LEARNING PATTERNS IN DYNAMIC GRAPHS WITH APPLICATION TO BIOLOGICAL NETWORKS

By

CHANG HUN YOU

A dissertation submitted in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

WASHINGTON STATE UNIVERSITY
School of Electrical Engineering and Computer Science

AUGUST 2009
To the Faculty of Washington State University:

The members of the Committee appointed to examine the dissertation of CHANG HUN YOU find it satisfactory and recommend that it be accepted.

___________________________________
Lawrence B. Holder, Ph.D., Chair

___________________________________
Diane J. Cook, Ph.D.

___________________________________
Shira L. Broschat, Ph.D.
ACKNOWLEDGMENT

I would like to show my gratitude to my advisor, Dr. Lawrence B. Holder, for his support, encouragement and advice throughout my study and research. I especially respect his patience even for depressed and tedious advancement. He has always guided me to find my own way rather than to direct to his way. I would also like to thank Dr. Diane J. Cook for her great comments and service as a committee member of two degrees. She has always perceived problems that I could not easily catch. I also thank to Dr. Shira Broschat for her comments and service on my Ph. D. committee. She also gave me a great help and advice for my future career.

I am also grateful to members of the graph mining group and artificial intelligence laboratory in both universities, Washington State University and University of Texas at Arlington. I especially thank Nikhil Ketkar, William Eberle and Jacek Kukluk for great collaboration and comments. Aaron Crandall and Jim Kusznir are good mentors for my research and life in Pullman.

I cannot find a proper word to thank my father and mother. My father is a model for my life. He always waits for and respects my decision on every moment in my life. I believe he has always worried about my decision, but he has never shown his prior opinion to respect my way. I should be the worst son to my mother, because of my long and terrible medical history. When I opened my eyes in a hospital, she was always there with me. After coming to the United States, she has always worried about my health and food. My sister, Young rae, and brother, Young hoon, they are always my close friends and mentors. Young rae is my great big sister who I can always believe in. Young hoon is my little brother, but he always cheers me up like a “big” brother.
I am grateful to my father-in-law, mother-in-law, Hye won Nam, and Young hoon Kim. They always pray for me and believe in me. I also thank them to allow their lastborn, my wife, to be with me in this lonely country.

My wife and I are not alone in the US, because we have a family here. Sook Lee, and her sons Won and Min make us a great family. Sook loves us like her son and daughter. Won is a great mentor to me and the best friend. Moreover, we are happy because we have a place to go in a holiday in the US.

I cannot imagine how I can survive without my wife, Hye In Nam, throughout my graduate study and life in the US. She has always been, and will be my soul and power behind me. I completed two best works of my life in the US, Ph.D. degree and marriage with her.

Lastly, I would thank everyone who guide and encourage me to be here. Even though I do not mention them, I have never forgotten them.
LEARNING PATTERNS IN DYNAMIC GRAPHS WITH APPLICATION TO BIOLOGICAL NETWORKS

Abstract

by Chang hun You, Ph.D.
Washington State University
August 2009

Chair: Lawrence B. Holder

We propose dynamic graph-based relational mining approach to learn structural patterns in graphs or networks as they change over time. There are a huge amount of data that can be represented as graphs, and a majority of the data have dynamic properties as well as structural properties. Most current graph-based data mining approaches focus on only static graphs, but few approaches address dynamic graphs. Our approach analyzes a dynamic graph containing a sequence of graphs, and discovers rules that capture the changes that occur between pairs of graphs in the sequence. These rules represent the graph rewrite rules that the first graph must go through to be isomorphic to the second graph. Then, our approach feeds the graph rewrite rules into a machine learning system that learns general transformation rules describing the types of changes that occur for a class of dynamic graphs. The discovered graph-rewriting rules show how graphs change over time, and the transformation rules show the repeated patterns in the structural changes.
We apply our approach to the analysis of the dynamics of biological networks with the cell. A cell is not only a basic unit to a life, but also an optimal system. This system is well-organized so that it can be represented as biological networks, which include various molecules and relationships between them. Moreover, biological networks also change their structure over time to express dynamics of the biological systems. In our research, we apply the dynamic graph-based relational mining approach to biological networks to understand how the biosystems change over time. We evaluate our results using coverage and prediction metrics, and compare our results to those in biological literature. Our results show important patterns in the dynamics of biological networks, for example, discovering known patterns in the biological networks. Results also show the learned rules accurately predict future changes in the networks.

We also evaluate our approach using two other data: synthetic data and Enron email data. We apply our approach to the synthetic data with several varied conditions, such as noise, size and density ratio. We also apply our approach to the Enron email data, and compare to an alternative approach.
# TABLE OF CONTENTS

ACKNOWLEDGEMENTS ........................................................................................................... iii 

ABSTRACT .................................................................................................................................... v 

1. INTRODUCTION ......................................................................................................................... 1 

2. RELATED WORK ............................................................................................................................ 4 

2.1 Graph-based Data Mining ........................................................................................................ 4 

2.2 Temporal Data Mining ............................................................................................................ 6 

2.3 Dynamic Graph Analysis ........................................................................................................ 7 

2.4 Structural Analysis of Biological Networks ............................................................................ 8 

2.5 Dynamics of Biological Networks .......................................................................................... 11 

2.6 Summary ..................................................................................................................................... 13 

3. PROBLEM DEFINITIONS .................................................................................................. 14 

3.1 Graph Rewriting Rules ........................................................................................................... 15 

3.2 Transformation Rules ............................................................................................................ 18 

3.3 Evaluation Metrics .................................................................................................................. 20 

3.4 Summary ..................................................................................................................................... 23 

4. Approach .................................................................................................................................. 25 

4.1 Substructure discovery ............................................................................................................ 26 

4.2 Learning Graph Rewriting Rules .......................................................................................... 27 

4.3 Learning Transformation Rules ............................................................................................ 30 

4.4 Complexity Issues ................................................................................................................... 33 

4.5 Summary ..................................................................................................................................... 35 

5. APPLICATION TO BIOLOGICAL NETWORKS .................................................................. 36 

5.1 Visualization Result using Graph Rewriting Rules ................................................................. 36 

vii
LIST OF FIGURES

Figure 2.1 TCA cycle biological network of Saccharomyces cerevisiae [24] ................................ 9

Figure 3.1 A framework of dynamic graph analysis. (A) A dynamic graph. (B) Learning graph rewriting rules from two sequential graphs. (C) Learning the entire set of graph rewriting rules. (D) Learning a transformation rule to abstract the learned graph rewriting rules (e.g., Sub is removed from G1 and then added back in G1 + 4). ................................................................. 15

Figure 3.2 (A) An example of application of graph rewriting rules, where the rule derives a graph H from a graph G by replacing a subgraph L by a subgraph R. (B) An example of application of graph rewriting rules, which shows a removal rule \{R, rc\} from a graph G and an addition rule \{A, ac\} to a graph H. The removal and addition substructures are connected to G and H by edges rc and ac. S represents the common subgraph between G and H. ................................................ 16

Figure 3.3 An instance of graph rewriting rules between graph G1 and G2 in a synthetic biological network. ....................................................................................................................... 17

Figure 4.1 Discovery of the best compressing subgraph in a set of graphs at iteration 1 (A), 2 (B), and 3 (C). ................................................................. 26

Figure 4.2 Examples of the temporal distance between removals and additions. (A) RAdist = 2 and ARdist = 1. (B) No addition between two successive removals (R1 and R4). .................. 32

Figure 5.1 (A) The oscillation curves of the changing gene expression values of three yeast genes: YNL071W, YER178W, and YBR221C. (B) An instance of the graph representation for a metabolic pathway. ....................................................................................................................... 37

Figure 5.2 Running time with the total size of running Algorithm 1 on ten dynamic graphs. .... 41

Figure 5.3 (A) A visualization of time points when the substructure including each gene is removed from or added to graphs representing the glycolysis pathway at the experiment of threshold 0.6. The points above the time axis represent the time points when the substructures including the specified genes or relation are removed (Genes with (-)). The points below the time axis represent the time points when the substructures including the specified genes or relation are added (Genes with (+)). Relation points represent the time points when the enzyme-enzyme relations are shown in the pathway. (B) A visualization of time points when a particular substructure is removed from or added to graphs representing the glycolysis pathway at the experiment of threshold 0.6. Each substructure includes a relation, which is an enzyme-enzyme relation between two genes, where ECrel(x,y) represents the relation, and x, y represent the id of enzymes................................................................. 42

Figure 5.4 Visualization of three periodic cycles in removals and additions of Enzyme Relations in TCA cycle (A), Pentose phosphate pathway (B), Urea cycle (C), Glutamate metabolism (C), Arginine and proline metabolism (E), Purine metabolism (F), Pyrimidine metabolism (G), and Glycerophospholipid metabolism (I) at the experiment of threshold 0.5. The experiment of Glycerolipid metabolism (H) is performed at threshold 0.4. The points marked as circles above the time axis represent removals, and the points marked as rectangles represent additions. ........ 43

Figure 5.5 Structural changes of a dynamic graph representing the partial glycolysis pathway. Gi denotes a graph at time i for 1 ≤ i ≤ 36. The blue arrows with boxed labels between two graphs,
$G_x$ and $G_y$, represent the transformation from $G_x$ to $G_y$, by application of the rule in the label of the arrow. Sub $P$ represents the substructure ($P = A$ to $F$, shown on the right) in each rule (removal and addition), where the red arrows with the dot boxed labels from the substructures represent the connection edges. For example, $G_1$ is transformed to $G_2$ by addition of Sub $A$, which is connected by a connection edge labeled “G_to_E”. ...................................................... 50

Figure 5.6 A visualization of discovered substructure of a removal rule in a dynamic graph representing the glycolysis pathway in our output (left) and on the KEGG glycolysis pathway map (right). Labels marked by “-[]” represent the removal rules and labels marked by “()” represent the connection edges (left). Red rectangles represent two genes and blue circles represent three compounds in the removal rule (right). When YKL060C is removed, two enzyme-relations between two genes are also removed. Two reactions R01015 and R01070, involved with the three compounds, are also removed. ................................................................ 51

Figure 5.7 KEGG Notch signaling pathway map and a portion of a graph that represents Notch signaling pathway. ........................................................................................................................ 53

Figure 5.8 (A) The best subgraph discovered in the graph rewriting rules of DG B, which is discovered as 9 instances in the 9 examples (4 removals and 5 additions). (B) Visualization of transformation rules in the learned graph rewriting rules of DG B. The above boxes denote the removals at the specified time. The below eclipses denote the additions at the specified time. The numbers on the arrow denote the temporal distances between two events like the removals and additions. (C) The repeated transformation rule of the temporal patterns in (B), where Sub 1 denotes an instance of (A). ............................................................................................................ 55

Figure 5.9 (A) The best subgraph (Sub 2) at iteration 2 includes the best subgraph (Sub 1) discover in DG D. Sub 1s are discovered at times 2, 5, 7, 10, 15, and 17, and Sub 2s are discovered at times 2, 5, 7, 15 and 17. (B) The visualization of temporal patterns in the learned graph rewriting rules including the Sub 1 and Sub 2 in (A). ............................................................................................................ 56

Figure 5.10 (A) The best subgraph discovered 8 times at the first iteration and (C) the third subgraph discovered 10 times at the third iteration. ............................................................................................................ 57

Figure 5.11 (A) The best subgraph (Sub1) discovered in TR1. Visualization of the graph rewriting rules including the subgraph in (A) in the training data (B) and the testing data (C)... 61

Figure 5.12 (A) Visualization of a portion of Sub1. Labels marked by “-[]” represent the removal rules and labels marked by “()” represent the connection edges. (B) A portion of the KEGG graphic map including the marked Sub1. Red circles (A) and rectangles (B) represent removed Active Cyclin:CDK and phosphorlated wee. Blue circles (A) and rectangles (B) represent SKP2 and CDC25 that are connected to the removal rule by the connection edges. ................. 63

Figure 5.13 (A) An instance of the best subgraph discovered in the experiment with model 171. (B) Visualization of the graph rewriting rules including the subgraph (A). (C) The transformation rule of the graph rewriting rules in (B). (D) Predicted graph rewriting rules from (C)............... 69

Figure 5.14 (A) Visualization of a portion of Sub1. Labels marked by “-[]” represent the removal rules and labels marked by “()” represent the connection edges. (B) A portion of the KEGG graphic map including the marked Sub1. Red circles (A) and rectangles (B) represent removed
the PER molecules. Blue circles (A) and rectangles (B) represent the TIM molecules that are connected to the removal rule by the connection edges.

Figure 6.1 An example of generating a random graph. (A) The input substructure contains 6 vertices and 5 edges. Vertices and Edges are 8 and 10. (B) Add two random vertices and reach 8 vertices. (C) Connect the starting graph to two added vertices using two random edges, and reach 7 edges. (D) Add three random edges and reach 10 edges.

Figure 6.2 An example of a dynamic graph using Core and Rule. The above orange subgraphs denote Rule (at time 1, 3, and 4), and the below red subgraphs denote Core. T denotes each time slice.

Figure 6.3 Three types of noise in a graph. (A) The addition noise represents random additions of vertices and/or edges. (B) The removal noise represents random removals of vertices and/or edges. (C) The replacement noise represents random replacements of labels for vertices and/or edges.

Figure 6.4 (A) Remove one random vertex (orange), and then remove edges (dashed) related to the removed vertex. (B) No more vertex removal, because of the extra edge removal. (C) The remainder of Vertices is transferred to Edges, and we remove one edge. (D) The final result has the correct size.

Figure 6.5 Prediction with noise rates in the four types of dynamic graphs: non-noise, addition noise, replacement noise and removal noise.

Figure 6.6 Running time vs. noise rates in the four types of dynamic graphs: non-noise, addition noise, replacement noise and removal noise.

Figure 6.7 Prediction for various sizes of Cores and Rules with the limited numbers of unique labels.

Figure 6.8 Running time for various sizes of Core and Rule with the sufficient numbers of unique labels.

Figure 6.9 Prediction for various density ratios of cores and rules with the limited numbers of unique labels. The Prediction axis shows the prediction value for the best transformation rule learned by DynGRL. The Core ratio axis shows the ratio of each core with the size (in the parenthesis). The Rule ratio axis shows the ratio of each rule with the size (in the parenthesis).

Figure 6.10 Running time for various ratios of cores and rules with the sufficient numbers of unique labels. The running time axis shows DynGRL’s running time. The core ratio axis shows the ratio of each core with the size (in parenthesis). The rule ratio axis represents the ratio of each rule with the size (in parenthesis).

Figure 7.1 Discovered best subgraphs in the dynamic graph contain 2,587 days’ email data. The best subgraph discovered at iteration 1 (A), and iteration 2 (B).

Figure 7.2 The three best subgraphs discovered by DynGRL in the dynamic graph contain 101 days of Enron email data. The best subgraph discovered at iteration 1 (A), iteration 3 (B) and iteration 6 (C).

Figure 7.3 A pattern discover by the Periodic Behavior Miner [23].
Figure 7.4 A visualization of removals and additions of $Sub_1$ discovered by DynGRL .......... 107
Figure 7.5 A visualization of removals and additions of $Sub_3$ discovered by DynGRL .......... 108
To my wife, Hye In Nam, who always listens to me and never hesitates to sacrifice for me.
1. INTRODUCTION

Graphs are natural ways to represent our world including entities and relationships between entities. From the small cell to human societies, most of our world is comprised of complex networks and structures that can be understood through graphs. Moreover, not only small cells but also human societies can change their structures and networks to maintain their life and create a better future. Analysis of dynamic graphs is an emergent area not only to understand a small cell, but also to protect our society from terrorists [1] or epidemic diseases [2].

Our bodies are well-organized and vigorous systems, which promote reproduction and sustain our lives. These well-organized systems can be defined by the attributes and structural properties of biological networks, which include various molecules and relationships between molecules. Vigorous systems refer to dynamic properties of biological networks, which continuously change, while an organism performs various biological activities. Therefore, analysis of the dynamics of biological networks is necessary to understand biosystems.

Our approach to analyzing biological networks starts with a dynamic graph containing a sequence of graphs representing biological networks changing over time. We propose a novel approach to analyze structural features along with temporal features in a dynamic graph to enhance our systems-level understanding of bio-organisms as well as to discover novel patterns in other structural data.

Discovered patterns that describe the dynamics of biological networks can represent significant information about a disease and help researchers develop new drugs. For example, lactose intolerance is the inability to digest lactose because of a lack of the lactase enzyme,
which breaks down lactose into galactose and glucose [3]. Our approach can discover novel patterns that describe the dynamics of the normal and/or lactase-lacked galactose metabolism pathway, i.e., abnormal patterns in the dynamics of the lactase-lacked pathway, and guide us to investigate a new treatment for this disease. During the development period, we can compare patterns with the treatment to patterns without any treatment. After development, the patterns can be used for evaluation of the new treatment. The combined structural and dynamic analysis of biosystems will show better performance than one-aspect analysis.

For analysis of dynamic graphs, we first structurally represent how one graph is transformed into another. Several approaches [4] can measure the difference between two graphs. But our first goal is to describe how two graphs are different, not merely that they are different or by how much. We use graph-rewriting rules to show how two graphs are different. The second step is to discover transformation rules that generalize the discovered graph rewriting rules into abstract patterns that represent the dynamics of a sequence of graphs.

This dissertation first introduces several preceding approaches related to analysis of dynamic graphs and biological networks in chapter 2. Chapter 3 presents our definition of graph-rewriting rules and more general transformation rules. We introduce our two step algorithm to discover graph-rewriting rules in a dynamic graph, and transformation rules in the discovered graph-rewriting rules using a compression-based approach in chapter 4. In our experiments, we generate dynamic graphs of the biological networks using artificial generation, microarray and mathematical modeling data, and then, we apply our approach. Chapter 5 shows results of learned graph rewriting rules (section 5.1) and transformation rules (section 5.2). We compare our compression-based approach with the frequent subgraph mining approaches [5; 6] in learning the transformation rules. In our results, graph rewriting rules represent the structural difference.
between two graphs, and describe how the graphs in the dynamic graph change over time. The transformation rules show which graph rewriting rules are repeated periodically and can help us predict the future changes of the dynamic graphs. Chapter 6 describes empirical evaluation our approach using synthetic data with several variations, such as noise, size and ratio. We try to identify how our system behaves with the varied data. We also apply our approach to another domain that can be represented as a dynamic graph in chapter 7. Chapter 8 concludes this dissertation with remaining problems.
2. RELATED WORK

With recent advances in computer science, we are now able to address computational approaches to the analysis of dynamic graphs. In the last decade, there has been great advance in graph-based data mining [7] as an approach to multi-relational data mining [8]. We now steer our focus to dynamic graphs to learn novel patterns that are ignored by most previous graph-based data mining approaches.

Systems biology is an emergent area in post-genomic bioinformatics. A biological network is a fundamental and indispensable way to describe a complex system in terms of both the structure and its dynamics. Analysis of dynamic graphs allows us to understand the dynamics of biological networks.

This chapter reviews related research in the analysis of dynamic graphs. First, we introduce several graph-based data mining approaches. Second, we introduce temporal data mining approaches. Third, we present the research in the analysis of dynamic graphs. Then, we present structural analysis of biological networks. Lastly, we present the related research in the dynamics of biological networks.

2.1 Graph-based Data Mining

The graph is an abstract data structure consisting of vertices and edges which are relationships between vertices. Graph-based data mining denotes a collection of algorithms for mining the relational aspects of data represented as a graph. Graph-based data mining has two major approaches: frequent subgraph mining and the compression-based approach. The
compression-based approach will be described in chapter 4. Frequent subgraph mining is focused on finding frequent subgraphs in a graph. There are two well-known approaches: Frequent SubGraph discovery and Graph-based Substructure Pattern Mining.

Frequent SubGraph discovery, FSG [5] is an approach that finds all connected subgraphs that appear frequently (more than a given minimum support threshold) in a set of graphs. FSG starts by finding all frequent single and double edge graphs. During each iteration, FSG expands the size of frequent subgraphs by adding one edge to generate candidate subgraphs. Then, it evaluates the candidates using the minimum support constraint, and prunes candidate subgraphs using the downward closure property, which observes that any subgraph that does not meet the minimum support constraint can be pruned along with all its descendants (supergraphs). In each step of the candidate generation and candidate pruning, canonical labeling is used to avoid the redundant generation of the same subgraph. This approach is successfully applied to classify chemical compounds [9]. In this research, chemical compounds are represented as two types of subgraph features: topological and geometric subgraphs. These two types of subgraphs and their frequency are used to generate a classification model using a support vector machine.

Graph-based Substructure Pattern Mining, gSpan [6], uses the depth-first search and lexicographic ordering to efficiently mine sets of graphs for frequent subgraphs. First gSpan sorts the labels, removes infrequent vertices and edges and relabels the remaining vertices and edges. Next it starts by finding all frequent one-edge subgraphs. The labels on these edges and vertices define a code for each graph. Larger subgraphs map themselves to longer codes. If the code of B is longer than A, the B code is a child of the A code in a code tree. If there are two non-unique codes in the tree, one of them is removed during the depth-first search traversal to reduce the cost of matching frequent subgraphs.
One challenge in frequent subgraph mining is that the approach can generate an exponential number of frequent subgraphs. CloseGraph [10] proposes a method for finding only closed frequent subgraphs. A graph \( G \) is closed if there is no supergraph of \( G \) that has the same support as \( G \). In this way, they can reduce the number of patterns generated in the discovery process. They apply this approach to discovery substructures in chemical compounds and compare to gSpan.

