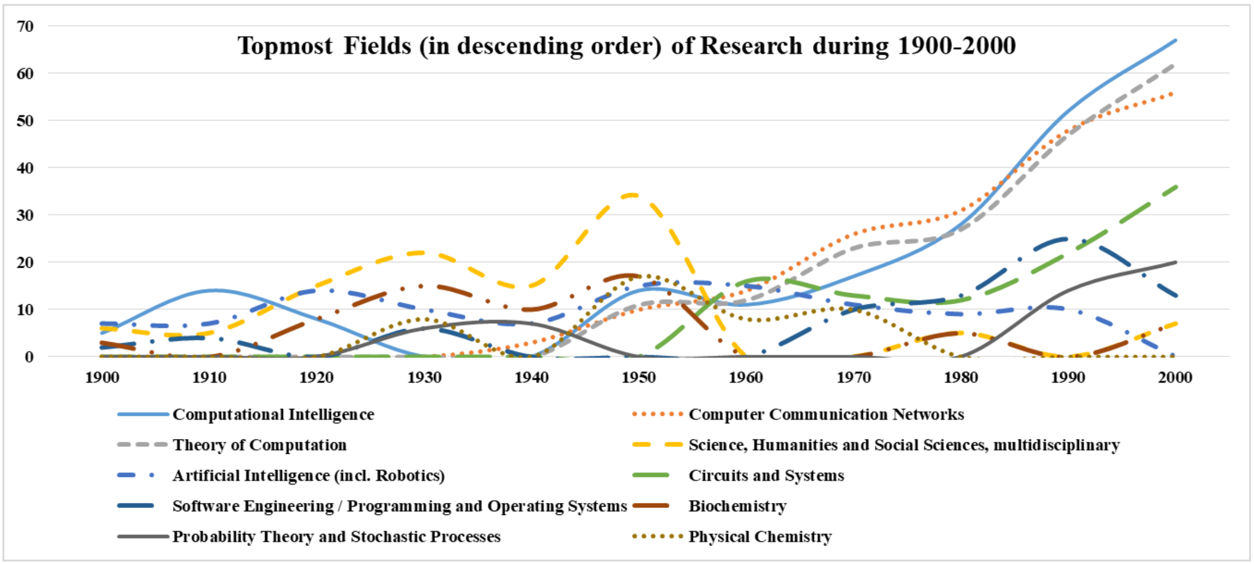


Figure 7.4 Comparison between the top-most scholarly fields, based on number of most frequent scholarly topics, during 2000 to 2017 as a state series where each state is for a year and window is for a decade.

On the extremely old data (of years 1800-1900), few of Springer's recommendations were different from the actual relationship between the scholarly concepts and the fields of research. Thus, we manually removed such wrong recommendation for the period of 1800-1900. Whereas, the recommendation for the period of 1900-2000 and 2000-2017 were appropriate, thus presented as it is. In Figures 7.2, 7.3, and 7.4, the results are aggregated over each decade separately.

In Figure 7.2, the top-most fields of research (or study) are presented for the duration in between 1800-1900. In this duration, the topmost fields are Science, Humanities and Social Sciences, multidisciplinary, Biochemistry, Probability Theory and Stochastic Processes, Analysis, and so on. In

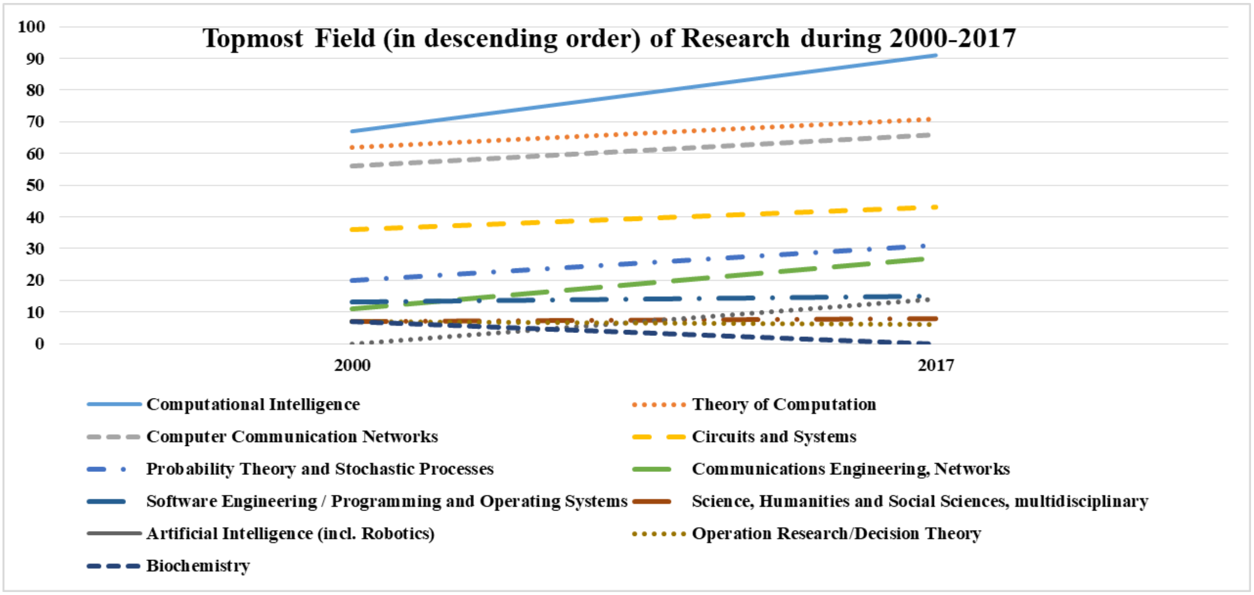


Figure 7.3 Comparison between the top-most scholarly fields of research based on their number of most frequent scholarly topics during 1900 to 2000 as a state series where each state is for a year and window is for a decade.

Figure 7.3, the top-most fields of research (or study) are presented for the duration in between 1900-2000. In this duration, the topmost fields are Computational Intelligence, Computer Communication Networks, Theory of Computation, Science, Humanities and Social Sciences, multidisciplinary, and so on. In Figure 7.4, the top-most fields of research (or study) are presented for the duration in between 2000-2017. In this duration, the topmost fields are Computational Intelligence, Theory of Computation, Computer Communication Networks, Circuits and Systems, and so on.

7.4 Comparison with other similar techniques

This section describes the current state-of-the-art in the fields of (a) evolution and change mining (b) big data rule mining or analytics, (c) KDD CUP 2016 dataset experimentations, and (d) scholarly big data mining. Work related to evolution and change mining are as follows. Recently, Namayanja and Janeja [220] presented big data processing techniques using MapReduce, to detect and monitor changes in large evolving computer networks by identifying the Time Periods of Change (TPC) associated with their consistency and inconsistency. Most recently, Basanta-Val [221] presented an approach for time-critical big-data system, which deals with requirements, analyses of the specific characteristics of some popular big-data applications. In this chapter, we extended the state-of-the-art of big data evolution analytics, which is demonstrated for evolving scholarly system.

Big Data Analytics or Mining uses the capability of extracting useful information from large datasets and data streams using data mining frameworks and services [222][223]. Conventional methods of data mining encounter several problems due to high computational costs. Therefore, a number of algorithms have been proposed to efficiently apply association rule mining of big data. Y. Chen *et al.* [224] proposed one such example named as Niche-Aided Gene Expression Programming (NGEP), which is compared with the FP-Growth and Apriori algorithms for environment pressure (Iris dataset) and an Artificial Simulation Database (ASD). Ölmezogullari *et al.* [225] discussed and developed a system to cope up with velocity of big data, specifically for big data mining. Sumbaly *et al.* [226] presented a survey of big data and machine-learning techniques used at LinkedIn and described how industries are applying big data mining techniques to their benefit.

The KDD Cup 2016 competition aimed at ranking affiliations of institutions based on predicting how many of their papers would be accepted in any upcoming top conferences [227]. It involved utilizing the given dataset, which included the Microsoft Academic Graph data and any other publically available data on the internet. Researchers all around the world proposed many solutions to the task. After the initial feature engineering process, many of the high ranked solutions incorporated attributes of various models for an optimal solution. Some notable examples included an approach by Qiu *et al.* [228] that combines Random Forest [229] and Gradient Boosting Regression Tree (GBRT) [230] as the dynamic approach of choosing different ensemble way in every phase. Qian *et al.* [232] described baseline, regression, and ranking SVM model to provide a final ensemble and average the output scores of all the models; conclusively stating that similar conference features were beneficial for prediction by dimensionality reduction. The winning paper was by Sandulescu and Mihai [233] that used the gradient boosting decision trees along with the mixed models approach, they inferred that the short-term factors and the longer-term contributions of an affiliation to a conference. All works [228][232][233] predict the relevance of the publication and institution. In contrast, our work performed scholarly evolution analysis.

Work related to the scholarly big data includes following. Xia Feng, et al [57] presented the survey of the background, relevant technologies, and analysis methods (like statistical analysis, social network analysis) that deals with big scholarly data. Tuarob *et al.* [234] proposed scalable techniques for AlgorithmSeer (a search engine for algorithms as part of CiteSeer) to identify and extract algorithm representations in a heterogeneous pool of scholarly documents. Datta *et al.* [231] studied factors that are related to the inter-domain presence of research topics. Zhou *et al.* [235] used integration of multi-source scholarly big data to study academic influence aware and multidimensional network analysis. Song *et al.* [236] proposed a Hadoop key-value based database named as FacetsBase, which store, index, and query scholarly big data. Zhang and Kabuka [237] proposed an implemented distributed hardware and software infrastructure for maintaining big scholar data as a large knowledge graph and calculating relationship among entities.

7.5 Conclusions

In this chapter, we presented the approach retrieved valuable information as *Big Data Evolution Rules* (BDERs) and *Big Data Evolution Concepts* (BDECs) about associations and trends in the evolving big data. For experimentation purpose, we had chosen scholarly system that produced a scholarly big data collected from KDD Cup 2016 of Microsoft Academic Graph as a publication bibliography dataset. This helped us to construct an evolving scholarly big data state series that we used to understand the impact made by various *evolving scholarly topics* (BDECs) that are frequently trending since last 2 centuries in scholarly publication titles. We were able to appreciate the gradual change in scholarly topics and fields with time. We also created visualizations of the data that is valuable for scholarly community (see Figures 7.2, 7.3, and 7.4). The figures demonstrate the top-most scholarly fields for each of the 22 decades in between the 2 centuries (1800-2017). In future, we will analyse the changeability, stability, and complexity of the EBD-system to do System Evolution Analytics.

1. SPMF, July-2018 [Online]. Available: http://www.philippe-fournier-viger.com/spmf/

2. acc-Motif: Accelerated Motif Detection, Jun 2017 [Online]. Available: http://www.ft.unicamp.br/docentes/meira/accmotifs/.

3. Sugomori Yusuke, “Deep learning source code” July 2018 [Online]. https://github.com/yusugomori/DeepLearning

Chapter 8

Service Evolution Analytics: Evolution and Change Mining of an Evolving Web Service System

An evolving distributed system is a kind of evolving system that often stored in a software repository to support its continuous evolution. Similar to systems in other domains [238], continuous changes to the requirements of distributed systems mandate their ongoing maintenance and evolution. This process becomes increasingly challenging as system complexity grows and thus tool support is needed to reduce the cost (or effort) of maintaining an evolving distributed system. We are interested in studying the *changeability* [5][239] and *evolvability* [240][241] of *distributed systems*. This study is based on *change mining*, which aims to discover change information, and *evolution mining*, which aims to discover evolution information.

Changeability and evolvability analysis can aid an engineer tasked with a maintenance or an evolution task. This chapter applies *change mining* and *evolution mining* to evolving distributed systems. First, we propose a *Service Change Classifier* based *Interface Slicing* algorithm that mines change information from two versions of an evolving distributed system. To compare old and new version, change classification labels are used: *inserted*, *deleted*, and *modified*. These labels are then used to identify subsets of the operations in our newly proposed Interface (WSDL) Slicing algorithm. Second, we proposed four *Service Evolution Metrics* that capture the evolution of a system's *version series* VS = {V_1, V_2 …V_N}. Combined the two form the basis of our proposed *Service Evolution Analytics* model, which includes learning during its development phase. We prototyped the model in an intelligent tool named *AWSCM* (*Automatic Web Service Change Management*). Finally, we present results from experiments with two well-known cloud services: Elastic Compute Cloud (EC2) from the Amazon Web Service (AWS), and Cluster Controller (CC) from Eucalyptus. These experiments demonstrate AWSCM's ability to exploit change and evolution mining.

8.1 Introduction

Computing models such as *Grid Computing*, *Cloud Computing*, *Utility Computing*, and *Service Oriented Computing* rely upon service frameworks. Grid Computing, the mother of all four, combines a distributed collection of computing resources to achieve scalability. *Cloud Computing* enables convenient, on-demand shared computing resources at several levels (e.g., at the infrastructure, platform, and software levels). *Utility Computing* enables an on-demand, pay-as-you-go billing method in which a service provider makes computing resources available to customers. *Service Oriented*

Computing supports loosely coupled distributed systems through the sharing of remote Web Service.

Our approach makes use of data mining techniques to extract information from an evolving distributed system. The particular distributed systems we focus on are web-based system and thus we focus our attention on web-based data mining. Traditionally, such web-mining techniques come in three types: web content mining, web usage mining, and web structure mining [242][243]. We consider the latter, *web structure mining*, which retrieves information from service interface documents (e.g., HTML, XML, and WSDL documents) as well as the underlying implementation. In doing so, we consider a cloud service as a time-varying dataset and apply techniques combining aspects of *change mining* and *evolution mining*.

Specifically this chapter applies *change mining* and *evolution mining* using an *analytic model* to study service changes and evolution. The analytic model exploits the history (e.g., as provided by a version-control system) to reduce the human effort required while analyzing the software evolution over the series of versions. In doing so, the chapter makes the following three contributions, which are presented in the next three sections:

- In Section 8.2, we propose a *service change classifier* algorithm that performs *change mining* based on two service versions. The change information is used to perform *interface slicing*, which results in an *interface slice* (a subset of service's interface). We also propose a technique for *evolution mining of multiple service versions* using four new *service evolution metrics*.
- In Section 8.3, we describe a *Service Evolution Analytics model* and its implementation as a prototype component in an existing tool, AWSCM, which performs *change mining and evolution mining of cloud services*.
- In Section 8.4, we present experiments applying AWSCM on evolving distributed systems.

