EFFICIENT PARTIALLY OBSERVABLE MARKOV DECISION PROCESS BASED FORMULATION OF GENE REGULATORY NETWORK CONTROL PROBLEM

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

UTKU ERDOĞDU

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF DOCTOR OF PHILOSOPHY
IN
COMPUTER ENGINEERING

APRIL 2012

Approval of the thesis:

EFFICIENT PARTIALLY OBSERVABLE MARKOV DECISION PROCESS BASED FORMULATION OF GENE REGULATORY NETWORK CONTROL PROBLEM

submitted by **UTKU ERDOĞDU** in partial fulfillment of the requirements for the degree of **Doctor of Philosophy in Computer Engineering Department, Middle East Technical University** by,

Prof. Dr. Canan Özgen
Dean, Graduate School of **Natural and Applied Sciences** _____

Prof. Dr. Adnan Yazıcı
Head of Department, **Computer Engineering** _____

Prof. Dr. Faruk Polat
Supervisor, **Computer Engineering Department, METU** _____

Prof. Dr. Reda Alhajj
Co-supervisor, **Computer Science Dept., Univ. of Calgary** _____

Examining Committee Members:

Prof. Dr. Varol Akman
Computer Engineering, Bilkent University _____

Prof. Dr. Faruk Polat
Computer Engineering, METU _____

Assoc. Prof. Dr. Tolga Can
Computer Engineering, METU _____

Assoc. Prof. Dr. Halit Oğuztüzün
Computer Engineering, METU _____

Assist. Prof. Dr. Mehmet Tan
Computer Engineering, TOBB ETÜ _____

Date: _____

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last Name: UTKU ERDOĞDU

Signature :

ABSTRACT

EFFICIENT PARTIALLY OBSERVABLE MARKOV DECISION PROCESS BASED FORMULATION OF GENE REGULATORY NETWORK CONTROL PROBLEM

Erdoğdu, Utku

Ph.D., Department of Computer Engineering

Supervisor : Prof. Dr. Faruk Polat

Co-Supervisor : Prof. Dr. Reda Alhajj

April 2012, 130 pages

The need to analyze and closely study the gene related mechanisms motivated the research on the modeling and control of gene regulatory networks (GRN). Different approaches exist to model GRNs; they are mostly simulated as mathematical models that represent relationships between genes. Though it turns into a more challenging problem, we argue that partial observability would be a more natural and realistic method for handling the control of GRNs. Partial observability is a fundamental aspect of the problem; it is mostly ignored and substituted by the assumption that states of GRN are known precisely, prescribed as full observability. On the other hand, current works addressing partially observability focus on formulating algorithms for the finite horizon GRN control problem. So, in this work we explore the feasibility of realizing the problem in a partially observable setting, mainly with Partially Observable Markov Decision Processes (POMDP). We proposed a POMDP formulation for the infinite horizon version of the problem. Knowing the fact that POMDP problems suffer from the curse of dimensionality, we also proposed a POMDP solution method

that automatically decomposes the problem by isolating different unrelated parts of the problem, and then solves the reduced subproblems. We also proposed a method to enrich gene expression data sets given as input to POMDP control task, because in available data sets there are thousands of genes but only tens or rarely hundreds of samples. The method is based on the idea of generating more than one model using the available data sets, and then sampling data from each of the models and finally filtering the generated samples with the help of metrics that measure compatibility, diversity and coverage of the newly generated samples

Keywords: Gene Regulatory Networks, Partially Observable Markov Decision Process, Control of GRN, Gene Expression Data, Data Enrichment

ÖZ

GEN AĞLARININ KISMİ GÖZLEMLENEBİLİR MARKOV KARAR SÜREÇLERİ İLE MODELLENEREK ETKİN OLARAK KONTROLÜ

Erdoğdu, Utku
Doktora, Bilgisayar Mühendisliği Bölümü
Tez Yöneticisi : Prof. Dr. Faruk Polat
Ortak Tez Yöneticisi : Prof. Dr. Reda Alhajj

Nisan 2012, 130 sayfa

Genlerin çalışma prensiplerini inceleme gereksinimi gen düzenleyici ağların (GDA) modellenmesi ve kontrolü üzerine bilimsel çalışmalar yapılmasına yol açmıştır. GDA'ları modellemek için değişik yaklaşımlar mevcuttur ve bu yaklaşımların çoğu genler arasındaki ilişkileri matematiksel modeller vasıtasıyla modellemektedir. Problemi daha zorlaştırmasına rağmen, GDA kontrol problemlerinin daha doğal ve gerçekçi çözülebilmesi için kısmi gözlemlenebilirliğin önerilmesi gerektiğini savunuyoruz. Kısmi gözlemlenebilirlik bu problemin temel bir bileşeni olmasına rağmen çoğunlukla gözardı edilmiş ve problemin çözümünde GDA'nın tüm durumlarının mükemmel olarak bilinebileceği varsayımı yapılmıştır, yani problem tam gözlemlenebilir kabul edilmiştir. Bir yandan da literatürdeki kısmi gözlemlenebilirliği dikkate alan yöntemler sınırlı adımdan oluşan bir problem tanımı ile GDA kontrol problemini çözen algoritmalar üretmeye çalışmaktadır. Bu çalışmada problemin kısmi gözlemlenebilir bir kurgu ile tanımlanması üzerinde çalışılmakta ve Kısmi Gözlemlenebilir Markov Karar Süreçleri (POMDP) bu kurguda kullanılmaktadır. Bu çalışmada problemin sonsuz adımdan oluşan bir hali problem POMDP modeline uygun bir şekilde tanımlanarak sunulmaktadır. POMDP problem-

lerinin boyutlardan kaynaklanan problemler yaşamasından dolayı POMDP problemlerin birbirinden bağımsız parçalarını ayırıp problemi otomatik olarak parçalayan ve bu parçaları çözerek problemin çözümünü bulan bir çözüm yaklaşımı da bu çalışmada sunulmuştur. Bu çalışmada ayrıca POMDP kontrol problemine girdi olarak verilen gen ifade verisini zenginleştirmek için de bir metot sunulmaktadır. Gen ifade verilerinde binlerce gen olmasına rağmen genelde onlarca, nadir olarak da yüz mertebesinde örneklem bulunduğundan böyle bir metot gerekli ve faydalıdır. Sunulan metot varolan veri kümesini kullanarak birden fazla model oluşturur; her modelden yeni veri noktaları üretikten sonra üretilen veri noktalarını veri kümesinin uygunluğunu, farklılığını ve genişliğini ölçen metrikler yardımıyla filtreyerek kullanıma hazır bir veri kümesi oluşturur.

Anahtar Kelimeler: Gen Düzenleyici Ağlar, Kısmi Gözlemlenebilir Markov Karar Süreçleri, GDA'ların Kontrolü, Gen İfade Verisi, Veri Zenginleştirme

To my grandparents

Hatice Oruç, İbrahim Oruç, Necibe Erdoğdu, Şehmus Erdoğdu

ACKNOWLEDGMENTS

I would like to thank my supervisor Professor Faruk Polat for his constant support, guidance and friendship. It was a great honor to work with him for the last twelve years and our cooperation influenced my academical and world view highly. I also would like to thank Professor Reda Alhajj for his support and guidance on my stay at Calgary. While away from my home, he not only supported me on my research but also provided that I feel welcome and personally attended all my needs and problems. He also motivated and influenced me highly in scientific context.

A lot of people influenced and supported this work scientifically and their contribution were most valuable for me. Assist. Prof. Mehmet Tan personally supplied a lot of important material for the real kick off of this work. His ideas and support made it possible that in a short time I were able to build the frame of this work. Members of my thesis committee Professor Varol Akman and Assoc. Prof. Halit Oğuztüzün always gave valuable feedback for the progress of this work, and were not hesitant to warn me of the shortcomings or risks of my work. Assoc. Prof. Tolga Can also provided valuable feedback for the future of this research, which I very appreciate. I would also like to thank Assoc. Prof. Douglas J. Demetrick of UofC, our talks about my research not only inspired me but also helped me to understand the biological aspects of the work.

This work is also supported by TÜBİTAK-BİDEB PhD scholarship (2211) and TÜBİTAK-ARDEB Scientific and Technological Research Project Program (1001) (Project No: #110E179).

In METU-CENG a lot of colleagues provided feedback or inspiration on this work, some of them were Ahmet Saçan, Sertan Girgin, and Utku Şirin. I also would like to thank Göktürk Üçoluk, Meltem Turhan Yöndem, Cevat Şener, Onur Tolga Şehitoğlu, Hakkı Toroslu for their friendship and making me feel like home at METU-CENG. I also like to thank Professor Müslim Bozyiğit and Professor Adnan Yazıcı who provided

me comfortable working environment after my research assistantship period.

I was also lucky to find another home in Calgary, Canada. I'd like to thank Mohammed Alshalalfa, Gao Shang, Ian Reinhart and Faraz Rasheed for their contributions and feedbacks on my research. I also like to thank Keivan Kianmehr for making me feel welcome there. I would also like to thank Alhajj family, who gave me a warm welcome in Calgary.

My family also provided invaluable support for this work. I would like to thank specially to my uncle Zeki Erdoğdu. He always make me feel loved and cared. I am also thankful for all the love and support by Sevilay Erdoğdu, Emin-Esra-Dicle-Deniz Erdoğdu, Hülya Oruç, Asiye Erdoğdu, Ahmet Erdoğdu, Sebahat-Serhun-Dilhun Ayaydın, Özlem-Metin-İpek Sarıbaş, Aynur-Kenan-Cihan-Berna Baykan. My father Abdulkadir Erdoğdu was not able to see this work completed, however his memory was with me all the way to the end.

And there are a lot of people that were with me in these eight years. They defined me, they made me who I am, they are true owners of this work. It is not possible to write down why each of them is important to me and this work, because it will take more space than the work itself. So I'll just give names of some of them; Neslihan Arslan, Selma Süloğlu, Hande Çelikkanat, Ömer Nebil Yaveroğlu, Gökdeniz Karadağ, Sinan Kalkan, Burçin Sapaz, Turan Yüksel, Aykut Erdem, Erkut Erdem, Ruken Çakıcı, Oğuz Özün, Semra Doğandağ, Burcu Büyükkağnıcı, Ayça Aksu, Mehmet Ersan Topaloğlu, Servet Kızıltaş, Emre Hamit Kök, Sedar Gökbulut, Özgür Kaya, Gültekin Kurtar, Serdar Özcan, Fatih Balkan and İsmet Yalabık.

And finally very special thanks to a group of people who've taught me real meaning of friendship lately. Special thanks to the residents of the most joyful office ever, A-206 : Kerem Hadımlı, Can Hoşgör, Okan Akalın, Merve Aydınlılar and Elvan Gülen.

TABLE OF CONTENTS

LIST OF TABLES

TABLES

LIST OF FIGURES

FIGURES

LIST OF ABBREVIATIONS

BN Boolean Network

GRN Gene Regulatory Network

MDP Markov Decision Process

PBN Probabilistic Boolean Network

POMDP Partially Observable Markov Decision Process

TF Transcription Factor

UHP Unbounded Horizon POMDP

CHAPTER 1

INTRODUCTION

In today's world, it's common knowledge that DNA is the code of life. DNA is the key molecule that is used in the development and functioning of living organisms. The information stored in DNA chain governs numerous biological phenomenon in the organism; from synthesis of basic protein molecules in the cell, to determining organism's traits in their phenotype such as eye color.

Science hasn't been able to understand and explain all the details of life, however in 20th century we witnessed a lot of important breakthroughs on the function of DNA molecules and their impact on the biological structure and organization of a living organism. Complex processes in cells were analyzed and explained by the biological and genetics research. Today, the role of DNA in these complex processes is not a mystery any more.

DNA molecules is made up of nucleobases that carry the genetic code of a living organism. This code is hereditary and responsible of all the genetic traits in a living organism. Moreover, this code is basically responsible for most of the biological activities carried out in a cell. All cells in a living organism carry the same DNA sequence. However, in complex organisms groups of cells are highly specialized and are organized into tissues or organs. In this case different parts of the DNA code actively participate in biological activities and thus different kind of cells can carry out different functions.

DNA molecules participate in the activities in a cell via a basic but fundamental biochemical process called *protein biosynthesis*. In this process, the code in a DNA chain is used to synthesize vital molecules for the cell, *proteins*. Proteins are bio-

chemical compounds that participate in virtually every process within cells. They are structured as chains of simpler compounds called *amino acids*. The role of the DNA molecules in protein synthesis is that parts of the DNA sequence encodes the order of amino acids in protein chains.

In the protein biosynthesis reaction, code on the DNA is copied to similar nucleobase compounds known as RNA. Different types of RNA molecules are responsible for carrying the genetic code from cell nucleus (where DNA molecules reside) to ribosomes (where actual protein biosynthesis takes place), directing the assembly of amino acids, and actual linking of amino acids to form the chains. Thus, DNA can be seen as a blueprint for the protein biosynthesis. The process of copying the DNA code into RNA molecules is known as the *transcription process*, and is one of the main steps of protein biosynthesis.