In our work, we mainly use the compression-based approach to discover graph rewriting rules and description rules. To evaluate our approach, we compare our methods with the frequent subgraph mining approach. While frequent subgraph mining could be used in place of our compression-based approach, we find that the compression-based approach can at times find more relevant subgraph patterns.

### 2.2. Temporal Data Mining

Temporal data mining attempts to learn temporal patterns in sequential data, which is ordered with respect to some index like time stamps, rather than static data [11; 12]. Mörchen's research [13] shows several unsupervised learning techniques for discovery of temporal patterns from symbolic temporal data. Symbolic temporal data can be categorized into time points and time interval data, as well as univariate and multivariate, and there are various approaches for each temporal data model. Laxman et al. [14] introduce an algorithm for finding frequent episodes in event streams. This approach measures frequency based on non-overlapping occurrences of the episodes. Allen et al. [15] formalized temporal logic for time intervals using 13 interval relations. This approach allows us to present temporal relations of sequential data.
There are several approaches to apply temporal data mining in biological data. Ho et al. [16] propose an approach to detect temporal patterns and relations between medical events of Hepatitis data. They represent medical information of patients as sequential events and classify temporal patterns and relations of medical testing results in the sequential events using the Naive Bayes classifier. Farach-Colton et al. [17] introduce an approach of mining temporal relations in protein-protein interactions. They model the assembly pathways of Ribosome using protein-protein interactions. This approach determines the order of molecular connections using the distance measure of each interaction between two proteins.

Temporal data mining is focused on the discovery of temporal relations or cause-effect associations. In other words, we can understand how or why the object changes rather than merely static properties of the object. Temporal data mining approaches discover temporal patterns in data, but they disregard relational aspects among entities. For example, they can identify temporal patterns of appearance of genes, such as a gene, YBR218C, appearing before another gene, YGL062W, but cannot identify how these two genes interact with each other.

2.3 Dynamic Graph Analysis

Most of our world can be represented as networks including entities and relationships between the entities. Many networks are not static in the real world. They change their structures and properties. The analysis of dynamic networks is an emergent area in data mining. It can be applied to various domains, such as biological networks, computer networks and social networks.
Several methods have addressed data mining in a dynamic graph. Sun et al. [18] propose a technique to discover communities and detect changes in graphs changing over time. They use matrix and encoding schemes to represent a dynamic graph. Other works [19; 4] propose several detection measures of abnormal changes in the sequence of graphs and graph distance measures between two graphs. These approaches can measure how much two graphs are different, but not show how they are different. Tensor analysis [20; 21] tries to analyze dynamic graphs using the tensor, which is a generalization of a matrix to present high-dimensional data. In this approach, the tensor is used to represent a dynamic graph.

Lahiri et al. introduce an approach to predict the future structure in a dynamic network using frequent subgraphs [22]. They also introduce a Periodic Behavior Mining System [23] in dynamic social networks. They discover frequent patterns as candidates, and keep either current periodic or possibly periodic patterns in a pattern tree. Then, they maintain the pattern tree containing periodic patterns over time. They try to discover changing structure in dynamic graphs, but there are several differences from our approach. First, they focus on patterns at each time slice, but not on changing state like our graph rewriting rules. Second, they use the frequency-based approach, but our approach uses the compression-based approach. Third, they maintain only frequent periodic patterns, but we represent all repeated patterns and find a transformation rule to describe the repeated changes.

2.4 Structural Analysis of Biological Networks

Biological networks have been studied from a structural aspect. Biological networks have various molecules and relations between them including reactions and relations among genes and
proteins. Biological networks include metabolic pathways, protein-protein interactions and gene regulatory networks [24]. The KEGG PATHWAY is a widely known database which contains information on various kinds of pathways including pathway image files [25]. The KEGG PATHWAY database has 96,277 pathways generated from 331 reference pathways (on June, 2009).

Figure 2.1 TCA cycle biological network of Saccharomyces cerevisiae [25]

Figure 2.1 shows a metabolic pathway called the TCA cycle (of the yeast), which is a metabolic pathway for the production of ATP (a fundamental energy molecule in a cell). A rectangle represents an enzyme (protein) or a gene, and a circle represents a chemical compound. Each arrow describes a relationship between these molecules. In the marked elements (two bold
rectangles and three bold circles near the lower left of Figure 2.1) a chemical compound Oxaloacetate (C00036) is changed to another compound (S)-Malate, C00149, by an enzyme (ec:1.1.1.37) that is generated from three genes YDL078C, YKL085W, and YOL126C. (S)-Malate is changed to another compound, Fumarate, as a product of an enzyme (ec:4.2.1.2) that is generated from a gene, YPL262W. These two simple biochemical reactions are also related to other reactions and form a metabolic pathway, the TCA cycle. A fundamental step to study metabolic pathways is the identification of structures covering a variety of biomolecules and their relationships. Dynamics and control methods of metabolic pathways are also included, because biological systems are interactive and well-controlled optimized systems. In addition to the structural aspect, we also consider the temporal aspect of biological networks, because the biosystems always change their properties and structures while interacting with other conditions.

Several graph-based data mining approaches have been developed for the structural analysis of biological networks [26; 27; 28]. First approach [26] represents metabolic pathways from KEGG as simplified graphs, where vertices represent enzymes and edges represent relationships between enzymes. They omit other entities, such as chemical compounds and reactions, because enzymes can cover major structures in metabolic pathways. They also represent enzymes as the same identification number for every species. In this way, their problem can be reduced to mining frequent edges, because each biological network uses the same enzyme name, and each edge can be identified by its incident vertices. Then, they try to find the frequent edges over the different biological networks, where the edges represent a series of reactions in the pathways. Their approach reduces the graph mining problem to the frequent edge mining problem. They gain benefit on the computational cost, but lose biological meaning due to oversimplification of the problem.
The other two approaches [27; 28] represent all information in the KEGG database as graphs, where their representation includes distinguishing features over different species. The first, an approach on discovery of node replacement graph grammars [29], discovers graph grammar patterns that can describe the structure of metabolic pathways using hierarchical graph grammars and their productions. The last approach [28] applies the compression-based graph mining approach to metabolic pathways to discover distinguishing patterns over two different groups of pathways and common patterns in one group of pathways. Their discovered patterns show common or distinguishing features over several groups of pathways.

There are two main points to consider for understanding biological networks: structural and temporal aspects. The former reminds us to focus on relations between molecules as well as a single molecule. The latter is necessary to understand biological networks as dynamic operations rather than static relations, because every biological process changes over time and interacts with inner or outer conditions. For this reason, we need an approach to analyze biological networks changing over time in both aspects: structural and non-structural properties.

2.5 Dynamics of Biological Networks

To investigate bio-organisms and understand the theory of life, we should consider our bodies are dynamic. Our bodies are well-organized and vigorous systems, which promote reproduction and sustain our lives. Well-organized systems refer to structural properties of biological networks, which include various molecules and relationships between molecules. Vigorous systems refer to dynamic properties of biological networks, which continuously change their structures and properties, while an organism performs various biological activities, such as
digestion, respiration and so on. We assume the structures of biological networks change over time as they interact with specific conditions, for instance, a disease.

In addition to the structural aspect, we also consider the temporal aspect of biological networks, because the biosystems always change their properties and structures while interacting with other conditions. There are two approaches to analyze dynamic properties of biological systems. The one approach is mathematical modeling, which is an abstract model to describe a system using mathematical formulae [30; 31]. Most of these approaches, as a type of quantitative analysis, model the kinetics of pathways and analyze the trends in the amounts of molecules and the flux of biochemical reactions. But most of them disregard relations among multiple molecules.

The other approach is the microarray. The microarray is a tool for measuring gene expression levels for thousands of genes at the same time [32; 33]. Most genes are co-expressed, as most proteins interact with other molecules. Co-expressed genes construct common processes or patterns in biological networks (gene regulatory networks or protein-protein interaction networks) in the specific condition or over time. Microarrays can also monitor patterns in gene expression levels for the period of time or at the different conditions. Patterns in gene expression levels can represent changes in the biological status or distinguish two different states, such as the normal and disease state. But the microarray analysis can overlook structural aspects, which show how the genes or expressed gene products are related to each other in biological networks.

In our approach we combine the mathematical modeling and microarray data with KEGG PATHWAY data to generate dynamic graphs representing biological networks that change over time. Then, we apply our dynamic graph-based relational approach to analyze in two aspects: structural and temporal.
2.6 Summary

In this chapter, we review some prior research related to the analysis of dynamic graphs and biological networks. Graph-based data mining approaches discover patterns in static graphs, but typically ignore dynamic properties. Temporal data mining approaches can learn temporal patterns in sequential or temporal data, but overlook relational patterns between entities. Dynamic graph analysis is an emergent area, where matrix-based and frequent subgraph mining approaches are addressed, but none of the approaches focus on changing states. Biological networks represent biosystems in two ways: structure and dynamics. But most approaches focus on one way and overlook the other. For this reason, we need to apply analysis of dynamic graphs to biological networks to enhance our system-level understanding.
3. PROBLEM DEFINITIONS

In this chapter, we define the dynamic graph learning problem. We define the graph rewriting rule and the transformation rule to describe the dynamics of a graph. Graph rewriting rules represent topological changes between two sequential versions of the graph, and transformation rules abstract the graph rewriting rules into the repeated patterns that represent the dynamics of the graph.

In our research, a graph $G$ denotes the directed labeled graph that is defined as $G = (V, E, L, L)$, where $V$ is a set of vertices, $E \subseteq V \times V$ is a set of edges. $L$ is a labeling function as $L: V \cup E \rightarrow L$ and $L$ is a set of labels for vertices and edges. Formally, we define $DG = \{G_1, G_2, \ldots, G_n\}$ as a dynamic graph, where each graph $G_i$ (time slice graph) is a graph at time $i$ for $1 \leq i \leq n$. The time slice graphs in a dynamic graph are not necessarily temporal or separated by the same amount. The dynamic graph is an ordered sequence of graphs, where the ordering may reflect a temporal ordering (which is the case in biological networks in chapter 5 and Enron email networks in chapter 7), but the ordering need not to be temporal. The time $i$ also does not have to be an equal unit of time. I.e., the time between $G_i$ and $G_{i+1}$ may be different than between $G_{i+1}$ and $G_{i+2}$.

Figure 3.1 shows the framework of our approach to analyzing dynamic graphs, which we call Dynamic Graph-based Relational Learning (DynGRL). A dynamic graph contains a sequence of graphs that are generated sampling static snapshots of the graph from a continuously-changing graph, e.g., a sequence of graphs represent one biological network that changes its structure over time. First, our approach learns graph rewriting rules including removals ($R_i$) and additions ($A_i$) between two sequential graphs $G_i$ and $G_{i+1}$ (figure 3.1 (B)), and
generates a list of the entire graph rewriting rules (figure 3.1 (C)). The removals and additions represent removal and addition subgraphs that will be defined in section 3.1. Then, the final step is to learn the transformation rules to abstract the structural change of the dynamic graph based on the repeated patterns in the graph rewriting rules.

Figure 3.1 The framework of Dynamic Graph-based Relational Learning (DynGRL). (A) A dynamic graph. (B) Learning graph rewriting rules from two sequential graphs. (C) Learning the entire set of graph rewriting rules. (D) Learning a transformation rule to abstract the learned graph rewriting rules (e.g., Sub is removed from $G_i$ and then added back in $G_{i+4}$).

The first section introduces graph rewriting rules. Then, we introduce transformation rules to generalize graph rewriting rules. Lastly, we present evaluation metrics for transformation rules.

3.1 Graph Rewriting Rules

Our research focuses on temporal and structural analysis of a dynamic graph. Our dynamic graph-based relational learning approach discovers graph rewriting rules in a series of graphs changing their structures over time. Each graph rewriting rule represents topological changes between two sequential graphs. Here, we define graph rewriting rules for our approach.
Graph rewriting is a method to represent topological changes of graphs using graph rewriting rules [34; 35]. Generally, graph rewriting rules identify subgraphs in a graph and modify them as shown in figure 3.2 (A). Each graph rewriting rule defines a transformation between $L$ and $R$, where $L$ and $R$ are subgraphs in two graphs $G$ and $H$ respectively, such that $L$ is replaced by $R$, $L$ is deleted, or $R$ is created [36]. There are also several algorithms to discover the node or edge replacement graph grammar using the minimum description length principle [29; 37]. They represent graphs using discovered patterns in the graphs and the production of the patterns. The graph grammar can be a way to represent structural changes between two graphs. But their productions use common patterns rather than different ones. Thus, their scope is limited to represent static graphs using embedded patterns.

Traditional approaches to the identification of graph rewriting rules determine which subgraphs will be replaced by other subgraphs. Our approach is focused on representing changing structures between two graphs rather than just what subgraphs change. First, we discover maximum common subgraphs between two sequential graphs $G$ and $H$. While this task
is NP-Complete [38], we use a tractable heuristic approach described later. Then, we derive removal substructures from \( G \) and addition substructures from \( H \). Figure 3.2 (B) shows an instance of this process. A maximum common subgraph (denoted by \( S \)) is discovered between two graphs, \( G \) and \( H \). Then the remaining structure in \( G \) and \( H \) becomes removal (denoted by \( R \)) and addition (denoted by \( A \)) substructures respectively. Our graph-rewriting rules contain connection edges. The connection edges are edges which are used to link removal (or addition) subgraphs to the original graphs. The edges with labels \( rc \) and \( ac \) in figure 3.2 (B) represent the connection edges between \( G \) (\( H \)) and removal rule \( R \) (addition subgraph \( A \)).

![Figure 3.3 An instance of graph rewriting rules between graph \( G_1 \) and \( G_2 \) in a synthetic biological network.](image)

For two consecutive graphs \( G_t \) and \( G_{t+1} \), we define \( S_{i,i+1} \) as the maximum common subgraph between \( G_t \) and \( G_{t+1} \). \( S_{i,i+1} \) can be a disconnected graph, i.e., describing the set of connected subgraphs common to \( G_t \) and \( G_{i+1} \). Then, we define a graph rewriting rule \( GR_{i,i+1} \) as follows.

\[
GR_{i,i+1} = \{(R_t, C_{R_t}), (A_t, C_{A_t})\}
\]

Then, a removal subgraph \( R_t \) and an addition subgraph \( A_t \) are defined as follows.

\[
R_t = G_t \setminus S_{i,i+1}, A_t = G_{i+1} \setminus S_{i,i+1}
\]
\(C_{R_i}\) and \(C_{A_i}\) are the sets of connection edges for \(R_i\) and \(A_i\) respectively. For example, the graph rewriting rule \(GR_{1,2}\) in figure 3.3 can be represented as follows.

\[
GR_{1,2} = \{(R_1, \{(s2, g3, bc), (s2, g4, bd)\}), (A_1, \{(g3, s1, de)\})\}
\]

The graph \(R_1\) denotes \(R\) (in \(G_1\)) that is linked by two connection edges labeled by 'bc' and 'bd'. \(A_1\) denotes \(A\) (in \(G_2\)) that is linked by one connection edge labeled by 'de'. In each edge, \(sX\) and \(gY\) denote the starting and ending vertices, where \(s\) denotes the vertex in the subgraph and \(g\) denotes the vertex in the original graph. A prior graph \(G_i\) is changed to a posterior graph \(G_{i+1}\) by application of a set of graph rewriting rules \(GR_{i,i+1}\) as follows.

\[
G_{i+1} = G_i \oplus GR_{i,i+1}
\]

After iterating this process for \(n\) graphs, i.e., the entire sequence in the dynamic graph, we have \(n-1\) Rs and \(n-1\) As as shown in figure 3.1 (C). Here, we consider a set of graphs \(L\) that is a list of graph rewriting rules learned in \(DG\). Then, \(L\) contains \(n-1\) Rs and \(n-1\) As like \(L = \{R_1, A_1, R_2, A_2, \ldots, R_{n-1}, A_{n-1}\}\). We arrange \(R\) and \(A\) in order of time when the event occurs. We assume \(R_i\) and \(A_i\) happen at the same time between \(G_i\) and \(G_{i+1}\). In the real world, removals and additions between two successive time slices can happen at different times between \(i\) and \(i+1\). So, we assume a temporal distance between \(R_i\) and \(A_i\) is 0. We also assume \(A_i\) represents an addition to \(G_{i+1}\) but not to \(G_i\).

3.2 Transformation Rules

Next, we discover transformation rules in the learned graph rewriting rules to abstract the structural changes in the dynamic graph as shown in figure 3.1 (D). A transformation rule is
defined as a pattern in the learned graph rewriting rules, where the pattern best abstracts (compresses) the learned graph rewriting rules to best describe structural changes. More description will be in chapter 4. If some structural changes are repeated in the dynamic graph, there exist common subgraphs in the Rs and As. Then, we can discover the common patterns over L as our transformation rules. Biologically speaking, if there exist repeated changes of the structure of a biological network, the change can be an important pattern in the network. Here, we propose one simple transformation rule $TR$, which represents repeated additions and removals (or vice versa), as follows.

$$TR_e = Sub_e (+T_a, -T_r)$$

In the case when the transformation rule represents only repeated additions (or removals), $-T_r$ (or $+T_a$) would be $\emptyset$, like $Sub_e (+T_a)$ (or $Sub_e (-T_r)$). $Sub$ represents a subgraph, which adds to and/or removes from the graph repeatedly. $+T_a$ represents the time interval from the last removal to the current addition, and $-T_r$ represents the time interval from the last addition to the current removal. If $+T_a$ is shown before $-T_r$, the addition precedes the removal. For instance, $Sub_e (+4, -2)$ denotes a repeated structure added after 4 time intervals from the last removal and removed after 2 time intervals from the last addition as shown in figure 3.1 (D). $e$ denotes the number of the transformation rules in one dynamic graph. There can be multiple patterns over $L$ to describe the structural change of the dynamic graph, where the transformation rules are ranked by our metric that will be described in the next section.

There are other forms of transformation rules besides repeated add/remove rules, such as patterns conditional on context, i.e., removal/addition of structure X if structure Y is present (or absent), or patterns that describe numeric changes in combination with structure, e.g., describing
trends in the concentration of an enzyme, not just appearance. We will consider other types of transformation rules in future work.

3.3. Evaluation Metrics

We use two metrics to evaluate the learned transformation rules. The first metric is Coverage that represents how well the rule describes the changes in the graphs. The Coverage of the BestSub discovered at iteration $i$ in Algorithm 2 is computed as follows.

$$\text{Coverage} = \frac{\text{size}(\text{BestSub}) \sum_{g \in \text{coveredAs,Rs}} \frac{1}{\text{size}(g)}}{2(n-1)}$$

where the covered As and Rs are the addition and removal subgraphs in L that contain BestSub. The size of a graph $G$ is calculated as $\text{size}(G) = |V| + |E|$. These graphs are efficiently identified during the discovery of BestSub, avoiding the need for costly subgraph isomorphism tests. Coverage represents the portion of the learned subgraphs (the removal or addition subgraphs) described by the transformation rule to be based on BestSub. For example, suppose we have $n=3$ graphs from which we find two graph-rewriting rules. Then, we have two removal and two addition subgraphs. Assume that the size of $R_1$ is 10, $R_2$ is 12, $A_1$ is 15, and $A_2$ is 10. Also assume that BestSub is found in $R_1$ and $A_2$, and that BestSub has a size of 5.

Coverage is computed as $\frac{5(\frac{1}{10} + \frac{1}{12})}{4} = 0.25$.

Higher Coverage indicates the subgraph can describe more significant (larger portions of) changes. Currently, Coverage does not consider the size of connection edges ($|C|$). Unless the subgraph is isomorphic with all As and Rs, Coverage will be less than 1. If we assume the
size of the connection edges is 0, the sum of Coverage should be 1. Practically, the sum of Coverage would be less than 1 in many cases. There may be other structural changes (As and Rs) that discovered BestSubs do not cover. Thus, Coverage can indicate the generality of our discovered transformation rules in the structural changes, and the sum of Coverage should be 1 as our target value to claim our transformation rules represent all structural changes in a dynamic graph. In other words, we can treat our Coverage as precision so that Coverage is the fraction of the discovered relations to real structural changes.

We might use the Minimum Description Length (MDL) principle instead of the Coverage. The MDL principle says the best hypothesis for a given set of data is the one that leads to the largest compression of the data. In other words, this approach tries to find a pattern that best compress the input graph. The MDL principle for our approach will be described in chapter 4. We can evaluate the transformation rule using the compression capability of the entire set of graph rewriting rules. But our coverage metric treats equally each graph rewriting rule, because the coverage is the sum of the ratio of the BestSub to each graph rewriting rule. Our coverage considers each structural change at different time intervals as the same size of change.

We define Prediction as our second metric to evaluate the prediction capability of the learned transformation rules as follows. We predict the future structural changes using the learned transformation rules over training data. Then, we evaluate the predicted change using testing data. The detail experiments will be described in section 5.2.

\[
\text{Prediction} = \frac{\sum_{i \in P} s(\text{RealSub}_i, \text{PredictedSub}_i)}{|P|}
\]
\(P\) is the set of positions where we predict the \( PredictedSub_i\) will show up, \( RealSub_i\) is the actual subgraph in the testing data found at position \(i\), and a graph similarity \( s(G_m, G_n)\) between two graphs is defined as follows.

\[
s(G_m, G_n) = \frac{|\text{mcs}(G_m, G_n)|}{|G_m \cup G_n|}
\]

\(s(G_m, G_n)\) is the graph similarity metric that is defined based on the graph distance by Bunke et al. [4; 39], where \(\text{mcs}(G_m, G_n)\) denotes the maximum common subgraph between \(G_m\) and \(G_n\). The similarity metric represents how two graphs are similar. If two graphs \(G_m\) and \(G_n\) are isomorphic, \(s(G_m, G_n) = 1\). If \(s(G_m, G_n) = 0\), then there is no similarity between the two graphs \(G_m\) and \(G_n\). For example, \(s(G_1, G_2)\) in figure 4.1 is \(\frac{11}{16}\), where \(\text{mcs}(G_1, G_2) = 11\) and \(G_1 \cup G_2 = 16\). \( Prediction\) represents how much the predicted subgraph covers the subgraphs in the testing experiments. For example, suppose we predict a subgraph \(s\) will be shown 3 times in the testing data. Then, we observe the subgraph \(rs\) that is partially different from \(s\) at one time point \(s(G_{rs}, G_s) = 0.5\), and we observe a subgraph isomorphic with \(s\) at another time point. \( Prediction\) is computed as \(\frac{0.5 + 1.0 + 0}{3} = 0.5\). \( Prediction\) measures the proportion of predicted structural changes that are actually observed in the future.