After these contributory sections, we present related works in Section 8.5 and concluding remarks in Section 8.6.

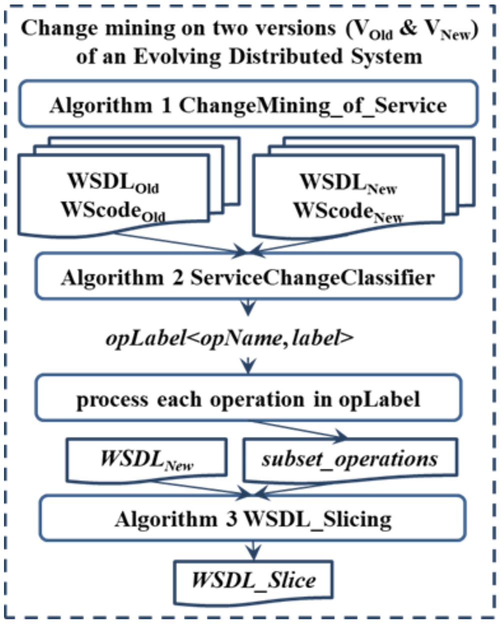
8.2 Change and Evolution mining of an Evolving Distributed system

This section describes how we perform *change mining* on two versions of an evolving distributed system and *evolution mining* on a version series from an evolving distributed system. Here a *version series* VS = $\{V_1, V_2 \dots V_N\}$ is a series of version snapshots taken at times $\{t_1, t_2, \dots, t_N\}$. Each version involves two key artifact: a specification written in a Web Services Description Language (WSDL) and the Web Service (WS) code itself.

The next two sub-sections provide details regarding the two kinds of mining used. The first sub-section explains change mining using change classifiers that are applied to two versions of a system. The crux of this step is classify changes between versions to make an Interface slice. Then the second sub-section introduces evolution mining and the four service evolution metrics used to study the evolution of an evolving distributed system. The crux of this step is to study evolution in a version series.

A. *Change mining of two versions: Service Change Classification*

The goal of the service change classification is to capture the changes between an old and a new version of a distributed system (thus the four parameters of the actual algorithm). We include changes at both the WSDL and the WS code levels. We first overview our *change mining* algorithm before considering each of its three steps in greater detail. The algorithm, ***ChangeMiningOfService***, is given as *Algorithm 8.1*. It takes old and new versions of the system as input and first invokes *Algorithm 8.2*, ***ServiceChangeClassifier***, which has label (*opLabel*) initialized operations with *unchanged* label. The *Algorithm 8.2* adds the following change labels to the WSDL and WS code of the new version: *inserted*, *deleted*, and *modified*. It then gathers up all operations (as *operation_j*) that are tagged inserted or modified. The gathered operations are kept is a set of operations (as *subset_operations*). Finally, it invokes *Algorithm 8.3*, ***WSDL_Slicing,*** to construct a Subset of the WSDL, referred to as the *Subset WSDL*, which is captured by the WSDL slice returned by the algorithm. These steps are summarized in Figure 8.1.



Change mining on two versions (V_{Old} & V_{New}) of an Evolving Distributed System
Algorithm 1 ChangeMining_of_Service
$WSDL_{Old}$ $WScode_{Old}$
$WSDL_{New}$ $WScode_{New}$
Algorithm 2 ServiceChangeClassifier
opLabel<opName, label>
process each operation in opLabel
$WSDL_{New}$
subset_operations
Algorithm 3 WSDL_Slicing
WSDL_Slice

Figure 8.1 Summary of the Change Mining of two versions.

In greater detail, the first step performs change classification on the WSDL and the WS code. Table 8.1 overviews the types of changes identified. Each change includes the level of the change (WSDL or WS code), the purpose of the change, and the WSDL Subsets (explained

```
Algorithm 8.1 ChangeMiningOfService (WSDL_Old, WSDL_New, WScode_Old, WScode_New)
opLabel<opName, label> ← ServiceChangeClassifier(WSDL_Old, WSDL_New, WScode_Old, WScode_New)
Initialize an empty set subset_operations
For each operation_j in opLabel where j represents operation number
  If label of operation_j is 'insert' or 'modify'
    subset_operations ← subset_operations ∪ {operation_j}
End for
WSDL_Slice = WSDL_Slicing(WSDL_New, subset_operations)
```

later in this section) that capture the change.

A WSDL description has six major parts [244], which are summarized in Table 8.2. The first part provides the *definitions* of the services provided. The second include the *schema XSD, which* represents the data structures for the input and output of each operation. The third part, *message calling*, supports communication between the client and the operation (we use the term "*operation*" to denote a procedure that is exposed as part of the service's external functionality available to a client). The fourth part provides the *port* used for each operation. Fifth is a *binding* that binds input and output data types to each operation. Finally, the sixth part defines the *service* that is accessed through a URL.

Table 8.1 Cloud service Change Classifiers

Label of the change	**Level**	**Purpose at location of change**	**Captured in Subset WSDL**
Insertion I_{WSDL}	WSDL	Insertion of an operation in WSDL or Datatype in XSD	DWSDL
Insertion $I_{WS\ Code}$	WS Code	Insertion of code for new operation	UWSDL
Deletion D_{WSDL}	WSDL	Deletion of operation from WSDL to make its disfunction to client	None. These changes are rare and unusual.
Deletion $D_{WS\ Code}$	WS Code	Deletion of operation from WS code to make disfunction at WSDL	
Modification M_{WSDL}	WSDL	Modification of XSD	DWSDL
Modification $M_{WS\ Code}$	WS Code	Modification at code lines without affecting WSDL	UWSDL

Table 8.2 Changes in WSDL Properties

Property	**Effect of changes on the properties of WSDL**
Definition change	Change in the xml version, name of service, xmlns or targetNamespace
XSD Type (schema)	The XSD includes sequence of two kind of data types in the form of XML elements: complex data types and simple data types. Changes may be in Simple Types or Complex Types, which are represented by elements, names, or data type. The simple data types contain only literal text, while the complex data types contain nested attributes and parameters.
Message	Changes may be in the message name of an operation. Changes may also be in the elements and part name of a message.
Port Type change	Change may be in the port type name or operation. Changes may also in the way message input and output were used.
Binding change	Change may be in the binding name and soap binding. Changes may also in the soap body (input - output).
Service change	Changes may be in the service name, binding address, or soap address location.

The goal of the first step is to classify the elements of the WSDL and the WS code. A change can be made to any of the following three service elements: the operation code, the messages used for communication, and the input-output parameters. A common approach from data mining used to classify items is to attach *classification labels*. We can therefore identify changed operations by labeling the three kinds of service elements. Step 2 simply gathers together all the operations (as *subset_operations*) marked inserted or modified.

The example shown in Figure 8.2 illustrates the first two steps. To begin with, the "Service version 1" is upgraded to make "Service version 2". In the "Service version 1", a *modified* 'Operation_11' calls *modified* 'Method_11' (which further calls 'Operation_12', which was *deleted*) and *modified* 'Operation_21'. Furthermore, 'Operation_13' calls 'Method_12', which were both *deleted* in "Service version 2". From the perspective of "Service version 2", *modified* 'Operation_11' calls *modified* 'Method_11' and *modified* 'Operation_21'. Furthermore, *inserted* 'Operation_12' calls *inserted* 'Operation_21' and subsequently *inserted* 'Method_21', and *inserted* 'Method_13'. Note the 'Operation_21' is provided by some other service.

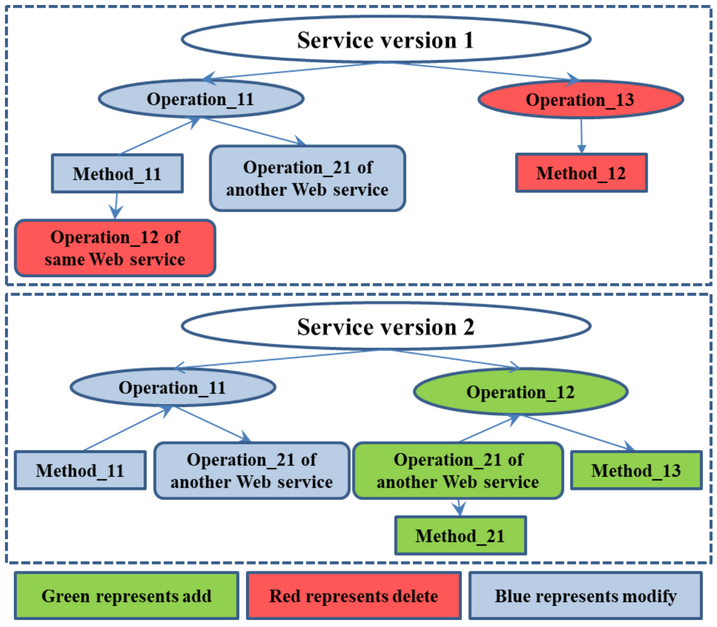


Figure 8.2 Simple representation of changes in a service.

*Algorithm 8.2, **ServiceChangeClassifier,*** takes four input artifacts: $WSDL_{New}$ (WSDL of new version), $WSDL_{Old}$ (WSDL of old version), $WScode_{New}$ (WS code of new version), and $WScode_{Old}$ (WS code of old version). The algorithm involves the following six steps.

1. Identify ***semantic WSDL difference*** between $WSDL_{New}$ and $WSDL_{Old}$ based on *semantic differencing*. Store the result in $WSDL_{Changes}$.

2. Identify ***semantic code difference*** between $WScode_{New}$ and $WScode_{Old}$ based on *semantic differencing*. Store the result in $WScode_{Changes}$.

3. Initialize the list *opLabel*, which will hold the labels of the differences between the old and new versions.

4. While scanning the $WSDL_{Changes}$ for operation changes:

   a. if an operation was added to the new version of the WSDL, then assign it the *inserted* label in *opLabel*.

   b. if an operation is deleted from the old version to create the new version of the WSDL, then assign it the *deleted* label in *opLabel*.

c. if an operation from the old version is modified while migrating to the new version of the WSDL, then assign it the *modified* label in *opLabel*.

5. While scanning the $WScode_{Changes}$ for operation changes:

a. if any line of code in an operation was added to the new version of the WS code, then assign it the *inserted* label in *opLabel*.

b. if any line of code in an operation is deleted to create new version of the WS code, then assign it the *deleted* label in *opLabel*.

c. If any line of code in an operation is modified while migrating to the new version of the WS code, then assign it the *modified* label in *opLabel*.

```
Algorithm 8.2 ServiceChangeClassifier($WSDL_{Old}$, $WSDL_{New}$, $WScode_{Old}$, $WScode_{New}$)

1. $WScode_{Changes}$ = semanticCodeDiff($WScode_{Old}$, $WScode_{New}$)
2. $WSDL_{Changes}$ = semanticWSDLDiff($WSDL_{Old}$, $WSDL_{New}$ )
3. Initialize array list opLabel<operation, label> with each operations with unchanged label
4. For each operation_j_change in $WSDL_{Changes}$
   a. If (operation_j_change  == insert)
        Assign label inserted to operation_j in opLabel
   b. If (operation_j_change  == delete)
        Assign label deleted to operation_j in opLabel
   c. If (operation_j_change == modify)
        Assign label modified to operation_j in opLabel
   End For
5. For each operation_j_change in $WScode_{Changes}$
   a. If (operation_j_change == insert)
        Assign label inserted to operation_j in opLabel
   b. If (operation_j_change == delete)
        Assign label deleted to operation_j in opLabel
   c. If (operation_j_change == modify)
        Assign label modified to operation_j in opLabel
   End For
Return opLabel<opName, label>
```

Note that in this algorithm the order of Steps 4 and 5, which scan the WSDL and the WS code, is important because the second scan may reset a label from the first. Thus, if the $WSDL_{Changes}$ analysis is done before the $WScode_{Changes}$ analysis, then the WS code analysis takes priority over the WSDL analysis. In contrast, if Step 4 and Step 5 are interchanged, then the WSDL analysis takes precedence. Furthermore, either step may be skipped enabling the approach to be applied separately by a cloud service owner or its consumer. In the implementation, finding the differences between the two versions of the WS code and the two versions of the WSDL is performed using a semantic differencing tool.

The algorithm depends on the accuracy of this tool in its ability to distinguish between the three kinds of changes: *insert*, *delete,* and *modify*.

The third step captures the changes between the old and new versions in a WSDL slice. We first formalize the concept of WSDL Slicing.

Definition: WSDL Slicing is a variation of interface slicing that extracts a subset of a given system's WSDL (as an interface) to capture a subset of the original WSDL's behaviour. The resulting interface slice is referred to as a WSDL slice and contains a subset of WSDL's operations.

For example, given a bank service that supports two operations 'deposit' and 'withdraw'. A WSDL slice might contain only the 'deposit' functionality.

As an interoperable subset of a service, a WSDL slice can aid an engineer (e.g., when testing) by focusing their attention on the relevant subset of the service. Furthermore, the slice of the distributed service's code can provide a similar focus. As seen in Table 8.1, our approach makes use of two WSDL subsets: the *Difference WSDL* (DWSDL) and the *Unit WSDL* (UWSDL) [58][59][60]. The DWSDL is constructed based on differences in the WSDL, while the UWSDL is constructed based on differences in the WS code.

The WS code differences are obtained from the WS change classification. This process first parses the old and the new versions of the code to separate-out the operations and methods of each, and then identifies differences including the location (file and line) of each change. On one hand, the DWSDL (a subset of the WSDL) is constructed to capture WSDL-level changes. The DWSDL is a type of WSDL Slice. On the other hand, building on the intra-operational and intra-procedural changes, we then identify operations that have inter-operational dependencies on changed operations and methods. These changed operations are used to construct the UWSDL, which is a WSDL Slice that captures changes at the WS code level. This makes the UWSDL useful to access and execute changes in the WS code. Based on the labels in opLabel, we capture changes in three ways:
- First, inserted operations are captured in both the DWSDL and the UWSDL.
- Second, deleted operations are ignored in the *Subset WSDLs* because they are not part of the new version of the service.
- Third, modified operations can occur in two places – in the WSDL and in the WS code. Modifications at the WSDL level mean changes in port, binding, and XSD (input, output, or both) as described in Table 8.2. The *DWSDL* captures these changes. Modifications at the WS code level are captured in the *UWSDL*.