In living species, all individuals synthesize same proteins. Thus we know that some parts of the DNA sequence is shared for all individuals of species. Similarly different genetic traits lead us to the know that some parts of the DNA sequence is unique to individuals. DNA sequence is organized in genes. Genes are the genetic unit of information on DNA code. When we express the genetic information and DNA code in terms of genes it is easier to assign functional roles to the specific parts of the codes and explain these specific parts with separate theories.

Differences in the genes of different individuals are used to explain different genetic traits displayed by them. However, different behavior and functionality on different parts of complex organisms can not be explained by different DNA codes, since all cells have the same code. For explaining this phenomenon, scientists are actively investigating how a specific gene is active in different kinds of cells in an organism.

Protein biosynthesis is a fundamental process and is regulated by numerous factors in the cell. The amount a specific protein synthesized is not fixed and determined by a complex process involving numerous external and internal molecules, enzymes and dynamics of the cell.

Some of the factors influencing the transcription process are proteins themselves; they bind to DNA sequences and promote or suppress the transcription process. They

are called *transcription factors* (TFs). Since TFs are proteins, they are the product of protein biosynthesis and other genes contain the genetic information necessary to carry on the biosynthesis of the TFs. One gene might influence the expression level of another gene via TFs. So, if we leave the external factors out and simplify our view, protein biosynthesis is controlled by interaction between a number of genes.

The amount of RNA produced by a gene is known as *gene expression level*; it is a fundamental metric for measuring how much a gene is involved in protein biosynthesis. In other words, upon a certain condition, cancer for example, genes start to behave differently due to some changes in the genetic code or due to an effect from other genes or from external conditions. Accurate identification of the genes that behave differently in diseases or in any set of conditions is essential to understand the changes in the cell as a system.

The contribution of each gene to transcription process and interactions between genes form a network of genes that effect each other. This network is called Gene Regulatory Network (GRN). Exploring the interaction between genes is an important research problem for biologists; thus computational methods are being developed for inferencing and modeling these interactions. GRN is a commonly used model for interactions between genes. It is a network structure with positive and negative links representing promoting and suppressing interactions, respectively.

In this work, we focus on the GRN control problem. The problem requires maintaining certain expression level for a single gene or certain expression levels for a group of genes. With the recent discoveries on functions of genes, we can identify harmful genes (e.g., genes causing cancer) or we can establish relationships between certain genes and certain biological conditions in organisms (e.g., genes causing insulin synthesis). By using this knowledge, it might be possible to promote or suppress a certain gene in order to prevent or promote related biological activities.

There are known TFs that can be used to control the expression levels of some genes. They are called *inputs*. However, it is not possible to control most of the genes directly. The GRN control problem can be solved by regulating known inputs in order to maintain desired expression levels for *target genes*.

Described in the literature, there are numerous mathematical models for simulating GRNs. However, not all modeling approaches are equally effective; there are some facts that could be used to give preference to certain models. For instance, the relationship between two genes is not purely functional, but stochastic. Also, the dynamic nature of the network should be represented by the model. Thus, probabilistic models, such as Probabilistic Boolean Networks (PBN) and Dynamic Bayesian Networks (DBN) are appropriate for modeling a gene regulatory network [26]. The control problem can also be modeled as a Markov Decision Process (MDP). The states and transition function of the MDP reflect network dynamics; actions and rewards represent the inputs to the network and target genes. Solving the MDP problem produces a policy which can be used for controlling the network.

One of the important aspects of the GRN control problem is that it is not possible to obtain complete state information. Expression levels of genes are typically measured imprecisely using different bio-techniques and sometimes it is not even possible to measure expression level of some genes. Due to the imprecise measurement of real states of a biological process, states can only be partially observable. Moreover, the problem of finding optimal/good policies for the control of GRNs gets complicated because of partial observability of real states.

In this work, we are proposing to model the GRN control problem in a more natural and realistic way, mainly as a Partially Observable Markov Decision Process (POMDP). In fact, there are some aspects of the problem that are not fully observable, and unfortunately all the above mentioned models assume full observability and hence simplify the problem. We argue that it is only possible to solve the GRN control problem in a realistic setting if partially observability is properly accounted in the model.

Another focus of this research is solving the POMDP problem more efficiently by taking advantage of the problem structure. GRN structure provides a natural factorization for the control problem formulation. Each gene contributes to problem dynamics to some extent. However, some of the genes are highly coupled in a cell and always interact together. These highly coupled genes are not affected by other genes to same extent.

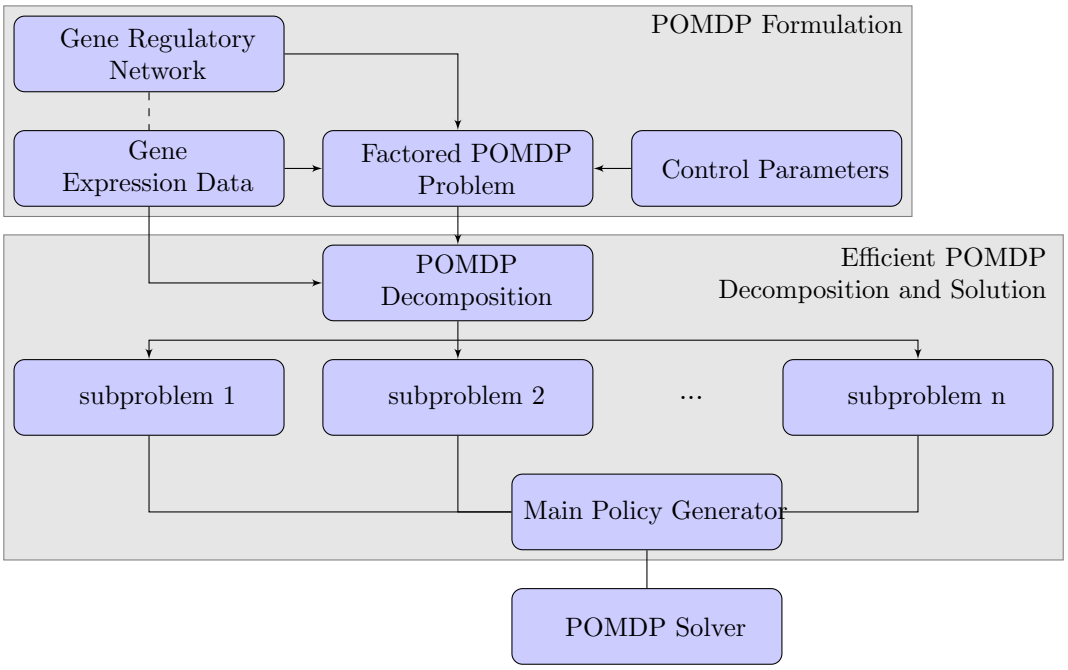

Figure 1.1: Block Diagram of the proposed POMDP Formulation and Solution Framework

When we have such a state space, it might be possible even necessary to distinguish between highly relevant and irrelevant problem components. Decomposing the problem space accordingly, we could reduce it to a smaller subproblem that can be solved in a much more efficient manner.

Having the POMDP model of GRN control problem, we further explore the possibility to decompose the problem to smaller subproblems in order to benefit from solving exponentially smaller subproblems. We are using the features of the gene expression data to guide us in the partitioning.

An abstract block diagram of the POMDP based GRN control problem modeling and solution is given in Figure 1.1. This diagram shows the main building blocks of our approach which is based on GRNs and gene expression data. It is possible to sample some gene expression data from a gene regulatory network; and similarly it is possible to infer a GRN from some given gene expression data. Our approach makes use of both alternatives; so if only one of them is available, it is necessary to infer or sample the other. In this work, we used existing gene regulatory networks inferred from real data. Also, in order to increase the variation in our experiments, we constructed

random gene regulatory networks and sampled synthetic gene expression data from these networks. We do not use an inference mechanism to generate gene regulatory networks from real data.

The first step of our approach is realizing the control problem in the POMDP framework based on gene expression data, the GRN and control parameters (i.e., input genes, target genes, and goal). We devise a method for constructing a factored representation of a POMDP problem by presenting all components of the problem in a factored way. The target of all the steps following POMDP formulation is to efficiently solve the POMDP problem. These steps make use of gene expression data for POMDP decomposition; however it is possible to adapt them to any POMDP problem by slightly modifying the decomposition scheme. The main idea in this part is decomposing the POMDP problem into subproblems. The motivation is the fact that POMDP problems are known to be intractable and can only be solved for small state spaces. In order to reduce the computational cost of the problem, we decompose the problem and solve it in terms of subproblems. This method utilizes the factored representation of the POMDP problem. The goal of the method is exploring the relationship between genes and partitioning the problem accordingly. Thus unrelated genes are classified into different groups. The produced subproblems have smaller state spaces and some of them can be completely ignored.

The main policy generator component is the part responsible for coordinating all the subproblems, solving them by interacting with a POMDP solver, and generating a policy by combining the policies generated as solutions to the subproblems. By performing the decomposition and solving the POMDP problem in terms of subproblems, what we are trying to achieve is solving the control problem in a more realistic setting without suffering from high computational cost. The method used is optimized for the GRN control problem; however, it is possible to adapt the method to similar problems in other domains, even to the general class of POMDP problems.

In this work we also present a possible pre-processing method to enrich gene expression data. For GRN control problem we are using gene expression data for our source of information on how genes interact and operate. There are different laboratory experiments designed for detecting gene expression levels. The common part of all

these techniques are that they produce data sets for further study. These data can be studied by specialists on molecular biology. However, the most important problem of this data is its size. This data is like a picture of the genes acting in the cell. Laboratory procedures produces huge amounts of data most of the time and it is rarely feasible for a human to analyze all of data produced. This is the focal point of bioinformatics, where computational methods help the preparation and analysis of the data.

Experimental techniques that identify gene expression profiles are typically working on a wide range of RNA molecules and expression profiles of thousands of genes are retrieved for a single experimental set up. However it is hard to repeat this set up to generate more samples of the gene expression profiles. To have a meaningful data set one should repeat the experiments on different biological samples, or create different conditions to repeat the experiment. Thus gene expression data sets commonly contain thousands of genes. On the contrary, the number of data points in the data set is usually very low compared to number of genes, typically around 20 to 100.

This work also addresses the sample size problem of the gene expression data. We propose a method to enrich the real life biological gene expression data by producing new samples that are compatible with the original data set and thus provide computational methods with a more reliable data set to process.

The main idea behind the data enrichment process is using multiple models for generating new data samples. Each of the models we used has distinct properties. We use

- Probabilistic Boolean Networks
- Hierarchical Markov Models
- Genetic Algorithm Model

to generate different models of GRNs. Using multiple distinct models gave us the chance to generate new samples in a robust way.

We also defined some simple statistical metrics for data generation. The metrics require new samples to be close to original sample set, but also be diverse from the

original data points. Our metrics also values new data sets to cover the data space as much as possible.

In summary, contributions of the problem solving method described in this work can be enumerated as follows:

1. We realized that the partially observability of the GRN control problem has been mostly ignored or has received little attention in the literature. Our work is among very few approaches that take partial observability into consideration; while all the other methods try to solve the GRN control problem in finite horizon, to the best of our knowledge, our work is the first attempt that considers both partial observability and infinite horizon. We focus on integrating existing POMDP solving techniques and POMDP solvers. Thus, the method proposed in this work can benefit from more efficient and robust POMDP solving algorithms. We evaluated our formulation by comparing the formulation and solution method we proposed with existing GRN control problem formulations that accompany partial observability.

2. We propose a method for decomposing and solving POMDP problems. The goal of the proposed method is to reduce the complexity of the POMDP problem solving method by utilizing domain specific properties of the problem. Thus, the proposed method contributes a novel way to handle the scalability problem of POMDP solving methods. We evaluated our formulation by comparing the solution cost and quality of a plain POMDP formulation with the decomposed version of the POMDP problem.

3. We presented a data enrichment method to enlarge gene expression data sets without changing the characteristics of the original data set. We used different generative methods to produce new samples and combined them to obtain a robust method to generate new data points that do not depend on any specific model. We formulated evaluation metrics that also guide us in selecting samples. We demonstrated our approach by running trials of experiments on a real biological data set.