Currently, our \( Prediction\) measure is not for a temporal prediction, i.e., the exact time the subgraph appears, but for a sequential prediction, i.e., whether the correct sequence of the subgraphs appears. If we also want to predict the exact time of appearance, rather than the order of appearance, we need to assess a penalty on some predictions. For example, we suppose that a pattern is predicted based on the transformation rule \(\text{Sub}_1(+4, -3)\). With the proper time appearance, we should give a penalty if the pattern appears as \((+3, -3)\) or \((+4, -2)\). Because we
choose the most frequent distance for the transformation rule, we need to consider these exceptional cases. However, this metric has the weakness, too. For example, when two predicted instances of a pattern appear with the right order, but one does not appear on the exact time, our metric gives the same score as when the two predicted instances appear on the exact time. We may include a slack with the proper time appearance, i.e., a pattern appearing within an allowed time (slack) can be considered as the right pattern. The other reason that the Prediction measure does not have temporal properties is that our algorithm for the discovery of transformation rules still needs to be updated with the temporal properties. When we complete our algorithm that can properly discover temporal patterns, i.e., with allowed time slacks, we will update our metric to evaluate the capability of the temporal prediction.

Even though we use the two metrics to evaluate our transformation rules, we still need better metrics to evaluate the quality of transformation rules. Unlike our Coverage, the new metric needs to focus on temporal properties as well as structural properties. We can measure how many instances of the best pattern exactly match with the learned transformation rule. In our algorithm, we choose most frequent temporal distances between two instances so that there can be several instances that do not match with the transformation rule. In this measure, we can represent the fraction of the matched instances to all instances of the best pattern.

3.4 Summary

Chapter 3 describes our problems and related definitions. A dynamic graph represents a sequence of graphs or snapshots of a network over time. Graph rewriting rules represent structural changes between two sequential graphs. Transformation rules represent patterns in the
graph rewriting rules to abstract the structural changes in the dynamic graph. We also present two metrics to evaluate our transformation rules.

In the next chapter, we present a two-step algorithm to discover graph rewriting rules and transformation rules.
4. Approach

This chapter describes our approach to analyzing dynamic graphs. We present a two step algorithm: Learning Graph Rewriting Rules [40; 41; 42] and Learning Transformation Rules [43; 44]. Graph rewriting rules represent structural changes between two sequential graphs in a dynamic graph. Transformation rules abstract the discovered rewrite rules and describe the periodicity of patterns within the rewrite rules. Step 1 learns graph rewriting rules in a dynamic graph to represent how pairs of sequential graphs are different. Step 2 learns the repeated transformation rules in the learned graph rewriting rules to describe how the graph changes over time, where the changes are actually represented as a sequence of revision graphs. For both algorithms we rely on a previously-developed method for finding the best-compressing subgraph in a set of graphs. For the first algorithm, repeated application of this method allows us to find the set of all subgraphs common to a pair of consecutive graphs. For the second algorithm this method allows us to find the subgraphs repeatedly added and removed in the dynamic graph. While we could use a frequent subgraph miner [5; 6] for this purpose, experiments have shown that the best-compressing patterns comparably capture the complete repeated structural changes [43].

First, we describe the compression-based substructure discovery. Section 4.2 describes the approach (step 1) to learn graph rewriting rules. Section 4.3 describes the approach (step 2) to learn transformation rules. Lastly, we discuss complexity issues in section 4.3.
4.1 Substructure discovery

We define the best-compressing subgraphs as those which minimize the description length of the input graph after being compressed by the subgraphs based on the Minimum Description Length (MDL) principle \[45; 46\]. Formally, the description length of the substructure \( S \) is represented by \( DL(S) \), the description length of the input graph is \( DL(G) \), and the description length of the input graph after compression is \( DL(G|S) \). The description length \( DL(G) \) is defined as the number of bits necessary to describe \( G \). The approach finds a substructure \( S \) that minimizes the Compression of the graph defined as follows.

\[
\text{Compression} = \frac{DL(S) + DL(G|S)}{DL(G)}
\]

This approach, which is called as DiscoverCommonSub in our algorithms, tries to maximize the Value of the subgraph, which is simply the inverse of the Compression. We present the Value with our learned subgraphs in the results section.

Figure 4.1 shows an example of the subgraph discovery by this compression-based approach. First, we can discover four instances of one common subgraph denoted by a red circle (A). After discovery, we compress each instance replacing by one vertex (\( S_1 \)), and we iterate the discovery process. In the second iteration (B), we discover two instances of the next common
subgraph, and compress them by one vertex \(S_2\). We stop the iteration because there is no more common subgraph, i.e., no more compression (C).

In addition to the MDL based approach, DiscoverSub() can use the size-based compression as follow.

\[
Compression = \frac{size(S) + size(G|S)}{size(G)}
\]

The size of a graph \(G\) is calculated as \(size(G) = |V| + |E|\), where \(|V|\) is the number of vertices and \(|E|\) is the number of edges. Size is a less accurate measure of compression, as it does not account for the compression of vertex and edge label information like MDL. However, size is faster to compute, and results based on size are typically consistent with those of MDL. In this research, we use both methods to discover subgraphs and compare our results with the frequent subgraph mining approach.

4.2 Learning Graph Rewriting Rules

Using the compression-based approach (as DiscoverCommonSub in the algorithm), we describe our two step algorithm. Algorithm 1 shows the learning graph rewriting rules algorithm, as depicted in figure 3.1 (C), where each iteration in the outer loop is depicted in figure 3.1 (B). First, the algorithm initializes \(L\) and \(C\) to store removal and addition subgraphs, and connection edges. At line 3, the algorithm prepares two sequential graphs as \(Graphs\), and then discovers one common subgraph by the compression-based approach. After compression, the algorithm discovers another subgraph at the next iteration until there is no more compression. In this way, the algorithm can discover the maximum common subgraph between two sequential graphs.
Algorithm 1: Learning Graph Rewriting Rules Algorithm

**Input:** Dynamic graph $DG = \{G_1, G_2, \cdots, G_n\}$

**Output:** Rewrite rules $L$, connection edges $C$

1. $L = \{\}, C = \{\}$
2. for $i = 1$ to $n - 1$
3.     $\text{Graphs} = \{G_i, G_{i+1}\}, S = \{\}$
4.     while More compression possible do
5.         $\text{BestSub} = \text{DiscoverCommonSub in Graphs}$
6.         $S = S \cup \text{BestSub}$
7.         Compress $\text{Graphs}$ by $\text{BestSub}$
8.     end
9.     $(R_i, C_{R_i}) = \text{mBFS}(S, G_i)$
10.    Add $R_i$ into $L$, and Add $C_{R_i}$ into $C$
11.    $(A_i, C_{A_i}) = \text{mBFS}(S, G_i)$
12.    Add $A_i$ into $L$, and Add $C_{A_i}$ into $C$
13. end

After compressing the two graphs by the maximum common subgraph, the algorithm identifies removal (or addition) subgraphs and connection edges (lines 9 and 11) using a modified Breadth First Search (mBFS), which adds each edge as well as each vertex into the queues as visited or to be visited. After compression, each maximum common subgraph is replaced by one vertex $S_t$. mBFS starts to search from one edge linked to $S_t$ to find one disconnected subgraph, and the starting edge is added into $C$. During the search, if there is one more edge between the disconnected subgraph and maximum common subgraph, the edge becomes the other connection edge. In this way, mBFS can find all disconnected subgraphs (without considering the link by the connection edges), and they become removal (or addition) subgraphs. mBFS stops the search when all connected edges are added in $C$. For example, in figure 4.1 (C), mBFS starts from one edge linked to $S_2$. (in case of $G_1$, choose $e_1$), and these starting edges are added into in $C$. Since there is one more linked edge ($e_2$) to $S_2$ in case of $G_2$, $e_2$ is added into $C$. Then, since there is no place to visit from the vertex $E$, $E$ becomes a disconnected subgraph as an addition subgraph. Since there is no place to visit from the vertex $F$
in $G_2$, $F$ becomes a disconnected subgraph as a removal subgraph. In this way, mBFS identifies removal subgraphs $R_i$ and addition subgraphs $A_i$ with connection edges.

**Function 1**: modified Breadth First Search (mBFS)

**Input**: Sub $S$, Graph $G$

**Output**: Rewrite rule $X$, connection edge $Y$

1. $X = \{\}, Y = \{\}, CEdges = \{\}$
2. for every sub in $S$ do
   3. Mark sub as visited
   4. foreach connection edge to sub do
      5. Add connection edge into $Y$
      6. Add incident vertex into toVisitVertex
      7. while toVisitVertex is not empty do
         8. Pop vertex from toVisitVertex
         9. if vertex is not visited then
            10. foreach connected edge to vertex do
               11. if two incident vertices are not visited then
                  12. Add connected edge to $CEdges$
                  13. Add incident vertex is not visited then
                     14. Push incident vertex to toVisitVertex
            end
            15. Mark vertex as visited
            16. Add vertex to $X$
         end
     end
   22. foreach edge in $CEdges$ do
      23. if two incident vertices of edge $\in X$ then
         24. Add edge into $X$
      25. else
         26. Add edge into $Y$
      end
   end
31. end
32. return $X, Y$

Function 1 shows the mBFS function. The function operates a basic breadth first search, visiting every vertex to find a connected graph (lines 4-23). But mBFS maintains every edge in $CEdges$ to find connection edges. The starting edge is saved as a connection edge $Y$ in line 5. The other edges that are connected to the visited vertices are saved into $CEdges$. After visiting all vertices from the starting (connection) edge, check all edges in $CEdges$ (lines 24-30). If one of
the incident vertices in an edge does not belong to Rule $X$, then the edge belongs to connection edge $Y$.

The output of Algorithm 1 includes $L$ and $C$. Each rule in $L$ is connected to the original graph by the connection edges in $C$. If a rule is not connected to the original graph, i.e., the maximum common subgraph is not connected to the remainder of the graph, the element of $C$ denotes $∅$. $L = \{R_1, A_1, R_2, A_2, \ldots, R_{n-1}, A_{n-1}\}$ is used in Algorithm 2 as an input. $C = \{C_{R_1}, C_{R_2}, C_{R_2}, \ldots, C_{R_{n-1}}, C_{A_{n-1}}\}$ is used not only to show the relations between the learned subgraphs and original graphs, but also to generate future graphs using our prediction results.

### 4.3 Learning Transformation Rules

From the result of Algorithm 1, we try to discover repeated rewrites, or transformation rules, to better understand how graphs change over time as shown in figure 3.1 (D). The transformation rule learning algorithm is shown in Algorithm 2. The input $L$ contains $2(n - 1)$ graphs: $(n - 1)$ $R$s and $(n - 1)$ $A$s. A user-specified parameter, $Limit$, specifies the number of different substructures to consider in each iteration. Note that each example (each $R$ or $A$) contains one or more graphs, which may not be connected to each other. We then use $DiscoverCommonSub$ again to find common subgraphs in $L$ (line 2). As described in figure 3.1, the best common subgraph in $L$ represents the subgraph in our transformation rule. After the discovery of the common subgraph, $L$ is compressed by this subgraph (line 3), and the discovery process is iterated until no more compression is achieved or we reach a user-defined maximum $Iter$ on the number of iterations. When the best subgraph at a latter iteration includes the best subgraph from a former iteration, the results can show the latter best subgraph includes a
previously-learned subgraph that is replaced by one vertex. More detail will be described with examples in the results section.

Algorithm 1: Learning Transformation Rules Algorithm

\textbf{Input}: L, Iter, Limit

\textbf{Output}: Transformation rules TR \( TR = \{ \} \)

\begin{algorithmic}
\While{More compression possible and Iter > 0}
\State \( \text{BestSub} = \text{DiscoverCommonSub in L} \)
\State \( \text{Compress L by BestSub} \)
\State Iter = Iter − 1
\For{Each Instance of BestSub}
\If{Instance is a removal}
\State \( R_i = \text{Instance} \)
\State Add RRdist between \( R_{i-1} \) and \( R_i \) into RRdistList
\If{\( A_{i-1} \) exists between \( R_{i-1} \) and \( R_i \)}
\State Add ARdist between \( A_{i-1} \) and \( R_i \) into ARdistList
\Else
\State Create \( A_{i-1} \) as \( A_{\text{Skip}} \)
\State Add ARdist\_Skip between \( A_{\text{Skip}} \) and \( R_i \) into ARdistList
\State Add RAdist\_Skip between \( R_{i-1} \) and \( A_{\text{Skip}} \) into RAdistList
\EndIf
\EndIf
\ElseIf{Instance is an addition}
\State \( A_i = \text{Instance} \)
\State Add AAdist between \( A_{i-1} \) and \( A_i \) into AAdistList
\If{\( R_{i-1} \) exists between \( A_{i-1} \) and \( A_i \)}
\State Add RAdist between \( R_{i-1} \) and \( A_i \) into RAdistList
\Else
\State Create \( R_{i-1} \) as \( R_{\text{Skip}} \)
\State Add RAdist\_Skip between \( R_{\text{Skip}} \) and \( A_i \) into RAdistList
\State Add ARdist\_Skip between \( A_{i-1} \) and \( R_{\text{Skip}} \) into RAdistList
\EndIf
\EndIf
\EndFor
\EndWhile
\If{\( R_{\text{Skip}} \) is the most frequent}
\State \( \text{Adist} = \text{the most frequent value in AAdistList} \)
\State Add BestSub\_\( e \)(+Adist) into TR
\ElseIf{\( A_{\text{Skip}} \) is the most frequent}
\State \( \text{Rdist} = \text{the most frequent value in RRdistList} \)
\State Add BestSub\_\( e \)(−Rdist) into TR
\Else
\State \( \text{Adist} = \text{the most frequent value in RAdistList} \)
\State \( \text{Rdist} = \text{the most frequent value in ARdistList} \)
\State Add BestSub\_\( e \)(+Adist,−Rdist) into TR
\EndElse
\EndIf
\State \text{return} TR
\end{algorithmic}
After the discovery process, Algorithm 2 generates a transformation rule at each iteration. When DiscoverCommonSub returns BestSub, BestSub contains all instances in $L$. Each instance is either a removal ($R$) or addition ($A$).

If the instance is a removal, we calculate a temporal distance ($RRdist_i$) between the current removal ($R_i$) and previous removal ($R_{i-1}$) (line 9). Then, we consider two possible cases as shown in figure 4.2. In figure 4.2 (A), Algorithm 2 discovers two removals and two additions. If the current removal ($R_i$) is $R_4$, the previous addition ($A_3$) is between the previous removal ($R_1$) and current removal ($R_4$). So, we can calculate $ARdist$ between $A_3$ and $R_4$ (line 11). But we have a problem in the case that there is no addition between two successive removals ($R_1$ and $R_4$) as shown in figure 4.2 (B). In that case, Algorithm 2 creates $A_{skip}$ between $R_1$ and $R_4$ without any specified time point (line 13). Therefore, $ARdist_{skip}$ and $RRdist_{skip}$ should be a null value that represents a non-specified distance (lines 14-15). In this way, we can calculate $AAdist$ and $ARdist$ related to additions (lines 17-27).

![Figure 4.2 Examples of the temporal distance between removals and additions. (A) RAdist = 2 and ARdist = 1. (B) No addition between two successive removals ($R_1$ and $R_4$).](image)

Then, Algorithm 2 generates a transformation rule for three cases. If the skipped removal (or addition) is the most frequent case, the transformation rule should be like $BestSub(+Adist)$.
(BestSub(−Rdist)), where Adist is the most frequent value in RDistList (ADistList) as shown in lines 29-34.

Otherwise, the transformation rule should be a general form like BestSub(+Adist, −Rdist), where Adist is the most frequent value in RDistList and Rdist is the most frequent value in ADistList (lines 36-38). When we generate TR_e, BestSub_e is included in the rule. BestSub denotes the best compressed subgraph over L, the e denotes the number of iterations. While DiscoverCommonSub iterates the discovery process, it returns the best subgraph in order of Value. In other words, the subgraph in the earlier iteration better compresses L than later iterations. If a transformation rule is discovered in the first iteration, the rule is labeled as TR_1, that includes the best compressed subgraph over all the iterations.

We might deploy the other approach to detect the periodic cycles, i.e., perform a graph isomorphism test for each time period k between G_l and G_l+k. However, the goal of transformation rules is the abstraction and description of structural changes across the entire dynamic graph, so that our algorithm discovers the best patterns and builds transformation rules to describe the patterns.

4.4 Complexity Issues

One challenge of our algorithm is to discover the maximum common subgraphs between two sequential graphs, because this problem is known to be NP-complete [38]. To address this issue we use a parameter, Limit, in DiscoverCommonSub to restrict the number of substructures to consider in each iteration. We can express Algorithm 1’s total runtime as N_1 = N_{DCS}(T − 1), where N_{DCS} is the runtime of DiscoverCommonSub and it runs for T − 1 times. Algorithm 2’s
running time is dominated by $N_{DCS}$. $N_{DCS}$ is restricted by $Limit$ that is calculated based on input data, specifically, the number of unique vertex and edge labels as follows. We use $Limit = UVL + 4\gamma(UEL - 1)$ in the application to biological networks, where UVL denotes the number of unique vertex labels, UEL denotes the number of unique edge labels and $\gamma$ denotes the user-specified parameter. We assume each vertex, usually a molecule, in biological networks has four relations with other vertices. Previous work [47] shows $N_{DCS}$ is polynomial in the $Limit$ parameter when $DiscoverCommonSub$ is run on a fully-connected graph (worst case). We can avoid the worst case in our domain, because biological networks are usually sparse graphs and there are not many instances due to mostly unique labels. But we still need to pursue reducing the running time for other domains. Also, our algorithm does not try to discover the entire set of maximum common substructures at once. In each step, the algorithm discovers a common, connected substructure and iterates the discovery process until discovering the entire set.

Graphs that represent biological networks usually contain unique vertex labels, because each vertex label usually denotes the name of the molecule. Because the maximum common subgraph problem in graphs with unique vertex labels is known to have quadratic complexity [48], discovery of the graph rewriting rules is still feasible. However, there will be a tradeoff between exactness and computation time when analyzing very large graphs.

The other issue we would discuss here is the use of MDL as a heuristic to discover the maximum common subgraphs and the likelihood that this approach does indeed to discover the maximum common subgraphs in step 1. We suppose two graphs $G_i$ and $G_{i+1}$ are connected graphs. $DiscoverCommonSubs$ tries to find the pattern that can best compress two input graphs as the maximum common subgraph between two graphs. Intuitively, it is impossible to discover a better compressing pattern than the maximum common subgraph. In other words, there is no
larger common graph, neither to best compress two input graphs, nor to be the maximum common graph. For this reason, DiscoverCommonSubs can discover the maximum common subgraphs between two graphs, if it can discover the best compressing pattern in the two graphs. The discovery process is a polynomial approximation restricted by Limit that is calculated based on the number of unique labels. As described above, the discovery process can be completed in polynomial time and reach the desired solution (maximum common subgraphs), because the graphs representing biological networks contain the sufficient numbers of unique labels for the sufficient search space. Chapter 6 contains several evaluation experiments regarding the number of unique labels.

4.5 Summary

In this chapter, we present our two-step algorithm for finding transformation rules describing patterns of change in dynamic graphs. Our algorithm is based on the compression-based substructure discovery approach. Step 1 discovers graph rewriting rules in a dynamic graph and returns a set of graph rewriting rules and a set of connection edges. Step 2 discovers transformation rules over the set of graph rewriting rules.

In the next chapter, we apply our approach to understand the dynamics of biological networks.
5. APPLICATION TO BIOLOGICAL NETWORKS

We performed two sorts of experiments to evaluate our approach. Section 5.1 shows an experiment using algorithm 1 on microarray data. The results in section 5.1 show algorithm 1 is useful for analyzing dynamic graphs of biological networks in both structural and temporal aspects. Then, section 5.2 shows results of transformation rules using algorithms 1 and 2 on artificial data, mathematical modeling and microarray data. The results in section 5.2 show transformation rules describing dynamics of biological networks and predicting future changes in the dynamic graphs.

5.1 Visualization Result using Graph Rewriting Rules

This section shows how discovered graph rewriting rules represent structural and temporal changes of dynamic graphs. First, we discuss temporal patterns in graph rewriting rules. Then, we represent how the discovered substructures in the rewriting rules link to the original graphs at the specific time.

5.1.1 Data Preparation

We prepare dynamic graphs representing the yeast metabolic pathways in combination with microarray data. As described in chapter 2, microarrays can be used in two ways: monitoring the change of gene expression levels over time or distinguishing patterns in two different states. Here, we use time-based microarray data to generate a dynamic graph, where each column of data represents the gene expression values at a particular time. The microarray
data used in our research observes periodic gene expression of *Saccharomyces cerevisiae* using microarray analysis [49]. The microarray data has 36 columns where each column represents one time slice. Their results show more than 50% of genes have three periodic cycles in the gene expression. We normalize each gene expression value of microarray data from 0 to 1, because we are focused on trends of the changes of gene expression values. Figure 5.1 (A) shows normalized gene expression values of three genes shown in the glycolysis pathway.