Finally, *Algorithm 8.3* ***WSDL_Slicing*** takes the WSDL of the new version and the identified subset operations as input; these two are used to construct the *WSDL_Slice* using the functions sliceXSD, sliceMessage etc., which each extract a subset of the WSDL. These subsets (captured as substrings) are combined (concatenated) to form the *WSDL slice*.

A WSDL slice cleanly captures a semantically meaningful subset of a service's functionality. As noted above, it can be computed in the absence of the code. However, as WSDL provides access to the WS code and thus the WSDL slice can also provide access to a reduced version of the code supporting the regression testing.

B. Evolution mining of a Version series: Service Evolution Metrics

This sub-section describes evolution mining of a service using the four novel *service evolution metrics*. The metrics are based on five important quantitative attributes: *number of operations*, *WSDL lines*, *parameters*, *message*s, and *operation code lines*. These five attributes are used to generate a quantitative evaluation of a version series VS = {V_1... V_i... V_N} such that V_i represents i^th version of the service. Using the five attributes, we define four *Service Evolution Metrics*. The intuition behind the Service Evolution Metrics is to represent cloud service evolution information by aggregating attributes over multiple versions.

Algorithm 8.3 **WSDL_Slicing(***$WSDL_{New}$*, *subset_operations***)**

```
String cStartDef, cXSD, cMsg, cPort, cBinding, cService, cEndDef
Array of String[] operationOfWSDL_New
1. [subset_operations ∈ operationOfWSDL_New | subset_operations = operation required to construct WSDL_Slice ]
2. cStartDef = sliceStartDefinition(WSDL_New)
cXSD = sliceXSD(WSDL_New, subset_operations)
cMsg = sliceMessage(WSDL_New , subset_operations)
cPort = slicePort(WSDL_New, subset_operations)
cBinding = sliceBinding(WSDL_New, subset_operations)
cService = sliceService(WSDL_New)
cEndDef = sliceEndDef(WSDL_New)
3. WSDL_Slice = cStartDef + cXSD + cMsg + cPort + cBinding + cService + cEndDef
Return WSDL_Slice }
```

1) Effective lines per operation in the WSDL: the first metric is the ratio of number of WSDL lines (non-comment and non-blank) to the number of operations in the WSDL. For version V_i, $LOWSDL_i$ denotes the Lines per Operation in the $WSDL_i$ and is defined as follows:

$$LOWSDLi = \frac{Number\ of\ source\ lines\ in\ the\ WSDL\ for\ version\ i}{Number\ of\ operations\ in\ version\ i}$$

This defines a set as LOWSDL =

{(V_1 , $LOWSDL_1$)… (V_i , $LOWSDL_i$)… (V_N , $LOWSDL_N$)}

2) Parameters per operation in the WSDL: the second metric is the ratio of the number of parameters in the XSD to the number of operations in the WSDL. This reflects the expected growth in a service's interfaces. It is denoted as PO_i the Parameters per Operation in the $WSDL_i$ for V_i and defined as follows:

$$PO\_i = \frac{Number\ of\ parameters\ in\ the\ WSDL\ for\ version\ i}{Number\ of\ operations\ for\ version\ i}$$

This defines a set as PO =

$\{(V_1, PO_1)\ldots(V_i, PO_i)\ldots(V_N, PO_N)\}$

3) Messages per operation in the WSDL: the third metric is the ratio of the number of messages to the number of operations in the WSDL. Under normal conditions, the number of messages per operation is two, which reflects having one message for input and one for output. The metric is denoted as MO_i the Messages per Operation in the $WSDL_i$ of V_i and defined as follows:

$$MO_i = \frac{Number\ of\ messages\ in\ the\ WSDL\ for\ version\ i}{Number\ of\ operations\ for\ version\ i}$$

This defines a set as MO =

$\{(V_1, MO_1)\ldots(V_i, MO_i)\ldots(V_N, MO_N)\}$

4) Code Lines per operation in the WS code: the fourth metric is based on the logic behind the WSDL. It is the ratio of the number of lines of code to the number of operations. This reflects the growth of service operations in terms of business logic. It is denoted as $WSCLO_i$ the WS Code Line per Operation in $WSCode_i$ of V_i

$$WSCLO\_i = \frac{Number\ of\ code\ lines\ of\ WS\ code\ in\ version\ i}{Number\ of\ operations\ in\ version\ i}$$

This defines a set as WSCLO =

$\{(V_1, WSCLO_1)\ldots(V_i, WSCLO_i)\ldots(V_N, WSCLO_N)\}$

These four novel metrics can be expressed together as $(V_i, LOWSDL_i, PO_i, MO_i, WSCLO_i)$. With respect to the versions, these four metrics are used to create four time series graphs that are useful in the study of the evolving distributed systems. The evolution mining of a version series using Service Evolution Metrics is summarized in Figure 8.3.

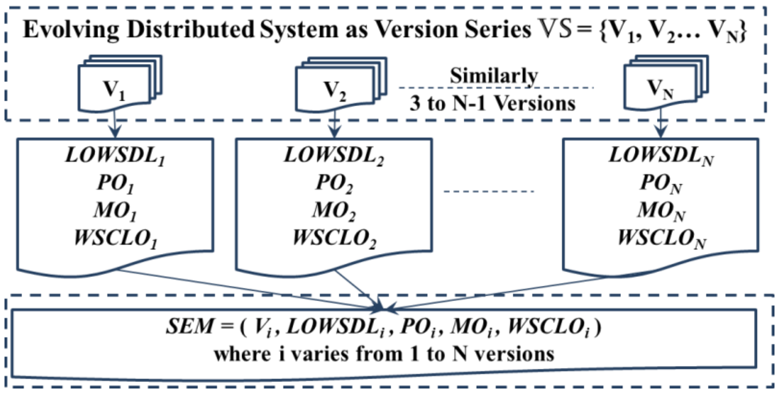

Evolving Distributed System as Version Series VS = {V₁, V₂… V_N}
V₁
V₂
Similarly
3 to N-1 Versions
V_N
LOWSDL₁ PO₁ MO₁ WSCLO₁
LOWSDL₂ PO₂ MO₂ WSCLO₂
LOWSDL_N PO_N MO_N WSCLO_N
SEM = (Vᵢ, LOWSDLᵢ, POᵢ, MOᵢ, WSCLOᵢ)
where i varies from 1 to N versions

Figure 8.3 Summary of the Evolution mining of version series based on Service Evolution Metrics.

8.3 Service Evolution Analytics

We now describe our *Service Evolution Analytics* model and tool. The model in Figure 8.4 is based on the WSDL and the WS code of a version series. As illustrated in Figure 8.5, tool construction includes a development phase (duration t_2) and a subsequent testing phase (duration t_3).

During the development phase, the tool learns from a prefix of the service's version series. A supervised learning approach is used: if the output is incorrect, the developer updates the tool's code and then recomputes the interface slice. Output correctness can be judged using two possible criteria: *software acceptance* and *human acceptance*. For software acceptance, an IDE (e.g., NetBeans and Eclipse) or testing framework (e.g., SoapUI and JMeter) must accept a subset of the WSDL as an interface slice. For human acceptance, an engineer determines if the tool's output is satisfactory. While generating a slice can be a complex task, a developer (or a tester) should be able to easily identify correct output. In this way, the tool learns to produce the correct interface slice from its developer. The process terminates when the IDE or testing framework accepts the tool's output as the desired interface slice.

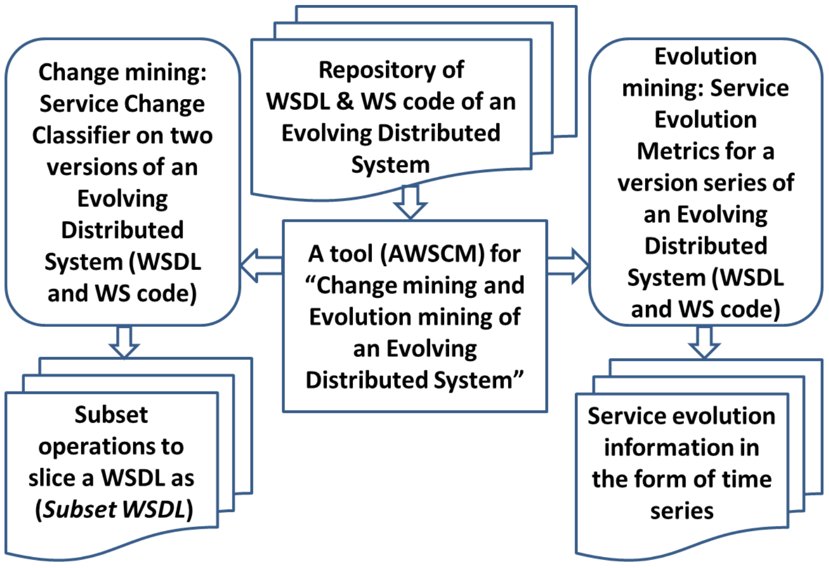

Change mining: Service Change Classifier on two versions of an Evolving Distributed System (WSDL and WS code)
Repository of WSDL & WS code of an Evolving Distributed System
Evolution mining: Service Evolution Metrics for a version series of an Evolving Distributed System (WSDL and WS code)
A tool (AWSCM) for "Change mining and Evolution mining of an Evolving Distributed System"
Subset operations to slice a WSDL as (*Subset WSDL*)
Service evolution information in the form of time series

Figure 8.4 The Service Evolution Analytics model.

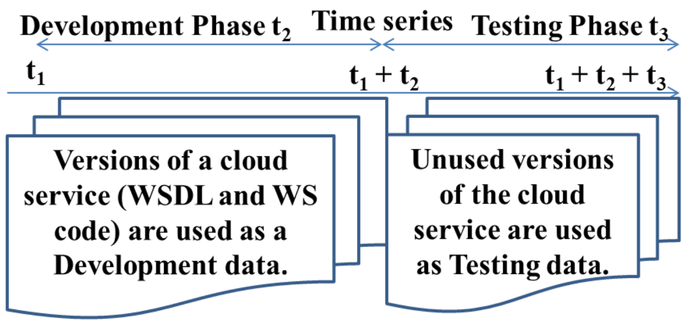

Development Phase t_2
Time series
Testing Phase t_3
t_1
$t_1 + t_2$
$t_1 + t_2 + t_3$
Versions of a cloud service (WSDL and WS code) are used as a Development data.
Unused versions of the cloud service are used as Testing data.

Figure 8.5 Development and testing phases of the Service Evolution Analytics tool. Assuming starting at time t_1 the *development phase* runs for time t_2. Let the subsequent testing phase starts at time $t_1 + t_2$ and runs for time t_3. Therefore, both the phases end at time $t_1 + t_2 + t_3$.

Next, in the testing phase the resulting tool is tested using the remainder of the service's version sequence. The performance of the tool is measured by comparing its output with the expected output, which includes the *WSDL (interface) slices*.

AWSCM: Using this model, we incrementally developed a new slicing component of our existing tool AWSCM ("Automated Web Service Change Management") [59]. AWSCM is an automated intelligent tool that seeks to understand the structure of the WSDL and the WS code. It slices the WSDL by performing *change mining of evolving distributed systems*. AWSCM constructs a *WSDL slice* based on semantic differences between two versions of the WSDL extracted using the Membrane SOA Model [277]. To find the semantic differences between two versions of the WS code, we use the jDiff [276]. To incorporate these semantic differences, we developed several functions, for example, a function for operation separation and a function that slices out portions of the WSDL. As described above, AWSCM's slicing code was repeatedly updated until the tool was able to independently generate the correct WSDL slice for each system studied in the next section. AWSCM was also updated to compute the four proposed service evolution metrics defined in Section II.B. This new feature was implemented in a new module named "Service Evolution Metrics".

8.4 Experiments on Distributed systems

This section describes two empirical experiments, which considers two self-made illustrative web services and two real-word web services (see Table 8.3). The first subsection considers the use of change classification in the construction of a WSDL slice. The second subsection presents experiments involving the Service Evolution Metrics.

a. WSDL Slice Construction

Using Algorithms 8.1, 8.2, and 8.3, the first experiment computes a WSDL slice. It applies change mining (Algorithm 8.1) to compute the change classification. It then uses this change classification to construct the resulting WSDL slice. The WSDL subsets produced for each system considered are shown in Table 8.3. Recall that, as described in Subsection II.A, we consider two Subset WSDLs: the *Difference WSDL* (DWSDL) and the *Unit WSDL* (UWSDL). In the table a 'Y' indicates that the Subset WSDL was constructed, while the single 'N' indicates the one case in which the source was not available to us. Figure 8.6 shows the changes that we made to the two web services: SaaS and BookService. The figure demonstrates the effects of changes in the form of dependency graphs of the two web services. In greater detail the following changes are involved:

- SaaS version 1 has three operations: Index, Searching and readingFile. We made following two changes to version 1 to create version 2. First, we inserted two new operations: edit and editFile, which are captured by the DWSDL. Second, we modified the code of the Searching operation, which is captured by the UWSDL.

Table 8.3 Changes between two Subsequent WSDLs

Example for Eucalyptus-Cluster Controller (CC)
Feb-13.Eucalytpus-CC.wsdl ➔ Aug-13.Eucalytpus-CC.wsdl **Operation Change:** ModifyNode inserted; MigrateInstances inserted. **Schema Change**: CT ccInstanceType has modified; CT virtualBootRecordType has modified; CT metricCounterType has modified; CT metricDimensionsType has modified.

May-15.Eucalytpus-CC.wsdl ➔ Dec-16.Eucalytpus-CC.wsdl
Operation Change: DescribePublicAddresses deleted; AttachNetworkInterface inserted; DetachNetworkInterface inserted;
Schema Change: CT DetachVolumeResponseType inserted; CT ccInstanceType has modified; CT netConfigType has modified;

Example for AWS-Elastic Compute Cloud (EC2)
Feb-13.ec2.wsdl ➔ Aug-13.ec2.wsdl **Operation Change:** DescribeReservedInstancesModifications inserted; ModifyReservedInstances inserted.
August-13.ec2.wsdl ➔ Oct-13.ec2.wsdl **Operation Change:** AcceptVpcPeeringConnection inserted; CreateVpcPeeringConnection inserted; DeleteVpcPeeringConnection inserted; DescribeVpcPeeringConnections inserted; RejectVpcPeeringConnection inserted.

* where CT stands for ComplexType input-output data structure

* for inserted operations, it is obvious that their input-output data-structure were also inserted, thus we skipped its details.