The rest of this dissertation is organized as follows. Chapter 2 briefly overviews the

related work on genes, gene expression data and gene regulatory network control. Chapter 3 introduces background material on POMDP model and presents related work on factored representations of POMDP problems. Chapter 4 presents the proposed method of using POMDP in handling the GRN control problem. This chapter also covers gene expression data analysis process which is an important part of POMDP formulation and decomposition of POMDP problem. Experimental results that analyze of the success of POMDP formulation method and gene expression data analysis are also presented in this chapter. Chapter 5 describes the process developed for the decomposition of POMDP problems and presents experimental results that compare performance of plain and decomposed gene regulatory network control problem solvers. Chapter 6 presents gene expression data enrichment process, models used for this process and experimental evaluation of the process in different settings. Chapter 7 is conclusions and future research directions

CHAPTER 2

RELATED WORK

In recent years genes and gene expression have been studied extensively in classical biological and bioinformatics research based on computational methods. Experimental biological techniques devise new ways for measuring gene expression levels and we learn more about genes by analyzing these data empirically or computationally.

Gene expression levels are the fundamental information we have about the dynamics of genes and all of the computational effort, which is the focus of this study, depends mainly on gene expression data. There are different techniques and approaches to determine expression levels of genes of different organisms.

Computational models are built on top of gene expression data and they are primary used for understanding and manipulating the biological structure underlying. There are well studied computational models of gene expression and relationship between expression levels of genes. In this work some of these models are used in different contexts and also we adapted models from different domains to represent gene expression or minor concepts related to gene expression.

The focus of this work is using computational models as a tool for controlling the underlying biological structure. Controlling the underlying biological structure essentially means manipulating expression levels of genes by using a model for underlying interaction mechanism. In literature generally well known control methods are applied to the gene expression control problem with certain assumptions and restrictions. Our research inspires from some of the existing work on control of genes, however our goal is to solve the problem at a more realistic setting by lifting some of the assumptions.

Another focus of this work is solving the gene control problem in a partially observable setting. Our approach for the solution makes use of factorization of the problem, Partially Observable Markov Decision Process (POMDP) model and we propose a decomposition method on top of these general concepts. This decomposition process leads to simplification of the problem and then we have a chance to solve the problem more efficiently. Thus, this work can also be interpreted as a case study about a method that solves partially observable control problems by decomposing the problem.

This chapter presents the literature on the main research areas outlined above. A complete discussion of experimental techniques that record expression profiles of genes is beyond the scope of this work. However we provided very brief information on the data generation methods we used extensively in this work. Most of the relevant discussion about the data is about synthetic gene expression data used in computational methods and models and how this data is created and manipulated. We provided a more detailed view on the computational models and methods used for representing and manipulating gene networks. We also discuss related work on partially observable control problems briefly. Fundamental concepts on partially observable problems and POMDP model is presented in the following chapter.

2.1 Gene Expression Research and Gene Expression Data

This work is based on gene expression data. We do not restrict our selection of data or our algorithms to work on a specific gene profile data. We gathered and used different data sets from different sources. We also used synthetic data which is created by different methods.

DNA Microarray technology is an experimental technique that is used to measure gene expression profile of a cell of tissue sample, among many other possible measurements. The use of DNA Microarray technology is common since the 80s. In the last three decades, there were lots of important studies on genetics and bioinformatics to analyze genetic profiles of living organisms and DNA Microarray technique was an important tool. DNA Microarray technique allowed measurement of profiles of many genes in parallel, thus using DNA Microarray technique it was possible to generate gene expression profile of many genes in a biological sample.

Emergence of DNA Microarray and similar other techniques made it possible to gather realistic gene expression profiles of even complex organisms. Many related research areas benefited from the gene expression profiles created. One of the data sets commonly used in bioinformatics, especially in GRN control research is a metastatic melanoma data created with DNA Microarray technique [8]. We also used this data in our studies for testing the control method devised and as base data set for data enrichment methods we used. This data is widely used since Bittner et al. identified the nine genes most relevant to the cancerous behavior on samples. Identification of these nine most relevant genes allows building small and realistic models based on this selection, and allows us to evaluate any computational method without dealing with the huge data set.

In 1990s, research on genes and gene expression accelerated with the *Human Genome Project* [31, 59, 16]. Attempt to fully sequence genes of living organisms were an important step in understanding and modeling how genes operate. Yeast specimen were the first living organisms whose genome is fully sequenced. Yeast organisms have genome structures that is similar to humans and other complex organisms, but the number of genes in yeast genome is much less than the human genome. Yeast are widely studied in biology and genetics due to their genome structure and ease to work on them on the laboratory environment. There is a huge community working on yeast organisms. In this study we also made use of some gene expression profile, made available by research on the yeast organisms.

A common problem about gene expression profiles is that the data sets generally have very small sample sizes compared to very high number of genes in the data set. The number of samples is typically small for successfully applying known computational models and methods. Also the dimensionality of the data is causes high costs of computation for the methods used. These issues are also addressed in literature mostly by feature selection approaches to reduce the dimensionality of the data, thus using methods appropriate to small number of samples.

2.2 Gene Regulatory Networks

Gene expression profile is the fundamental information we have about the actual behavior of genes in organisms. However, in order to explain the interactions or to use computational methods it is more feasible to build a model of how genes interact. Gene expression data, which contains gene expression profiles can be used to build such models.

There are numerous graphical models used for modeling gene regulation in the literature. Graphs, Boolean networks [26, 3, 39, 47, 46], differential equations and stochastic master equations are commonly used as models for GRNs. In this work we mainly used Probabilistic Boolean Networks (PBN) as a computational model for GRNs. We also used graphs as a condensed and simpler representation for the relationships between genes.

All these models are derived from the gene expression data. These models are always constructed to fit the data. It is possible to use generic pattern recognition models that are not aware of the genome structure of the underlying data. However a more useful model would also include information about the relationships or dependencies between genes. In our studies we focused on PBN based GRN representations. PBN models are simplest graphical models that are rich in information about relationships between genes.

One of the major assumptions of GRN control method in this work is both gene expression data and GRN is available to all components of the method we designed. Mainly we make use of the gene expression data to formulate the partially observable control problem and as a guide to decompose the control problem. However dependency information about the genes is widely used in the decomposition phase of the method. Thus the availability of a GRN is always assumed.

This assumption is realistic, since it is always possible to infer a GRN by using the gene expression data we have. In this work we extensively inferred PBN models of GRNs. Inferring PBNs from gene expression data is carried on by using a method proposed Lähdesmäki by [30] . The method uses the coefficient of determination concept introduced by Dougherty et al. [19] and is based on the evaluating all possible

PBNs to find a model that best fits the gene expression data by evaluating models using an error function and coefficient of determination.

A Boolean Network (BN) consists of a single Boolean function $f^{(i)}_{P(g_i)}$ for each gene g_i. whose parents in the network are denoted by $P(g_i) \subseteq G$ (set of genes). For each gene in the Boolean network it is possible to find the best set of parent genes and best Boolean function by evaluating each possible set of parent genes and each possible Boolean function by using the error function

$$\epsilon(g_i, f^{(i)}_{P(g_i)}, P(g_i)) = \sum_{t=0}^{s-1} \begin{cases} 1 & f^{(i)}_{P(g_i)}(D^t(g_{i1}), D^t(g_{i2}), ..., D^t(g_{ik})) \neq D^{t+1}(g_i) \\ 0 & \text{else} \end{cases}$$

where $D^t(g)$ is value of gene g on the t^{th} sample in the gene expression data and $P(g_i) = g_{i1}, ..., g_{ik}$, parent function mentioned above. This error function simply gives the number of values in gene expression data that are not predicted correctly by the Boolean function. By using this error function, coefficient of determination can be defined as

$$\theta^{P(g_i)}_{f^{(i)}_{P(g_i)}} = \frac{\epsilon(g_i) - \epsilon(g_i, f^{(i)}_{P(g_i)}, P(g_i))}{\epsilon(g_i)}$$

$\epsilon(g_i)$ is the error of best constant Boolean function. A constant Boolean function is a function that has 0 arity and always have the same value, 0 or 1. There are two possible constant Boolean function for each gene and best constant Boolean function is found by using the error function defined above. Since the function is constant, $f^{(i)}_{P(g_i)}(D^t(g_{i1}), D^t(g_{i2}), ..., D^t(g_{ik}))$ is always equal to 0 or 1, depending on function. Thus it is straight forward to calculate error of best constant Boolean function. This error is basically the maximum possible error and coefficient of determination is the normalization of error function w.r.t. this error value.

The coefficient of determination can be used to construct a PBN. Each node in the network is based on several Boolean functions and parent sets as parameters of these functions. Coefficient of determination values both determine the closest estimator functions and parent sets; and they are also used to build a probability distribution

of chosen functions and parent sets. Constructed PBN is the closest network to fit to gene expression data, when data is conceived as a time series.

We also used a simpler graph method of GRN. In our studies we always had this graph model available for the datasets we used and for synthetic data, graph models were used as primary means to sample data. However when a graph model is not readily available, it is always possible to infer one. There are numerous methods in literature to infer GRN from gene expression data [7, 56].

2.3 Gene Expression Data Enrichment

An important problem about gene expression data is although most of the time we have excessive amounts of data, the huge size of the data is due to number of genes involved in the laboratory process. On the contrary, the number of repeated data points for a given gene is not very high, since generally each data point is produced by repeating the laboratory procedure and repetition of the procedure many times is never feasible [60].

Most of the computational methods working on this kind of data, first employ a pre-processing phase. One of the goals of pre-processing, which is specific to gene expression data, is detecting the genes relevant to the task at hand. This kind of pre-processing is almost a requirement for gene expression data, whenever possible. Without reducing the dimension of data to manageable levels it is very difficult to process the data in any way.

Such a pre-processing on data and applying statistical [25] and computational methods afterwards has been the standard way of processing gene expression data. However there is also a trend in gene expression research that challenges this convenient way. Some contemporary research, especially research on regulating gene expression levels, tries to use gene expression data as a source of information for the changes in expression levels, too. Interpreting samples of the data as elements of a time series helps us to view the data as a changing view of expression levels. This interpretation of data is analogous to a video, if we carry on the picture analogy.

However there are two problems with this interpretation. On particular problem is,

the data is not essentially a time series view of gene expression levels. The laboratory processes does only guarantee that different samples are different snapshots of the expression levels of genes. Most of the processes does not put any constraints on the uniformity or order of the samples. It turns out that this problem is not very critical, since the studies that view the data as time series produce valuable results in finding interactions between genes in time.

Another important problem of this time series interpretation is the few number of samples available. It is always a fact that sample size is very little compared to dimension in gene expression data ($n >> s$). The practical outcomes of this fact are numerous. The most important outcome is that the few number of samples affect the quality of computational methods that rely on a very descriptive sample set. Limited in number, the gene expression samples might fail to describe the expression levels of a gene properly. Even when the limited number of samples draw an accurate picture of the expression levels, since we have so little samples, confidence in the results on any computational method is limited [43, 14, 20, 27].

Indirect approaches were proposed for enrichment of data, using biological techniques. [32, 40, 58] discuss the impact of sample size to the distinction of gene expression values and try to formulate necessary sample size by using experiment parameters. The common goal in these works is to determine the number of samples necessary for the experiment to be useful, and if the number of samples are limited, then to use an appropriate experimental technique to utilize the available samples. [32] also suggests repetition of microarray experiments for increasing sample size.

In this work we've used an alternative approach that benefits from generative models for gene expression data. Our data enrichment method builds multiple models of same data set and produce new samples by using these models. By using different models, our goal was combining different expressive tools to produce a robust and rich data enrichment process.

Models used in this generative process are *Probabilistic Boolean Network* model,*Hierarchical Markov* model and *Genetic Algorithm* model.

Details of Probabilistic Boolean Networks and how they are constructed is discussed

in Section 2.2. The formulation presented there also forms the foundation of the PBN model we used for data enrichment.

Hierarchical Markov model is based on hierarchical dictionary methods and the idea of using hierarchical phrase structures proposed by Witten [61]. Hierarchical Markov models are mainly used for text mining. Witten formulates a way for storing text with a set of phrase hierarchies which are hierarchical rule sets similar to context-free grammars. They can be used to represent any string and can be used in generative manner for producing new strings. Witten also discusses different ways to deduce phrase hierarchies from a given text, both online and offline.

Thornton [57] applied the phrase hierarchies to the music generation problem with a specific emphasis on uncertainty. A very successful method for formulating phrase hierarchies would give us a model that generates musical pieces nearly identical to the original one. In order to provide some diversity, uncertain rules are introduced. An example is

$$S \rightarrow abXcY|d|e$$

The last symbol in the rule is one of the symbols Y,d or e. Providing these kind of uncertain rules increases the expression power of the phrase hierarchy and leads to generation of musical pieces that divert from the original ones.

Representing the gene expression data at hand is possible by using a similar technique. Existing data can be used to produce a set of phrase hierarchies. These phrase hierarchies can be used as a generative model for producing new data strings .Phrase hierarchies and similar computational structures are well studied in literature and it is almost an elemental problem to devise a grammar from a set of strings. However for phrase hierarchies with uncertain rules, no explicit methodology for building the phrase hierarchies is defined. Thornton [57] emphasizes a number of possible criteria for a desirable phrase hierarchy and directs the user to use any grammar construction algorithm in literature, like the ones proposed by Witten [61].