![Figure 5.1 (A) The oscillation curves of the changing gene expression values of three yeast genes: YNL071W, YER178W, and YBR221C. (B) An instance of the graph representation for a metabolic pathway.](image)

Here, we prepare 10 dynamic graphs, each of which contains 36 consecutive graphs representing one yeast metabolic pathway changing over time (36 time slices) corresponding to 36 columns in the microarray data. The 10 dynamic graphs represent 10 metabolic pathways: glycolysis (00010), TCA (00020), Pentose phosphate pathway (00030), Purine metabolism (00230), Pyrimidine metabolism (00240), Urea cycle (00220), Glutamate metabolism (00251), Arginine and proline metabolism (00330), Glycerolipid metabolism (00561) and Glycerophospholipid metabolism (00564), where each number denotes the identification number of the pathways in the KEGG data [25]. The first three pathways are involved in the carbohydrate metabolism, the second two pathways are involved in the nucleic acids metabolism,
where purine and pyrimidine are organic compounds that are used as building blocks of DNA and RNA, the next three pathways are involved in the amino acids metabolism and the last two pathways are involved in the lipid metabolism. First, we generate a static graph to represent each metabolic pathway from the KEGG PATHWAY database [25], where vertices represent compounds, genes, enzymes, relations and reactions, and edges represent relationships between vertices. Figure 5.1 (B) shows an example of the graph representation. “ECrel:Compound” represents a relation between two enzymes (gene products). One enzyme is produced by one or more genes, which is represented as edges “G_to_E”. “RN:Rxxxx” represents a reaction and “cpd:Cyyyyy” represents a chemical compound, where xxxxx and yyyyy represent the identification number in the KEGG database. Here, we assume only genes change over time based on gene expression values, and other molecules, like compounds, remain at the same amount.

We use a threshold $th$ to determine what level of the numeric gene expression values results in the presence or absence of a gene in the graph. At each time, we assume a gene, which has more than $th$ gene expression value, is shown in the graph. The threshold is specified to maximize structural change in a dynamic graph. Generally, a smaller threshold provides more structural changes with more noise. From the biologists’ opinion, the threshold should be specified for each gene and based on specific conditions. However, it is impossible to specify the threshold for every gene in a cell in combination with various conditions. In this research, we normalize gene expression values to be between 0 and 1, so that we specify the threshold as a constant rate to evenly affect every gene.

One particular point is our graph representation has enzyme vertices, which do not exist in the KEGG data. One enzyme needs one or more genes to synthesize. At a specific time, only
one gene can be expressed out of two genes, which are needed for one enzyme. Naturally, the enzyme is not synthesized at that time. We use enzyme vertices to represent this scheme. Only when all genes are expressed, is the enzyme vertex shown in the graph. At that time, the reaction, which is catalyzed by the enzyme, is also shown. In this way, we can observe the structure of the metabolic pathway at each time based on microarray gene expression.

5.1.2 Result: Temporal patterns

Table 1 shows the running time of Algorithm 1 on the ten dynamic graphs representing the ten metabolic pathways. Most cases are finished within one minute. As shown in figure 5.2 and table 1, larger graphs generally take longer running time. However, the largest graph (00230) does not take the longest time. 00251 takes the longest time in this experiment. Especially, 00251 takes more than two times longer than 00010 although it has a similar size with 00251. We assume the label diversity of data causes this result. Many redundant labels make the discovery process much slower, because there are likely to be more instances of each pattern and therefore more time is spent matching patterns to instances. In metabolic pathways, the number of unique vertex labels is varied for each pathway, because each pathway has different numbers of compounds and genes that are the major portion of unique vertex labels. Pathway 00010 has 23 compounds and 47 genes, and pathway 00251 has 19 compounds and 29 genes. Because 00010 has more unique vertex labels, the running time is faster than 00251. Pathway 00230 also has large numbers of unique vertex labels, where the number of compounds is 61 and number of genes is 81. The size of 00230 is larger than 00251. But the larger number of unique labels in 00230 provides enough search space to DiscoverCommonSubs to discover patterns, so that the running time is even faster than 00251. We will discuss this issue in more detail in chapter 6.
Table 5.1 Running time of Algorithm 1 on ten dynamic graphs. Pathway denotes the name of the pathway represented by the dynamic graph. Max. Size and Min. Size denote the maximum and minimum size of a graph in the dynamic graph. Total Size denotes $\sum \text{size}(G_i)$ for $G_i \in DG$. Time is in seconds

<table>
<thead>
<tr>
<th>Pathway</th>
<th>Max. Size</th>
<th>Min. Size</th>
<th>Total Size</th>
<th>Time (sec.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>00010</td>
<td>522</td>
<td>65</td>
<td>7738</td>
<td>69.86</td>
</tr>
<tr>
<td>00020</td>
<td>294</td>
<td>46</td>
<td>4667</td>
<td>9.44</td>
</tr>
<tr>
<td>00030</td>
<td>192</td>
<td>57</td>
<td>4069</td>
<td>3.82</td>
</tr>
<tr>
<td>00220</td>
<td>236</td>
<td>58</td>
<td>4147</td>
<td>4.58</td>
</tr>
<tr>
<td>00251</td>
<td>394</td>
<td>110</td>
<td>7928</td>
<td>172.88</td>
</tr>
<tr>
<td>00330</td>
<td>184</td>
<td>61</td>
<td>4277</td>
<td>4.65</td>
</tr>
<tr>
<td>00561</td>
<td>183</td>
<td>44</td>
<td>2425</td>
<td>3.38</td>
</tr>
<tr>
<td>00564</td>
<td>231</td>
<td>57</td>
<td>4937</td>
<td>4.96</td>
</tr>
<tr>
<td>00230</td>
<td>643</td>
<td>161</td>
<td>10259</td>
<td>54.06</td>
</tr>
<tr>
<td>00240</td>
<td>486</td>
<td>85</td>
<td>6040</td>
<td>18.03</td>
</tr>
</tbody>
</table>

Because the result of the microarray data [49] represents three periodic cycles of gene expression, we observe similar temporal patterns in the graph rewriting rules. Here, we are focused on graph rewriting rules involving enzyme-enzyme relations as well as genes. One or more genes produce an enzyme, and the enzyme can have a relation with one other enzyme. The relation vertex labeled as “ECrel:Compound” exists, only when there exist two enzyme vertices. Each enzyme vertex exists only when the linked genes exist (biologically, the linked genes produce the enzyme).
Figure 5.2 Running time of Algorithm 1 on the ten dynamic graphs ordered by size.

Figure 5.3 (A) shows a visualization of the periodic changes in a portion of the glycolysis pathway that includes only the three genes shown in figure 5.1 (A). The complete pathway is shown in figure 5.5 (Sub F). The points above the time axis represent the time points when the substructures including the specified genes or relation are removed. The points below the time axis represent the time points when the substructures including the specified genes or relation are added. The points on the axis represent the time when the relation appears. The result shows the temporal patterns in removal and addition rules as three cycles. Three genes are added and the relation is shown in the pathway. After several time intervals, one of three genes starts to be removed from the pathway and the relation disappears. Like the microarray research [49], we can notice the genes are added and removed three times periodically. In addition, we discover the removal and addition of some relations also show periodic cycles. Suppose there are two genes
and a relation between two genes. One gene is always shown in the pathway, and the other is shown three times periodically. The relation is also shown three times like the latter gene, because the relation is activated only when both genes are activated. Because most genes and proteins work together, the temporal patterns in the relations between the molecules are also important as well as the temporal patterns in the existence of genes and proteins.

Figure 5.3 (A) A visualization of time points when the substructure including each gene is removed from or added to graphs representing the glycolysis pathway at the experiment of threshold 0.6. The points above the time axis represent the time points when the substructures including the specified genes or relation are removed (Genes with (-)). The points below the time axis represent the time points when the substructures including the specified genes or relation are added (Genes with (+)). Relation points represent the time points when the enzyme-enzyme relations are shown in the pathway. (B) A visualization of time points when a particular substructure is removed from or added to graphs representing the glycolysis pathway at the experiment of threshold 0.6. Each substructure includes a relation, which is an enzyme-enzyme relation between two genes, where $ECrel(x,y)$ represents the relation, and $x, y$ represent the id of enzymes.

Figure 5.3 (B) shows a visualization of three periodic cycles of 10 relations in the glycolysis pathway. In this experiment, the dynamic graph with threshold 0.6 shows a maximum of 13 relations at each time slice. 10 out of the 13 relations clearly show periodic cycles three times.
Figure 5.4 Visualization of three periodic cycles in removals and additions of (A) Enzyme Relations in TCA cycle (00020) and (B) Pentose phosphate pathway at the experiment of threshold 0.5. The points marked as red circles above the time axis represent removals, and the points marked as blue rectangles represent additions.
Figure 5.4 (cont.) Visualization of three periodic cycles in removals and additions of Enzyme Relations in (C) Urea cycle (00020) and (D) Glutamate metabolism (00251) at the experiment of threshold 0.5. The points marked as red circles above the time axis represent removals, and the points marked as blue rectangles represent additions.
Figure 5.4 (cont.) Visualization of three periodic cycles in removals and additions of Enzyme Relations in (E) Arginine and proline metabolism (00330) and (F) Purine metabolism (00230) at the experiment of threshold 0.5. The points marked as red circles above the time axis represent removals, and the points marked as blue rectangles represent additions.
Figure 5.4 (cont.) Visualization of three periodic cycles in removals and additions of Enzyme Relations in (G) Pyrimidine metabolism (00240) at the experiment of threshold 0.5 and (H) Glycerolipid metabolism (00561) is performed at threshold 0.4. The points marked as red circles above the time axis represent removals, and the points marked as blue rectangles represent additions.
Figure 5.4 (cont.) Visualization of three periodic cycles in removals and additions of Enzyme Relations in (I) Glycerophospholipid metabolism at the experiment of threshold 0.5. The points marked as red circles above the time axis represent removals, and the points marked as blue rectangles represent additions.

Figure 5.4 shows the similar temporal patterns in the 9 other pathways, TCA cycle (A), Pentose phosphate pathway (B), Urea cycle (C), Glutamate metabolism (C), Arginine and proline metabolism (E), Purine metabolism (F), Pyrimidine metabolism (G), Glycerolipid metabolism (H), and Glycerophospholipid metabolism (I). The points (marked as circles) above the time axis represent the patterns of removals and the points (marked as the rectangles) below the time axis represent the patterns of additions. Each experiment is performed at threshold 0.5 except the Glycerolipid metabolism experiment. The Glycerolipid metabolism experiment is performed at threshold 0.4 to show more numbers of relations, because the static graph of the Glycerolipid metabolism has only 5 enzyme relations. We choose the user-specified threshold to
include a sufficient number of genes and enzyme-relations in the graph to show structural changes. If the threshold is chosen smaller, there may be noise data in the graph. But there are only a small number of molecules in the graph, if the threshold is too large.

Two time points at the same distance from the axis represent the removals and additions of the same subgraphs. The nine visualizations show the temporal patterns in the graph rewriting rules of the major metabolic pathways. Even though there are some time points that do not show clear cycles, all ten pathways show the three periodic cycles of enzyme-enzyme relations. We can conclude that the removals and additions of the subgraphs including genes and relations show the temporal patterns of three periodic cycles.

Our results show three periodic cycles of enzyme-relations and maplink-relations over ten major metabolic pathways. We can observe similar temporal patterns in the four major categories of pathways. These temporal patterns of relations describe periodic cycles in the behaviors of the yeast biosystem corresponding to the periodic cycles of the gene expression of the yeast. The major events and behaviors of the biosystems accord with the metabolic cycles [49].

The experiments show that algorithm 1 discovers graph rewriting rules from dynamic graphs representing the yeast metabolic pathways changing over time. These graph rewriting rules represent temporal patterns that describe how the structure of the metabolic pathways change over time by showing which elements change periodically. These temporal patterns and graph rewriting rules help us to understand the dynamic properties of metabolic pathways. The results show temporal patterns in structural changes of metabolic pathways.
5.1.3 Result: Structural Patterns

The other goal of this research is to show structural patterns in metabolic pathways as well as temporal patterns. Because an advantage of the graph representation is visualization, we can understand metabolic pathways better using structural analysis with temporal analysis. This section illustrates the use of the substructures discovered within the graph rewriting rules. Figure 5.5 shows structural changes of the dynamic graph representing the partial glycolysis pathway introduced in figure 5.1 (A). $G_i$ represents the graph at time $i$. This dynamic graph contains 36 time series of graphs starting with a single vertex graph in time 1 to no vertex in time 36. The blue edge with the boxed labels between two sequential graphs represents the graph transformation using removal (-) or addition (+) of one of the six substructures (Sub $A$ to $F$) shown on the right of figure 5.5. For example, graph $G_5$ is transformed to $G_6$ with removal of Sub $C$ and addition of Sub $B$. The red edges with the dot-boxed labels in the rules represent the connection edges as described previously. The connection edges describe how the discovered substructures connect to the original graph.
Figure 5.5 Structural changes of a dynamic graph representing the partial glycolysis pathway. $G_i$ denotes a graph at time $i$ for $1 \leq i \leq 36$. The blue arrows with boxed labels between two graphs, $G_x$ and $G_y$, represent the transformation from $G_x$ to $G_y$, by application of the rule in the label of the arrow. Sub $xP$ represents the substructure ($x = \{A, B, C, \ldots, F\}$ shown on the right) in each rule (removal and addition), where the red arrows with the dot boxed labels from the substructures represent the connection edges. For example, $G_1$ is transformed to $G_2$ by addition of Sub $A$, which is connected by a connection edge labeled “G_to_E”.

As described previously, we show the graph rewriting rules between two graphs as a formula. Here, we show two examples of graph rewriting rules $GR_{1,2}$ and $GR_{5,6}$ as follows,

$$GR_{1,2} = \left\{ \left( A_{Sub_A}, C_{Sub_A} \right) \right\}, C_{Sub_A} = \left\{ (s2, g2, G\_to\_E) \right\},$$

$$GR_{5,6} = \left\{ \left( A_{Sub_B}, \emptyset \right), \left( R_{Sub_C}, \emptyset \right) \right\},$$

where $A_{Sub_A}, A_{Sub_B}$, and $R_{Sub_C}$ denote the substructures (Sub $A$, $B$, and $C$) in figure 5.5. $C_{Sub_A}$ denotes the connection edges, where the connection edge with a label “G_to_E” links Sub $A$ to a gene YER178W in $G_1$ so that an enzyme is activated by two genes, YBR221C and YER178W, and a relation is created with the other enzyme that is activated by a gene, YNL071W. But the connection edges in $GR_{5,6}$ are all $\emptyset$, because there is no connection edge between the substructures ($A_{Sub_B}$ and $R_{Sub_C}$) and the original graphs ($G_5$ and $G_6$) respectively.
Figure 5.6 A visualization of discovered substructure of a removal rule in a dynamic graph representing the glycolysis pathway in our output (left) and on the KEGG glycolysis pathway map (right). Labels marked by “[-]” represent the removal rules and labels marked by “()” represent the connection edges (left). Red rectangles represent two genes and blue circles represent three compounds in the removal rule (right). When YKL060C is removed, two enzyme-relations between two genes are also removed. Two reactions R01015 and R01070, involved with the three compounds, are also removed.

Figure 5.6 shows our visualization results of a removal rule. Figure 5.6 (A) shows a removal rule in our output, and figure 5.6 (B) shows the same rule marked on the KEGG pathway map. The labels marked by “[-]” represent the labeled vertices and edges belonging to the substructures of removal rules. The labels are marked by “[+]” in the case of addition rules. Connection edges between the discovered substructures and original graphs are marked by “()”. The removal of a gene YKL060C causes the removal of two enzyme-relations with one other gene YDR050C and a reaction R01070, which is catalyzed by an enzyme produced by YKL060C. The graph also loses several connection edges between the removal structures and original graph. Our approach helps us visualize removal or addition rules on the original graph with the connection edges. The results show how the substructures in graph rewriting rules are structurally connected to the original graphs and how the graphs change after removal or addition rules are applied.
In addition to the change of one element, our results show how the changes are related to other elements (i.e., which elements are removed or added at the same time) as shown in the discovered subgraphs and how the subgraphs are linked to the original graphs. Our results show patterns in the structural changes, not merely changes of amount. It allows us to better understand the structural properties as the pathways change over time.

In summary, we evaluated algorithm 1 in the experiments with 10 dynamic graphs each containing 36 graphs representing the yeast metabolic pathways in combination with the microarray data of yeast. 35 sets of graph rewriting rules for removals and additions are discovered during 35 time intervals. Temporal patterns in the graph rewriting rules show a number of substructures are removed and added periodically, showing three cycles. The graph rewriting rules and our visualization results describe how the discovered substructures are connected to the original graph and how the structures of graphs change over time. These temporal patterns and graph rewriting rules help us to understand temporal properties as well as structural properties of biological networks. Some discovered temporal and structural patterns in a specific disease can show us how these patterns are different from normal patterns and help us investigate the disease and develop new treatments.

5.2 Results using Transformation Rules

This section shows how transformation rules discovered by algorithm 2 describe the dynamics of biological networks. To evaluate our approach, we prepare three data sets: artificial data, mathematical modeling and microarray data. In artificial data, we use a real biological network, but we remove and add some subgraph randomly to generate the dynamic graphs. For
real world data, we use mathematical modeling and microarray data to generate dynamic graphs. This section has four subsections as follows. The first section describes evaluation metrics to evaluate our transformation rules and compare with other approaches. The second section describes an experiment with artificial data. The third section describes an experiment with mathematical modeling data, and the comparison with the frequent subgraph approach.

5.2.1 Application to Artificial Data

The artificial biological network represents the Notch signaling pathway in humans generated from the KEGG data. We remove and add some subgraphs repeatedly with this biological network to generate the artificial dynamic graphs. We process four different removals and additions (A to D), and generate four artificial dynamic graphs, which contain 20 time-slice graphs in each dynamic graph. The Notch signaling pathway contains 46 genes in our experiments, and we assume that each gene can be shown at most once at each time slice. Figure 5.7 shows the Notch pathway graphic map [25] and a portion of the graph representation for the pathway.

Figure 5.7 KEGG Notch signaling pathway map and a portion of a graph that represents the Notch signaling pathway.
The original dynamic graph contains all genes for every 20 graphs. We remove some genes at a specific time, and observe how the graph structures change over time. Because of the biological semantics, the removal (addition) of even one gene can cause the removal (addition) of one or more larger subgraphs.

Table 5.2 Coverage of the best subgraphs in Artificial Data. DG $x$ denotes an artificial biological network for $x = \{A, B, C, D\}$. The number in each iteration denotes $y(z)$, where $y$ denotes the number of the discovered subgraphs and $z$ denotes the coverage by the best subgraph discovered at the iteration. Total denotes the total coverage from 1 to 3 iterations.

<table>
<thead>
<tr>
<th>Data</th>
<th>Itr. 1</th>
<th>Itr. 2</th>
<th>Itr. 3</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>DG A</td>
<td>19 (1.0)</td>
<td>NA</td>
<td>NA</td>
<td>1.0</td>
</tr>
<tr>
<td>DG B</td>
<td>9 (1.0)</td>
<td>NA</td>
<td>NA</td>
<td>1.0</td>
</tr>
<tr>
<td>DG C</td>
<td>8 (0.16)</td>
<td>4 (0.032)</td>
<td>10 (0.05)</td>
<td>0.242</td>
</tr>
<tr>
<td>DG D</td>
<td>6 (0.15)</td>
<td>5 (0.125)</td>
<td>2 (0.045)</td>
<td>0.320</td>
</tr>
</tbody>
</table>

In dynamic graph (DG) A, we remove one gene (hsa:9541) at every other time slice ($t = 1, 3, 5, \cdots, 19$). In DG B, we remove one gene (hsa:3516) at every fourth time slice ($t = 1, 5, 8, \cdots, 18$). The next DGs, C and D, have a more complex construction. The two matrices below represent the existence of genes in DG C and D respectively. Each of the 20 columns after the colon represent a time slice, where 0 denotes non-existence of the corresponding gene and 1 denotes the existence. The numbers in the leftmost column denote the identification number of the genes like hsa:3066. For example, the first line in DG C represents has:3066 gene appears at time 4, 5, 6, 10, 11, 12, 16, 17, and 18.

DG C

| 3066 : 00011 10001 11000 11100 |
| 3516 : 01101 10110 11011 01101 |
| 4242 : 00111 11011 11101 11101 |
| 4851 : 01101 10110 11011 01101 |
| 4853 : 00111 00111 00111 00111 |

DG D

| 1387: 00111 00111 00111 00111 |
| 9794: 00111 00111 11001 00111 |

| 4851 : 01101 10110 11011 01101 |
| 4853 : 00111 00111 00111 00111 |
Figure 5.8 (A) The best subgraph discovered in the graph rewriting rules of DG B, which is discovered as 9 instances in the 9 examples (4 removals and 5 additions). (B) Visualization of transformation rules in the learned graph rewriting rules of DG B. The above boxes denote the removals at the specified time. The below eclipses denote the additions at the specified time. The numbers on the arrow denote the temporal distances between two events like the removals and additions. (C) The repeated transformation rule of the temporal patterns in (B), where Sub 1 denotes an instance of (A).