* for deleted operations, it is obvious that their input-output data-structure were also deleted, thus we skipped its details.

- BookService version 1 is upgraded to version 2 with two modifications at the code level that affect the following operations. Code Change 1 affects the two operations: bgAllVerse and bgOp, which further affect their parent nodes in the graph. Code Change 2 affects the two operations: bibleAllVerse and bibleOp, which further affected their parent nodes in the graph. Overall, the two code changes affect three out of four BookService operations.

Table 8.4 provides two examples of the three classification labels (insertion, deletion, and modification) for the two large-scale evolving distributed systems based cloud services: Eucalyptus Cluster Controller (CC) and the Amazon Web Service Elastic Compute Cloud (AWS-EC2). Looking at the output it is clear that the accuracy of the change detection is dependent on the semantic differencing tool. In other words, AWSCM depends directly on the capability and accuracy of semantic differencing tool that it uses.

Table 8.4 Subset WSDL for Change analysis

Web services	DWSDL	UWSDL	Available at
SaaS	Y	Y	Self-made
BookService	Y	Y	Self-made
Eucalyptus	Y	Y	GithHub
AWS	Y	N	GithHub

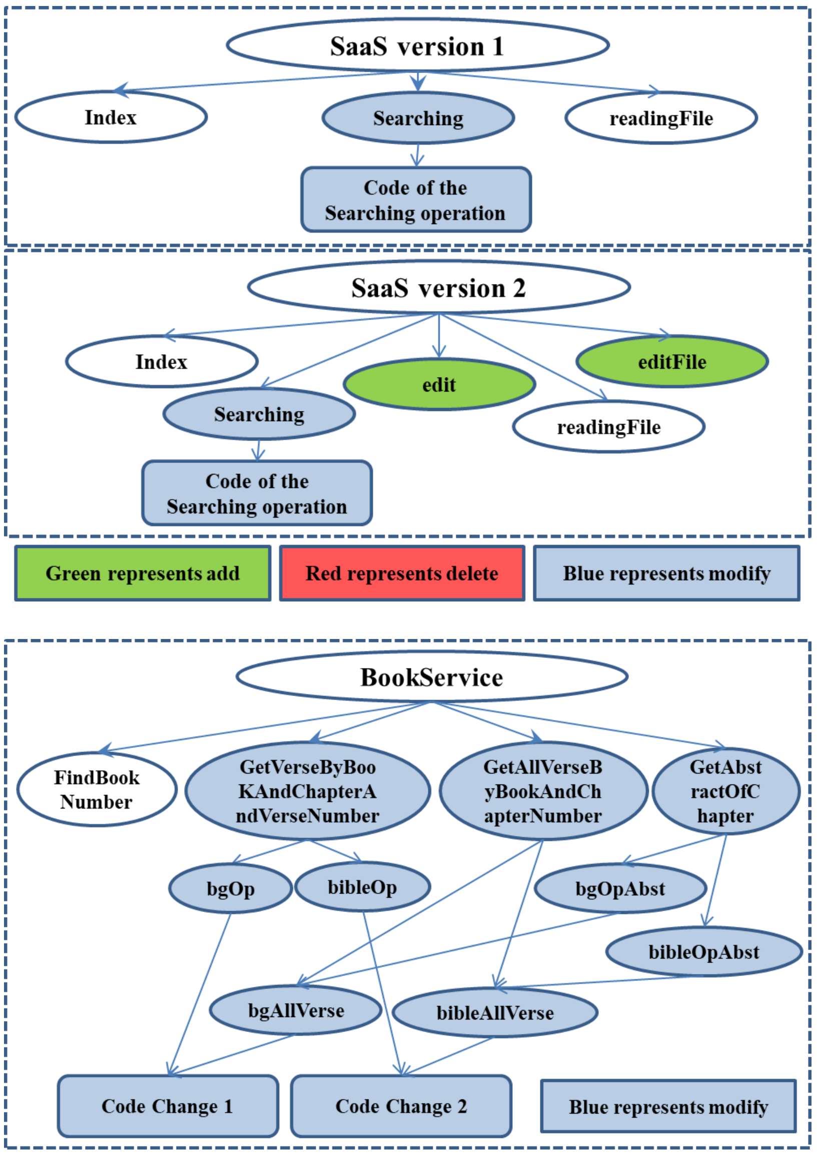


Figure 8.6 Simple representation of changes in two self-made web services, SaaS and BookService.

b. Service Evolution Metric Study

We present a study of the two cloud services that are belonging to large-scale evolving distributed systems: Eucalyptus Cluster Controller (Eucalyptus-CC) and Amazon Web Service – Elastic Compute Cloud (AWS-EC2). To study our four service evolution metrics, we use the repositories available from Github [278][279][279]. The number of lines, parameters, lines of WS code, and WSDL operations are given in Table 8.5. These counts are used to calculate the service evolution metrics; the resulting calculation values are shown in Figure 8.7.

We deduce the following from Figure 8.7. First, the number of lines per operation in WSDL of Eucalyptus-CC is slightly larger than that of AWS-EC2. In contrast, AWS-EC2 has more parameters per operation as compared to Eucalyptus-CC. Both projects have two messages per operations for all versions. Eucalyptus-CC has almost 200 lines of code per operation, which is found in the file *handlers.c* inside module ‘cluster’.

Table 8.5 Information of Eucalyptus-CC & AWS-EC2 to calculate Service Evolution Metrics.

Time Sequence	Eucalyptus Cluster Controller (CC)						Amazon Web Service-Elastic Compute Cloud (EC2)		
Time for last commit	Version	# Commits	# Lines of WSDL	# Parameter	# Code lines	# Operations	# Lines of WSDL	# Parameter	# Operations
June-06	The project was not published on GitHub during this period.						865	96	14
January-07							1157	130	19
Feb-08							1560	181	26
July-09							3568	523	65
November-09							4565	714	81
March-10	1.0	2612	912	50	3272	15	Data is not available for these months.		
Feb-09	1.5	512	848	45	2086	14			
June-10	2.0	4054	1087	59	3663	18	4077	762	87
Nov-10	Data is not available for these months.						4536	890	95
Nov-11							5637	1116	119
Feb-12	3.0	13647	1464	83	4822	24	Data is not available for this month.		
June-12	3.1.0	14252	1684	101	4870	28	6513	1359	137
August-12	3.1.1	14310	1465	84	4870	24	7021	1455	144
Feb-13	3.2.1	16770	1580	98	5606	26	7252	1501	149
August-13	3.3.1	19974	1684	101	6438	28	7392	1535	151
Oct-13	3.4	20621	1783	102	6672	30	Data is not available for this month.		
Feb-14	Data is not available for this month.						7612	1579	156
May-14	4.0	22130	1836	102	6869	31	Data is not available for these months.		
May-15	4.1.1	23208	1840	104	7066	31			
Dec-16	4.3.1		1905	105	6346	32			

To keep experimental result simple, external code dependencies are not considered. All the graphs suggest that both projects are growing at a steady pace. In general, the metric values can help a manager to take decisions regarding a cloud service's evolution task such as regression testing and service monitoring. In a time series graph of a service evolution metric, a sudden change (glitch) may denote an anomaly such as an incorrect upgrade. In this case, the project manager can go for validating the reason of the anomaly. Therefore, in Figure 8.7, all the glitches in time series graphs are the point of deviation from normal trend.

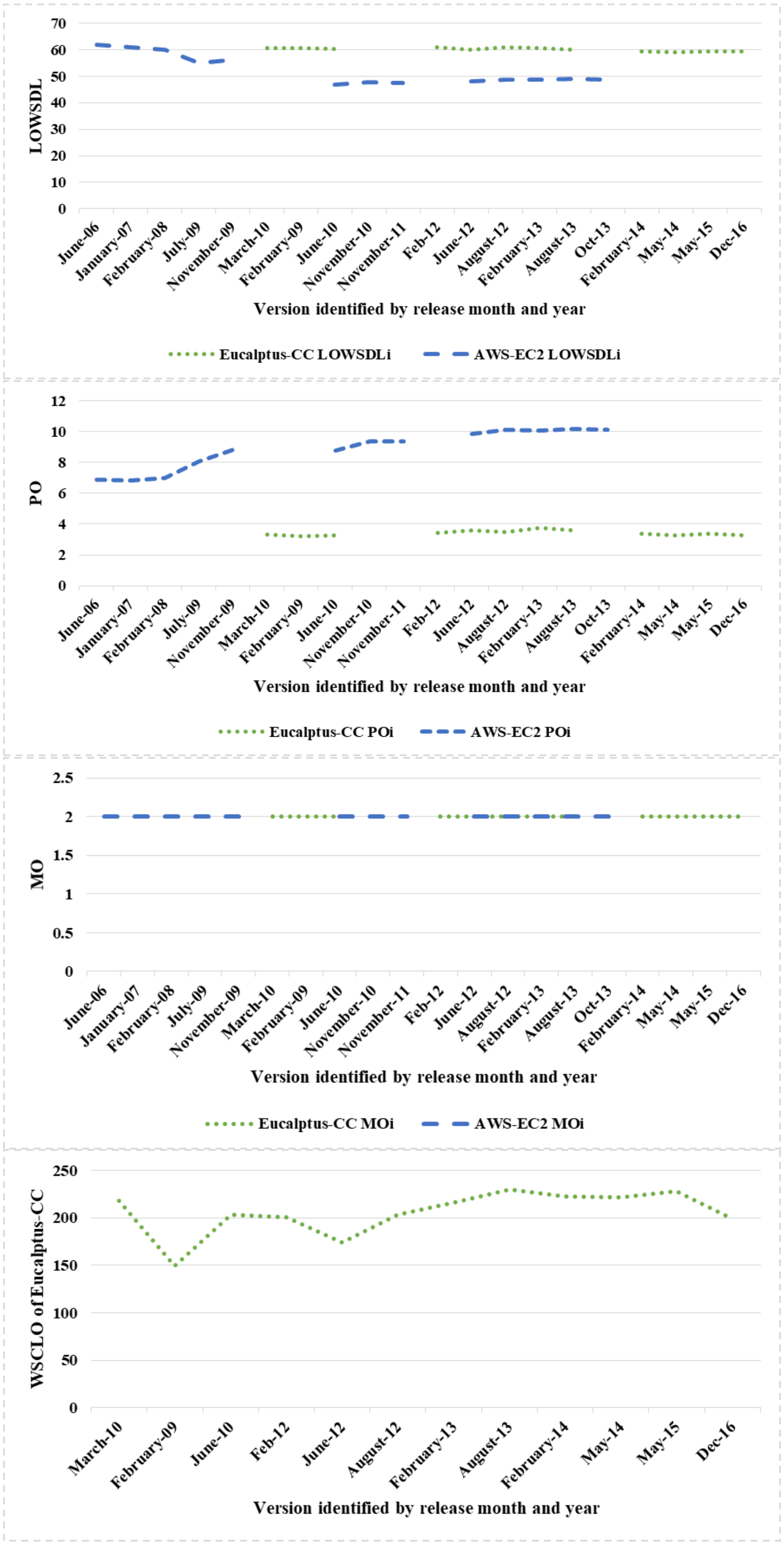


Figure 8.7. Time-series graphs for four Service Evolution Metrics of Eucalyptus-CC & AWS-EC2.

For example, consider the following three changes shown in Table 8.5. First, in August of 2006, public beta of Amazon EC2 was released[1]. This is reflected in the first row (June-06). In October of 2007, two new instance types were added[2] to EC2, Large and Extra-Large. Thereafter, in May of 2008, EC2 added[3] two more instance types, High-CPU Medium and High-CPU Extra Large. These four additions are reflected in Table 8.5 in the second row (January-07), third row (Feb-08), fourth row (July-09), and fifth row (November-09). These changes are evident in the PO_i time series of AWS-EC2 shown in Figure 8.7. To begin with the initial point at (June-06). Second is the almost constant growth between (January-07) and (Feb-08). Third is the sudden increase (glitch) from the (Feb-08) to the (July-09) and (November-09).

In summary, the change classification enables AWSCM to construct a WSDL slice that captures the operations changed from an old to a new version. The evolution mining provides information about an entire version series, which helps to compare the two services empirically. Collectively, the two mining techniques retrieve different information, which helps in service evolution analysis.

8.5 Related Works and Discussions

This section discusses related contributions. To begin with, we used fine-grained change mining information based on the three change classifications (insertions, deletions, and modifications). This change information aids in extracting various WSDL subset slices, which are combined to construct a *WSDL slice*. Chaturvedi *et al.* [245][246][247], these WSDL slices (or Subset WSDLs) were used to select a subset of the test cases for regression-testing of a subset service. Furthermore, the algorithm and definition for WSDL slicing is novel work as compared to the Subset WSDL construction algorithm and interface slicing definition by Chaturvedi *et al.* [58][59][60]. Building on this work, this chapter describes a mining algorithm and the service evolution metrics.

A WSDL slice is also helpful in subset service analysis, which reduces the effort required in regression testing, reverse engineering, and refactoring. The effort is reduced because WSDL slice construction takes only a few millisecond. The provider can also control the usage of a service with the help of an interface subset (Subset WSDL). Finally, the Subset WSDL is also helpful in accessing a subset of a service, which helps clients focus on subsets of the service's functionalities. The WSDL slices are useful to study impact of the effect of change in one operation to the other operations in a distributed system. Using a reduced regression test suite can save effort, and thus cost [246][247][60]. Regression testing is a key part of maintaining quality of service and service-level agreements.

1. Jeff Barr, (August 25, 2006). "Amazon EC2 Beta". https://aws.amazon.com/blogs/aws/amazon_ec2_beta/. Link accessed on February 2019.
2. Jeff Barr, (October 16, 2007). "Amazon EC2 Gets More Muscle". Link accessed on February 2019.
3. Jeff Barr, (May 29, 2008). "More EC2 power". https://aws.amazon.com/blogs/aws/more-ec2-power/. Link accessed on February 2019.

Other related works include work related to *software analysis* and *mining software repositories* [248] in the context of *change impact analysis* [249] includes the following. For example, Ying *et al.* [250] presented an approach to determine change patterns and evaluated their work on the code repositories for Eclipse and Mozilla. Likewise, we evaluate our work by analyzing the repositories of *Eucalyptus-CC* and *AWS-EC2*. More recently, Negara *et al.* [251] identified previously unknown frequent code change patterns from an old repository. Thereafter, they analyzed some of the mined unknown code change patterns and some high-level program transformations.