Witten [61] studies different alternative approaches for building the phrase hierarchy. Each of the approach aims to provide advantage in some specific setting, such as online processing of data, fast offline construction of phrase hierarchies or constructing a minimal phrase structure. In this work we developed a custom method that does

not emphasize on one criteria strongly. Our methodology is based on representing string in a fairly compact (not necessarily minimal) phrase hierarchy. We try to use uncertain rules as much as possible and our algorithm is based on repetitions in the gene expression data.

Genetic algorithms are well studied methods that are proposed by Holland et. al [22] and used for major genetic tasks such as DNA sequence prediction [41] or protein structure prediction [42]. However to the best of our knowledge using genetic algorithms for generating gene expression profiles is an approach that wasn't tried before.

2.4 Gene Regulatory Network Control

The GRN control problem is a popular problem that has gained more attention with the advancements in molecular biology in the last decade. The idea of intervening with the genes became possible with the research on transcription factors and other dynamics in the protein biosynthesis. From the biological point of view, simple pathways of genes were formulated for explaining certain phenomena in the cell.

Lately, the research community has focused more on generic and complex interactions between genes, and GRNs became an important tool; intervention problems have been formulated using GRNs [18, 38, 37]. Different mathematical frameworks and approaches have been used for devising mechanisms of intervention with genes in order to accomplish some non-trivial goals. There are numerous useful examples of this intervention mechanisms. For instance, targeted therapy is a cancer treatment technique based on suppressing the expression of some genes in order to prevent cancerous behavior in cells [21]. Employing mathematical models and computational techniques in undertaking the intervention problem would increase our ability to devise more complex intervention mechanisms.

Modeling the GRN control problem using PBNs is an approach widely used by the research community [26]. PBN models network dynamics and different approaches can be used for solving control problems defined on this representation. Expressing the problem in terms of an Markov Decision Process (MDP) and using general purpose

MDP solving algorithms is an approach well suited to PBN representation. Examples of methods that use Markovian and MDP concepts are the work of Shmulevich et al. [46] and the work of Datta et al. [18] who studied the dynamics of PBN model in the probabilistic context of Markov chains. The work of Pal et al. [36] explores an alternative model, namely context-sensitive PBN for the intervention problem.

Our research group has already contributed to the control and intervention of GRN by developing novel approaches based on PBN formulation of the GRN and different formulations of the intervention problem, including MDP formulation [1, 2, 53, 54, 55]. The main focus on these previous works by our group was formulating the GRN in PBN model and trying to solve the MDP problem in different settings and exploring different aspects of the problem such as finite or infinite horizon reward mechanisms, factored representations of the MDP problems, and improved modeling and solution techniques for plain and factored MDP problems. Although this approach has been effective in controlling GRNs as demonstrated at [1, 2, 53, 54, 55], the need for incorporating partial observability in the problem definition is a strong requirement for a realistic model. Accordingly, the target of our approach described in this work is to develop appropriate solutions for the problem augmented to be partially observable.

The need to cover partial observability has been realized by some other researchers; however, the problem has not yet received enough and comprehensive attention. Datta et al. [17] developed a finite horizon control algorithms for GRNs; their algorithms can work with imperfect information. Their approach was a revision of the MDP based fully observable optimization approach they used in their previous study [18]. In the previous study they modeled the GRN as a PBN and used Markovian probabilities for their control algorithm. They had modeled the optimization problem in terms of states of a GRN and Markovian probabilities specifies possible successive states given a state of GRN. Thus, it is possible to find expected reward of all states in a finite horizon by using a dynamic programming algorithm similar to value iteration.

For the partially observable version of the problem a similar dynamic programming algorithm is devised by using Markovian state transition probabilities and observation probabilities of the system. By using an observation state description based on observation history, it is possible to solve a similar dynamic programming task when the

horizon is limited. The size of the dynamic programming task increases exponentially as the horizon increases, thus the computational cost of the solution is very high and the problem is not tractable when horizon exceeds very small numbers. (Refer to Section 4.3 for an analysis of the performance of this method).

Bryce et al. [11] were first to model the GRN control problem by using POMDP model. Their approach makes use of PBN representation of the problem and their POMDP formulation is similar to MDP formulation based on PBN. Their focus was on solving the control problem in finite horizon. They did not use a POMDP solver for solving the defined POMDP problem. They applied a custom probabilistic planning algorithm for extracting an intervention mechanism for a finite horizon. Their planning algorithm is a modified version of the algorithm used by Datta et al. at [17]. They reformulated the algorithm as a search algorithm appropriate for a planning problem. Thus, they are able to prune some of the configurations of the problem, leading a more efficient solution in the given horizon. However this improvement only improves the computational cost by removing some redundant node expansions. Feasible values of horizon is increased by the improvements introduced, however the solution algorithm-is still intractable for even reasonable horizon values.

The goal of this work is to solve this control problem in a partially observable setting, not limited by a low boundary on horizon value. Our approach described in this work is a major contribution to the literature because to the best of our knowledge it is the first effort to develop a method that properly and comprehensively incorporates partial observability in handling the control of GRNs.

CHAPTER 3

BACKGROUND ON PARTIALLY OBSERVABLE CONTROL PROBLEMS

In this chapter, we present the fundamental concepts on partially observable control problems. Our main goal in this work is to model GRN control problem in a realistic setting. POMDP is the model we've used for this purpose. POMDPs are general models for representing partially observable dynamic systems. POMDP provides a convenient representation for partially observable problems and they provide a common ground for solution efforts. Solving POMDP problems is known to be hard due to a number of reasons outlined below. However there are important research efforts that try to solve real life POMDP problems.

POMDP model is important in this work since we aim to formulate the GRN control problem in POMDP setting and thus, aim to benefit from the existing efforts for solving POMDP problems efficiently. We are also proposing a method to decompose the POMDP problem so that in certain circumstances we can reduce the problem to a smaller form and solve this reduced form more efficiently. Thus POMDP model can be viewed as heart of all the research effort in this work.

3.1 Markov Decision Processes

Markov Decision Process (MDP) is a framework for studying decision making problems in probabilistic domains, proposed by Richard Bellman [4]. An MDP is based on stochastic processes, decision problems and Markovian property.

Probabilistic decision making can be modeled similar to a stochastic process. In AI perspective an agent interacts with an environment and after each action executed by the agent, environment changes in a probabilistic way.

The Markovian property indicates that, at any state, the following state will not be related to previous states. Formally,

$$\Pr(s^{k+1}|s^k, s^{k-1}, \ldots, s^0) = \Pr(s^{k+1}|s^k).$$

A decision problem with a Markovian property has distinct states and at each state the following states are determined by actions taken and world dynamics. Most of the probabilistic decision problems fall into this category. In general, for real-life problems and even for simple demonstrative problems, different histories of execution that lead to the same state have no distinct importance. Thus any solution attempt does not have to deal with execution histories.

3.1.1 Formulation

An MDP is formulated as

- Set of state space, S
- Set of actions, A
- State transition function, $T : S \times S \times A \to [0, 1]$

These three factors determine *how the world changes*. As the nature of a decision problem dictates, some states are desirable and some states are not desirable in an MDP. Reward function captures this notion.

- Reward Function, $R : S \times S \times A \to \mathbb{R}$

$T(s, s', a)$ is the probability that taking action a at state s leads to state s'. $R(s, s', a)$ is the amount of reward received *immediately* when system changes from state s to state s' via action a.

3.1.2 Deterministic Case

When state transition function is formulated as

$$T : S \times A \to S$$

or

$$T : S \times S \times A \to \{0, 1\}$$

the system is deterministic, i.e. there is a single next state for a given state-action pair. In this setting the problem reduces to a classical planning problem.

3.1.3 Acting on an MDP Problem

In AI perspective, an agent acting on an MDP reacts to the environment via S, A, T and R. At any state s, agent chooses an action among A. This action changes the world, according to T and the world enters a new state s'. Agent is noticed the reward received upon this transition. Thus, a time-tick is completed. Agent now chooses a new action for s'.

An execution history is a string of states and actions of such an agent acting on an MDP. It is a snapshot of lifetime of an agent in a given time frame.

$$H = s^0 a^0 s^1 a^1 s^2 \dots a^{n-1} s^n$$

The cumulative reward gained at such an execution history is the sum of rewards gained at each step.

$$CumulativeReward(H) = \sum_{i=1}^{i=n} R(s^{i-1}, s^i, a^{i-1})$$

The agent may decide which action to take according to its internal architecture. For MDP problems, it is assumed that the agent knows all problem dynamics (S, A, T and R) perfectly. Also it is assumed that at each time-tick, the new state s' is perfectly observable by the agent (See Section 3.2 for generalization). For MDP problems general practice is constructing a policy in form of a mapping from states to actions. Thus acting on an MDP environment is simply executing the action that the policy dictates for the current state.

3.1.4 Solving an MDP Problem

In simple terms, solving an MDP problem is finding a way to act optimally for a decision problem expressed in terms of an MDP.

The optimality is measured generally in terms of cumulative reward. Cumulative reward of an execution history might be quite precise to measure the optimality of a solution. However there are two factors that complicate this view:

1. There is an infinite number of execution histories, since there is no boundary for the length of an execution history and the formulation of MDP does not specify any concept of termination.
2. Even in the case of restricting execution histories to some fixed length, since state transition is probabilistic in nature, applying same actions starting with the same initial state might lead to different execution histories.

Since there are infinitely many execution histories, it is essential to formulate an *expected cumulative reward* of a policy δ, given a fixed execution history length t (which is commonly called *horizon*).

$$\begin{aligned} V_t^{\delta}(s) &= \sum_{s'} T(s, s', \delta(s))[R(s, s', \delta(s)) + V_{t-1}^{\delta}(s')] \quad t > 0 \\ V_1^{\delta}(s) &= \sum_{s'} T(s, s', \delta(s))[R(s, s', \delta(s))] \end{aligned}$$

$V_t^{\delta}(s)$ is the expected cumulative reward of applying policy δ for $t+1$ steps, starting at state s. The value function is defined recursively. The value of any state is the sum of expected immediate reward and expected value of the next state. The base case, value of a single step MDP, is defined only in terms of immediate reward. In this case after executing a single action no more reward is collected, i.e. the value of next state does not have any meaning for the actor.

3.1.5 Algorithms for Solving MDP Problems

3.1.5.1 Value Iteration

Value Iteration is a dynamical programming method used in learning and decision problems[51]. The pseudo code for the algorithm is given in Figure 1.

In a state oriented view of the world each state has a value, based on the reward received at that state, and possible next states. Value Iteration algorithm tries to approximate this value iteratively. A modified version of value iteration algorithms try to approximate value of state-action pairs.

In a dynamical system, value of a state is the expected cumulative reward gained after that state. It is possible to formulate this statement recursively. At a given state, there are finite number of possible next states, even we do not have a restriction for the action taken. Thus, if we know the value of all the next states, we can calculate the value of the current state.

Value of a state is denoted by V^* The main difference between V_t and V^* is V_t depends on the horizon value, i.e. a state s have different values (different expected future reward) when reached in different times in execution history. This might be the case for some control problems, however it is always possible to formulate a state description that does not cause value function to depend on time. In this case a real value function exists and maps a state to a real number, regardless of the t value.

Below formulation was used for calculating expected cumulative reward with an arbitrary horizon t:

$$V_t^{\delta}(s) = \sum_{s'} T(s, s', \delta(s))[R(s, s', \delta(s)) + V_{t-1}^{\delta}(s')]$$

We can use this formula for formulating an iterative form that converges to V^*. After V^* is found, $\delta(s)$ follows as:

$$\delta(s) = \underset{a}{\operatorname{argmax}} \sum_{s'} T(s, s', a)[R(s, s', a) + V^*(s')]$$

With combining these two ideas, we can formulate an algorithm to calculate V^*. Algorithm starts with an initial random guess of V^*, called V^0.A refined version of V^0, V^1 can be constructed by using the cumulative reward formula and using V^0 for value of the next state (operator := symbolizes *update*)

$$V^1(s) := \sum_{s'} T(s, s', \delta(s))[R(s, s', \delta(s)) + V^0(s')]$$

It's usually preferred to use a discount factor $\gamma(0 < \gamma < 1)$ to speed up the convergence to V^*:

$$V^1(s) := \sum_{s'} T(s, s', \delta(s))[R(s, s', \delta(s)) + \gamma.V^0(s')]$$

Thus the actual reward values influence V^1 more than V^0 values.