Table 5.2 shows the coverage of the best subgraph discovered at each iteration of algorithm 2. The first two DGs, A and B, can be covered by one subgraph, because the removals and additions are simple and regular. Figure 5.8 (A) shows the best subgraph discovered in DG B experiments. The instances of the best subgraphs are discovered in the 9 examples (4 removals and 5 additions). The results of algorithm 2 help us to visualize description rules in the learned graph rewriting rules as shown in figure 5.8 (B). The above boxes denote the removals at the specified time. The below eclipses denote the additions at the specified time. The numbers on the arrow denote the distance of the time intervals between two events like the removals and additions. The first addition occurs at time 1, and the first removal occurs after 3 time intervals. From the first addition at time 1 to the last addition at time 17, every removal is repeated after 3 time intervals from the last addition, and every addition is repeated after 1 time interval from the last removal. The repeated description rule can be represented as shown in figure 5.8 (C) and can be expressed as $TR = Sub_1(+3, -1)$. 

55
Figure 5.9 (A) The best subgraph (Sub 2) at iteration 2 includes the best subgraph (Sub 1) discovered in DG D. Sub 1s are discovered at times 2, 5, 7, 10, 15, and 17, and Sub 2s are discovered at times 2, 5, 7, 15 and 17. (B) The visualization of temporal patterns in the learned graph rewriting rules including the Sub 1 and Sub 2 in (A).

As described in section 4.2, our discovered best subgraphs can include a previous best subgraph after several iterations. Figure 5.9 (A) shows an example of a hierarchical cluster consisting of two subgraphs. The first subgraph (Sub 1) is discovered at times 5, 10, 15 as removals and at times 3, 8, 18 as additions. The second subgraph (Sub 2) includes Sub 1 and is discovered at times 5, 15 as removals, and at times 3, 8, 18 as additions.

Unlike DG B, the repeated transformation rule cannot be easily represented in the case of DG D. As shown in figure 5.9, visualization (B) does not seem understandable. Sub 1 is removed at time 10, and removed again at time 15 without any addition. Per our assumption, each gene can be shown only once at each time. Sub 1 cannot be removed consecutively without any addition. The solution to the riddle is that the gene, hsa:1387, is added in a different subgraph at time 12. Sub 2 has similar cases. The gene, hsa:9794, is removed at time 12 and added at time 14 without any relation to Sub 1. If we look at the above construction matrix of DG D, we can easily understand what causes this result. Both genes, hsa:1387 and hsa:9794 are added just after
time 2, 7 and 17 appearing in the graphs at time 3, 8, 18. But they are separately added at time 12 (hsa:1387) and 14 (hsa:9794), so the “group” vertex and “component” edges in figure 5.9 (A) are added with hsa:9794 at time 14 following our assumption that a “group” vertex exists when all its component genes exist. Because our approach is focused on the structural changes of the biological networks, not merely the change of one element, such cases present a challenge to discovering consistent transformation rules. If our approach focuses on one gene, hsa:1387, the addition at time 12 can be detected. But our approach looks for the best compressed pattern to show general structural changes, so that it returns the discovered subgraphs without considering this inconsistent pattern. To overcome this challenge, our approach should consider relationships among instances, i.e., two discovered subgraphs at two different times, where one is removed before the other is added.

Figure 5.10 (A) The best subgraph discovered 8 times at the first iteration and (B) the third subgraph discovered 10 times at the third iteration.

Here, we discuss the advantage of the compression-based subgraph discovery. In DG C, the first best subgraphs are discovered 8 times. Actually, the third best subgraphs are discovered 10 times. Because the Value (1.56) of the first subgraph is larger than the Value (1.07) of the third subgraph, our approach prefers the first subgraph. The larger Value means the subgraph can
better compress the graph. The size of the first subgraph is 51, and the size of the third subgraph is 5. Also, the coverage (0.16) of the first subgraph is larger than the coverage (0.05) of the third subgraph. The reason for the low coverage for DG C and B is that the structural changes are distributed over several graph rewriting rules, but the coverage metric considers only the portion of the dynamic graph covered by the subgraph comprising the individual transformation rule. We will discuss this issue in the section 5.2.2. For this reason, the compression-based approach can be more useful than frequent graph mining in the analysis of dynamic graphs. Figure 5.10 shows the first subgraph (A) and third subgraph (B).

5.2.2 Application to Mathematical Modeling Data: Prediction Test

We also apply our approach to a dynamic graph based on the mathematical modeling data. The dynamic graph represents the cell cycle signaling pathway [50]. The cell cycle signaling network in our experiment contains 14 molecules (genes and compounds) and 11 reactions between molecules. We produce the graph based on the KEGG database. If there is a difference between the KEGG representation and mathematical modeling data, e.g., molecule name, we follow the mathematical modeling data. Table 5.3 and 5.4 show the mathematical modeling data of the cell cycle signaling network. Our simulation is performed using CellDesigner [51] and Systems Biology Workbench (SBW) [52]. 14 molecules are grouped into 5 complex groups. Each group has several formulae and initial parameters to simulate the regulation of the complex. CellDesigner is a diagram editor for drawing biological networks, and SBW is an open source framework containing several software applications for systems biology research. One of the software components is a simulator of reaction networks that is used for simulating our mathematical models. CellDesigner can show a diagram of biomodels
(mathematical modeling data) that are downloaded from [53], and SBW is installed as a component in the cell designer for simulating the biomodels. The BioModels Database [53] is a repository of mathematical models of biological networks that can be stored, searched and retrieved by users. The biomodels are saved in XML format.

Table 5.3 Variables used in the mathematical modeling data of the cell cycle signaling network. The first column represents a full name of each molecule, the second column represents the variables used in the formulae in table 5.4 [50].

<table>
<thead>
<tr>
<th>Name</th>
<th>Variables</th>
</tr>
</thead>
<tbody>
<tr>
<td>Free cyclin</td>
<td>( y )</td>
</tr>
<tr>
<td>Inactive Cyclin:CDK complex</td>
<td>( x_1 )</td>
</tr>
<tr>
<td>Active Cyclin:CDK complex</td>
<td>( x )</td>
</tr>
<tr>
<td>Total CDK</td>
<td>( c_0 )</td>
</tr>
<tr>
<td>Free CDK</td>
<td>( c )</td>
</tr>
<tr>
<td>Unphosphorylated CDC25</td>
<td>( z_0 )</td>
</tr>
<tr>
<td>One-site phosphorylated CDC25</td>
<td>( z_1 )</td>
</tr>
<tr>
<td>Two-site phosphorylated CDC25</td>
<td>( z_2 )</td>
</tr>
<tr>
<td>Unphosphorylated wee1</td>
<td>( w_0 )</td>
</tr>
<tr>
<td>Phosphorylated wee1</td>
<td>( w_1 )</td>
</tr>
<tr>
<td>Active CK</td>
<td>( u )</td>
</tr>
<tr>
<td>Free CKI</td>
<td>( i )</td>
</tr>
<tr>
<td>Cyclin:CDK:CKI complex with CKI unphosphorylated</td>
<td>( i_x )</td>
</tr>
<tr>
<td>Cyclin:CDK:CKI complex with CKI phosphorylated</td>
<td>( i_{xp} )</td>
</tr>
</tbody>
</table>
Table 5.4 Differential equations and parameters for mathematical modeling data of the cell cycle signaling network [50]. First five groups (rows 2 to 6) represent formulae (second column) used for each complex regulation that is represented at the first column. The last row represents initial parameters for the formulae.

<table>
<thead>
<tr>
<th>Regulation Name</th>
<th>Formula</th>
</tr>
</thead>
</table>
| Cyclin and CDK regulation | \[ y = k_1 + k_4 x_1 - k_3 y c - (k_2 + k_{2u} u)y \]  
  \[ x_1 = k_3 y c + (k_6 + g(w))x - k_4 x_1 - (k_5 + f(z))x_1 \]  
  \[ x = (k_5 + f(z))x_1 - (k_6 + g(w))x - (k_7 + k_{7u} u)x - k_{14} x i + k_{15} i_x + (k_{16} + k_{16u} u)i_{xp}, \]  
  where \( c = (c_0 - x - x_i - i_x - i_{xp})/c_0. \] |
| CDC25 regulation      | \[ z_0 = k_8 + k_7 z_1 - k_5 z_0 - k_9 z_0 \]  
  \[ z_1 = k_7^+ z_0 + k_7^- z_2 - k_5^- z_1 - k_5^+ z_1 - k_9 z_2, \]  
  where \( k_7^+ = b_x + c_x x \) is the rate constant for CDC25 phosphorylation, \( k_7^- = a_x \) is for dephosphorylation, \( b_x \) is the rate constant for CDC25 phosphorylation not catalyzed by active Cyclin:CDK, and \( c_x x \) is for phosphorylation catalyzed by active Cyclin:CDK. |
| wee1 regulation       | \[ w_0 = k_{10} + k_{w} w_1 - k_{w}^- w_0 - k_{11} w_0 \]  
  \[ w_1 = k_{w}^+ w_0 - k_{w}^- w_1 - k_{11} w_1, \]  
  where \( k_{w}^+ = b_w + c_w x \) is the rate constant for wee1 phosphorylation, \( k_{w}^- = a_w \) is for dephosphorylation, \( b_w \) is the rate constant for wee1 phosphorylation not catalyzed by active Cyclin:CDK, and \( c_w x \) is for phosphorylation catalyzed by active Cyclin:CDK. |
| SKP2 or APC regulation | \[ u = (h(x) - u)/\tau, \text{ where } h(x) = x^2/(a^2 + x^2). \] |
| CKI regulation        | \[ i = k_{12} - k_{13} i - k_{14} x_i + k_{15} i_x \]  
  \[ i_x = k_{14} x_i - k_{15} i_x + k_{1}^- i_{xp} - k_{1}^+ i_x \]  
  \[ i_{xp} = k_{1}^+ i_x - k_{1}^- i_{xp} - (k_{16} + k_{16u} u)i_{xp}, \]  
  where \( k_{1}^+ = b_i + c_i x \) is the rate constant for CKI phosphorylation, \( k_{1}^- = a_i \) is for dephosphorylation, \( b_i \) is the rate constant for CKI phosphorylation not catalyzed by active Cyclin:CDK, and \( c_i x \) is for phosphorylation catalyzed by active Cyclin:CDK. |
| Default parameters    | \( k_1 = 300, k_2 = 5, k_3 = k_4 = 30, k_5 = 0.1, k_6 = 1, k_7 = 10, \)  
  \( k_9 = 100, k_9 = 1, k_{10} = 10, k_{11} = 1, k_{12} = 0, k_{13} = k_{14} = k_{15} = 1, k_{16} = 2, k_{2u} = 50, k_{7u} = 0, k_{16u} = 25, c_0 = 200, a = 4, \tau = 25, \)  
  \( a_x = a_w = a_i = 10, b_x = b_w = b_i = 0.1, \) and \( c_x = c_w = c_i = 1. \) |
We normalize the concentrations of 14 molecules from 0 to 1, because we are focused on trends in the changes, and the concentrations of different molecules vary significantly. When we normalize the data, there are four molecules that have constant values over time: $c_0$, $i$, $i_x$, and $i_{xp}$. We set these variables to 1 after the normalization so that these molecules always appear in the graph with any threshold.

We use a threshold $th$ to activate each compound or gene. At each time, a compound or gene, which has more than $th$ amount, is shown in the graph. In other words, the biological network contains a portion of the 14 molecules with related reactions at each time. Here, we choose $th = 0.2$ so that every molecule can be shown more than once in the graphs. We use the KEGG database for our graph representation. Because the simulation is performed for 700 seconds and we take a snapshot every 10 seconds, we have 51 time slices ($t=1$ to 51) of data for training and the following 20 time slices for testing.

Figure 5.11 (A) The best subgraph ($Sub_1$) discovered in $TR_1$. Visualization of the graph rewriting rules including the subgraph in (A) in the training data (B) and the testing data (C).

Figure 5.11 (A) shows the best subgraph ($Sub_1$) in $TR_1$ discovered at 16 time slices as visualized in figure 5.11 (B). Our approach discovers $TR_1 = Sub_1(+5, -2)$, where the temporal distances, +5 and -2, are chosen because they are most frequent. Even though this rule is off in three places figure 5.11 (A), our $TR_1$ can describe the general structural changes. The vertices
containing “Rct” in the labels denote reactions like “Rct:+p_CDC”. The vertices without “Rct”
denote molecules (genes or proteins). The three edges, “Rct_to_R”, “Rct_to_P” and
“Rct_to_M”, denote how the molecules are related to the reactions as reactant, product and
multiplier respectively. These results are biologically significant, because they describe the
repeated structural changes in the networks. Qu et al. [50] describe periodic changes of
molecules (i.e., amount of molecules). Specifically, they mention several molecules such as
Active Cyclin:CDK and Free Cyclin that show periodic increase and decrease, where the cycles
correspond to the change of the cell size. Figure 5.11 (B) shows the subgraph including Active
Cyclin:CDK, that is added and removed periodically corresponding to periodic changes in the
amount of the molecule. In addition, figure 5.11 (A) shows how the changes are related to other
elements (i.e., which elements are removed or added at the same time) as shown in the
discovered subgraphs, and figure 5.12 shows how the subgraphs are linked to the original graphs.
The Active Cyclin:CDK complex and phosphorelated Wee are removed at time 3, where they are
originally connected to unphosphorelated CDC25 and Active SKP2 by the connection edges.
The representation of our graph looks a little different from the KEGG graphic map, because
they represent only molecule names, no matter whether the molecules are phosphorelated or
active. Our results show patterns in the structural changes, not merely changes of amount.
Figure 5.12 (A) Visualization of a portion of $Sub_1$. Labels marked by “[]” represent the removal rules and labels marked by “()” represent the connection edges. (B) A portion of the KEGG graphic map including the marked $Sub_1$. Red circles (A) and rectangles (B) represent removed Active Cyclin:CDK and phosphorlated Wee. Blue circles (A) and rectangles (B) represent SKP2 and CDC25 that are connected to the removal rule by the connection edges.

The coverage is calculated as 0.127. The coverage looks low, because it represents only the portion of structural changes covered by $TR_1$. The sum of coverage at all iterations is closer to 1, where the coverage of $TR_2$ is 0.018 and the coverage of $TR_3$ is 0.012. There is also another consideration for the coverage metric. Except for 16 of 100 graph rewriting rules, the size of the graph rewriting rules is less than 7. The $TR_1$, $TR_2$ and $TR_3$ rules above are included in the 16 graph rewriting rules, where their sizes are greater than 31. Our coverage metric treats equally
these two sizes of graph rewriting rule, because the *coverage* is the sum of the ratio of the *BestSub* to each graph rewriting rule. We can calculate the *coverage* as \( \frac{\sum \text{BestSubs}}{\sum \text{Rs} + \sum \text{As}} \) for this problem. In that case, we need to ignore the ratio of graph rewriting rules. For example, if a graph rewriting rule with the size 1 is discovered from a graph with size 1 to the next graph with size 1, the change covers 100% between the two graphs. If the same size (1) of graph rewriting rule is discovered from a graph (size 100) to the next graph (size 100), the change covers only 1%.

Based on \( TR_1 \) this rule, we predict the future change as shown in figure 5.11 (C). The rule predicts 6 graph changes in the testing range from time 52 to 71. The temporal distance and graph rewrites denoted by the bold fonts in figure 5.11 (C) represent correct predictions with the testing data. Five of the six predictions match the rewrites observed in the testing data. The 6th prediction, that \( Sub_1 \) is removed at time slice 70 is not entirely correct, as the subgraph removed at time slice 70 is not an isomorphic match to \( Sub_1 \): \( d(Sub_1, R_{70}) \) is computed as 0.833. Thus, the *prediction* value of \( TR_1 \) is 0.972.

Next, we perform a more extensive prediction experiment. Because our research is focused on patterns in graph rewriting rules (i.e., patterns in structural changes), we can predict which graph rewriting rules appear (i.e., which structural changes occur). To evaluate prediction ability of the learned transformation rules, we perform ten prediction experiments using the above modeling data.

We modify some initial parameters shown in table 5.5 in the model to generate different dynamic graphs. The parameters are defined by biologists based on experiments to simulate a general cell environment. We modify 10 parameters from the default values to different values to
simulate the mathematical model in the different environments. The modified parameters and values are shown in table 5.5. Like the above experiment, we use 51 time slice graphs as training and 20 time slice graphs as testing. Table 5.5 shows the results. The modified mathematical model $M_1$ shows the transformation rule $Sub_1(+7,-1)$ that describes $Sub_1$ is added after 7 time units from the last removal and is removed after 1 time unit from the last addition. For example, the above $Sub_1$ is added at time 11 (during the time from 10 to 11), and is removed at time 12 (during the time from 11 to 12).

Table 5.5 Results of the prediction experiments with the modified mathematical model. Name denotes the name of the case. Variable denotes the name of the modified parameter. Mod. denotes the modification (X/Y), where X denotes the new value and Y denotes the default value. Uppercase “Size” denotes the size of each dynamic graph in training. TR denotes the learned transformation rule. Lowercase “size” denotes the size of the subgraph ($Sub_1$) in the transformation rule. Coverage denotes the Coverage metric of the learned rule, and Prediction denotes the Prediction metric of the learned rule.

<table>
<thead>
<tr>
<th>Name</th>
<th>Variable</th>
<th>Mod.</th>
<th>Size</th>
<th>TR</th>
<th>size</th>
<th>Coverage</th>
<th>Prediction</th>
</tr>
</thead>
<tbody>
<tr>
<td>$M_1$</td>
<td>k1</td>
<td>200/300</td>
<td>645</td>
<td>$Sub_1(+7,-1)$</td>
<td>27</td>
<td>0.115</td>
<td>1.0</td>
</tr>
<tr>
<td>$M_2$</td>
<td>k2</td>
<td>3/5</td>
<td>1541</td>
<td>$Sub_1(+4,-2)$</td>
<td>30</td>
<td>0.153</td>
<td>0.962</td>
</tr>
<tr>
<td>$M_3$</td>
<td>k4</td>
<td>50/30</td>
<td>835</td>
<td>$Sub_1(+12,-1)$</td>
<td>27</td>
<td>0.051</td>
<td>1.0</td>
</tr>
<tr>
<td>$M_4$</td>
<td>k5</td>
<td>0.2/0.1</td>
<td>1530</td>
<td>$Sub_1(+4,-2)$</td>
<td>25</td>
<td>0.155</td>
<td>1.0</td>
</tr>
<tr>
<td>$M_5$</td>
<td>k7</td>
<td>6/10</td>
<td>1880</td>
<td>$Sub_1(+5,-5)$</td>
<td>28</td>
<td>0.084</td>
<td>1.0</td>
</tr>
<tr>
<td>$M_6$</td>
<td>k8</td>
<td>60/100</td>
<td>1007</td>
<td>$Sub_1(+8,-1)$</td>
<td>27</td>
<td>0.080</td>
<td>0.864</td>
</tr>
<tr>
<td>$M_7$</td>
<td>k10</td>
<td>20/10</td>
<td>1741</td>
<td>$Sub_1(+4,-3)$</td>
<td>27</td>
<td>0.119</td>
<td>0.852</td>
</tr>
<tr>
<td>$M_8$</td>
<td>k11</td>
<td>0.5/1</td>
<td>1003</td>
<td>$Sub_1(+9,-1)$</td>
<td>27</td>
<td>0.066</td>
<td>0.944</td>
</tr>
<tr>
<td>$M_9$</td>
<td>k2u</td>
<td>300/50</td>
<td>886</td>
<td>$Sub_1(+18,-1)$</td>
<td>27</td>
<td>0.033</td>
<td>1.0</td>
</tr>
<tr>
<td>$M_{10}$</td>
<td>tau</td>
<td>15/25</td>
<td>1402</td>
<td>$Sub_1(+3,-1)$</td>
<td>27</td>
<td>0.185</td>
<td>1.0</td>
</tr>
<tr>
<td>Average</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0.1041</td>
<td>0.962</td>
</tr>
</tbody>
</table>

As shown in table 5.5, the average of the predictions is larger than 0.9, indicating that our approach is able to learn accurate rules across the different conditions yielding different dynamic graphs. In the case of $M_1$, $M_3$, $M_5$, $M_6$, and $M_8$, they show relatively small coverage, because
some elements in the best subgraph are removed (or added) separately. This issue is discussed further in section 5.2.1. We can also observe a correlation between the period of the regularity and the coverage. In case of $M_3$ and $M_9$, they have small numbers of cycles. $M_3$ and $M_9$ contain the entire sequences of discovered subgraphs in the transformation rule, but the cycles of the oscillations in the two models are few, 3 cycles for $M_3$ and and 2 cycles for $M_9$. This is the same issue for the lower coverage that we discussed in the previous section. Because the two models contain fewer cycles, the bestSubs cover only a small number of the graph rewriting rules. As described, the small number of graph rewriting rules that include the bestSub causes smaller coverage. In most cases, the oscillation shows more than 5 cycles (i.e., figure 5.11). The results show that our algorithm can predict the future structural changes from the learned transformation rules of the graph rewriting rules that represent the structural changes of dynamic graphs.

5.2.3 Application to the Mathematical Modeling Data: Comparison

As described previously, we can use the frequency-based graph mining approach for our DiscoverCommonSub. Here, we compare the compression-based approach to the frequency-based approach when finding common subgraphs in the rewriting rules in Step 2 of the algorithm. We prepare eight dynamic networks representing biological networks that structurally change over time. The eight biological networks are generated from the mathematical modeling data in the literature [30; 31; 50; 54; 55; 56; 57; 58], which are downloaded from [53].