Work related to change analysis, mining, and learning of an evolving service repository includes that of Xiao *et al.* [249] who proposed a formula to quantify the cost of a business process change in a service-oriented architecture. Dam *et al.* [252] analyzed change propagation using extensions of the well-known UML framework in SOA as SoaML. Papazoglou *et al.* [253][254] present a unifying theoretical framework for controlling the evolution of services using distinguished shallow and deep changes. Alam *et al.* [255] presented a literature survey of 60 relevant studies of change propagation in SOA and Business Process Management (BPM) based on four research questions. Romano and Martin [256] proposed a tool WSDLDiff for gathering fine-grained changes between subsequent versions of a service. They then use this information to analyze the evolution of four web services. Recently, Karn *et al.* [257] proposed a methodology for selection and tuning a machine-learning model, which is applied on dynamic changing environment to achieve higher accuracy in prediction of security attacks.

Intuitively, our approach is useful in both static and dynamic program analysis for testing and verification [258]. For example, in support of mobile app testing [259], Facebook researchers have developed CT-Scan [260], which can also aid in regression testing [261]. The application of change impact analysis based interface slicing, such as WSDL slicing, can also be used in regression testing of web services [60].

The second kind of related work involves evolution mining of an evolving distributed system. Here, we proposed four service evolution metrics that are helpful when evaluating the risk associated with evolving a cloud service. Additionally, these metrics can be helpful in the - maintenance of existing versions and development of future versions. The metrics can also be helpful to a consumer while analyzing functionalities of the WSDL, XSD, and WS code. It is also helpful in the regression testing of the upgraded cloud service system.

The work related to software and service evolution metrics includes the following. Constantinou *et al.* [262] presented six metrics to measure the architectural stability and evolution of software projects for reuse. Wang *et al.* [263] proposed a service evolution model based on four service evolution patterns used to analyze service dependencies, identify changes on services, and estimate impact on consumers. Recently, Kohar *et al.* [264] proposed a service evolution metric, a service client-code evolution metric, and service usefulness evolution metric. Similarly, we consider service evolution analysis using the four novel service evolution metrics. Their analysis produced useful interpretations (e.g., see Figure 8.7 and Table 8.5).

Fokaefs *et al.* [265] presented WSDL evolution analysis through the empirical study of their VTracker algorithm for XML differencing. The algorithm can recognize and analyze changes between subsequent versions of various WSDL web-services. Thereafter, Fokaefs *et al.* [266] introduced WSDarwin, a specialized differencing method for WSDL and WADL (Web Application Description Language), which produces a set of rules. Li *et al.* [267] reported an empirical study on web API evolution and then drew conclusions for application developers. In contrast to these works, our tool AWSCM can use change information to generate interface (WSDL) slices, which can execute subsets of a cloud/web service. Additionally, AWSCM can also retrieve cloud/web service evolution information that can lead to the extraction of facts about the evolution of an evolving distributed system.

Mohamed *et al.* [268] presented SaaS evolution platform based on Software Product Lines (SPLs) for a multi-tenant single instance; the platform provides evolution rules based on feature modeling to govern evolution decisions. Jamshidi *et al.* [269] reviewed cloud migration research and include a comparative study between SOA and Cloud migration. Ghosh *et al.* [270] proposed SelCSP framework to facilitate selection of a trustworthy and competent service provider. Similarly, our service evolution metrics generate empirical reports that may be helpful in the Cloud/SOA migration and in the selection of a trustworthy service provider. Recently, Noor *et al.* [271] described a trust management framework named as CloudArmor, which is based on reputation and credibility model.

Both of our approaches are useful for establishing and maintaining Service Level Agreements (SLAs) and Quality of Service (QoS). Service providers often update or enhance a service to meet new requirements, thus this may cause a potential SLA violation where an incorrect change leads to an incorrect behaviour. Thus, QoS monitoring is required to check and re-establish the specific QoS described in the SLA [272][273]. Reduced cloud/web service maintenance effort can further reduce the effort required to guarantee the QoS and the SLA [274][275].

8.6 Conclusions

This chapter describes approaches for *change mining and evolution mining of an evolving distributed system*. The first approach uses a *service change classifier* algorithm to assign change labels to a service's operations and then extracts a *WSDL slice*. The second approach uses four *service evolution metrics* to study cloud service evolution and to deduce facts from a version series. The experiments demonstrate how the tool can be used to study the evolution of cloud services. Our *Service Evolution Analytics* model is also helpful in subset regression testing, which in turn helps to maintain the QoS required by a SLA. In the future, we plan to apply change and evolution mining to other APIs.

Chapter 9

Conclusions and Future Works

System entities (or components) in an evolving system keeps on evolving, which makes a set of states that we referred as state series. There exist connections (or relationships) between entities, which also evolve over system state, and make a series of evolution representor e.g. a set of evolving networks. We can use these evolution representor (like evolving networks) to do mining and learning over evolving system states for system evolution analysis. Our main objective is to introduce System Evolution Analytics (SysEvo-Analytics), which is helpful to do system evolution analysis. For this, we proposed end-to-end approaches for data mining and machine learning of evolving system; here end-to-end means a set of steps that contains data pre-processing, information retrieval, result post-processing, and graphical representation. To accomplish these tasks, we proposed novel algorithms and developed few analytic tools based on fundamental mining and learning techniques. The tools are majorly coded in Java technologies. Our tools are used to do exhaustive experimental analysis over various evolving systems. These experiments resulted in different kind of SysEvo-Analytics.

We introduced SysEvo-Analytics model, which uses *network evolution mining* over a state series of an evolving system. This mining is a hybrid mining of *network rule mining* and *network subgraph mining* with *evolution mining* over multiple states of a system. This mining is realized as two mining techniques: *Network Evolution Rule Mining* and *Network Evolution Subgraph Mining*. Both these techniques retrieve evolution information. The first generates *Network Evolution Rules* (NERs) and the latter generates *Network Evolution Subgraphs* (NESs). We have implemented these two approaches as two prototype tools of SysEvo-Analytics model, which mines multiple states of an evolving system. For evolving system mining, we developed tool for evolution rule mining. The tool is based on the Rule-Growth algorithm of SPMF[1] rule mining tool. Our tool extends this existing tool to accomplish the application of network evolution rule mining for evolving systems. We conducted experiments and considered network evolution rules with their support, confidence, and stability. We developed tool for evolving system subgraph mining. The tool is based on the acc-motif[2]. Our tool extends this existing tool to accomplish the application of network evolution subgraph mining on evolving systems. The developed tools are used to conduct experiments. In the experimentation, the tools are used for six different evolving systems. For each evolving system, the tools retrieved the NERs and NESs, which are interesting and non-obvious information. The information is helpful in system development and maintenance.

We have also introduced a model for SysEvo-Analytics, which is based on *system network* *evolution learning* over a state series of an evolving system. This learning is a hybrid of *graph learning* with *evolution learning* over multiple states of a system. This learning is realized with the deep learning techniques: RBM, DBN, and dA. The proposed approach is used to implement a SysEvo-Analytics tool that learns multiple states of an evolving system. For evolving system learning, we developed a tool based on java technology. We developed the tool for deep evolving system learning. The tool is based on the code available for deep learning[3]. Our tool extends the code to accomplish the application of deep learning on evolving systems. We used the tool to do experiments on six different evolving systems. In the experiments, we have generated the *System Neural Network* (SysNN) for each evolving system. The SysNN can be used to do recommendation or prediction. For recommendation purpose, the SysNN reconstructed a network matrix. To measure the quality of recommendation, we compared binary output matrix against a testing matrix of a state that remain unused during training phase. The SysNN can be helpful in system development and maintenance.

We also presented two application of SysEvo-Analytics. First, we also presented an approach to do *Big Data Evolution Analytics* (BDE-Analytics) for a given *Big Data State Series*, which is constructed from an *Evolving Big Data System* (EBD-System). We applied the BDE-Analytics approach on big scholarly data collected from KDD Cup 2016 of Microsoft Academic Graph as a publication bibliography dataset. Second, we also describes a Service Evolution Analytics model and tool (AWSCM) that supports change and evolution mining of an evolving distributed system. We applied this on real-world evolving distributed system based on Web Services having service-oriented architecture enabled with WSDL. We performed four several case studies to construct WSDL slices of four web services. We also demonstrated the service change classifier and service evolution metrics using two well-known cloud services: Eucalyptus-CC and AWS-EC2.

In future, some other pattern mining techniques (like rule and subgraph mining) can be used with evolution and change analysis to make another hybrid pattern mining. Similarly, some other machine learning techniques (like deep learning) can be used with evolution and change learning to make another hybrid evolution learning. It is also possible to perform similar studies of the changeability and stability for an evolving system. It is also possible to present enhanced experimentations on similar evolving systems, which demonstrate few more applications of our tool for other domains. Other mining techniques with network evolution mining techniques can produce another hybrid mining approach. Additionally, our approach can be applicable to the large networks or big graphs. Our approach is applicable to various domains such as system biological networks (of DNA, genome, protein-protein interactions), food web, computer networks, social networks, and so on. Our SysEvo-Analytics model can be done using other evolution mining or learning techniques. We plan to use other techniques for studying system evolution. Further, we plan to execute our tool on other types of evolving systems. We plan to strengthen our experimentation with more number of datasets from other domains like software system and natural language.

REFERENCES

[1] T. P. Hughes. “The evolution of large technological systems.” The social construction of technological systems: New directions in the sociology and history of technology 82 (1987).

[2] J. H. Holland. “Complex adaptive systems.” Daedalus (1992): 17-30.

[3] P. Angelov, and N. Kasabov. “Evolving intelligent systems, eIS.” IEEE SMC eNewsLetter 15 (2006): 1-13.

[4] R. Valerdi, A. M. Ross, and D. H. Rhodes. “A framework for evolving system of systems engineering.” CrossTalk. 2007.

[5] A. M. Ross, D. H. Rhodes, and D. E. Hastings. “Defining changeability: Reconciling flexibility, adaptability, scalability, modifiability, and robustness for maintaining system lifecycle value.” Systems Engineering 11.3 (2008): 246-262.

[6] P. Angelov, D. P. Filev, and N. Kasabov, eds. Evolving intelligent systems: methodology and applications. Vol. 12. John Wiley & Sons, 2010.

[7] P. Angelov, and N. Kasabov. “Evolving computational intelligence systems.” Proceedings of the 1st International Workshop on Genetic Fuzzy Systems. 2005, 76-82.

[8] S. A. Frost, and M. J. Balas. “Evolving Systems and Adaptive Key Component Control”. In: Arif, T.T. (ed.) Aerospace Technologies Advancements. ISBN: 978-953-7619-96-1 (2010).

[9] T. Mens, A. Serebrenik, and A. Cleve. “Evolving Software Systems.” Heidelberg: Springer Eds. Vol. 190, 2014.

[10] M. Böttcher, D. Nauck, C. Borgelt, and R. Kruse. “A framework for discovering interesting business changes from data”. BT Technology Journal 24.2 (2006): 219-228.

[11] T. Palpanas. “Data series management: The next challenge.” IEEE 32nd International Conference on Data Engineering Workshops (ICDEW). IEEE, 2016.

[12] J. Han, J. Pei, and M. Kamber. Data mining: concepts and techniques. Elsevier, 2011.

[13] U. Fayyad, G. Piatetsky-Shapiro, and P. Smyth. “From data mining to knowledge discovery in databases.” AI magazine 17.3 (1996): 37.

[14] M. Böttcher, F. Höppner, and M. Spiliopoulou. “On exploiting the power of time in data mining.” ACM SIGKDD Explorations Newsletter 10.2 (2008): 3-11.

[15] M. Boettcher. “Contrast and change mining.” Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery 1.3 (2011): 215-230.

[16] B. Wu, et al. “Resume mining of communities in social network.” 7th IEEE International Conference on Data Mining Workshops. 2007.

[17] M. Takaffoli, et al. “Community evolution mining in dynamic social networks.” Procedia-Social and Behavioral Sciences 22 (2011): 49-58.

[18] T. Abraham, and J. F. Roddick. “Incremental meta-mining from large temporal data sets.” International Conference on Conceptual Modeling. Springer Berlin Heidelberg, 1998.

[19] G. Bin, V. S. Sheng, Z. Wang, D. Ho, S. Osman, and S. Li. “Incremental learning for ν-support vector regression.” Neural Networks 67 (2015): 140-150.

[20] G. Ditzler, et al. “Learning in nonstationary environments: A survey.” IEEE Computational Intelligence Magazine 10.4 (2015): 12-25.

[21] D. Binkley. “Source code analysis: A road map.” 2007 Future of Software Engineering. IEEE Computer Society, 2007.

[22] C. W. Günther, S. Rinderle, M. Reichert, and W. V. D. Aalst. “Change mining in adaptive process management systems.” On the Move to Meaningful Internet Systems 2006: CoopIS, DOA, GADA, and ODBASE. Springer Berlin Heidelberg, 2006 309-326.

[23] G. Dong, J. Han, L. VS Lakshmanan, J. Pei, H. Wang, and P. S. Yu. “Online mining of changes from data streams: Research problems and preliminary results.” Proceedings of the 2003 ACM SIGMOD Workshop on Management and Processing of Data Streams. 2003.

[24] M.C. Chen, A.L. Chiu, and H.H. Chang. “Mining changes in customer behavior in retail marketing.” Expert Systems with Applications 28.4 (2005): 773-781

[25] B. Liu, W. Hsu, H. S. Han, and Y. Xia. “Mining changes for real-life applications.” DaWaK. Vol. 1874. 2000.

[26] M. J. Shih, D. R. Liu, and M. L. Hsu. “Discovering competitive intelligence by mining changes in patent trends.” Expert Systems with Applications 37.4 (2010): 2882-2890.

[27] H. S. Song, J. kyeong Kim, and S. H. Kim. “Mining the change of customer behavior in an internet shopping mall.” Expert Systems with Applications 21.3 (2001): 157-168.

[28] A. Chaturvedi and A. Tiwari. “System Evolution Analytics: Deep Evolution and Change Learning of Inter-Connected Entities”. IEEE International Conference on Systems, Man, and Cybernetics (SMC), 2018, 3075-3080.

[29] A. Chaturvedi and A. Tiwari. “System Evolution Analytics: Evolution and Change Pattern Mining of Inter-Connected Entities”. IEEE International Conference on Systems, Man, and Cybernetics (SMC), 2018, 3877-3882.