Since delta formula and this formula are similar, it is possible to rewrite this formula with eliminating delta:

$$V^1(s) := \max_a \sum_{s'} T(s, s', a)[R(s, s', a) + \gamma.V^0(s')] \qquad (0 < \gamma < 1)$$

Note that this formula should be applied to each state. At each application, we have a maximization over all actions and we have a summation over all states. Generalizing this formula we have:

$$V^{k+1}(s) := \max_a \sum_{s'} T(s, s', a)[R(s, s', a) + \gamma.V^k(s')] \qquad (0 < \gamma < 1)$$

We can apply this formula repeatedly to find better approximations to V^*. It is possible to show that this iteration converges to V^*, if it exists.

It's also possible to forget to maintain separate V^k and V^{k+1} values, or we can use a dynamic programming approach, eliminating the indexes and iterating continuously for all states:

$$V(s) := \max_a \sum_{s'} T(s, s', a)[R(s, s', a) + \gamma.V(s')] \qquad (0 < \gamma < 1)$$

When V converges to V^*, right and left sides of the formula get equal and no update is done. Thus, it is possible to terminate the iteration by examining the *update amount*:

$$\Delta := \max_s |V(s) - \max_a \sum_{s'} T(s, s', a)[R(s, s', a) + \gamma.V(s')]$$

When Δ gets low enough, algorithm is terminated. Each iteration of Value Iteration Algorithm (Do-While loop) takes $\mathcal{O}(|S|^2|A|)$ operations, due to maximization on actions and summation over next state. The number of iterations is polynomial in terms of $|S|$[24].

3.1.5.2 Policy Iteration

Policy Iteration is an algorithm based on improving a policy for a learning problem or a decision problem. The main idea behind policy iteration is, given a policy, it is possible to calculate a value function based on this policy. And given a value function, it is possible to refine the policy.

Algorithm 1: Value Iteration Algorithm

Input: Environment dynamics T, R ; Learning Rate γ;

Termination threshold Θ

Initialize V arbitrarily, e.x. $V(s) := 0$, for all s;

repeat

  $\Delta := 0$;

  foreach $s \in S$ **do**

    $v := V(s)$;

    $V(s) := \max_a \sum_{s'} T(s, s', a)[R(s, s', a) + \gamma.V(s')]$;

    $\Delta := \max(\Delta, |v - V(s)|)$;

  end

until $\Delta < \Theta$;

$\delta(s) := \operatorname{argmax}_a \sum_{s'} T(s, s', a)[R(s, s', a) + V^*(s')]$;

Policy iteration algorithm is similar to value iteration, in the sense that both algorithms try to improve the solution in iterations. Solution of policy iteration algorithm is policy itself.

Policy iteration algorithm starts with an initial arbitrary policy π. Each iteration of algorithm has two phases: *Value calculation phase* and policy improvement phase.

In the *Value calculation phase* phase the value function of the policy π is calculated. The classical value function formula is sufficient for this calculation:

$$V_\pi(s) = \sum_{s'} T(s, s', \pi(s))[R(s, s', \pi(s)) + \gamma.V_\pi(s')]$$

Writing this equation for each $s \in S$ gives us $|S|$ different equations. $V_\pi(s)$ and $V_\pi(s')$ are unknowns in these equation, for each $s, s' \in S$. We have a linear system of $|S|$ variables and $|S|$ unknowns, it is possible to find a solution. The solution for $V_\pi(s)$ gives us value function of the policy.

In the policy refinement phase, a refined version of the policy is calculated by using the value function. This phase is identical to policy extraction step of value iteration algorithm.

$$\pi'(s) := \operatorname{argmax}_a(\sum_{s'} T(s, s', a)[R(s, s', a) + \gamma.V(s')])$$

This two phases are iterated and policy $\pi'(s)$ converges to optimal policy. After the policy $\pi'(s)$ converges to optimal policy, value function calculated at the first phase converges to optimal value function. Thus the refined policy extracted at the second phase is the same policy extracted in the previous iteration. This fact gives us a sufficient termination condition. When an iteration does not refine the policy anymore, i.e. refined policy is equal to the previous one, the algorithm terminates, converging to optimal policy.

Each iteration of the algorithm has two phases. Value calculation phase is simply solving a linear system with $|S|$ equations. This phase has polynomial complexity, specifically $\mathcal{O}(|S|^3)$ if Gaussian elimination is used. Policy improvement phase has complexity $\mathcal{O}(|S|)$.

At the worst case, the policy algorithm iterates over all possible policies until finding the optimal one. Since there is $|A|^{|S|}$ possible policies, the number of policies have exponential complexity. However it is shown that the running time is pseudo polynomial[33, 24].

3.2 Partially Observable Markov Decision Processes

A POMDP is a framework for modeling probabilistic decision making problems featuring partial observability. The framework is similar to MDPs, where there is no partial observability and the state information is known perfectly. However, in the POMDP framework, it is not possible to observe the state perfectly, instead an imperfect observation is available [23, 12, 13].

The model is formulated as

- Set of state space, S
- Set of actions, A
- State transition function, $T : S \times S \times A \to [0, 1]$

These model elements are identical to elements of an MDP. The distinction between MDPs and POMDPs is in a POMDP system any executing action does not have any

knowledge about current world state $s \in S$. An agent acting on a POMDP perceives an *observation* at each time-tick, which is different than the state information.

- Set of observations, O
- Observation function, $\emptyset : S \times A \times O \rightarrow [0, 1]$

The observation function is a function of observations, states and actions, gives the probability of perceiving given observation *after* the world goes to given state with the effect of given action. The reward function is also a function of observations, reward given at any time-tick is dependent to the observation perceived.

- Reward Function, $R : S \times S \times A \times O \rightarrow \mathbb{R}$

The major impact of partially observability is that the system no longer has the Markovian property *by agent's view*. The underlying state based model is still Markovian. However observation based view of the system does not hold this property. Any observation to be perceived in the future is not only dependent to current observation, since this current observation might not uniquely map to the current state. Different observation might be perceived at any state and an observation might be perceived at a number of states.

Also since the observations are probabilistic it is not generally possible to establish a mapping between states and observations. Still some of the solution methods try to establish such a mapping, even if it is probabilistic.

An agent acting on a POMDP problem has a similar view with the MDP version. The only major difference is the agent perceives an observation at each time-tick, instead of a new world state. Thus an execution history has form

$$H = o^0 a^0 o^1 a^1 o^2 \dots a^{n-1} o^n$$

at agent's perspective.

Since the reward function is dependent to state information and state information is inaccessible to agent, it is not possible to calculate a cumulative reward in any way

except summing the actual rewards. Thus reward information might be added to execution history.

$$H = o^0 a^0 r^0 o^1 a^1 r^1 o^2 \dots a^{n-1} r^{n-1} o^n$$

Solving a POMDP problem is still about finding a way to act optimally. For POMDP problems, an agent may act upon a policy but it is not possible to construct a policy simply as a mapping.

It is possible to construct the policy as a state-action mapping by solving the MDP problem underlying. However since state information is not accessible to agent, this solution has little use from the agent's perspective. Even when such a policy is constructed, the policy should also contain a way to determine or predict the current state to use these state-action pairs.

It is also not meaningful to construct the policy as an observation-action mapping since this view does not hold the Markovian property and current observation does not implicate current state. Acting upon observations is a very primitive way to act in most decision problems and completely ignores the state of the system.

Unfortunately such simple approaches are not sufficient for overcoming the the partially observability and build policies appropriate for POMDP problems. It is necessary to construct a more sophisticated policy model for agents acting at POMDP problems.

Since there is no simple policy structure for a POMDP, it is also not possible for a simple optimality metric. The principal of optimality is same with the MDP case. A policy is optimal if and only if expected cumulative reward received by applying the policy is optimal. Calculation of expected cumulative reward should be similar to MDP case for any policy type. It is important to note that next state should be dependent to observations too.

3.2.1 Algorithms for Solving POMDP problems

3.2.1.1 History Based Algorithms

Most important challenge of solving POMDP problems is their lack of Markov property, in terms of observations and actions. However, if an execution history is available, it can be used for defining a new MDP problem in terms of history states.

In order to apply a history based algorithm, one should have sufficient history data on all the state space. Thus an history-based algorithm can be configured in a generative way, which involves producing history data from known world dynamics; or it can be configured as a learning problem.

Set of states of the new problem is strings of execution histories of the POMDP problem, containing alternating actions and observations. Set of actions of the new problem is set of action of the POMDP problem.

An important probability function is *belief state probability function*, which is a probability distribution of current state, given a history-state. This function can be computed iteratively, by using initial state distribution, state transition function of POMDP and observation distribution function of POMDP.

State transition function of the new problem is formulated by calculating the probabilities of perceiving an observation of POMDP problem, given a history-state and an action. Next state in this formulation is the history-state, concatenated with given action and observation. This transition function can be computed using transition and observation functions of POMDP with *belief state probability function.*

Reward function of the new problem is similarly probability of receiving a certain reward after executing a given action at a given history-state. This function can also be computed using original reward function and observation distribution functions of POMDP with the *belief state probability function.*

The calculation of both functions just consists of taking expected value over the belief state probability function. With all the components of an MDP system, the derived MDP problem can be solved using Value Iteration or any other MDP solving method.

This approach is successful in the sense that, even we don't know the transition and observation distributions of the POMDP, we can generate estimates from the execution history and use them in the algorithm. This algorithm can be constructed as a pure model-free method to solve a POMDP online. However it is apparent that the state space of the constructed MDP is exponential in terms of state and action spaces of the original POMDP. Thus even if this approach provides a way to successfully solve POMDP problems, it does not provide a scalable approach.

3.2.1.2 Belief State Based Algorithms

History-state based approach suffers from the huge state space of the constructed MDP problem. Using belief state probability function provides a simpler approach if the system dynamics are well known. When the transition function and the observation function are known, *belief state probability function* can be used to maintain a policy.

The general approach of all belief state based algorithms is similar to the value iteration algorithm of MDP problems. As state information, we have belief states, which is the domain of the *belief state probability function.* A belief state is a probability distribution over set of states and represents *agent's prediction on the current state he is in.*

A belief state can be formulated as a vector. The value function over belief state space is a piecewise linear function. So similarly value function can be represented as a set of vectors, too.With such vectorial representation, value of an arbitrary belief state, w.r.t to value function is the maximum value of the dot product of each vector in value function with the belief state vector.

The belief state space is the set of all $|S|-dimensional$ probability distribution vectors, and this space is continuous . Using value iteration and other similar algorithms over this state space in not straight forward but not impossible. The value functions of a POMDP based on belief states is known to be piecewise linear. Thus it is possible to use a simplified discrete version of the belief state space and formulate solution algorithms accordingly.

Different algorithms are proposed for maintaining a discrete view of the belief state

space.

- Sondik/Monahan's enumeration algorithm tries to enumerate all possible useful belief states and apply value iteration algorithm only considering these belief states [35]. The algorithm is originally proposed by Monahan, but mentioned by Sondik before in [49] and is commonly known as Sondik/Monahan's algorithm.
- Sondik's one pass algorithm tries to find a value function component for a single belief state, and then explore the belief state interval, on which this component of value function is dominating by examining possible actions and outcomes [49].
- Cheng's linear support Algorithm tries to build the value function directly [15]. The approach is similar to the one pass algorithm, however linear support algorithm relies of geometric properties of piecewise linear value function, rather than using possible actions and their outcomes.
- Witness Algorithm is similar to Linear Support Algorithm, but it also uses observations in order to simplify the calculation of real value of belief points [23].

3.3 Factoring of POMDP Problems

3.3.1 Factored MDP and POMDP

There are a number of challenges, of which all POMDP algorithms, especially belief based ones, suffer from:

- Continuity of belief space is an important challenge for belief state based POMDP algorithms. A belief state based algorithm has to find a way to deal with the continuous state space as a first task. Luckily a value function of a PODMP problem is piecewise linear and we have a simple finite mean to represent it. Thus different methods to overcome continuous nature of the problem are possible.
- *Curse of Dimensionality* is the difficulty of manipulating value function for huge problems, which have many states and actions. Value function should be calculated by considering all states and actions even in the simplest possible form.

Thus huge number of states and actions increases both the size of value function and time/space necessary to compute it. This term is attributed to Richard Bellman in his famous *Dynamic Programming* book [5].

- *Curse of Horizon* is an effect of Curse of Dimensionality. Most of the algorithms proposed for solving POMDP problems are iterative. Solving a POMDP problem optimally also requires iterations over horizon. As in MDP case, it is not possible to formulate the value function independent of the horizon since the belief state changes with time-ticks. Optimal solutions of POMDP problems always generate all possible belief states by iterating over execution horizon. Each iteration produces a policy for a specific horizon. At each iteration, horizon is increased. Since the horizon increases, each iteration deals with a more complex belief state, a more complex value function. We can simply say that Curse of Dimensionality gets worse as the horizon increases.

It is possible to formulate more efficient algorithms to overcome this issues. It is even possible to use approximation algorithms to completely evade some of the challenges presented here. As an example, there are point based algorithms that sample from belief state space uniformly. Thus it is possible to solve POMDP problems without iterating over horizon.