We generate dynamic graphs from the KEGG database in combination with the mathematical modeling data in the same way as shown in the previous section. As similar with the last experiment, we follow the way of the mathematical modeling if the representation is different between the KEGG data and mathematical modeling data. We use the user-defined...
threshold $th$ to activate a molecule in a graph at each time slice. We use $th = 0.30$ for the models 42 and 110, $th = 0.35$ for the models 90 and 168 and $th = 0.40$ for other models. We determine the thresholds for each dynamic graph to show dramatic changes over time. Each mathematical model has a different time frame, but we still extract 71 time slices from each model. Then, we have 51 time slices ($t=1$ to 51) of data for training and the following 20 time slices for testing.

We apply five different methods to compare two graph-based data mining approaches: compression-based and frequent subgraph mining approaches. We use MDL and size-based compression methods for the compression-based approach, and use three methods for the frequent subgraph mining approach: size-based compression, largest most frequent, and largest frequent patterns. The size-based compression is computed as $\max_i(size(Sub_i) \cdot num(Sub_i))$, where $num(Sub_i)$ denotes the number of instances of the subgraph $Sub_i$. The largest most frequent pattern denotes the largest of the most frequent subgraphs, and the largest subgraphs denote the largest subgraphs in all discovered subgraphs that meet the minimum support. The minimum support for the subgraphs is 4.0%, because we desire the subgraphs to appear more than two times in each removal and addition rule. Our approach first learns graph rewriting rules (100 total removals and additions) in the dynamic graph. Then, algorithm 2 discovers the best subgraphs in the learned graph-rewriting rules as transformation rules using the above five methods. We will compare these approaches based on two metrics: coverage and prediction. In addition to this quantitative assessment, we also visualize the transformation rules to better understand the dynamic graph.

Table 5.6 shows the results of our experiments. In case that two or more methods have the same coverage and prediction, they discover the same subgraph. As shown in Table 5.6, the
compression-based approach is comparable to the frequent subgraph mining approach in terms of coverage and prediction except two cases. In all cases, prediction is greater than 70%. However, there are several challenges for successful prediction. In most cases, higher-valued subgraphs discovered using the compression-based approach can show higher coverage in the graph rewriting rules and transformation rules. Only two cases show the frequency-based approach outperform the compression-based approach, because these two models show regularity in terms of only small subgraphs, not relatively larger ones, i.e., after one subgraph is added (or removed), portions of the subgraph are removed (or added) in other ways. We discuss the same issue as one weakness of our approach in section 5.2.1.

Table 5.6 Coverage of the best subgraphs in eight dynamic graphs, and prediction in the testing data. “No” denotes the identification number of the mathematical model in [53] for each dynamic graph. Comp.-MDL and Comp.-Size denote the MDL and size-based compression methods. Freq.-Size, Freq.-MF and Freq.-Largest denote the size-based compression, the largest subgraph in the most frequent subgraphs, and the largest subgraph in the frequent subgraph mining approach. Cover. and Pre. denote coverage and prediction. The best coverage values are in bold. The last row represents the average for each column.

<table>
<thead>
<tr>
<th>No</th>
<th>Comp.-MDL</th>
<th>Comp.-Size</th>
<th>Freq.-Size</th>
<th>Freq.-MF</th>
<th>Freq.-Largest</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>0.025</td>
<td>1.0</td>
<td>0.050</td>
<td>1.0</td>
<td>0.050</td>
</tr>
<tr>
<td>42</td>
<td>0.073</td>
<td>1.0</td>
<td>0.099</td>
<td>1.0</td>
<td>0.099</td>
</tr>
<tr>
<td>90</td>
<td>0.043</td>
<td>0.90</td>
<td>0.042</td>
<td>1.0</td>
<td>0.043</td>
</tr>
<tr>
<td>99</td>
<td>0.127</td>
<td>0.74</td>
<td>0.127</td>
<td>0.74</td>
<td>0.127</td>
</tr>
<tr>
<td>110</td>
<td>0.096</td>
<td>1.0</td>
<td>0.096</td>
<td>1.0</td>
<td>0.096</td>
</tr>
<tr>
<td>168</td>
<td>0.051</td>
<td>0.76</td>
<td>0.051</td>
<td>0.76</td>
<td>0.051</td>
</tr>
<tr>
<td>170</td>
<td>0.040</td>
<td>1.0</td>
<td>0.054</td>
<td>0.78</td>
<td>0.040</td>
</tr>
<tr>
<td>171</td>
<td>0.073</td>
<td>1.0</td>
<td>0.073</td>
<td>1.0</td>
<td>0.073</td>
</tr>
<tr>
<td>Avg.</td>
<td>0.066</td>
<td>0.93</td>
<td>0.074</td>
<td>0.91</td>
<td>0.072</td>
</tr>
</tbody>
</table>
Figure 5.13 (A) An instance of the best subgraph discovered in the experiment with model 171. (B) Visualization of the graph rewriting rules including the subgraph (A). (C) The transformation rule of the graph rewriting rules in (B). (D) Predicted graph rewriting rules from (C).

Figure 5.13 (A) shows an instance of the best subgraph discovered in the experiment with model 171. The instances of the subgraph are discovered at time 11, 23, 35 and 47 as removals, and 5, 17, 29 and 41 as additions as shown in figure 5.13 (B). Like the previous section, the above rectangles in the figure represent the removals at the denoted time slice and the below eclipses represent the additions at the denoted time slice. The temporal distance from the removal to addition is 6, and from the addition to removal is 6 as shown in figure 5.13 (B). From the discovered subgraphs in figure 5.13 (A), we can learn the transformation rule $Sub_1(+6, -6)$ as shown in figure 5.13 (C). From the learned description rule, we can predict the future changes of the graph. As noticed, our research is focused on the prediction of the future structural changes, not the structure of the graph at the specific time. We predict three graph rewrites, i.e., three structural changes, at time 53, 59, and 65. As we predict, the instances of the subgraph in figure 5.13 (A) are added at time 53 and 65, and removed at time 59 (figure 5.13 (D)). We might think there should be one more removal at time 71. But the removal at time 71 represents the absence of the subgraph in the graph at time 72, but which does not exist in the testing data.
These results are biologically significant, because they describe the repeated structural changes in the biological networks. The researches related to these mathematical models [30; 31; 50; 54; 56; 55; 57; 58] describe that there are periodic changes of molecules (i.e., concentrations) to perform the metabolism or other processes in a cell. In addition, our results show patterns in the structural changes, but not merely changes of amount. The mathematical modeling [58] shows the trends of changing amounts of two proteins (PER and TIM) and related molecules, where they show periodic increase and decrease. Figure 5.13 (B) shows the graph rewriting rules including PER protein and related molecules that correspond to periodic changes in the amount of the molecule. In addition to the change of one element, our results show how the changes are related to other elements (i.e., which elements are removed or added at the same time) as shown in the discovered subgraphs (figure 5.13 (A)), and how the subgraphs are linked to the original graphs as shown in figure 5.14. Our results show patterns in the structural changes, not merely changes of amount.

Figure 5.14 (A) Visualization of a portion of $Sub_1$. Labels marked by “-[]” represent the removal rules and labels marked by “()” represent the connection edges. (B) A portion of the KEGG graphic map including the marked $Sub_1$. Red circles (A) and rectangles (B) represent removed PER molecules. Blue circles (A) and rectangles (B) represent the TIM molecules that are connected to the removal rule by the connection edges.
5.3 Summary

Dynamic network analysis is important and necessary not only for biological network analysis, but also for many other domains, such as social networks, web mining and so on. There are also many challenges to overcome.

In this chapter, we show our results in two ways: graph rewriting rules and transformation rules. Graph rewriting rules can help us understand the dynamics of biological networks with temporal and structural patterns. Temporal patterns can show how biological networks change over time. Structural patterns show how the changing patterns are related to each other. Transformation rules can abstract discovered graph rewriting rules so that they can describe repeated structural changes in the dynamic networks, and can predict future changes.

In the next chapter, we will represent experiments with synthetic data to further empirically evaluate our approach.
6. EMPIRICAL EVALUATION WITH SYNTHETIC DATA

This chapter presents an empirical evaluation of our approach using synthetic data. We generate several dynamic graphs that contain some variations, such as noise, size, and density ratio ($|V|:|E|$). We apply our Dynamic Graph-based Relational Learning approach, which we refer to as DynGRL, to these dynamic graphs in order to identify the strengths and weaknesses of the approach across various dimensions of the problem. For these experiments, we propose algorithms to generate graphs and dynamic graphs randomly. In addition, these generation algorithms can add three types of noises to dynamic graphs. Section 6.1 describes the algorithms for generating graphs and dynamic graphs. Section 6.2 shows the experiment with dynamic graphs including three types of noise. Section 6.3 shows the comparison of performance with the different size of dynamic graphs. Lastly, section 6.4 shows the comparison of performance with the different density ratio ($|V|:|E|$) of dynamic graphs. The final goal of this chapter is to show how DynGRL behaves with the various types of dynamic graphs.

6.1 Data Preparation

This section introduces the data preparation for the described experiments. Basically, we randomly generate dynamic graphs with or without noise. For this process, we first introduce our static graph generation approach. The graphs generated from the static graph generation constitute a sequence of graphs in a dynamic graph. Then, we introduce the dynamic graph generation and how the three types of noise can be applied to a dynamic graph.
6.1.1 Generating Static Graphs

The static graph is generated based on five input parameters: the number of vertices and edges, the lists of vertex labels and edge labels, and one seed substructure. The number of vertices and edges specify the maximum number of vertices and edges of the graph that will be generated. The lists of vertex labels and edge labels are used to randomly generate a vertex and edge with an appropriate label. The seed substructure is a starting point to generate a graph, and the substructure can be a single vertex.

An Input for Static Graph Generation

Vertices 14
Edges 20

VertexLabels {
cpd:C00001 0.003
cpd:C00002 0.003
cpd:C00003 0.003
...
}

EdgeLabels {
C_to_Rct 0.05
Rct_to_C 0.05
...
}

Substructure {
v 1 sce:YAL00A
v 2 enzyme
d 1 2 G_to_E
}

The above input for the static graph generation contains the five parameters. Vertices and Edges with following numbers in the first two lines denote the maximum number of vertices and edges. VertexLabels and EdgeLabels denote the list of labels. The following numbers denote a
probability of the label’s appearance. The sum of probabilities in each VertexLabels and EdgeLabels should be 1.0. The last, Substructure, denotes a starting graph (can be a single vertex) that is used when the user wants to include a specific substructure.

Algorithm 3: Generating Random Graph Algorithm

Input: Vertices, Edges, Sub, VertexLabels, EdgeLabels
Output: Graph
1 \( G = \text{Sub} \)
2 while NumVertices < Vertices do
3 \hspace{1em} AddRandomVertexWithinVertexLabels(G)
4 end
5 while NumEdges < Edges do
6 \hspace{1em} ConnectGraphWithinEdgeLabels(G)
7 end
8 while NumEdges \leq Edges do
9 \hspace{1em} AddRandomEdgesWithinEdgeLabels(G)
10 end
11 return \( G \)

Algorithm 3 presents an approach to randomly generate a static graph. First, the algorithm generates a starting graph that is isomorphic to the input substructure. Then, the algorithm randomly adds vertices until it reaches the maximum number of vertices (Vertices), where the added vertices use labels according to the distribution in the list of vertex labels (VertexLabels). Then, the algorithm tries to connect the graph with the added vertices using edges according to the label distribution from EdgeLabels. This process randomly adds edges between two disconnected graphs, but it is possible that there are still disconnected graphs after this process, if the process reaches Edges before connecting the graph. This process stops when reaching Edges. If the number of edges in the current graph is still less than Edges, the algorithm randomly adds edges until reaching Edges. This process randomly chooses two existing vertices and randomly adds an edge between two chosen vertices according to the label distribution from EdgeLabels. There can also be multiple edges between two vertices. EdgeLabels also determines
whether the edge is directed or undirected. In this chapter, we assume every edge is directed following the way of biological networks.

An example is shown in figure 6.1. Here, the starting graph contains 6 vertices and 5 edges shown in (A). Vertices and Edges are specified as 8 and 10 for the final graph. Two (orange) vertices are randomly added and the final number of vertices (Vertices) is reached in (B). Then, the starting graph is connected to two vertices using two random (dashed) edges in (C). We can still add more edges to reach the final Edges. Three random (dashed) edges are added to reach Edges in (D).

In this way, Algorithm 3 can generate a random static graph. This static graph generation process is used to generate the Core graph in each time slice graph and the Rule graph (what is added/removed) in the dynamic graph, as described in the next section.

6.1.2 Generating Dynamic Graphs with Noise

This section describes the dynamic graph generation with the inclusion of noise. When generating a dynamic graph, our approach uses two static graphs, Core and Rule, generated from
the static graph generation in the previous section. We suppose the Core graph always appears at every time slice in a dynamic graph. Then, the Rule graph appears periodically so that DynGRL can learn a transformation rule in the Rule appearances. Figure 6.2 shows an example of a dynamic graph that contains the Core and Rule graphs. The red graph shown at every time denotes the Core, and the orange graph shown at time 1, 3 and 4 denotes the Rule. When Rule appears, Core and Rule are connected using a random edge that does not have to be the same edge at every time as shown in figure 6.2.

Figure 6.2 An example of a dynamic graph using Core and Rule. The above orange subgraphs denote Rule (at time 1, 3, and 4), and the below red subgraphs denote Core. T denotes each time slice.

In the experiments of this chapter, we generate three variations of dynamic graphs, as along the dimensions of noise, size and density ratio. For the size and ratio variations, we can generate dynamic graphs using different numbers of Vertices and Edges used in the static graph generation.

For the noise variation, we make two assumptions. First, we assume there are three types of noise: addition, removal and replacement. Each case is shown in figure 6.3. Addition noise represents random additions of vertices and/or edges as shown in (A). The number of additions is specified by an error rate. In case of (A), the error rate is 0.2, because there are four additions.
(two vertices and two edges) to the graph, the size of which is 20. If we specify the error rate as 0.1, we need to add two additions. The removal noise represents random removals of vertices and/or edges as shown in (B). There are four removals for a 0.2 error rate, where one vertex and edge from the above orange subgraph and two edges from the below red subgraph are removed. The replacement noise represents random replacements of labels for vertices and/or edges. There are four replacements on two vertices and two edges in (C).

![Figure 6.3 Three types of noise in a graph. (A) The addition noise represents random additions of vertices and/or edges. (B) The removal noise represents random removals of vertices and/or edges. (C) The replacement noise represents random replacements of labels for vertices and/or edges.](image)

In real world, the three types of noise may appear together. However, we consider only one type of noise for each experiment. The second assumption is that the noise is applied to each time at the same rate. If we specify a noise rate of 0.1, the noise rate of 0.1 is applied to every graph in a dynamic graph. But the noise is randomly applied, so there are different modifications at each time. We can employ other ways of applying noise. For example, we can apply $\sqrt{\text{NoiseRate}}$ noise to $\sqrt{\text{NoiseRate}}$ of the time slices. In this way, we choose $\sqrt{\text{NoiseRate}}$ of the time slices, and apply $\sqrt{\text{NoiseRate}}$ noise to each chosen graph. However, we simulate biological networks in this experiment. When we generate biological networks from biological data, i.e., microarray data, the noise would be applied when we generate a graph at each time rather than particular time slices, i.e., there is a greater chance of noise while taking each
snapshot using microarray. For this reason, we assume our noise rate is applied to every graph in a dynamic graph.

The example below represents an input file for the dynamic graph generation algorithm with the noise application. The input contains NumTime, RuleAppearance, Noise, VertexLabels, EdgeLabels, Rule and Core. NumTime denotes the number of time slices in the dynamic graph. RuleAppearance specifies the appearance of Rule using binary digits. In case of this example, RuleAppearance has four digits, because NumTime is four. Each digit represents the appearance of Rule, where 1 denotes appearance and 0 does not. VertexLabels and EdgeLabels are the lists of labels for vertices and edges.

**An Input for Dynamic Graph Generation with Noise Application**

% Number of Time Series
NumTime 4

% Rule Appears 1 Appear, 0 Not
RuleAppearance
0
1
1
0

NoiseRate 0.01
NoiseType add

VertexLabels {
cpd:C00001 0.003
cpd:C00002 0.003
cpd:C00003 0.003
...}

EdgeLabels {
C_to_Rct 0.05
Rct_to_C 0.05
...
}
The lists should be the same lists that are used for generating Rule and Core, because the error application uses the lists of labels. Rule and Core contain the static graphs that are generated from Algorithm 3. In each experiment, we use the same list of labels for Rule, Core and the noise application to provide the same condition for each case. There is an issue related to the limited number of labels. We will discuss this issue with the size variation. We assume there is a single connection between Rule and Core as shown in figure 6.2, but there could be more than one edge because of the noise application.

Algorithm 4 describes the approach to generate a dynamic graph with noise application. For each time slice, if there appears Rule, the algorithm generates a graph containing Rule and Core. Otherwise, a graph containing only Core is generated. If the Rule is added, then a random edge is chosen, based on the distribution in EdgeLabels, to connect a randomly-chosen vertex in Rule to a randomly-chosen vertex in Core. If the NoiseRate is 0.0, then the algorithm generates a noise-free dynamic graph. If the NoiseRate is greater than 0.0, then the type of noise (as given in NoiseType) is added to the graph, which is then added to the final dynamic graph.
If the NoiseType is addition, then the ApplyNoise procedure proceeds similar to Algorithm 3 using the two internal parameters, Vertices and Edges, that specify the maximum number of vertices and edges in the graph. But Vertices and Edges contain the number of vertices and edges in Core (or Core + Rule) and increments by the error rate. For example, if $|V_{Core}| = 8$, $|E_{Core}| = 18$, $|V_{Rule}| = 12$, $|E_{Rule}| = 12$, and NoiseRate = 0.1. Then, totally $|V| = 20$ and $|E| = 30$. After the addition noise at rate 0.1, $|V| = 22$ and $|E| = 33$. Therefore, Vertices = 22 and Edges = 33 in this case, and ApplyNoise randomly adds the noise in the same way as Algorithm 3.

Algorithm 4: Generating Random Dynamic Graph With Noise Application

**Input:** NumTimes, Core, Rule, RuleAppearance, NoiseRate, NoiseType, VertexLabels, EdgeLabels

**Output:** Dynamic graph DG

1. $DG = []$
2. for $i = 1$ to NumTimes do
3.   $G = Core$
4.   if RuleAppearance[$i$] = 1 then
5.     $G = G + Rule + connectingEdge$
6.   end
7.   if NoiseRate > 0 then
8.     ApplyNoise($G$, NoiseRate, NoiseType)
9.   end
10. $DG = DG + G$
11. end
12. return $DG$

The procedure is more complex if the NoiseType is removal. We can specify Vertices and Edges, but should be careful in the removal case. If we remove one vertex, this may cause the removal of multiple edges connected to this vertex, as shown in figure 6.4 (A). If we remove one orange (thick) vertex, two related (dashed) edges are also removed. The easiest way is to remove only edges up to the noise rate times the graph size. Based on the types of removal we see in real
graphs, we try to remove vertices and edges at the same time. The procedure is shown in figure 6.4. We suppose \(|G| = 9\) after the noise removal, where \(|V| = 4\) and \(|E| = 5\).

![Diagram of vertex and edge removal process](image)

**Figure 6.4** (A) Remove one random vertex (orange), and then remove edges (dashed) related to the removed vertex. (B) No more vertex removal, because of the extra edge removal. (C) The remainder of \(\text{Vertices}\) is transferred to \(\text{Edges}\), and we remove one edge. (D) The final result has the correct size.

First, we try to remove one random vertex (orange), and then remove edges (dashed) related to the removed vertex. We can remove one more vertex in (B), but in that case we need to remove extra edges so that \(\text{CurEdges}\) will be less than \(\text{Edges}\). In that case, we transfer the remainder (to remove) of vertices to edges. Then, we remove one more edge in (C), and provide the final result in (D). Even though \(\text{Vertices}\) and \(\text{Edges}\) are different than we planned, the size of the graph is as we planned.

The procedure if \(\text{NoiseType}\) is replacement is simply to replace the number of desired (noise rate) labels. This procedure randomly chooses vertices and edges, and then replaces their labels according to the distribution in the list of labels as provided in the input.

We generate data for three experiments varying noise, size and density ratio. For the noise experiment, we generate four dynamic graphs: non-noise graph, removal noise graph, addition noise graph, and replacement noise graph. For the size experiment, we generate 25
dynamic graphs in combination with five Cores and Sizes. These five Cores and Sizes are different. For the ratio experiment, we generate 25 dynamic graphs in combination with five Cores and Sizes, where they have different ratios of number of vertices ($|V|$) to number of edges ($|E|$). To evaluate our results, we use the prediction measure introduced earlier.

6.2 Synthetic Data with Noise Variation

This section describes experiments with noise variation. The goal of this experiment is to show how DynGRL behaves with the noise variations. We prepare four types of dynamic graphs: non-noise, addition noise, replacement noise, and removal noise. The non-noise graph contains Core and Rule in twenty time slices. Rule appears at times 2, 3, 7, 8, 12, 13, 17, and 18 (eight times), where Rule is added at time 1, 6, 11 and 16, and removed at time 3, 8, 13 and 18. The transformation rule is $TR = Rule(+3, -2)$. The size of Core is 272, where $|V|=112$ and $|E|=160$. The ratio of $|V|$ to $|E|$ is 0.7 which was targeted based on the ratios of the ten metabolic pathways we used in the previous microarray experiment. The ratios of $|V|$ to $|E|$ in the ten metabolic pathways vary, but the average is 0.7. Rule and Core are connected with a single edge. The addition noise dynamic graph adds random vertices and edges according to the noise rate. The noise rate is varied from 0.01 to 0.8 as shown in table 6.1. The dynamic graphs with the removal noise are generated by removing vertices and edges at the noise rate. The dynamic graphs with the replacement noise are generated by replacing the labels on vertices and edges at the noise rate.
Table 6.1 The result of the noise variation. First column shows the four types of the dynamic graph: non-noise, addition noise, removal noise and replacement noise graphs. The other columns show the prediction for each noise rate.