[30] C. Aggarwal, and K. Subbian. “Evolutionary network analysis: A survey.” ACM Computing Surveys (CSUR) 47.1 (2014): 10.

[31] P. Holme, and J. Saramäki. “Temporal networks.” Physics Reports 519.3 (2012): 97-125.

[32] L. A. Zadeh. “Time-varying networks, I.” Proceedings of the IRE49.10 (1961): 1488-1503.

[33] J. E. Aronson. “A survey of dynamic network flows.” Annals of Operations Research 20.1 (1989): 1-66.

[34] M. Kivelä, et al. “Multilayer networks.” Journal of complex networks 2.3 (2014): 203-271.

[35] M. Kivelä, and M. A. Porter. “Isomorphisms in multilayer networks.” IEEE Transactions on Network Science and Engineering (2017).

[36] R. Agrawal, T. Imielinski, and A. Swami, “Mining Association Rules Between Sets of Items in Large Databases,” ACM SIGMOD Conference Management of Data, May 1993.

[37] J. M. Ale, and G. H. Rossi. “An approach to discovering temporal association rules.” Proceedings of the 2000 ACM Symposium on Applied Computing-Volume 1. ACM, 2000.

[38] B. Liu, Y. Ma, and R. Lee. “Analyzing the interestingness of association rules from the temporal dimension.” IEEE International Conference on Data Mining. ICDM. IEEE, 2001.

[39] C. Aaron and N. Eagle. “Persistence and periodicity in a dynamic proximity network”. DIMACS Workshop on Computational Methods for Dynamic Interaction Networks, 2007.

[40] R. Agrawal, and R. Srikant. “Mining sequential patterns.” 11th International Conference on Data Engineering, IEEE, 1995.

[41] P. Fournier-Viger, C. Wu, V. Tseng, L. Cao, & R. Nkambou. “Mining Partially-Ordered Sequential Rules Common to Multiple Sequences” IEEE Transactions on Knowledge and Data Engineering, vol. 27, no. 8, August 2015.

[42] P. Fournier-Viger, R. Nkambou, and V. Shin-Mu Tseng. “RuleGrowth: mining sequential rules common to several sequences by pattern-growth.” ACM Symposium on Applied Computing 2011.

[43] P. Fournier-Viger, et al. “SPMF: a Java open-source pattern mining library.” Journal of Machine Learning Research 15.1 (2014): 3389-3393.

[44] M. Berlingerio, et al. “Mining graph evolution rules.” Joint European Conference on Machine Learning and Knowledge Discovery in Databases. Springer Berlin Heidelberg, 2009.

[45] C. W. Leung, et al. “Mining interesting link formation rules in social networks.” 19th ACM International Conference on Information and Knowledge Management. ACM, 2010.

[46] W. Fan, et al. “Association rules with graph patterns.” VLDB Endowment 8.12 (2015): 1502-1513.

[47] E. Scharwächter, et al. “Detecting Change Processes in Dynamic Networks by Frequent Graph Evolution Rule Mining.” IEEE 16th International Conference on Data Mining (ICDM). IEEE, 2016.

[48] M. H. Namaki, et al. “Discovering Graph Temporal Association Rules.” Proceedings of the 2017 ACM on Conference on Information and Knowledge Management. ACM, 2017.

[49] A. Inokuchi et al. “An apriori-based algorithm for mining frequent substructures from graph data.” European Conference on Principles of Data Mining and Knowledge Discovery. Springer Berlin Heidelberg, 2000.

[50] M. Kuramochi, and G. Karypis. “Frequent subgraph discovery.” IEEE International Conference on Data Mining (ICDM). IEEE, 2001.

[51] X. Yan and J. Han. “gspan: Graph-based substructure pattern mining.” IEEE International Conference on Data Mining (ICDM). IEEE, 2002.

[52] R. Milo, S. Shen-Orr, S. Itzkovitz, N. Kashtan, D. Chklovskii, U. Alon. “Network Motifs: Simple Building Blocks of Complex Networks.” Science. 298, 824—827 (2002).

[53] U. Alon, “Network motifs: theory and experimental approaches” Nature Rev. Genetics (2007) 450–461.

[54] N. Pržulj, D. G. Corneil, and I. Jurisica. “Modeling interactome: scale-free or geometric?.” Bioinformatics 20.18 (2004): 3508-3515.

[55] T. Milenkoviæ, and N. Pržulj. “Uncovering biological network function via graphlet degree signatures.” Cancer informatics 6 (2008): 257.

[56] Y. LeCun, Y. Bengio, and G. Hinton. “Deep learning.” Nature 521.7553 (2015): 436-444.

[57] F. Xia, et al. “Big Scholarly Data: A Survey.” IEEE Transactions on Big Data 3.1 (2017): 18-35.

[58] A. Chaturvedi. “Subset WSDL to Access Subset Service for Analysis.” Cloud Computing Technology and Science (CloudCom), 2014 IEEE 6th International Conference on. IEEE, 2014.

[59] A. Chaturvedi. “Automated Web Service Change Management AWSCM-A Tool.” Cloud Computing Technology and Science (CloudCom), 2014 IEEE 6th International Conference on. IEEE, 2014.

[60] A. Chaturvedi and D. Binkley, “Web Service Slicing: Intra and Inter-Operational Analysis to Test Changes,” IEEE Transactions Services Computing, 2018.

[61] Y.-Y. Liu, J.-J. Slotine, and A.-L. Barabási. “Controllability of complex networks.” Nature 473.7346 (2011): 167.

[62] Y.-Y. Liu, J.-J. Slotine, and A.-L. Barabási. “Observability of complex systems.” Proceedings of the National Academy of Sciences 110.7 (2013): 2460-2465.

[63] N. Przulj, “Biological network comparison using graphlet degree distribution,” Bioinformatics, vol. 23, no. 2, pp. e177–e183, 2007.

[64] O. N. Yaveroglu et al., “Revealing the hidden language of complex networks,” Sci. Rep., vol. 4, 2014, Art. no. 4547.

[65] W. Ali, T. Rito, G. Reinert, F. Sun, and C. M. Deane, “Alignment-free protein interaction network comparison,” Bioinformatics, vol. 30, no. 1 pp. i430–i437, 2014.

[66] A. E. Wegner, L. Ospina-Forero, R. E. Gaunt, C M. Deane, and G. Reine, “Identifying networks with common organizational principles,” J. Complex Netw., 2017, Art. no. cny003.

[67] J. Abonyi, B. Feil, and A. Abraham. “Computational intelligence in data mining.” Informatica 29.1 (2005), 3-12.

[68] H. Liu & J. Zeleznikow “Intelligent computation for association rule mining”. In Workshop on Computational Intelligence in Data Mining in conjunction with the 5th IEEE International Conference on Data Mining 2005, 49-53.

[69] C.-K. Ting, W.-M. Zeng, and T.-C. Lin. “Linkage discovery through data mining [research frontier].” IEEE Computational Intelligence Magazine 5.1 (2010): 10-13.

[70] J. M. Luna, et al. “Mining Context-Aware Association Rules Using Grammar-Based Genetic Programming.” IEEE Transactions on Cybernetics (2017).

[71] J. Xuan, et al. “Topic model for graph mining.” IEEE Transactions on Cybernetics 45.12 (2015): 2792-2803.

[72] V. Jakkula, and D. J. Cook. “Mining sensor data in smart environment for temporal activity prediction.” Poster session at the ACM SIGKDD, San Jose, CA (2007).

[73] K.-H. Liu, et al. “Association and temporal rule mining for post-filtering of semantic concept detection in video.” IEEE Transactions on Multimedia 10.2 (2008): 240-251.

[74] X. Qin, R. Ahsan, X. Lin, E. A. Rundensteiner, and M. O. Ward. “Interactive temporal association analytics.” In EDBT, pp. 197-208. 2016.

[75] K. Verma, and O. P. Vyas. “Efficient calendar based temporal association rule.” ACM SIGMOD Record 34.3 (2005): 63-70.

[76] K. Nørvåg, T. Ø. Eriksen, and K.-I. Skogstad. “Mining association rules in temporal document collections.” International Symposium on Methodologies for Intelligent Systems. Springer, Berlin, Heidelberg, 2006.

[77] N. M. Khairudin, A. Mustapha, and M. H. Ahmad. “Effect of temporal relationships in associative rule mining for web log data.” The Scientific World Journal 2014 (2014).

[78] M. A. Madaio, et al. “Using Temporal Association Rule Mining to Predict Dyadic Rapport in Peer Tutoring.” EDM. 2017.

[79] M. Shokoohi-Yekta, et al. “Discovery of meaningful rules in time series.” 21st ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. ACM, 2015.

[80] T. Rolfsnes, et al. “Improving change recommendation using aggregated association rules.” 2016 IEEE/ACM 13th Working Conference on Mining Software Repositories (MSR). IEEE, 2016.

[81] T. Rolfsnes, L. Moonen, and D. Binkley. “Predicting relevance of change recommendations.” Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering. IEEE Press, 2017.

[82] A. Gyorgy, and M. Arcak. “Pattern Formation over Multigraphs.” IEEE Transactions on Network Science and Engineering (2017).

[83] R. Ahmed, and G. Karypis. “Algorithms for mining the coevolving relational motifs in dynamic networks.” ACM Transactions on Knowledge Discovery from Data (TKDD) 10.1 (2015): 4.

[84] Y. Sun, et al. “Co-evolution of multi-typed objects in dynamic star networks.” IEEE Transactions on Knowledge and Data Engineering 26.12 (2014): 2942-2955.

[85] N. Shah, et al. “TimeCrunch: Interpretable dynamic graph summarization.” Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. ACM, 2015.

[86] L. Kovanen, et al. “Temporal motifs in time-dependent networks.” Journal of Statistical Mechanics:

Theory and Experiment 2011.11 (2011): P11005.

[87] A. Paranjape, A. R. Benson, and J. Leskovec. “Motifs in temporal networks.” Proceedings of the Tenth ACM International Conference on Web Search and Data Mining. ACM, 2017.

[88] P. Holme. “Analyzing temporal networks in social media.” Proceedings of the IEEE 102.12 (2014): 1922-1933.

[89] K. M. Borgwardt, K. Hans-Peter, and P. Wackersreuther. “Pattern mining in frequent dynamic subgraphs.” Data Mining, 2006. ICDM'06. Sixth International Conference on. IEEE, 2006.

[90] B. Wackersreuther, et al. “Frequent subgraph discovery in dynamic networks.” Proceedings of the Eighth Workshop on Mining and Learning with Graphs. ACM, 2010.

[91] P. Patel, E. Keogh, J. Lin, and S. Lonardi, “Mining motifs in massive time series databases.” IEEE International Conference on Data Mining (ICDM). IEEE, 2002.

[92] B. Chiu, E. Keogh, and S. Lonardi. “Probabilistic discovery of time series motifs.” 9th ACM SIGKDD International Conference on Knowledge Discovery and Data mining. ACM, 2003.

[93] D. Braha, and Y. Bar-Yam. “Time-dependent complex networks: Dynamic centrality, dynamic motifs, and cycles of social interactions.” Adaptive Networks. Springer Berlin Heidelberg, 2009. 39-50.

[94] Q. Zhao, Y. Tian, Q. He, N. Oliver, R. Jin, and W.-C. Lee, “Communication motifs: A tool to characterize social communications,” in Proc. 19th ACM International Conference Inf. Knowl. Manage., 2010, pp. 1645–1648.

[95] D. Jurgens, and T.-C. Lu. “Temporal Motifs Reveal the Dynamics of Editor Interactions in Wikipedia.” ICWSM. 2012.

[96] Y. Meng, and H. Guo. “Evolving network motifs based morphogenetic approach for self-organizing robotic swarms.” Conference on Genetic and Evolutionary Computation. ACM, 2012.

[97] P. Bhattacharya, et al. “Graph-based analysis and prediction for software evolution.” 34th International Conference on Software Engineering. IEEE Press, 2012.

[98] Y. Hulovatyy, H. Chen, and T. Milenkovic, “Exploring the structure and function of temporal networks with dynamic graphlets.” Bioinformatics, vol. 31, no. 12, pp. i171–i180, 2015.

[99] A. JM Martin, et al. “Graphlet Based Metrics for the comparison of gene regulatory networks.” PloS one 11.10 (2016): e0163497.

[100] E. Fricke, and A. P. Schulz. “Design for changeability (DfC): Principles to enable changes in systems throughout their entire lifecycle.” Systems Engineering 8.4 (2005).

[101] H. McManus, and D. Hastings. “3.4. 1 A Framework for Understanding Uncertainty and its Mitigation and Exploitation in Complex Systems.” INCOSE International Symposium. Vol. 15. No. 1. 2005.

[102] M. W. Maier. “Architecting principles for systems-of-systems.” Systems engineering 1.4 (1998):

267-284.

[103] A. M. Ross, and D. H. Rhodes. "11.1. 1 Using Natural Value-Centric Time Scales for Conceptualizing System Timelines through Epoch-Era Analysis." INCOSE International Symposium. 2008, 18.1, 1186-1201.

[104] M. E. Fitzgerald, A. M. Ross, and D. H. Rhodes. A method using epoch-era analysis to identify valuable changeability in system design. MASSACHUSETTS INST OF TECH CAMBRIDGE, 2011.

[105] M. E. Fitzgerald, and A. M. Ross. "Sustaining lifecycle value: Valuable changeability analysis with era simulation." IEEE International Systems Conference (SysCon). IEEE, 2012, 1-7.

[106] M. E. Fitzgerald, A. M. Ross, and D. H. Rhodes. "8.4. 1 Assessing Uncertain Benefits: a Valuation Approach for Strategic Changeability (VASC)." INCOSE International Symposium. 22.1. 2012, 1147-1164.

[107] E. CY Koh, N. HM Caldwell, and P. J. Clarkson. "A technique to assess the changeability of complex engineering systems." Journal of Engineering Design 24.7 (2013): 477-498.

[108] B. Fluri. "Assessing changeability by investigating the propagation of change types." Companion to the proceedings of the 29th International Conference on Software Engineering. IEEE Computer Society, 2007, 97-98.

[109] J. Xuan, X. Luo, G. Zhang, J. Lu, and Z. Xu. "Uncertainty analysis for the keyword system of web events." IEEE Transactions on Systems, Man, and Cybernetics: Systems 46, no. 6 (2016): 829-842.

[110] J. Avalos, M. W. Grenn, and B. Roberts. "Assessment of Complex Adaptive System Changeability Using a Learning Classifier System." IEEE Systems Journal (2018).