However one of the underlying reasons of these difficulties is that POMDP models use representations more inefficient than it should be. Most of the real life problems do not have distinct unique states, actions and observations. Most of the real world problems exhibit some structure in action space, state space and observation space [45].

Factored POMDPs could be seen as the most important attempt made to exhibit the structure of POMDP problems [9]. The idea was originally used for MDP problems. Instead of representing the problem with a set of states and actions, the factored approach represents MDP problems with *state variables* and actions. The state notion is simply the cross product of all state variables. This idea is also applicable to POMDP problems where the state space is only partially observable. Partial observability provides a natural factorization among states. This is known as mixed observability. Using factorization structure of the state space can be used to simplify

the problem [48, 29]. Our intention is to use the factored PODMP in tackling the GRN control problem.

Factored POMDP can be demonstrated by using an example . RockSample is a simple 2-D robot problem. A robot is navigating in an obstacle free environment with some rocks available to collect. Some rocks have value and others do not. The robot knows its location and the locations of rocks, but does not know the quality of samples, which makes the problem partially observable. The robot can use its long range sensors to determine the quality of any rock. The sensor reading is noisy proportional to the distance to the rock sensed. The robot have specific start and goal locations.

A RockSample problem with 4 rocks and 4×4 environment can be represented in factored form as

- $S = \{L, RQ1, RQ2, RQ3, RQ4\}$ where $Val(L) \in \{(a, b)|1 \leq a, b \leq 4\}$ and $Val(RQ1), Val(RQ2), Val(RQ3), Val(RQ4) \in \{havevalue, novalue\}$
- $A = \{ML, MR, MU, MD, SR1, SR2, SR3, SR4, G\}$
- $O = SR$ where $SR \in positive, negative$

In this representation, $L, RQ1, RQ2, RQ3$ and $RQ4$ are state variables where L is the location of agent, $RQ1, RQ2, RQ3$ and $RQ4$ are qualities of 4 rocks in the environment. Action set contains four actions for moving (MMR, MU and MD), four actions for sensing the rocks in the environment ($SR1, SR2, SR3$ and $SR4$) and a grab action to collect the rock at the agent's location. After trying to sense any rock in the environment, agent receives an observation of positive or negative. Without activating the senses, agent does not receive any observation.

This representation not only fully explains the simple problem at hand, but also separates independent parts of the problem. Different components of the problem are represented by different variables or sets of variables. These variables change over time, but again the changes in each variable are dependent to a small set of variables, such as changes in robot location are independent of rock sample quality. Thus this system can be modeled efficiently with a dynamic Bayesian network.

In Figure 3.1, three dynamic Bayesian Networks that represents the above state variables is given. Each network shows casual dependencies between variables for a specific action. In the first figure effect of moving left with action ML is demonstrated. Location of agent changes as an effect, other variables remain same. In the second figure effect of sensing rock sample 2 is demonstrated. When agent activates its sensor and senses the rock sample, the result of the observation is an observation variable denoted by SR. The result of the observation depends on the actual rock quality and the location of agent, since sensor readings are noisy with a noise function inversely proportional to the distance of agent to the sample. In the last figure, effect of grabbing a sample is demonstrated. The effect of grabbing depends on the location of the agent. Grabbed rock sample disappears and the corresponding rock quality variable is updated accordingly. Other rock quality variables are not affected.

The problem complexity is still unchanged with this representation, however since the independencies between variables is exposed in this representation it is possible to exploit these independencies and try to fully separate unrelated parts of the problem. The possibility of such a decomposition is one of the goals of this research and explored for the gene regulatory network control problem in Chapter 5.

3.4 POMDP Solving and Fundamental Representation Schema

This work focuses on presenting a POMDP formulation of GRN control problem and formulating more a efficient solution method by decomposing the problem elements. The efficient POMDP solution process we want to create does not involve solving a POMDP problem more efficiently, but involves simplifying or reducing the problem into a smaller problem, or partitioning a problem into multiple subproblems and combining the solutions to obtain a policy for the GRN control problem at hand.

Thus we did not attempt to build a custom POMDP solver in this work, but we have chosen a POMDP solver that is appropriate for our purposes. Most important requirement of this work is solving POMDP problems in factored form since all our representation depends heavily on factored forms.

We used *Symbolic Perseus* developed by Pascal Poupart for solving POMDP problems

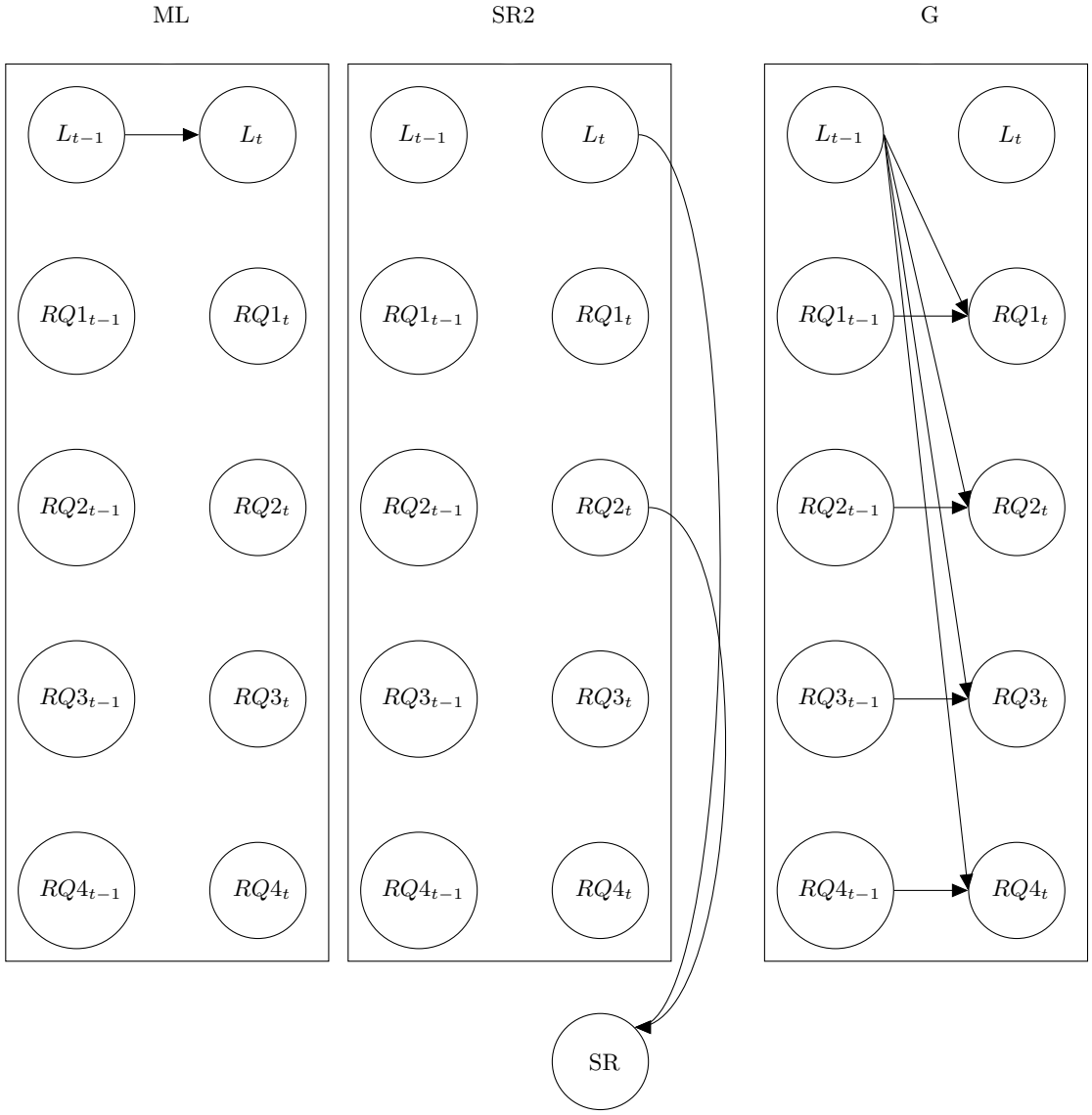

Figure 3.1: Factored dependencies of the problem for three different action: ML (Move Left), SR2 (Sense Rock Sample 2) and G (Grab)

in this work [44]. Symbolic Perseus is based on *Perseus* solver by Matthijs Spaan and Nikos Vlassis [50]. Perseus solves flat POMDP problems with a point-based value iteration algorithm. Symbolic Perseus uses a similar algorithm for solving POMDP problems and it specifically works on POMDP problems with factored representations. Thus Symbolic Perseus is well fit for solving for our gene regulatory network control problem.

Symbolic Perseus is implemented in Matlab and Java. The execution system we built deals with POMDP problems. These problems can easily be exported to a format recognizable by Symbolic Perseus and a policy can be generated by running the solver on the exported problem. Thus our execution system handles all the problem formulation, decomposition and integration tasks while using Symbolic Perseus for solving any POMDP problem it needs.

CHAPTER 4

MODELING AND SOLVING GRN CONTROL PROBLEM USING POMDP

In order to use the POMDP model for handling a partially observable GRN control problem in hand, we need to define each model element in terms of the GRN control problem. Our goal is to solve the formulated POMDP problem by using a conventional POMDP solver. However, we intend to decompose the problem into subproblems before attempting to produce a solution; and we intend to make use of the factored representation in this decomposition. Thus, the POMDP model formulation presented in this work constructs a POMDP instance in a factored form.

This chapter presents POMDP formulation used for solving GRN control problem. All components of POMDP model are addressed individually and it is discussed how each model component is expressed for gene regulatory network control problem.

The calculation of transition properties depend on the gene expression data analysis process and the exact formulation of transition function is also presented in this chapter. The goal of the data analysis is exploring similarities between expression profiles of genes, and then using these similarities both in formulation of the POMDP problem and in determining how to decompose the POMDP problem into sub problems.

Finally we present experimental results evaluating the POMDP formulation and gene expression data analysis algorithm outlined. There are two sets of experiments presented in this chapter.

In the first set of experiments we compared POMDP formulation with similar finite horizon methods. The goal of this experiment set is to verify the benefit of POMDP

formulation for the GRN control problem. We've shown that POMDP formulation we propose allows us to solve the gene regulatory network control problem for problem sizes where other finite horizon methods fail to produce any solution within reasonable computational time. In this experimental study, we compared our work with two existing work on partially observable GRN control problems: optimization based algorithm of Datta et al.[17] and POMDP based plan theoretic algorithm of Bryce et al. [10].

In the second set of experiments we tested our data analysis method with different permutations of gene expression data and with synthetic gene expression networks of different connectivity levels from sparse to very dense. The goal of this experiment set is to explore how our gene expression data analysis method is affected by the nature of the gene regulatory network and gene expression data sampled from this network.

4.1 POMDP Model Formulation for GRN Control Problem

4.1.1 States

A joint expression level vector of all genes in the network is a state in the sense of representing the gene expression control problem as a POMDP. For a flat representation of states, we need to consider all the joint values of expression levels. However, for a factored representation, it is easy to represent each gene as a state variable separately.

The most important question about accuracy of this state representation is whether all the necessary genes are included in this state vector. In the ideal sense, since we do not perfectly know all the interactions in the cell, we can never be certain that the expression level of a gene that is not included in the problem does not affect the expression levels of genes included. However in reality, for known problems it might be possible to identify related genes, and we can assume that the state vector contains all genes related to the control problem. This assumption might be crude for some gene regulation control problems, but the assumption is strong in the generalized case.

Another important point to consider is related to components of the gene expression vector. Our source of information for gene expression levels is generally experimental

data, possibly from multiple sources. The data format only affects the solution process. However, if the gene expression levels are continuous then we have a continuous state POMDP, and this should be taken into consideration. In gene regulation processes, minor differences between gene expression levels do not have major impact on the behavior of genes. And for the general case, it is possible to restrict the ideas here to discrete state POMDP problems. In this work, we assumed all gene expression levels are discretized to a multi-value set.

In all examples and experiments, we discretized gene expression data into binary values. So each state variable has two values: `off` and `on`. For a network of n genes, we have n state variables in factored form, each with 2 values; this corresponds to 2^n states for a flat representation.

4.1.2 Actions

Actions in the GRN problem are intervening with the network, setting expression levels of some inputs. These inputs might be any biological agent. As long as the input is part of the network and the relationship between the input and any of the genes can be expressed as the relationship between two genes, the biological characteristics are not significant.

In this work, we assume that some of the nodes in the GRN can be controlled directly. There are no joint actions. Each action sets the expression level of a single gene either `off` or `on`. We also assume that action `noop` is available to represent the case of not intervening with the dynamics of the network. The set of actions are specified as control parameters.