<table>
<thead>
<tr>
<th>Type</th>
<th>Prediction for Noise Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0.01</td>
</tr>
<tr>
<td>Non-noise</td>
<td>1.00</td>
</tr>
<tr>
<td>Addition Noise</td>
<td>0.91</td>
</tr>
<tr>
<td>Replacement Noise</td>
<td>0.71</td>
</tr>
<tr>
<td>Removal Noise</td>
<td>0.73</td>
</tr>
</tbody>
</table>

The results show how DynGRL works on data with noise. Prediction measures what portion of the desired rules DynGRL discovers in the dynamic graph. There are four additions and four removals of Rule in a dynamic graph. If prediction is 1.0, DynGRL discovers all four additions and all four removals. A lower prediction value means DynGRL does not discover the entire Rule. We could include a penalty for not predicting the correct time when Rule is removed or added, but we focus more on whether Rule is correctly discovered. Figure 6.5 shows the prediction curve of four dynamic graphs with the noise rates. Clearly, DynGRL discovers all of the substructures in the non-noise dynamic graph. The addition noise dynamic graphs show the best performance among the three types of noise. DynGRL achieves more than 0.8 prediction with less than 0.05 noise rate and 0.5 prediction with less than 0.2 noise rate. DynGRL shows worse performance on the removal noise dynamic graphs show than the addition noise dynamic graphs. DynGRL achieves less than 0.5 prediction with noise rates larger than 0.05. With noise rates larger than 0.2, prediction goes down below 0.1. The replacement noise dynamic graphs result in the worst performance. With a replacement noise rate larger than 0.05, prediction is less
than 0.1. With the 0.4 noise rate, prediction seems larger than other cases. But this seems to happen accidentally after multiple replacements.

Figure 6.5 Prediction with noise rates in the four types of dynamic graphs: non-noise, addition noise, replacement noise and removal noise.

As the results show, DynGRL is least sensitive to addition noise in dynamic graphs, more sensitive to removal noise, and most sensitive to replacement noise. In the case of the addition noise, the desired rules should still reside in the dynamic graphs, because the noise just adds more vertices and edges to the graphs. However, after adding some vertices and edges, the common portion of each graph in the dynamic graph is growing. In other words, the maximum common subgraphs can be growing with larger noise, and some portions of the rule can be included into the common portions with added noise. In this way, some portions of the desired rule cannot be discovered by DynGRL. In the case of removal noise, the performance is worse
due to the possibility that some removals occur inside the rule substructure, which makes the rule substructure smaller or divided into several substructures. In these cases, DynGRL identifies substructures different from the desired substructure. In the case of replacement noise, the performance is the worst, because even a few single label replacements can make a different substructure. The removal case can only make the substructure smaller unless the removal breaks the graph into several disconnected graphs. But the replacements in the substructure can make the substructure completely different.

Table 6.2 The result of the noise variation. First column shows the four types of dynamic graph: non-noise (non), addition noise (add), replacement (rep) noise and removal (rem) noise. The other columns show running time for each noise rate.

<table>
<thead>
<tr>
<th>Type</th>
<th>Running Time (sec.) for Noise Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0.01</td>
</tr>
<tr>
<td>Non</td>
<td>65.51</td>
</tr>
<tr>
<td>Add</td>
<td>122.43</td>
</tr>
<tr>
<td>Rep</td>
<td>101.1</td>
</tr>
<tr>
<td>Rem</td>
<td>77.43</td>
</tr>
</tbody>
</table>

Table 6.2 and figure 6.6 show running time vs. noise rate in the four types of dynamic graphs. The results show some trends in the running time for four types of dynamic graphs. Running time of the addition noise case is longest. In DynGRL, discovery of the maximum common subgraph is a bottleneck process. After adding noise, the maximum common subgraph is bigger and running time becomes slower. For replacement noise, there are several replacements in the graphs, and the discovery process takes more time to discover common patterns between two arbitrarily changed graphs. Running times for addition and replacement
noise are not monotonic as the noise rate increases, because of the running time for Step 2. Running time in Step 2 is varied for each case, where it takes various amounts of time to discover the bestSub over the differing graphs due to noise. Specifically, we compare the cases of addition and replacement noise at noise rates 0.6 and 0.8. We investigated in detail the substructure discovery process in these two cases. The process discovers more candidate substructures at noise rate 0.6 than at 0.8 in both addition and replacement noise cases. The sizes of the substructures at noise rate 0.6 are much larger than at 0.8 in both noise cases. In this way, the process performs more work at noise rate 0.6 than at 0.8 with the arbitrarily changed graphs. In the removal noise case, running time can be faster, because the input graphs become smaller. But the discovered patterns become much different from the desired ones as described previously.

This section shows how DynGRL behaves in the presence of various types of noise. In general, DynGRL can discover the best rules in the addition noise case, but replacement noise tends to more quickly obscure the desired rule.
Figure 6.6 Running time vs. noise rates in the four types of dynamic graphs: non-noise, addition noise, replacement noise and removal noise.

6.3 Synthetic Data with Size Variation

This section presents experimental results to show how well DynGRL learns desired patterns as the dynamic graphs vary in size. We prepare five different sizes of Rules and Cores, and generate 25 different dynamic graphs based on these Rules and Cores. Then, we run DynGRL on the 25 dynamic graphs, and observe the variance of DynGRL’s performance in terms of both prediction and running time.

Table 6.3 shows the five Rules and Cores. Each graph is two times bigger than the previous one. The minimum size of Rule is 34 and the maximum size of Rule is 544. The minimum size of Core is 68 and the maximum size of Core is 1088. We generate 25 dynamic
graphs using the combinations of \textit{Rules} and \textit{Cores}, such as \(C1\odot R1, \ C1\odot R2, \ C1\odot R3, \ldots, \ C5\odot R4, \ C5\odot R5\), where \(R\) and \(C\) denote \textit{Rule} and \textit{Core} with a specific number, and \(\odot\) denotes that two graphs are connected by a single random edge. With this data, we study how DynGRL’s performance varies with various sizes of \textit{Cores} and \textit{Rules}.

Table 6.3 The number of vertices and edges, and size of graphs for each \textit{Core} and \textit{Rule}. Each column denotes an index number of \textit{Core} and \textit{Rule}.

<table>
<thead>
<tr>
<th>Type</th>
<th>Number for Rule / Core</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
</tr>
<tr>
<td>Rule</td>
<td>(</td>
</tr>
<tr>
<td></td>
<td>(</td>
</tr>
<tr>
<td></td>
<td>(</td>
</tr>
<tr>
<td>Core</td>
<td>(</td>
</tr>
<tr>
<td></td>
<td>(</td>
</tr>
<tr>
<td></td>
<td>(</td>
</tr>
</tbody>
</table>

When we generate 25 dynamic graphs, we also need to set the parameter that determines the number of unique labels for vertices and edges. The number of unique labels is closely related to the search space of DynGRL. First, \textit{Limit} is specified based on the number of unique labels, as described in chapter 4. Recall that \textit{Limit} is a bound on the number of difference substructures considered by the substructure discovery process. We set \textit{Limit} as \(UVL + 4\gamma(UEL - 1)\), where \(UVL\) represents the number of unique vertex labels and \(UEL\) represents the number of unique edge labels. \(\gamma\) is a user specified parameter, where we set \(\gamma = 1\) as a default. Therefore, \textit{Limit} (and thus DynGRL’s search space) is in proportion to the number of unique labels. Second, the discovery of the maximum common substructure is a NP-hard problem, but it has been proven to be a quadratic problem with unique labels [48], as also described in chapter 4. For these two reasons, the numbers of unique labels can affect the performance of DynGRL. We first run an experiment with 25 dynamic graphs generated with the limited numbers of unique labels.
labels. Then, we run an experiment with another set of 25 dynamic graphs generated with the sufficient numbers of unique labels so that the Limit, and thus the search space, is sufficient to find the desired structures. In this experiment, we simulate biological networks. In biological networks, the number of unique edge labels is limited. Our dynamic graphs contain only nine unique edge labels. But the number of unique vertex labels can be larger, because vertices represent various names of molecules, reactions and pathways. Here, the sufficient numbers of unique labels means the sufficient numbers of unique vertex labels. For the limited number of unique labels, we limit the number of unique vertex labels to 143 for all five different sizes. For the sufficient number of unique labels, we specify 5 different numbers of unique labels: 100, 200, 400, 800, and 1600.

Table 6.4 Prediction for various sizes of Cores and Rules for the limited numbers of unique labels. The first column represents the size of Cores, and the first row represents the size of Rules. Each element represents Prediction of the dynamic graph consisting of the connected Core and Rule.

<table>
<thead>
<tr>
<th>Core</th>
<th>Rule</th>
<th>R1 (34)</th>
<th>R2 (68)</th>
<th>R3 (136)</th>
<th>R4 (272)</th>
<th>R5 (544)</th>
</tr>
</thead>
<tbody>
<tr>
<td>C1 (68)</td>
<td>R1 (34)</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0.76</td>
</tr>
<tr>
<td>C2 (136)</td>
<td>R2 (68)</td>
<td>1</td>
<td>0.98</td>
<td>0.96</td>
<td>0.97</td>
<td>0.72</td>
</tr>
<tr>
<td>C3 (272)</td>
<td>R3 (136)</td>
<td>1</td>
<td>1</td>
<td>0.92</td>
<td>1</td>
<td>0.8</td>
</tr>
<tr>
<td>C4 (544)</td>
<td>R4 (272)</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0.96</td>
<td>0.76</td>
</tr>
<tr>
<td>C5 (1088)</td>
<td>R5 (544)</td>
<td>1</td>
<td>0.73</td>
<td>0.94</td>
<td>0.97</td>
<td>0.85</td>
</tr>
</tbody>
</table>
Table 6.4 and figure 6.7 show the results of prediction with the limited numbers of unique labels. 12 cases of out 25 show 1.0 prediction. More than 50% of the cases cannot discover the entire desired patterns. Most likely, larger core and rule sizes result in smaller prediction values, because a larger core and rule require a more extensive, and thus sometimes unsuccessful, search for the complete desired substructure. There is an exceptional case with core size 1088 and rule size 68, due to the small limit on the search space during substructure discovery. In the above limit formula $UVL + 4\gamma (UEL - 1)$, we set $UVL = 143, UEL = 9$ and $\gamma = 1$ (as default), which gives a limit of 175. If we increase the user-specified constant $\gamma$ to 2, limit will increase from 175 to 207. With this small increment, DynGRL learns the entire desired
rule with \textit{prediction} = 1.0. Generally, the larger graph size results in worse performance when the number of unique labels is low. With the limited number of unique labels, i.e., limited search space, DynGRL shows lower \textit{prediction} values on the larger graph. Running time is also slower when the dynamic graph is larger. Especially, running time is much slower with larger \textit{rule} size (544). Running of step 2 in DynGRL takes several days as shown in table 6.5.

Table 6.5 Running time for various sizes of \textit{Cores} and \textit{Rules} with the limited numbers of unique labels. The first column represents the size of \textit{Cores}, and the first row represents the size of \textit{Rules} with each index number. Each element represents running time (seconds) of the dynamic graph consisting of the connected \textit{Cores} and \textit{Rules}.

<table>
<thead>
<tr>
<th>Core\Rule</th>
<th>R1 (34)</th>
<th>R2 (68)</th>
<th>R3 (136)</th>
<th>R4 (272)</th>
<th>R5 (544)</th>
</tr>
</thead>
<tbody>
<tr>
<td>C1 (68)</td>
<td>2.21</td>
<td>5.93</td>
<td>23.27</td>
<td>677.32</td>
<td>15086.82</td>
</tr>
<tr>
<td>C2 (136)</td>
<td>7.70</td>
<td>10.06</td>
<td>65.93</td>
<td>1063.19</td>
<td>38907.40</td>
</tr>
<tr>
<td>C3 (272)</td>
<td>40.79</td>
<td>43.60</td>
<td>77.27</td>
<td>5495.58</td>
<td>18259.97</td>
</tr>
<tr>
<td>C4 (544)</td>
<td>223.14</td>
<td>281.08</td>
<td>365.62</td>
<td>3571.62</td>
<td>127729.17</td>
</tr>
<tr>
<td>C5 (1088)</td>
<td>3129.72</td>
<td>6453.38</td>
<td>5487.92</td>
<td>6122.77</td>
<td>372581.47</td>
</tr>
</tbody>
</table>

In step 2, DynGRL discovers common subgraphs in the discovered graph rewriting rules from step 1. In that procedure, there are many redundant labels that make the discovery process much slower.

With the sufficient numbers of unique labels, DynGRL discovers the entire desired patterns. The reason is that the sufficient number of unique labels increases the search space in every case. Here, we observe running time in each case to show the performance for the varied size of dynamic graphs. Because we have enough unique labels and enough search space to discover the patterns, the entire desired pattern is discovered (\textit{prediction} = 1.0) in every case. Table 6.6 and figure 6.8 show the results of running time with the sufficient numbers of unique labels. Similar to the trend for limited unique labels, running time follows a quadratic curve with increasing size of a dynamic graph. As shown in our results, the size of \textit{core} has more effect than
the size of rule, because the discovery of maximum common subgraphs is a bottleneck process in DynGRL.

Table 6.6 Running time for various sizes of Cores and Rules with the sufficient numbers of unique labels. The first column represents the size of Cores, and the first row represents the size of Rules with each index number. Each element represents running time (seconds) of the dynamic graph consisting of the connected Cores and Rules.

<table>
<thead>
<tr>
<th>Core\Rule</th>
<th>R1 (34)</th>
<th>R2 (68)</th>
<th>R3 (136)</th>
<th>R4 (272)</th>
<th>R5 (544)</th>
</tr>
</thead>
<tbody>
<tr>
<td>C1 (68)</td>
<td>2.19</td>
<td>3.53</td>
<td>13.58</td>
<td>116.33</td>
<td>2356.64</td>
</tr>
<tr>
<td>C2 (136)</td>
<td>9.10</td>
<td>9.29</td>
<td>25.57</td>
<td>143.26</td>
<td>2460.17</td>
</tr>
<tr>
<td>C3 (272)</td>
<td>58.64</td>
<td>63.15</td>
<td>72.93</td>
<td>242.81</td>
<td>2694.15</td>
</tr>
<tr>
<td>C4 (544)</td>
<td>502.62</td>
<td>573.20</td>
<td>632.54</td>
<td>809.22</td>
<td>3952.14</td>
</tr>
<tr>
<td>C5 (1088)</td>
<td>6337.92</td>
<td>6987.49</td>
<td>7057.20</td>
<td>7865.34</td>
<td>9002.24</td>
</tr>
</tbody>
</table>

Figure 6.8 Running time for various sizes of Core and Rule with the sufficient numbers of unique labels.

We study how the sizes of dynamic graphs affect the performance of DynGRL. Generally, running with a larger dynamic graph shows worse performance and running with a
smaller dynamic graph shows better performance in both aspects: prediction and running time. The size of the search space, the limit in DynGRL, is also an important parameter for the performance. If DynGRL has enough search space, either by choosing a larger value for the user-specified $\gamma$ or by having a sufficient number of unique labels, DynGRL performs better in both aspects: running time and prediction.

6.4 Synthetic Data with Density Ratio Variation

This section presents experimental results that show the performance of DynGRL as the dynamic graphs vary in terms of the ratio between the number of vertices and edges (density). As in the size varied experiment, we prepare five different rules and cores, that have different ratios of number of vertices to number of edges as shown in table 6.7. We generate 25 different dynamic graphs based on these rules and cores. Then, we run DynGRL on the 25 dynamic graphs, and investigate the effect on DynGRL’s performance both in terms of prediction and running time.

Table 6.7 Rules and Cores used in the density ratio variation experiment. The first row represents index numbers for each rule or core. The next three rows show the numbers of vertices, edges, and size for rule. The last three rows show the numbers of vertices, edges, and size for core.

<table>
<thead>
<tr>
<th>Type</th>
<th>Number for Rule / Core</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1 (1:1)</td>
</tr>
<tr>
<td>Rule</td>
<td>$</td>
</tr>
<tr>
<td></td>
<td>$</td>
</tr>
<tr>
<td></td>
<td>$</td>
</tr>
<tr>
<td>Core</td>
<td>$</td>
</tr>
<tr>
<td></td>
<td>$</td>
</tr>
<tr>
<td></td>
<td>$</td>
</tr>
</tbody>
</table>
Table 6.7 shows the five rules and cores. Each rule (core) has a different ratio of the number of vertices to number of edges. Each rule and core with the same index has the same ratio. Rule 2 (Core 2) has the ratio of 7:10, which is the average ratio of the ten metabolic pathways from chapter 5. The other four rules (cores) with the indices, 1, 3, 4, and 5 have different ratios of 1:1, 1:2, 1:3 and 1:4, respectively. While the number of vertices in the five rules (cores) is constant, the number of edges is increasing. We generate 25 dynamic graphs using the combinations of rules and cores, such as $C1 \odot R1$, $C1 \odot R2$, $C1 \odot R3$, $\cdots$, $C5 \odot R4$, $C5 \odot R5$, where $R$ and $C$ denote rule and core with an index number, and $\odot$ denotes the two graphs are connected by a single random edge. With this data, we study how DynGRL behaves with various density ratios of dynamic graphs.

We also consider the limit issue here. This experiment has two sets of labels for dynamic graphs. One set has the limited number (143) of unique vertex labels, and the other has the sufficient number (500) of unique vertex labels. As with the size variation experiments, the number of unique labels in the dynamic graphs is also an important issue for experiments with varying density ratios.

Table 6.8 Prediction for various density ratios of cores and rules with the limited number of unique labels. The first column represents the ratio of cores and the first row represents the size of rules. Each entry in the table shows the prediction value for the dynamic graph consisting of the connected core and rule.

<table>
<thead>
<tr>
<th>Core\Rule</th>
<th>1:1 (112)</th>
<th>7:10 (136)</th>
<th>1:2 (224)</th>
<th>1:3 (224)</th>
<th>1:4 (280)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1:1 (224)</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>7:10 (272)</td>
<td>1.00</td>
<td>0.92</td>
<td>0.96</td>
<td>1.00</td>
<td>0.93</td>
</tr>
<tr>
<td>1:2 (336)</td>
<td>1.00</td>
<td>0.97</td>
<td>0.88</td>
<td>0.88</td>
<td>0.86</td>
</tr>
<tr>
<td>1:3 (448)</td>
<td>0.98</td>
<td>0.88</td>
<td>0.88</td>
<td>0.98</td>
<td>0.94</td>
</tr>
<tr>
<td>1:4 (560)</td>
<td>0.73</td>
<td>0.68</td>
<td>0.86</td>
<td>0.78</td>
<td>0.98</td>
</tr>
</tbody>
</table>
Table 6.8 and figure 6.9 show the results for the limited numbers of unique labels. In 8 cases out of 25, DynGRL discovers the entire desired pattern as $prediction = 1.0$. Where the dynamic graphs are denser (i.e., with lower ratio), $predictions$ become lower.

![Figure 6.9](image)

Figure 6.9 $Prediction$ for various density ratios of $cores$ and $rules$ with the limited numbers of unique labels. The $Prediction$ axis shows the $prediction$ value for the best transformation rule learned by DynGRL. The $Core$ ratio axis shows the ratio of each $core$ with the size (in the parenthesis). The $Rule$ ratio axis shows the ratio of each $rule$ with the size (in the parenthesis).

The first observation is that the ratios in $core$ have a greater effect on $prediction$. We can increase the $\gamma$ parameter to provide more search space and to reach the full $predication$ value. But in that case, running time increases considerably. For instance, if we increase the $\gamma$ parameter to 2.0, the running time of only step 1 becomes 50,357 seconds (about 14 hours).
Then, even several thousand more seconds of running time in step 2 cannot discover the entire pattern. We understand this to be the same reason that the dynamic graph does not have the sufficient numbers of unique labels, and therefore DynGRL does not have enough search space (limit).

We also need to consider that in this experiment, we do not change the number of vertices, and just change the number of edges. From the change, the size of the graph is also increased. Therefore, we should consider whether the effect of ratio actually just results from the effect of size. We also built five dynamic graphs with a constant density ratio having the same size of core and rule used in the other dynamic graphs, where rule sizes are 112, 136, 168 and 280 and core sizes are 224, 272, 336, 448 and 560. But these rules and cores have the same ratios as 1:1. The combinations of five dynamic graphs are \( C1\otimes R1, \ C2\otimes R2, \ C3\otimes R3, C4\otimes R4, \) and \( C5\otimes R5 \). The prediction in the constant ratio data is 1.0 in every case. Therefore, we can claim that the density ratio is the major factor affecting DynGRL’s performance in this experiment.

Table 6.9 Running time for various ratios of cores and rules with the sufficient numbers of unique labels. The first column shows the ratio of Cores, and the first row shows the ratio of Rules. Each entry in the table shows the running time (seconds) of DynGRL on the dynamic graph consisting of the connected core and rule.