[111] V. N. Belykh, I. V. Belykh, and M. Hasler. "Connection graph stability method for synchronized coupled chaotic systems." Physica D: nonlinear phenomena 195.1 (2004): 159-187.

[112] L. Moreau. "Stability of multiagent systems with time-dependent communication links." IEEE Transactions on Automatic Control 50.2 (2005): 169-182.

[113] J-C. Delvenne, S. N. Yaliraki, and M. Barahona. "Stability of graph communities across time scales." Proceedings of the National Academy of Sciences 107.29 (2010): 12755-12760.

[114] M. T. Angulo, Y.-Y. Liu, and J.-J. Slotine. "Network motifs emerge from interconnections that favour stability." Nature Physics 11.10 (2015): 848.

[115] M. Ogura, and V. M. Preciado. "Stability of spreading processes over time-varying large-scale networks." IEEE Transactions on Network Science and Engineering 3.1 (2016): 44-57.

[116] R. Wolff, and A. Schuster. "Association rule mining in peer-to-peer systems." IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics) 34.6 (2004): 2426-2438.

[117] S. M. Pincus. "Approximate entropy as a measure of system complexity." National Academy of Sciences 88.6 (1991): 2297-2301.

[118] R. Rosen. "Complexity and system descriptions." Facets of Systems Science. Springer US, 1991.

477-482.

[119] A. M. Abdullah, et al. “Modification of standard function point complexity weights system.” Journal of Systems and Software 74.2 (2005): 195-206.

[120] J. Gignoux, et al. “Emergence and complex systems: The contribution of dynamic graph theory.” Ecological Complexity 31 (2017): 34-49.

[121] T. J. McCabe. “A complexity measure.” IEEE Transactions on Software Engineering 4 (1976): 308-320.

[122] P. Smolensky. Information processing in dynamical systems: Foundations of harmony theory. No. CU-CS-321-86. Colorado Univ at Boulder Dept of Computer Science, 1986.

[123] G. W. Taylor, G. E. Hinton, and S. T. Roweis. “Modeling human motion using binary latent variables.” Advances in Neural Information Processing Systems. 2006.

[124] G. E. Hinton, S. Osindero, and Y. W. Teh. “A fast learning algorithm for deep belief nets.” Neural computation 18.7 (2006): 1527-1554.

[125] T. Tieleman. “Training restricted Boltzmann machines using approximations to the likelihood gradient.” Proceedings of the 25th International Conference on Machine learning (ICML). ACM, 2008.

[126] P. Vincent, H. Larochelle, Y. Bengio, and P. A. Manzagol. “Extracting and composing robust features with denoising autoencoders.” Proceedings of the 25th International Conference on Machine learning (ICML). ACM, 2008.

[127] G. E. Hinton “Training products of experts by minimizing contrastive divergence.” Neural computation 14.8 (2002): 1771-1800.

[128] Y. Bengio, P. Simard, and P. Frasconi. “Learning long-term dependencies with gradient descent is difficult.” IEEE Transactions on Neural Networks, 5.2 (1994): 157-166.

[129] Y. Bengio, P. Lamblin, D. Popovici, and H. Larochelle. “Greedy layer-wise training of deep networks.” Advances in neural information processing systems 19 (2007): 153.

[130] Y. Bengio. “Learning deep architectures for AI.” Foundations and trends in Machine Learning 2.1 (2009): 1-127.

[131] Y. Yang, et al. “Concept graph learning from educational data.” Proceedings of the 8th ACM International Conference on Web Search and Data Mining. ACM, 2015.

[132] P. Yanardag, and S. V. N. Vishwanathan. “Deep graph kernels.” Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. ACM, 2015.

[133] S. Cao, W. Lu, and Q. Xu. “Deep Neural Networks for Learning Graph Representations.” AAAI. 2016.

[134] T. D. Bui, S. Ravi, and V. Ramavajjala. “Neural Graph Machines: Learning Neural Networks Using Graphs.” arXiv preprint arXiv:1703.04818(2017).

[135] S. Somanchi, and D. B. Neill. "Graph Structure Learning from Unlabeled Data for Early Outbreak Detection." IEEE Intelligent Systems 32.2 (2017): 80-84.

[136] N. Nori, D. Bollegala, and M. Ishizuka. "Interest prediction on multinomial, time-evolving social graph." IJCAI. Vol. 11. 2011.

[137] F. Ganz, P. Barnaghi, and F. Carrez. "Automated semantic knowledge acquisition from sensor data." IEEE Systems Journal 10.3 (2016): 1214-1225.

[138] Y.-Q. Zhang, et al. "Human interactive patterns in temporal networks." IEEE Transactions on Systems, Man, and Cybernetics: Systems 45.2 (2015): 214-222.

[139] D. Di Nucci, et al. "Dynamic selection of classifiers in bug prediction: An adaptive method." IEEE Transactions on Emerging Topics in Computational Intelligence 1.3 (2017): 202-212.

[140] E. Cambria, et al. "Computational intelligence for natural language processing." IEEE Comput Intell Mag 9.1 (2014): 19-63.

[141] X. Luo, Z. Xu, J. Yu, and X. Chen "Building association link network for semantic link on web resources." IEEE Transactions on Automation Science and Engineering 8.3 (2011): 482-494.

[142] O. Nasraoui, et al. "A web usage mining framework for mining evolving user profiles in dynamic web sites." IEEE Transactions on Knowledge and Data Engineering 20.2 (2008): 202-215.

[143] K. Huang, Y. Fan, and W. Tan. "Recommendation in an evolving service ecosystem based on network prediction." IEEE Transactions on Automation Science and Engineering 11.3 (2014): 906-920.

[144] E. M. Jin, M. Girvan, and M. EJ Newman. "Structure of growing social networks." Physical review E 64.4 (2001): 046132.

[145] Q. Zhao, et al. "Communication motifs: a tool to characterize social communications." Proceedings of the 19th ACM International Conference on Information and Knowledge Management. ACM, 2010, 1645-1648.

[146] R. Cloutier, B. Sauser, M. Bone, and A. Taylor. "Transitioning systems thinking to model-based systems engineering: systemigrams to SysML models." IEEE Transactions on Systems, Man, and Cybernetics: Systems 45, no. 4 (2015): 662-674.

[147] X. Zhang, et al. "Dynamic motifs in socio-economic networks." EPL (Europhysics Letters) 108.5 (2014): 58001.

[148] S. Gurukar, S. Ranu, and B. Ravindran. "Commit: A scalable approach to mining communication motifs from dynamic networks." Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data. ACM, 2015.

[149] R. Pascanu, et al. "How to construct deep recurrent neural networks." arXiv preprint arXiv:1312.6026 (2013).

[150] M. Hermans, and B. Schrauwen. "Training and analysing deep recurrent neural networks."

Advances in Neural Information Processing Systems. 2013.

[151] A. Graves, A.-R. Mohamed, and G. Hinton. “Speech recognition with deep recurrent neural networks.” IEEE International Conference on Acoustics, Speech and Signal Processing. IEEE, 2013.

[152] H. Sak, A. W. Senior, and F. Beaufays. “Long short-term memory recurrent neural network architectures for large scale acoustic modeling.” INTERSPEECH. 2014.

[153] O. Irsoy, and C. Cardie. “Opinion Mining with Deep Recurrent Neural Networks.” EMNLP. 2014.

[154] K. Gregor, et al. “DRAW: a recurrent neural network for image generation.” Proceedings of the 32nd International Conference on International Conference on Machine Learning-Volume 37. 2015.

[155] M. F. Valstar, and M. Pantic. “Fully automatic recognition of the temporal phases of facial actions.” IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics) 42.1 (2012): 28-43.

[156] P. Rodriguez, et al. “Deep Pain: Exploiting Long Short-Term Memory Networks for Facial Expression Classification.” IEEE Transactions on Cybernetics (2017).

[157] C. Castro-Herrera, J. Cleland-Huang, and B. Mobasher. “A recommender system for dynamically evolving online forums.” Proceedings of the third ACM Conference on Recommender Systems. ACM, 2009.

[158] G. A. Susto, et al. “An adaptive machine learning decision system for flexible predictive maintenance.” 2014 IEEE International Conference on Automation Science and Engineering (CASE). IEEE, 2014.

[159] H. Yin, et al. “Dynamic user modeling in social media systems.” ACM Transactions on Information Systems (TOIS) 33.3 (2015): 10.

[160] C. Zeng, et al. “Online context-aware recommendation with time varying multi-armed bandit.” Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. ACM, 2016.

[161] B. Kehoe, et al. “A survey of research on cloud robotics and automation.” IEEE Transactions on Automation Science and Engineering 12.2 (2015): 398-409.

[162] Y. Koren, R. Bell, and C. Volinsky. “Matrix factorization techniques for recommender systems.” Computer 42.8 (2009).

[163] N. P. Whitehead, W. T. Scherer, and M. C. Smith. “Systems thinking about systems thinking a proposal for a common language.” IEEE Systems Journal 9.4 (2015): 1117-1128.

[164] Y. Zhao, Z. Liu, and M. Sun. “Representation Learning for Measuring Entity Relatedness with Rich Information.” IJCAI. 2015.

[165] J. Lian, et al. “Towards Better Representation Learning for Personalized News Recommendation: a Multi-Channel Deep Fusion Approach.” IJCAI. 2018.

[166] K. P. Unnikrishnan, et al. “Network reconstruction from dynamic data.” ACM SIGKDD

Explorations Newsletter 8.2 (2006): 90-91.

[167] Z. Yue, et al. “Linear Dynamic Network Reconstruction from Heterogeneous Datasets.” arXiv preprint arXiv:1612.01963 (2016).

[168] M. T. Angulo, et al. “Fundamental limitations of network reconstruction from temporal data”. Journal of The Royal Society Interface 14.127 (2017): 20160966.

[169] M. Porfiri, and M. R. Marin. “Information flow in a model of policy diffusion: an analytical study.” IEEE Transactions on Network Science and Engineering (2017).

[170] D. Chen, S. L. Sain, and K. Guo. “Data mining for the online retail industry: A case study of RFM model-based customer segmentation using data mining.” Journal of Database Marketing and Customer Strategy Management 19.3 (2012): 197-208.

[171] M. Hu, and B. Liu. “Mining and summarizing customer reviews.” 10th ACM SIGKDD International Conference on Knowledge Discovery and Data mining. ACM, 2004.

[172] B. Liu, M. Hu, and J. Cheng. “Opinion observer: analyzing and comparing opinions on the web.” 14th International Conference on World Wide Web. ACM, 2005.

[173] T. Hocevar and J. Dem ˇ sar, “A combinatorial approach to graphlet counting,” Bioinformatics, vol. 30, no. 4, pp. 559–565, 2014.

[174] J. Ugander, L. Backstrom, and J. Kleinberg. “Subgraph frequencies: Mapping the empirical and extremal geography of large graph collections.” Proceedings of the 22nd International Conference on World Wide Web. ACM, 2013.

[175] S. Wernicke, and F. Rasche. “FANMOD: a tool for fast network motif detection.” Bioinformatics 22.9 (2006): 1152-1153.

[176] Z. RM Kashani, et al. “Kavosh: a new algorithm for finding network motifs.” BMC Bioinformatics 10.1 (2009): 318.

[177] X. Li, et al. “NetMODE: network motif detection without Nauty.” PloS One 7.12 (2012): e50093.

[178] L. AA Meira, et al. “acc-Motif: accelerated network motif detection.” IEEE/ACM Transactions on Computational Biology and Bioinformatics (TCBB) 11.5 (2014): 853-862.

[179] acc-Motif: Accelerated Motif Detection, http://www.ft.unicamp.br/docentes/meira/accmotifs/ accessed Jun 2017.

[180] N. K. Ahmed, R. A. Rossi, N. G. Duffield, and T. L. Willke, “Graphlet decomposition: Framework, algorithms, and applications,” Knowl. Inf. Syst., vol. 50, no. 3, pp. 689–722, 2017.

[181] N. K. Ahmed, T. L. Willke, and R. A. Rossi, “Estimation of local subgraph counts,” in Proc. IEEE International Conference Big Data, 2016, pp. 559-565, 2014.

[182] S. D. Eppinger, and T. R. Browning. Design structure matrix methods and applications. MIT press, 2012.

[183] D. V. Steward. “The design structure system: A method for managing the design of complex

systems.” IEEE Transactions on Engineering Management 3 (1981): 71-74.

[184] T. R. Browning. “Design structure matrix extensions and innovations: a survey and new opportunities.” IEEE Transactions on Engineering Management 63.1 (2016): 27-52.

[185] A. Yassine, and D. Braha. “Complex concurrent engineering and the design structure matrix method.” Concurrent Engineering 11.3 (2003): 165-176.

[186] B. Wehrwein. “Augmented design structure matrix visualizations for software system analysis.” U.S. Patent No. 8,799,859. 5 Aug. 2014.

[187] D. M. Powers. “Evaluation: from precision, recall and F-measure to ROC, informedness, markedness and correlation.” (2011).

[188] Y. Bengio, and Y. Grandvalet. “No unbiased estimator of the variance of k-fold cross-validation.” Journal of machine learning research 5.Sep (2004): 1089-1105.

[189] J. D. Rodriguez, A. Perez, and J. A. Lozano. “Sensitivity analysis of k-fold cross validation in prediction error estimation.” IEEE Transactions on Pattern Analysis and Machine Intelligence 32.3 (2010): 569-575.

[190] R. Kohavi, and G. H. John. “Wrappers for feature subset selection.” Artificial intelligence 97.1-2 (1997): 273-324.

[191] S. Yusuke. “Java deep learning essentials.” (2016).

[192] S. Yusuke. “Deep learning source code” https://github.com/yusugomori/DeepLearning accessed in 2016.

[193] H. He, and E. A. Garcia. “Learning from imbalanced data.” IEEE Transactions on Knowledge and Data Engineering 21.9 (2009): 1263-1284.

[194] A. Karniel, and Y. Reich. “Formalizing a workflow-net implementation of design-structure-matrix-based process planning for new product development.” IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans 41.3 (2011): 476-491.

[195] A. Kosari, M. H. Jafari, and M. Fakoor. “On Equivalency Between Numerical Process DSM and State-Space Representation.” IEEE Transactions on Engineering Management 63.4 (2016): 404-413.