4.1.3 State Transition Function

The state transition function of the GRN problem expresses the interaction between genes. The GRN control problem itself does not dictate a transition function. If a GRN for the control problem is known (which is our assumption all through this work) the computational model for the GRN specifies dependencies between genes in the network.

As an example, PBN formulation of GRN contains probability distributions for each Boolean function a gene is associated with. By using these probabilities, it is possible to infer expected value of the whole network moving from one state to another. By combining action descriptions with these expected values, it is possible to formulate the state transition function of the POMDP model.

However, it is also possible to deduce the transition function by using the gene expression data available. We developed a method for analyzing the gene expression data and extracting state transitions by using the available analysis results. This method has two advantages : *i*) Since already computed values are used there is no need to re-process the data; *ii*) using analysis results leads to a formulation more consistent with the decomposition, which is also done using the analysis results. The details of the method are described in Section 4.2.

In the ideal case, this transition function formulation is an approximation and we can never be sure that the constructed transition function captures all the relationships between genes. However, this is a minor problem for transition functions, since we can use methods that construct a transition function that models the premises (the probabilistic graphical models of the GRN or the gene expression data) as close as possible. Assuming our construction actually builds a close model, the only reason behind missing influences in the transition function could be imprecise premises. We may have the chance for building up better premises to construct the transition function. However, if it is not the case, we have to stick to the premise in hand, and we have to model a transition function that fits the premises as much as possible.

4.1.4 Observations and Observation Function

The gene expression problem has many unknown components in reality. Most of the time, what we know about genes is one of the following:

- Gene expression profiles that describe expression levels of a set of genes as snapshots

- Correlations between expression levels of genes, derived from biological research

In order to formulate a model for GRNs, it is common practice to use any modeling structure and to assume that such structure approximates the GRN. If the dynamics of the system is accessible from the point of view of a manipulator, then the problem is fully observable. This approach is successful in modeling simple isolated gene expression networks; it is capable of producing policies for controlling the modeled networks. If we focus on representing GRNs as MDP problems, this modeling approach assumes that the GRN is fully observable. The generated policies are expressed in terms of the expression levels of genes. This view is only sound under certain assumptions. In reality, there are a number of aspects that are not fully observable

The traditional biological approach explains certain processes in the cell by pathways. Each gene causes another gene to get activated or suppressed and at the end of the chain reaction a target gene is activated. The control policies defined on simple linear pathways are simply intervening with the gene that starts the chain reaction. Since in reality there is no restriction on the interactions between genes, more complex interactions are possible. The control problem turns to be more complicated in many ways when the interactions between genes become more complex. When the problem becomes more complicated, one important point related to observability is the fact that only more complicated policies can achieve optimal or near-optimal results for the control problem. Typically, a more complicated policy requires monitoring the gene expression levels when executing the policy, and taking different actions for different situations. This is not a strong assumption. When intervening with a set of genes; generally it might not be possible to observe the activation levels simultaneously. This fact is our primary motivation for building a partially observable model of the control problem.

The gene regulatory network modeling the interactions between genes is usually deduced from gene expression data by computational methods, or empirically. Regardless of the confidence level of the method used, there is always room for error and the actual interaction scheme between genes can not be fully observed. Imprecise measurements is always a source of partial observability for control problems. An important problem about imprecise measurement is unless you are sure that you are absolutely free of imprecise measurements, you should always make room for partially observability in order to have a realistic model. This fact is our secondary motivation

for building the partially observable model.

If we concentrate on the first motivation mentioned, instead of defining the control problem as fully observable, it is more realistic to identify an observation set. This observation set is typically direct or indirect information on the expression levels of some genes. One might be the capability to monitor the expression levels of some of the genes directly, or via some marker related to the controlled genes.

Of course, for modeling the problem as POMDP, we also need to build an observation function. The observation function is directly related to the definition of the observation set. However, there is one important thing to mention here: Each observation on the GRN is related to a single gene if the observation is simply the expression level of a gene. Then, the observation function can be defined trivially. It might be possible to define more complex observations (e.g., a marker that is related to two genes and observed only when both genes are expressed) and thus the observation function might get more complicated.

In this work, we assume that a given proper subset of genes in the GRN are observable and observation related to these genes is perfect. Other genes in the GRN are not observable and we do not have any direct information on their expression levels. The set of observable genes are specified as control parameters and denoted by O, where $O \subset S$, i.e. observation set contains genes that are observable.

The observation function can be formulated by using the notation for plain representation. Remember that the formulation for state and action sets is given in the factored form. Thus the plain form of state and observation function is a vector. In state description each component of the state vector s represents expression level of a gene. Similarly observation vector contains expression levels for observable genes. In this fashion observation function can be defined as

$$\emptyset(s, a, o) = \begin{cases} 1 & \text{if } \forall g \in O, s_g = o_g, \\ 0 & else \end{cases} \tag{4.1}$$

where O is the observation set, s_g and o_g denotes the component of state and observation vectors related to gene g. The observation function states that probability of receiving observation o in state s is 1 if for all the genes in the observation set, the

observed expression level in the observation vector and the actual expression level in the state vector are matching. The observation o can not be perceived if the expression levels in observation vector and state vector do not match for any gene, thus the observation probability is 0 in this case.

4.1.5 Reward Function

Generally, the goal of controlling a gene regulatory network is to satisfy some condition over genes (e.g., prevent the expression of a gene) by intervening with the expression level of some other genes. We can express the condition to be satisfied with a Boolean function. If the state representation is discrete, it is possible to enumerate states satisfying the goal. In general, it is always possible to test a state in order to determine whether it satisfies the goal or not.

In this work, we use a reward template which is specified as follows:

$$R(s,a) = \begin{cases} \alpha & \text{if } \mathit{goal}(s) \text{ and } a = \mathit{noop}, \\ \alpha - 1 & \text{if } \mathit{goal}(s) \text{ and } a \neq \mathit{noop}, \\ 0 & \text{if } \neg\mathit{goal}(s) \text{ and } a = \mathit{noop}, \\ -1 & \text{if } \neg\mathit{goal}(s) \text{ and } a \neq \mathit{noop}. \end{cases}$$

This reward template is based on two factors: (1) whether the goal description is satisfied or not, and (2) the action executed; the *noop* action is a special *no action*. It is possible to read this template as a linear combination of two reward templates. The goal reward template assigns reward α whenever the goal description is satisfied and the action reward template assigns reward -1 whenever an action is executed. Since these two events are independent, their summation is used as the general reward template. If the goal description is satisfied, a reward of α is granted, reduced by 1 whenever an action is executed. Similarly, no reward is granted when the goal is not realized, even reduced by -1 whenever an action is executed.

In the experiments, we typically used $\alpha = 10$. The ratio of this value and the cost of an action (1 in this work) determine the significance of satisfying the goal compared to the cost of intervention; it is a parameter of the control problem that should be specified.

4.2 Gene Expression Data Analysis

The motivation behind this analysis is the importance of the gene expression data for GRN related problems. Mostly, gene expression data is the primary source of information for gene interactions. Using this data, GRNs are constructed by empirical studies or computational methods. One can use the POMDP (or any other) representation of GRNs to explore similarities between genes; however, analyzing gene expression data would give a much more precise result.

The basic analysis algorithm used in this study outputs a similarity matrix for genes. Each entry $e_{i,j}$ represents the degree of similarity in the expression levels of the two genes i and j. Genes labeled as similar are also returned as a list of pairs.

The algorithm works on the discretized version of the gene expression data. The data is padded with don't care values and all genes have the same number of samples. For each gene pair, the algorithm compares the expression levels of genes, for each sample in a fixed size window. The window is shifted all through the data circularly and for each position of window local similarity is measured. These local similarity values make up the global similarity of gene pairs.

For a single position of window, for each pair of genes, the frequency of identical samples inside the window is the local similarity of the two genes for that window location. For two genes to be locally similar this similarity measure should exceed a *local threshold*.

Similarly, if the frequency of the window positions with similar expression levels is above some *global threshold*, the two genes are labeled as globally similar. This frequency is recorded as the degree of similarity; it roughly reflects the percentage of data for which the two genes behave similarly.

The above described similarity analysis is illustrated in Figure 4.1, where we analyze the gene expression data of 3 genes; sample size of 10 and window size is 3. Values of both local and global similarity thresholds are 50%. The two expression levels `on` and `off` are shown with values 1 and 0, respectively. Table 4.1 presents the results of the analysis. Each row in the table corresponds to a step of the algorithm, and each + or -

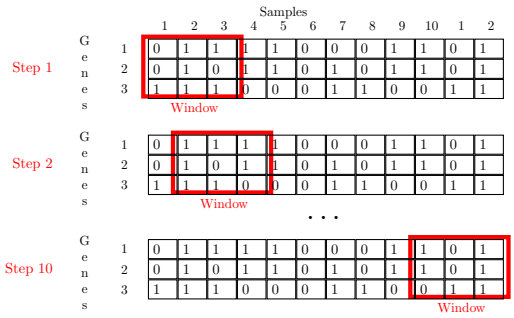

Figure 4.1: An example of shifting window for similarity analysis. Each position of the windows is a step in the algorithm. At each step, expression levels of all samples in the window are compared for each pair of genes. Window is shifted circularly.

Table 4.1: Results of the example similarity analysis carried on in Figure 4.1

	1-2	1-3	2-3
Step 1	+	+	-
Step 2	+	+	-
Step 3	+	-	-
Step 4	+	-	-
Step 5	+	-	+
Step 6	+	-	+
Step 7	+	-	-
Step 8	+	-	-
Step 9	+	-	-
Step 10	+	-	-
Global	+	-	-

sign is a positive or negative result of the local similarity test. For example, the entry + at row *Step 5* and column *2-3* indicates that the two genes 2 and 3 exhibit similar expression levels when the window is on samples 5, 6 and 7. This result follows from $gene_2[5] \neq gene_3[5]$, $gene_2[6] = gene_3[6] = 0$ and $gene_2[7] = gene_3[7] = 1$. Since $2/3 = 0.67\%$ of the samples are equivalent (which is greater than the local threshold 50%), the two genes 2 and 3 are locally similar for *Step 5*. Note that only the two genes 1 and 2 are globally similar by considering all the steps; for all other pairs, the frequencies of local similarities are less than 50%.

Figure 4.1 visualizes the search process with the sliding window. At each step, for each positioning of the window, local similarity measure between each pairs of genes

are calculated as following:

$$\gamma^p(g_i, g_j) = \sum_{k=p}^{p+WS} \begin{cases} 1 & \text{if} EL_k(g_i) = EL_k(g_j) \\ 0 & \text{else} \end{cases} \tag{4.2}$$

$$SML^p(g_i, g_j) = \frac{\gamma^p(g_i, g_j)}{WS} \tag{4.3}$$

Function γ calculates the number of identical positions for a specific positioning of the window (at $[p, p + WS]$) where WS is the window size and $EL_k(g)$ is the expression level of gene g at sample k. Thus SML function, calculating the frequency of samples with identical expression values for both genes, gives the local similarity measure between two genes for a specific positioning of the window. By using the SML function, it is possible to calculate the global similarity measure as following:

$$\delta(g_i, g_j) = \sum_{k=1}^{\#ofsamples} \begin{cases} 1 & \text{if} SML^k(g_i, g_j) > \theta_l \\ 0 & \text{else} \end{cases} \tag{4.4}$$

$$SM(g_i, g_j) = \frac{\delta(g_i, g_j)}{\#ofsamples} \tag{4.5}$$

Function δ calculates the number of window positionings where local similarity measure SML exceeds a threshold θ_l. Thus by calculating the frequency of local similarities exceeding this threshold value, we obtain similarity measure function SM which maps a pair of genes to $[0, 1]$.

Another important feature of the algorithm is compensating noise in the gene expression data. For this purpose the calculation of local similarity measures are carried on with flexible alignment of samples from different genes. For each window position, the comparison is actually performed multiple times, each time shifting the expression data of one of the genes for a small amount (typically 1 to 3 samples). The genes are labeled similar for the actual window position if any of these comparisons succeeds.

Among the two threshold values used, the first one is called *local similarity threshold*; it is used in comparing the expression values of genes for a fixed placement of the window. This threshold determines how strong we define the similarity concept for our purpose. When this threshold is low, for each placement of the window, the expression levels of the two analyzed genes are marked similar more often. When this

threshold is high, the two analyzed genes have similar expression levels for a placement of the window only when they follow exactly the same pattern.

The second threshold value is called *global similarity threshold*; it is used in deciding whether two genes are globally similar. This threshold determines the allowed degree of locality for observing the similarity between two genes in order to mark them globally similar. When this threshold is high, the similarity between the two genes should be observed in the majority of the gene expression data (i.e., for most placements of the window) for marking them similar. When this threshold is low, only a small number of local similarities might be sufficient for marking the two genes similar.