<table>
<thead>
<tr>
<th>Core\Rule</th>
<th>1:1 (112)</th>
<th>7:10 (136)</th>
<th>1:2 (224)</th>
<th>1:3 (224)</th>
<th>1:4 (280)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1:1 (224)</td>
<td>9.18</td>
<td>19.33</td>
<td>68.27</td>
<td>469.81</td>
<td>1251.74</td>
</tr>
<tr>
<td>7:10 (272)</td>
<td>64.66</td>
<td>79.30</td>
<td>122.98</td>
<td>524.40</td>
<td>1399.95</td>
</tr>
<tr>
<td>1:2 (336)</td>
<td>300.77</td>
<td>272.27</td>
<td>346.24</td>
<td>920.04</td>
<td>3602.61</td>
</tr>
<tr>
<td>1:3 (448)</td>
<td>4762.66</td>
<td>5412.41</td>
<td>6241.33</td>
<td>8854.93</td>
<td>11270.27</td>
</tr>
<tr>
<td>1:4 (560)</td>
<td>22592.95</td>
<td>21507.07</td>
<td>22299.27</td>
<td>23699.52</td>
<td>26403.36</td>
</tr>
</tbody>
</table>
Table 6.9 and figure 6.10 show running time results when using the sufficient numbers of unique labels. With the sufficient numbers of unique labels, and therefore a sufficient sized search space, DynGRL can learn the entire desired pattern in every case. The results show the performance in running time (seconds). The higher density dynamic graphs, with more edges, need more time to discover the entire patterns. As with the size variation, the ratios in the cores have a greater effect than the ratios in the rules. This result also supports the fact that the discovery of maximum common subgraph is a bottleneck process in DynGRL. We study how the density ratios of dynamic graphs affect the performance of DynGRL. Generally, running with a denser dynamic graph shows worse performance and running with a sparser dynamic graph
shows better performance in both aspects: prediction and running time. The increasing size of the graphs, as we increase the number of edges while holding constant the number of vertices, does have some effect on performance. But our results show that the desity ratio, independent of graph size, is also an important factor in the performance of DynGRL.

6.5 Summary

In this chapter we evaluate our approach, DynGRL, using various synthetic data. We generate dynamic graphs with several variations. First, we generate dynamic graphs with the noise variation, where we add three types of noise: addition, replacement and removal noise. Performance is best with the addition noise and worst with the replacement noise. Generally, the performance worsens after applying a higher rate of noise. Second, we generate dynamic graphs that vary in size. The larger size dynamic graphs show worse performance in both aspects: prediction and running time. Lastly, we generate dynamic graphs with varying density ratios. The denser dynamic graphs show worse performance in both aspects: prediction and running time. In the last two experiments, we also consider the issue of the number of unique labels, which can also be view as an issue of the search space size. The sufficient number of unique labels also provides better performance in both aspects, but the actual number of unique labels will depend on the domain.

We observe how DynGRL behaves with three variations of dynamic graphs in both aspects: capability of discovery and running time. We also need to evaluate DynGRL’s ability to learn other types of changes, e.g., patterns with varying periods, patterns applied to a limited sequence. We leave these experiments for our future work.
7. Application to the Enron Email Networks

We developed DynGRL mainly to analyze the dynamics of biological networks. The dynamic graphs representing biological networks usually describe specific biological activities that most likely behave regularly or periodically over time. Even though we developed our approach for the regular behaviors in dynamic networks, our final goal is to analyze any types of dynamic networks whether the networks are regular or not.

Enron email data [59; 60] is a proper dataset for the analysis of dynamic graphs, because it represents dynamic email exchanges collected over a period of 3 years. The raw data contains 619,446 messages that are retrieved from 158 users. The Enron email data is much different from biological networks for two reasons. First, the data includes large networks and a long period. Second, the data does not show regular behaviors like biological networks. Even though there are many regular or periodic patterns in the data, the email exchanges as a whole do not follow regular patterns. In contrast, biological networks are relatively small and they show specific behaviors, because the networks are organized by specific functions. For these reasons, we choose the Enron email data as the second domain.

Another benefit to using the Enron data is the existence of results on this data from another dynamic graph mining system, the Periodic Behavior Miner [23]. While this system uses a different approach than DynGRL, we can compare the results qualitatively and identify some of the advantages and disadvantages of the different approaches.

In this chapter, we apply DynGRL to the Enron email data, to show how DynGRL behaves with other domains. Because there are few alternative approaches to DynGRL that focus on the patterns of change, it is hard to find a comparable approach. But we try to compare our
results with a close alternative approach called the Periodic Behavior Miner [23], which was discussed in Chapter 2.

7.1 Experiment with the Enron Data

The Enron data used in this experiment is downloaded from [61]. The data is preprocessed for the Periodic Behavior Miner [23]. We convert this data to a dynamic graph, and run DynGRL on the dynamic graph. The Enron data uses one day as a unit time slice, and contains email exchange information from January 1, 1997 to February 3, 2004 (2,587 days).

The Enron data has two data format as follows.

**Enron.itemset**

*4_Dec_1998 91 92  
*5_Dec_1998 91 100  
*7_Dec_1998 92 100 101  
%End of File

**Enron.map**

bruno.gaillard@enron.com mona.petrochko@enron.com 91  
bruno.gaillard@enron.com jeff.dasovich@enron.com 92  
hvc2qtr1cn@hotmail.com frk4hpdcqvr@hotmail.com 93  
postmaster@grtn.it richard.shapiro@enron.com 100  
duran@enron.com richard.shapiro@enron.com 101  
%End of File

Enron.itemset contains a set of links at each time. This file has 2,587 lines, and each line represent one day. If a line is empty, there is no email exchange on that day. The third line
between 5\textsuperscript{th} and 7\textsuperscript{th} of December 1998 in the above example represents December 6, 1998 and there is no email data on that day. Each number from the second column at each day represents an index for a unique link in the Enron.map file that represents a sender sending an email to a receiver. In the above example, a link, 91, at December 6, 1998 denotes a link that \textit{bruno.gaillard@enron.com} sends an email to \textit{mona.petrochko@enron.com}. The link, 91, is a unique index for the sending of email from \textit{bruno.gaillard@enron.com} to \textit{mona.petrochko@enron.com}.

We convert these two files to a dynamic graph having 2,587 time slices. Our conversion process first reads Enron.map and builds a list of all the links. Then, the process reads Enron.itemset. For each one line (one day), the process creates one graph for one time unit. After reading one link index in each day, the process finds the link with two emails from the list containing information from the Enron.map. The link is converted to an edge using the index as a label. The two email addresses are converted to two incident vertices. Each vertex number is assigned starting from 1. If the retrieved email address is already included in the graph, the process uses the previous one without generating a redundant vertex. Then the process builds a list of vertices and edges for the graph for one time unit (one day). The example below represents the converted dynamic graph from the Enron.map and Enron.itemset above. Each time denotes one day in the Enron.itemset. Figure 7.1 shows the graph representation for the example below.

\textbf{Enron.dg}

\begin{verbatim}
TIME 1
v 1 bruno.gaillard@enron.com
v 2 mona.petrochko@enron.com
v 3 jeff.dasovich@enron.com
d 1 2 91
d 1 3 92
\end{verbatim}
Figure 7.1 The graph representation for the dynamic graph, Enron.dg.

The total number of vertices is 722,945 and number of edges is 951,971. The size of the dynamic graph is 1,674,916. The average size of a graph in the dynamic graph is 647, where on average $|V| = 279$ and $|E| = 368$. 1,221 graphs out of 2,587 have one or more vertices, so 1,366 graphs do not contain any vertex. In other words, there is no email data on those 1,366 days. The average size of 1,221 graphs is 1,371, where on average $|V| = 592$ and $|E| = 779$.

First, we run DynGRL on the dynamic graph. But the running time is extremely slow, because the dynamic graph is too big (the file size is 35 megabytes). Even though we use a small
number (50) for the limit, the running time is more than two weeks. Figure 7.2 shows a result of running with limit = 50. It is possible to discover larger subgraphs if we use a larger limit, but currently the slow running time is prohibitive.

Figure 7.2 Discovered best subgraphs in the dynamic graph contain 2,587 days’ email data. The best subgraph discovered at iteration 1 (A), and iteration 2 (B).

Figure 7.2 shows the transformation rule substructures discovered at iteration 1 (Sub₁) and iteration 2 (Sub₂). The transformation rule for Sub₁ discovered at iteration 1 is Sub₁(₊2, −5) so that Sub₁ is added two times after the last removal and removed 5 times after the last addition. The transformation rule for Sub₂ discovered at iteration 2 is Sub₂(₊2, −1) so that Sub₂ is added two times after the last removal and removed one time after the last addition.

We would like to compare our results to those from the Periodic Behavior Miner system [23]. However, the results in figure 7.2 are not our optimal result, because we use a very small limit. In the next section, we analyze a subset of the Enron data, and compare our result to that found by the Periodic Behavior Miner system.

7.2 Experiment and Comparison with a Subset of the Enron Data

In this section, we prepare 101 days of data from April 17, 2000 to June 27, 2000 as 101 graphs in a dynamic graph. The size of the dynamic graph is 71,506, where |V| = 31,661 and
The average size of a graph in the dynamic graph is 708, where on average $|V| = 313$ and $|E| = 395$. The running time is about 24 hours, where Limit is set as default, and the number of iterations in step 2 is 10. The numbers of unique vertex labels and edge labels for each graph in the Enron dynamic graph vary considerably. The number of unique vertex labels is from 1 to 1,643, and the number of unique edge labels is from 1 to 2,097. The averages are 484.8 for vertex labels and 608.2 for edge labels. The Limits for each pairwise learning are calculated from 19 to 13,786, and the average is 5,408.8.

Figure 7.3 shows the three best subgraphs discovered at iteration 1, 3 and 6 in step 2. The transformation rules for these three subgraphs are $Sub_1(+2, -1)$, $Sub_3(+3, -2)$ and $Sub_6(+6, -1)$. In this way, DynGRL discovers how the dynamic graph changes its structure over time.

Now, we try to compare our result to one from the Periodic Behavior Miner (PBM) system. As described in chapter 2, the PBM system has two major differences from DynGRL. First, PBM tries to discover frequent patterns across all time points, not patterns describing the changing state like DynGRL’s graph rewriting rules. In other words, PBM requires the repeated patterns over the dynamic graph; whereas, DynGRL does not require the patterns over the dynamic graphs. One advantage of PBM is that PBM can find patterns with “jitter” so that the periodic pattern can appear at $p \pm J$ timeslices instead of exactly $p$, where $p$ denotes a period and $J$ denotes the jitter.
Figure 7.3 The three best subgraphs discovered by DynGRL in the dynamic graph contain 101 days of Enron email data. The best subgraph discovered at iteration 1 (A), iteration 3 (B) and iteration 6 (C).

Second, PBM keeps only periodic patterns out of the frequent patterns. After discovering a frequent pattern, PBM discards the pattern if it is not periodic. This is one reason why PBM is faster than DynGRL. DynGRL tries to discover the best compressing pattern and then find a transformation rule that covers all discovered instances. Below is a partial result by PBM on the 101 days of Enron data. PBM enumerates 377 results that satisfy the support of 2 and jitter 0. We compare a partial result (3 of the 377) below to our result.

Partial result of Periodic Behavior Miner on 101 days of Enron data

- sup 2 p 17 pos 4 [2268]
- sup 3 p 3 pos 15 [2268]
- sup 3 p 4 pos 78 [13661 13663 13666 13667 19449 19450 19451 32943]
At the first line in the above result, ‘sup 2’ denotes the subgraph in figure 7.3 (A) is discovered 2 times, and ‘p 17’ denotes the period between instances is 17. ‘pos’ denotes the starting point of this pattern. The third line denotes a periodic pattern that the subgraph is discovered at time 78, 82 and 86. If we convert the links in the third result above to a graph, the graph can be represented as shown in figure 7.4. The graph looks similar to the Sub3 in figure 7.3. One vertex and one edge in Sub3 are changed to two vertices and two edges of the graph in figure 7.3, where the different vertices and edges are denoted as bold.

Figure 7.4 A pattern discovered by the Periodic Behavior Miner [23].

As described previously, Periodic Behavior Miner tries to discover patterns at each time slice, and DynGRL discovers patterns over changing states that are represented as graph rewriting rules. For this reason, it is hard to compare the two systems directly. However, both systems try to discover patterns in dynamic graphs, so that there can be comparable points. The results by DynGRL show the repeated additions and removals in the dynamic graphs. From this information, we can predict the substructures that should be present at a specific time. Here, we try to compare the results in this way.

Figure 7.5 shows a visualization of removal and addition of Sub1 in figure 7.3. The additions and removals look more irregular, unlike the patterns in the biological networks. Algorithm 2 chooses the most frequent temporal distance between removals and additions or
between additions and removals. Algorithm 2 chooses 2 for the distance between removal and addition, and 1 for the distance between addition and removal. Then, $Sub_1(+2, -1)$ is learned as the transformation rule.

From the result by PBM, $Sub_1$ appears at time 4 and 21 as the first pattern, and at time 15, 18 and 21 as the second pattern in the above result. In figure 7.5, we notice $Sub_1$ is added at 3, 10, 14, 17, 20 and so on. $Sub_1$ is also removed at time 4, 11, 15, 18, 21 and so on. As described in chapter 3, an addition of a substructure at time $i$ represents the substructure is added to time $i+1$. Therefore, $Sub_1$ appears at time 4, 11, 15, 18, 21 and so on. In the case of the appearances at time 4, 11, 15 and 21, $Sub_1$ is removed and therefore absent by the next time slice (5, 12, 16 and 22). The appearance at time 4 and 21 has a period 17 that matches with the first pattern discovered by PBM. The appearance at time 15, 18 and 21 shows a period 3 that is the same pattern as the second one of PBM.

Figure 7.6 shows a visualization of removal and addition of $Sub_3$ in figure 7.3. Algorithm 2 chooses the most frequent temporal distance between removals and additions or between additions and removals. Algorithm 2 chooses 3 for the distance between removal and addition, and 2 for the distance between addition and removal. Then, $Sub_1(+3, -2)$ is learned as the transformation rule.
Figure 7.6 A visualization of removals and additions of $Sub_3$ discovered by DynGRL.

In figure 7.6, we notice $Sub_3$ is added at 81 and 86, and removed at 82 and 88. Therefore, $Sub_3$ exists at time 81, 82, 86, 87 and 88 based on the above additions and removals. The appearance at 82 and 86 shows 2 appearances with a period 4, which partially synchronizes with the last result of PBM. In other words, $Sub_3$ is added at 81, and 86 and then further augmented to become the substructure in figure 7.3 discovered by PBM. The problem is the pattern at time 78. The result of PBM shows the appearance at time 78. But the pattern does not appear at time 78 from figure 7.5. If we investigate the dynamic graph in detail, we notice the answer. A portion of $Sub_3$ is added at time 77, i.e., to time 78. But a substructure including an edge labeled as “13664” and one incident vertex labeled as “kenengish@txu.com” is added later at time 80. As we notice, the missing portion is not included in the pattern of PBM. The reason why two approaches discover different patterns is that DynGRL discovers patterns on the changing state; whereas PBM discovers patterns at time points.

Even though there can be similar results, our final transformation rules are much different from those of the PBM system. This may be a result of the Enron data being more irregular than the biological network data. The other reason is how both systems discover patterns. PBM tries to discover frequent and periodic patterns. The periodic patterns can cover all or a portion of the time series. DynGRL tries to discover patterns that cover the whole time series, so that DynGRL tries to build a rule over the entire set of discovered graph rewriting rules. If the data behaves
irregularly in one portion of the dynamic graph and regularly in another portion, PBM discovers patterns that are periodic in certain portions. But DynGRL tries to build a general rule to abstract the entire changes of a dynamic graph.

One last thing we compare to PBM is the metric to evaluate the discovered pattern. They propose \( \text{purity} \) as follows.

\[
\text{purity} = \frac{\text{periodic support}}{\text{total support in period}}
\]

\( \text{Periodic support} \) denotes the number of appearances of periodic patterns. \( \text{Total support in period} \) denotes the number of appearances of the patterns in the period that includes non-periodic appearances. If \( \text{purity} = 1.0 \), the pattern is purely periodic in the period.

The first advantage of this metric is to measure how perfectly the patterns are periodic. This metric measures the purity of the periodicity, i.e., it focuses on temporal properties, i.e., how well the patterns adhere to the learned regularity. In contrast, our \textit{coverage} and \textit{prediction} metrics focus on structural properties, i.e., how well the patterns describe the structural aspect of the changes in the dynamic graph. If we combine temporal properties with our measure, we can evaluate patterns better.

The main goal of the two approaches is to analyze dynamic graphs. PBM discovers the patterns at time points, whereas DynGRL discovers the patterns in the changing states. PBM discovers the periodic patterns that are frequent, whereas DynGRL discovers the best compressing patterns and then builds a transformation rule that attempts to cover all instances of the pattern.
7.3 Summary

In this chapter, we apply DynGRL to a different domain, Enron email data, and compare DynGRL to the similar Periodic Behavior Miner. The results of DynGRL can be similar in meaning to the results of Periodic Behavior Miner, but the systems use somewhat different ways to discover patterns.

Since dynamic network analysis is an emergent field, there are few approaches to compare, and these few systems each have their own unique view on dynamic networks and the types of patterns to be discovered from them. Because DynGRL was motivated from biological networks, there are a few disadvantages when applied to other domains. For example, a transformation rule in irregular data may not generalize well. But DynGRL has proven successful for dynamic graphs derived from biological networks. Further comparisons of DynGRL to other approaches will be fruitful to improve DynGRL’s success in other domains.
8. Conclusion

This research introduces the use of graph rewriting rules to describe structurally changing networks, and more general transformation rules abstracting the graph rewriting rules. We also present a two step algorithm to discover graph rewriting rules and transformation rules in a dynamic graph. The algorithm is evaluated using dynamic graphs representing biological networks in combination with the artificial generation, mathematical modeling and microarray data. The graph rewriting rules show how one graph is transformed into another. The learned transformation rules over the graph rewriting rules can describe repeated patterns in the series of the structural changes.

We mainly apply our approach to biological networks to help our understanding of biological processes at the system-level. Our results show important patterns in the dynamics of biological networks, for example, discovering known patterns in the various networks. Results also show the learned rules accurately predict future changes in the networks. The edges connecting the changing patterns to the unchanging portion of the graph can help us understand how the learned patterns relate to the original pathway at each time. Our approach also helps us visualize the change of subgraphs at each time to show how the networks structurally change, helps us better explore how networks change over time, and guides us to understand the structural behaviors of the dynamic network. In the biological domain, the dynamic patterns in structural changes of the network under specific conditions (e.g., infection) can provide essential information for drug discovery or disease treatments.

We also evaluate our approach using synthetic data with several variations, such as noise, graph size, and graph density ratio. We also consider the issue of search space that is related to
the number of unique labels in the dynamic graphs. Graphs with a sufficient number of unique labels lead to better performance. We apply our approach to the Enron email data and compare it to an alternative approach for finding frequent periodic patterns. The comparison highlighted the differences between the approaches, but also identified a weakness of our approach when the data exhibits irregular behavior.

For future work there are many directions we should address. First, we need to develop better representations for dynamic graphs. A sequence of snapshots might not be the best solution to represent a dynamic graph. For biological networks, we need to represent the changing amount of molecules rather than just their appearance or disappearance. For computer networks or traffic networks, we need to represent the changing flows of (network) traffic. An enhanced representation should be developed to represent numerical values with graphs.

Second, there are several challenges with respect to complexity issues. In our approach, discovery of the maximum common subgraph requires significant computational cost. In case that we know the structure of the original network, i.e., a static biological network, we can apply the unique labeling approach to discover the maximum common subgraphs. Using the sufficient numbers of unique labels enhances the performance, but we still need a better way to analyze larger graphs. Theoretically, we need to prove the bounds on the performance for our approach. There are two major operations in our approach: the maximum common subgraph issue in step 1 and graph matching issue in step 2.

For our transformation rule, we need to apply temporal mining techniques to discover temporal relations in dynamic graphs. DynGRL currently discovers structural patterns and when they appear and disappear. If we can discover temporal relations among different substructures,
i.e., substructure A always appears prior to substructure B, we can better understand dynamic graphs. In that case, we need to modify our metric to evaluate temporal issues.

The dynamic graph mining problem is an important emergent area of interdisciplinary research. A graph is a natural way to represent heterogeneous data, and dynamic graphs offer the potential to represent and find patterns in the fast-moving stream of changes in the world’s domains. Dynamic graphs that include both structural and dynamic properties are fundamental tools to describe the complex and active world. Dynamic graph mining will learn novel patterns in complex and active data that have yet to be integrated into mainstream data mining systems and will impact multiple areas. Biologists can expand their research from molecules to networks and from one time observation to long term observation at the same time. A stock price can be analyzed in multi-dimensional and relational ways, e.g., historical price and relations to other economic conditions. Ultimately, dynamic graph mining will play a fundamental role in the understanding of our complex and dynamic society.
Bibliography


6. X. Yan and J. Han. gSpan: Graph-based substructure pattern mining. *Proceedings of IEEE International Conference on Data Mining (ICDM)*. 2002.


10. X. Yan and J. Han. CloseGraph: Mining Closed Frequent Graph Patterns. *Proceedings of ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 2003.


42. —. Graph-based Temporal Mining of Metabolic Pathways with Microarray Data. Proceedings of ACM SIGKDD Conference on Knowledge Discovery and Data Mining, Workshop on Data Mining in Bioinformatics (BIOKDD). 2008.

43. —. Graph-based Data Mining in Dynamic Networks: Empirical Comparison of Compression-based and Frequency-based Subgraph Mining. Proceedings of IEEE International Conference on Data Mining (ICDM), Workshop on ADN. 2008.

44. —. Learning Patterns in the Dynamics of Biological Networks. Proceedings of the ACM SIGKDD Conference on Knowledge Discovery and Data Mining. 2009.


46. —. Graph-Based Data Mining. IEEE Intelligent Systems. 2000, Vol. 15, 2, pp. 32-41.