[196] W. Zaremba, I. Sutskever, and O. Vinyals. “Recurrent neural network regularization.” arXiv preprint arXiv:1409.2329 (2014).

[197] J. Weston, S. Chopra, and A. Bordes. “Memory networks.” arXiv preprint arXiv:1410.3916 (2014).

[198] S. Sukhbaatar, J. Weston, and R. Fergus. “End-to-end memory networks.” Advances in Neural Information Processing Systems. 2015.

[199] A. Kumar, et al. “Ask me anything: Dynamic memory networks for natural language processing.” International Conference on Machine Learning. 2016.

[200] J. N. Bruck “Decades-long social memory in bottlenose dolphins.” Proceedings of the Royal Society of London B: Biological Sciences 280.1768 (2013): 20131726.

[201] G. Martin-Ordas, D. Berntsen, and J. Call. “Memory for distant past events in chimpanzees and orangutans.” Current Biology 23.15 (2013): 1438-1441.

[202] H. E. Watts, and K. E. Holekamp. “Interspecific competition influences reproduction in spotted hyenas.” Journal of Zoology 276.4 (2008): 402-410.

[203] A. Gulli, F. Tanganelli, and A. Savona. “System and method for monitoring evolution over time of temporal content.” U.S. Patent Application No. 11/313,584.

[204] G. Rishwaraj, S. G. Ponnambalam, and C. K. Loo. “Heuristics-Based Trust Estimation in Multiagent Systems Using Temporal Difference Learning.” IEEE Transactions on Cybernetics (2017).

[205] K. Li, et al. “LRBM: A restricted boltzmann machine based approach for representation learning on linked data.” 2014 IEEE International Conference on. Data Mining (ICDM), IEEE, 2014.

[206] S. Poria, et al. “Convolutional MKL based multimodal emotion recognition and sentiment analysis.” 2016 IEEE 16th International Conference on Data Mining (ICDM). IEEE, 2016.

[207] W. Maass. “Networks of spiking neurons: the third generation of neural network models.” Neural networks 10.9 (1997): 1659-167.

[208] TechAmerica Foundation’s Federal Big Data Commission. (2012). Demystifying big data: practical guide to transforming the business of Government.

[209] I. Taleb, R. Dssouli, and M. A. Serhani. “Big data pre-processing: A quality framework.” 2015 IEEE International Congress on Big Data (BigData Congress). IEEE, 2015.

[210] G. Amir, and Murtaza Haider. “Beyond the hype: Big data concepts, methods, and analytics.” International Journal of Information Management 35.2 (2015): 137-144.

[211] N. Zhong, Y. Li, and S.-T. Wu. “Effective Pattern Discovery for Text Mining.” IEEE Transactions on Knowledge and Data Engineering 24.1 (2012): 30-44.

[212] X. Wu, et al. “Data mining with big data.” IEEE Transactions on Knowledge and Data Engineering 26.1 (2014): 97-107.

[213] J. Hipp, U. Güntzer, and G. Nakhaeizadeh. “Algorithms for association rule mining—a general survey and comparison.” ACM sigkdd explorations newsletter 2.1 (2000): 58-64.

[214] A. Ceglar, and J. F. Roddick. “Association mining.” ACM Computing Surveys (CSUR) 38.2 (2006): 5.

[215] Z. Matei, et al. “Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing.” Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation. USENIX Association, 2012.

[216] H. Li, et al. “PFP: Parallel FP-Growth for Query Recommendation.” Proceedings of the 2008 ACM conference on Recommender systems. ACM, 2008.

[217] “Frequent Pattern Mining - RDD-based API” https://spark.apache.org/docs/latest/mllib-frequent-

pattern-mining.html, September 2017.

[218] J. Han, J. Pei, and Y. Yin. “Mining frequent patterns without candidate generation.” ACM sigmod record. Vol. 29. No. 2. ACM, 2000.

[219] J. Han, and J. Pei. “Mining frequent patterns by pattern-growth: methodology and implications.” ACM SIGKDD explorations newsletter 2.2 (2000): 14-20.

[220] J. M. Namayanja, and V. P. Janeja. “Change detection in temporally evolving computer networks: A big data framework.” 2014 IEEE International Conference on Big Data (Big Data). IEEE, 2014.

[221] P. Basanta-Val, et al. “Architecting time-critical big-data systems.” IEEE Transactions on Big Data 2.4 (2016): 310-324.

[222] D. Talia. “Clouds for scalable Big Data Analytics.” Computer 46.5 (2013): 98-101.

[223] W. Fan, and A. Bifet. “Mining Big Data: current status, and forecast to the future.” ACM SIGKDD Explorations Newsletter 14.2 (2013): 1-5.

[224] C. Yunliang, F. Li, and J. Fan. “Mining association rules in big data with NGEP.” Cluster Computing 18.2 (2015): 577-585.

[225] E. Ölmezogullari, and I. Ari. “Online Association Rule Mining over fast data.” 2013 IEEE International Congress on Big Data (BigData Congress). IEEE, 2013.

[226] R. Sumbaly, J. Kreps, and S. Shah. “The big data ecosystem at linkedin.” Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data. ACM, 2013.

[227] KDD Cup 2016, https://kddcup2016.azurewebsites.net/

[228] Z. Qiu, et al. “Heterogenous Graph Mining for Measuring the Impact of Research Institutions”. (2016).

[229] L. Breiman. Random forests. Machine learning, 45(1):5–32, 2001.

[230] J. H. Friedman. Greedy function approximation: a gradient boosting machine. Annals of statistics, pages 1189–1232, 2001.

[231] S. Datta, R. Lakdawala, and S. Sarkar. “Understanding the Inter-domain Presence of Research Topics in the Computing Discipline: An Empirical Study.” IEEE Transactions on Emerging Topics in Computing (2018).

[232] Y. Qian, et al. “Feature Engineering and Ensemble Modeling for Paper Acceptance Rank Prediction.” arXiv preprint arXiv:1611.04369 (2016).

[233] V. Sandulescu, and M. Chiru. “Predicting the future relevance of research institutions-The winning solution of the KDD Cup 2016.” arXiv preprint arXiv:1609.02728 (2016).

[234] S. Tuarob, et al. “AlgorithmSeer: A system for extracting and searching for algorithms in scholarly big data.” IEEE Transactions on Big Data 2.1 (2016): 3-17.

[235] X. Zhou, et al. “Academic Influence Aware and Multidimensional Network Analysis for Research

Collaboration Navigation Based on Scholarly Big Data." IEEE Transactions on Emerging Topics in Computing (2018).

[236] J. Song, et al. "FacetsBase: a Key-value Store Optimized for Querying on Scholarly Data." IEEE Transactions on Emerging Topics in Computing (2018).

[237] D. Zhang, and M. Kabuka. "Distributed Relationship Mining over Big Scholar Data." IEEE Transactions on Emerging Topics in Computing (2018). S. A. Frost, and M. J. Balas. "Evolving Systems and Adaptive Key Component Control." (2009).

[238] M. Efatmaneshnik, S. Shoval, and L. Qiao. "A Standard Description of the Terms Module and Modularity for Systems Engineering." IEEE Transactions on Engineering Management (2018).

[239] M. A. Chaumun, H. Kabaili, R. K. Keller, F. Lustman, and G. Saint-Denis. "Design properties and object-oriented software changeability." In Proceedings of the Fourth European Conference on Software Maintenance and Reengineering IEEE, 2000: 45.

[240] H. P. Breivold, I. Crnkovic, and P. J. Eriksson. "Analyzing software evolvability." Annual IEEE International Computer Software and Applications Conference. IEEE, 2008.

[241] S. V. Loo, G. Muller, T. Punter, D. Watts, P. America, and J. Rutgers. "The Darwin project: evolvability of software-intensive systems." In IEEE International Workshop on Software Evolvability, pp. 48-53. IEEE, 2007.

[242] T. Srivastava, P. Desikan, and V. Kumar. "Web mining–concepts, applications and research directions." Foundations and advances in data mining (2005): 275-307.

[243] H. Chen, and M. Chau. "Web mining: Machine learning for Web applications." Annual review of information science and technology 38 (2004): 289-330.

[244] R. Chinnici, J.-J. Moreau, A. Ryman, S. Weerawarana. "Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language" https://www.w3.org/TR/wsdl20/ W3C Recommendation 26 June 2007.

[245] A. Chaturvedi. "Change Impact Analysis Based Regression Testing of Web Services." arXiv preprint arXiv:1408.1600 (2014).

[246] A. Chaturvedi, and A. Gupta. "A tool supported approach to perform efficient regression testing of web services." Maintenance and Evolution of Service-Oriented and Cloud-Based Systems (MESOCA), 2013 IEEE 7th International Symposium on the. IEEE, 2013.

[247] A. Chaturvedi. "Reducing cost in regression testing of web service." Software Engineering (CONSEG), 2012 CSI Sixth International Conference on. IEEE, 2012.

[248] A. E. Hassan. "The road ahead for mining software repositories." Frontiers of Software Maintenance, FoSM 2008. IEEE.

[249] H. Xiao, J. Guo, and Y. Zou. "Supporting change impact analysis for service oriented business applications." Systems Development in SOA Environments, 2007. SDSOA'07: ICSE Workshops

2007. International Workshop on. IEEE, 2007.

[250] A. T. Ying, et al. “Predicting source code changes by mining change history.” IEEE Transactions on Software Engineering, vol. 30 no. 9 pp. 574-586 (2004).

[251] S. Negara, et al. “Mining fine-grained code changes to detect unknown change patterns.” 36th International Conference on Software Engineering. ACM, 2014.

[252] D. H. Khanh, and A. Ghose. “Supporting change propagation in the maintenance and evolution of service-oriented architectures.” 2010 Asia Pacific Soft. Engineering Conference. IEEE, 2010.

[253] M. P. Papazoglou, V. Andrikopoulos, and S. Benbernou. “Managing evolving services.” IEEE Software 28.3 (2011): 49-55.

[254] V. Andrikopoulos, S. Benbernou, and M. P. Papazoglou. “On the evolution of services.” IEEE Transactions on Software Engineering 38.3 (2012): 609-628.

[255] K. A. Alam, et al. “Impact analysis and change propagation in service-oriented enterprises: A systematic review.” Information Systems 54 (2015): 43-73.

[256] D. Romano, and M. Pinzger. “Analyzing the evolution of web services using fine-grained changes.” Web Services (ICWS), 2012 IEEE 19th International Conference on. IEEE, 2012.

[257] R. Karn, P. Kudva, and I. M. Elfadel. “Dynamic Autoselection and Autotuning of Machine Learning Models for Cloud Network Analytics.” IEEE Transactions on Parallel and Distributed Systems (2018).

[258] M. Harman and P. O’Hearn. From start-ups to scale-ups: Open problems and challenges in static and dynamic program analysis for testing and verification (keynote paper). In International Working Conference on Source Code Analysis and Manipulation, 2018.

[259] https://code.facebook.com/posts/1708075792818517/managing-resourcesfor-large-scale-testing/

[260] https://code.facebook.com/posts/300815046928882/the-mobile-device-lab-at-the-prineville-data-center/

[261] https://code.facebook.com/posts/924676474230092/mobile-performance-tooling-infrastructure-at-facebook/

[262] E. Constantinou, and I. Stamelos. “Architectural stability and evolution measurement for software reuse.” Proc. of the 30th Annual ACM Symposium on Applied Computing. ACM, 2015.

[263] S. Wang, et al. “Service Evolution Patterns.” Web Services (ICWS), 2014 IEEE International Conference on. IEEE, 2014.

[264] R. Kohar, and N. Parimala. “A metrics framework for measuring quality of a web service as it evolves.” International Journal of System Assurance Engineering and Management8.2 (2017): 1222-1236.

[265] M. Fokaefs, R. Mikhaiel, N. Tsantalis, E. Stroulia, and Alex Lau. “An empirical study on web service evolution.” In IEEE International Conference on Web Services (ICWS), pp. 49-56. IEEE,

2011.

[266] M. Fokaefs, and E. Stroulia. “WSDarwin: Studying the evolution of web service systems.” In Advanced Web Services, pp. 199-223. Springer, New York, NY, 2014.

[267] J. Li, Y. Xiong, X. Liu, and L. Zhang. “How does web service API evolution affect clients?.” In IEEE 20th International Conference on Web Services, pp. 300-307. IEEE, 2013.

[268] F. Mohamed, M. Abu-Matar, R. Mizouni, M. Al-Qutayri, and Z. A. Mahmoud. “SAAS dynamic evolution based on model-driven software product lines.” 2014 IEEE 6th International Conference on Cloud Computing Technology and Science (CloudCom). IEEE, 2014.

[269] P. Jamshidi, A. Ahmad, and C. Pahl. “Cloud migration research: a systematic review.” IEEE Transactions on Cloud Computing 1 (2013): 1.

[270] N. Ghosh, S. K. Ghosh, and S. K. Das. “SelCSP: A framework to facilitate selection of cloud service providers.” IEEE Transactions on Cloud Computing 3.1 (2015): 66-79.

[271] T. H. Noor, et al. “CloudArmor: Supporting reputation-based trust management for cloud services.” IEEE Transactions on Parallel and Distributed Systems 27.2 (2016): 367-380.

[272] Y. Du, et al. “Dynamic Monitoring of Service Outsourcing for Timed Workflow Processes.” IEEE Transactions on Engineering Management (2018).

[273] Z. Li, G. Nan, and M. Li. “Advertising or Freemium: The Impacts of Social Effects and Service Quality on Competing Platforms.” IEEE Transactions on Engineering Management (2018).

[274] G. Canfora, M. Di Penta, “Testing services and service-centric systems: challenges and opportunities” IT Professional vol. 8, may 2006.

[275] D. S. Linthicum, “The Evolution of Cloud Service Governance.” IEEE Cloud Computing 2.6 (2015): 86-89.

[276] T. Apiwattanapong, A. Orso, and M. J. Harrold, “JDiff: A Differencing Technique and Tool for Object-Oriented Programs,” Journal of Automated Soft. Engineering, vol. 14, no. 1, pp 3-36, March 2007.

[277] “Membrane SOA Model,” http://membrane-soa.org/soa-model/, March 2017.

[278] “AWS EC2,” March 2017, https://github.com/chilts/awssum-amazon-ec2, March 2017.

[279] “AWS EC2 Release,” https://aws.amazon.com/releasenotes/Amazon-EC2, March 2017.

[280] “Eucalyptus,” https://github.com/eucalyptus/eucalyptus, March 2017.