In our experiments, we selected values for both thresholds empirically by running some initial tests and we compared different values. In this work, for the gene expression data, it is important for us to have high confidence in the similarity of the expression levels. And the gene expression data is known to have very small number of samples. So, we always used very high values for both similarity thresholds, typically over 75%.

Another parameter used in the method is window size. The size of the window determines the number of samples compared at each iteration of the algorithm. In a way, window size defines the locality measure. When the window size is small, only a small number of samples are compared at each step. When the window size is large, a significant portion of the whole data is compared for each step. Here it is worth noting that we are using a twofold comparison scheme; first we compare all the values inside a sample and decide on local similarity; and second we repeat this process by sliding the window through the data and accordingly decide on global similarity. In the conducted experiments, we concluded that when both thresholds have similar values the twofold process produces similar results for any choice of the window size. It is also possible for the reader to grasp this result by considering the two extreme cases, where window size is 1 and windows size is equal to the sample size, respectively.

When the window size is equal to 1, values for a single sample are compared at each step. Two genes are marked as similar if they have the same expression level, regardless of the value of the local similarity threshold (since there is a single sample, it is not possible to calculate a frequency). For deciding whether two genes are similar globally, we test whether the frequency of the local similarity is higher than the

global similarity threshold. Consider two genes with expression data of 10110101 and 01000110. When the window size is equal to one, each position of the data produces a separate local similarity. And we decide whether they are similar or not, based on the global similarity threshold. If this threshold is 50%, we mark these two genes as not similar since only 2 of 8 local similarity checks succeed.

When the window size is equal to the sample size, there is a single step for testing local similarity, where all samples are compared (the window surrounds the whole data). Two genes are marked as similar if the frequency of the samples at which the two genes have the same expression level is larger than the local similarity threshold. This process is not repeated and two genes are marked as globally similar if they are locally similar. Consider the same genes again with expression data of 10110101 and 01000110. When the window size is equal to 8, we determine local similarity based on the whole data. If the local similarity threshold is 60%, then the whole data will not be marked as locally similar, since only 2 samples are equal. Since the only local similarity check does not succeed, two genes are marked as dissimilar.

Note that these two cases become identical in case the global similarity threshold and the local similarity threshold are equivalent. In both cases, the two genes are similar if the frequency of samples with similar expression levels for the two genes is above the threshold. This conclusion similarly follows for any window size.

Since we are using a window to compare gene expression values locally, our approach is sensitive to the order of samples. The windows used is shifted through the data circularly and theoretically it is possible that two different orderings of the samples might produce different results for the analysis we carry on.

In this work, we assumed that the gene expression data is given in a specific ordering which is treated as a problem parameter, where the sampling methods or strategy might have some meaning as in the case of time-series data. We do not attempt to reorder the samples, or consider different permutations of samples. We assume that the order in which the samples are given is better than any other alternative, unless we have prior knowledge. There are two reasons for this assumption. First, without any further knowledge on the correct ordering of data, the only optimal way to analyze the data is to repeat the analysis for all permutations of samples, which

is a computationally intractable task. Second, especially in the case of real biological data we know that the original order of the data might have a real life impact on the dataset.

Note that the effect of ordering on analysis results is also related to the window size. Consider expression data for two genes 10101010 and 11111111. For a window of size 1 and global similarity threshold 60%, these two genes would be marked as dissimilar, since none of the local similarity test succeeds. For a window size of 4 with local similarity threshold of 60% and global similarity thresholds 25%, these genes would be marked dissimilar again. When we reorder the samples of the first gene as 11110000, first analysis still produces the same result; however second analysis would mark the genes similar, since three of the local similarity tests succeed. Larger windows decrease the coincidental positive or negative effects of ordering and leads to more accurate analysis results generally, despite the fact that with larger windows sizes the impact of local similarities decrease.

In Section 4.3 we present an independent study on the order of samples. This study also shows that preserving the original ordering of data does not seem to reduce the correctness of the analysis, on the contrary original ordering produces slightly more confident results than the average case.

The complete gene expression data analysis algorithm is given in Figure 2.

The result of algorithm in Algorithm 2 is used to serve two purposes in this work, namely identifying classes of genes and constructing the transition function; details are discussed in the sequel.

4.2.1 Identifying Classes of Genes

The similarity relation calculated in the previous section is used in our work for decomposing the formulated POMDP problem into subproblems; each subproblem contains problem components closely related to each other, but not coupled with other subproblems and other problem component.

For the GRN control problem, the similarity relation provides an appropriate way for

Algorithm 2: Gene Expression Data Analysis

Input: $m \times n$ gene expression data matrix D (m genes, n samples), gene set G (m elements), window size w, window shift amount s, local similarity threshold θ_l, global similarity threshold θ_g

Result: Similarity matrix SM, similarity list SL

Initialize SM to all zero

for $i = 0$ *to* $n - w$ **do** shift window through all samples circularly

  $wp_1 \longleftarrow [i, i + W]th$ columns in D

  foreach *gene pair* $g_1, g_2 (g_1 ! = g_2)$ **do**

    for $j = -s$ *to* s **do** shift one window

      $wp_2 \longleftarrow D[i + j, i + W + j]th$ columns in D

      sim $\longleftarrow$ number of identical entries in g_1^{th} row of wp_1 and g_2^{th} row of wp_2

      if $sim > \theta_l * w$ **then**

        $SM[g_1, g_2] + = 1$

        $SM[g_2, g_1] + = 1$

        break for

      end

    end

  end

end

Divide all entries in SM by n

foreach *gene pair* $g_1, g_2 (g_1 ! = g_2)$ **do**

  if $SM[g_1, g_2] > \theta_g$ **then** add (g_1, g_2) to SL

end

guiding the decomposition. In the POMDP formulation of the problem, the expression level of each gene is a state variable, intervening and observing the expression level of each gene is a potential action and observation. So it is possible to define each subproblem in terms of a set of genes. Each subproblem defined in this fashion, is only related to the set of genes, as if these genes are closely coupled as a group; isolating this group leads to isolate a part of the control problem.

The similarity relation introduced in the previous subsection is symmetric and reflexive. To convert it into a equivalence relation we can take the transitive closure of the similarity relation. Thus, the transitive closure of the similarity relation can be used to partition the gene set into classes. Each class is simply a set of genes. Our aim is to define a POMDP subproblem for each class. In the classification process, we use the similarity list produced by the gene expression data analysis. The process for constructing classes is described in Figure 3. This algorithm is a simple transitive closure algorithm.

Chapter 5 presents how subproblems are created by using these classes.

Algorithm 3: Forming classes of genes

Input: Similarity list SL, gene set G (m elements)

Result: A partition of gene set P

$P \longleftarrow \emptyset$ **foreach** *gene* g *in* G **do**

  if *empty(P)* **then**

    $P \longleftarrow \{\{g\}\}$

  else

    foreach *class* c *in* P **do**

      if *SL contains* (g, g') *for each* $g' \in c$ **then** add g to c

    end

    if g *is not added to any class* **then** $P \longleftarrow P \bigcup \{\{g\}\}$

  end

end

Table 4.2: An Example on Inferring Conditional Probabilities Between Genes

$gene_j$	$gene_k$	$gene_i$	
		On	Off
On	On	0.85 * 0.55 = 0.47 / 0.87	0.15 * 0.45 = 0.07 / 0.13
On	Off	0.85 * 0.45 = 0.38 / 0.83	0.15 * 0.55 = 0.08 / 0.17
Off	On	0.15 * 0.55 = 0.08 / 0.17	0.85 * 0.45 = 0.38 / 0.83
Off	Off	0.15 * 0.45 = 0.07 / 0.13	0.85 * 0.55 = 0.47 / 0.87

4.2.2 Constructing the State Transition Function

The similarity matrix produced from the gene expression data analysis is used in the POMDP formulation of the GRN control problem. The entry at position (i,j) in the matrix is used as the conditional probability value $Pr(gene_i = on|gene_j = on)$. This probability value is used for constructing the transition function of the POMDP problem.

When constructing the state transition function for the factored representation, we do not need a joint probability distribution over all genes. For each state variable s, we need a probability distribution of the value of the state variable, given values of other related states at the previous state of s. By *related variables*, we mean state variables that are known to influence the next value of s.

For the GRN control problem, since each state variable is a gene, we need a probability distribution over the expression level of the gene, given the expression levels of related genes. The *related genes* can easily be found by using the GRN. For constructing the probability distributions, we assumed that the relationship between two genes is always independent of the other relationships. Thus, if a gene is influenced by multiple genes, each interaction is considered as an independent event. This assumption simplifies the calculation of the transition function. Also we assumed that genes behave according to a linear model. Thus a gene only effects other genes positively or negatively and the effected gene is influenced by a linear combination of all influences.

Our transition function formulation has two steps:

1. Gene expression data is analyzed and similarity on the behavior of genes is

extracted. Details of this analysis is presented in Section 4.2. The output of this analysis step is a function SM defined $G \times G \rightarrow [0,1]$ where G is the set of genes in GRN. This function represent the similarity of any two genes, where value 1 indicates two genes behave similarly for the data we analyzed.

2. We use the similarity measure function SM produced at the previous step and the GRN to formulate the transition function of the POMDP model. We make use of the factored representation in this step to simplify the process. Each gene is conceived as a state variable and the transition function is calculated for each gene separately.

   We assume that only a limited number of other genes directly effect the expression level of a gene. This assumption is also used in PBN model, where each node is connected to only a limited number of other nodes. For determining which genes effect a given gene directly, we refer to the gene regulatory network. A gene g_i is assumed to be directly influenced by another gene g_j if there is a directed edge (g_j, g_i) in the gene regulatory network.

   Assume that for each gene g_i, the gene is influenced by only genes $g_i^1, g_i^2, ...g_i^j$ directly. Then we formulate factored transition function T that determines the expression level of gene g_i at next time step, given expression levels of genes $g_i^1, g_i^2, ...g_i^j$ by using probabilities:

$$Pr(g_i = on | g_i^1, g_i^2, ...g_i^j) = \prod_{x=1}^{j} \begin{cases} SM(g_i, g_i^x) & \text{if } g_j^x = on, \\ 1 - SM(g_i, g_i^x) & \text{if } g_j^x = off. \end{cases}$$

$$Pr(g_i = off | g_i^1, g_i^2, ...g_i^j) = \prod_{x=1}^{j} \begin{cases} SM(g_i, g_i^x) & \text{if } g_j^x = off, \\ 1 - SM(g_i, g_i^x) & \text{if } g_j^x = on. \end{cases}$$

   For factored representation, the state space is expressed in terms of each state variable separately. Thus it is possible to directly use these probability values as descriptors of state variables. Symbolic Perseus uses a factored representation and will take care of calculating the plain transition function where necessary.

   It is possible to formalize this function in the plain POMDP notation equivalently. One need to calculate joint probability values by assuming that each probability calculated is independent of each other. This assumption is compat-

ible with our previous assumption, where each gene is only influenced by genes it is connected to in the GRN.

Calculating joint transition probabilities by using these values gives us the transition function with no intervention action *noop* performed (such as $T'(s, s')$ that does not depend to any action. If A is the action function over $S \rightarrow S$ we can derive T for other actions as:

$$T(s, noop, s') = T'(s, s') \tag{4.6}$$

$$T(s, a, s') = T'(A(s), s') \tag{4.7}$$

As we stated before, since we use the factored POMDP representation, our formulation only need to express transition probabilities for each gene and action descriptions. When action descriptions and state transitions are given together, plain representation of the state transition function can easily be generated by the planner.

To illustrate the process, assume gene i is influenced by genes j and k. The similarity matrix contains the values $SM(j, i) = 0.85$ and $SM(k, i) = 0.55$. Then the conditional probability distribution of gene i is computed as given in Table 4.2.

Note that these values are not actual probability distribution. We need to normalize these values in order to use them as transition probabilities. The values shown after the '/' in Table 4.2 are normalized values. Note that values in a row add up to 1.

4.3 Experimental Evaluation of POMDP Formulation and Data Analysis

For the experiments, we built implementations of our POMDP formulation method and two other gene regulatory network control problem settings mentioned above.

For the realization of the optimization method developed by Datta et. al. we used two alternate implementations. One of the implementations strictly follows the algorithm proposed by Datta et. al. at [17]. Gene regulatory network control problem is formalized as a finite horizon, stochastic optimization problem with imprecise knowledge about the system. A dynamic programming algorithm is used to solve the

optimization problem and extracting a policy. We designated this implementation as Optimization-1.

We also implemented the alternative realization of this algorithm, as formulated in [10] by Bryce et.al. This version of the method uses the same algorithm, however suggests an identical graph search instead of dynamic programming. Bryce et.al. used this alternative realization since this realization is more similar to their method and it is easier to compare two, then comparing a dynamic programming algorithm with a graph search algorithm. We designated this implementation as Optimization-2.

Similarly we also realized the AO* method proposed by Bryce et. al. by implementing the graph search algorithm they proposed. Alternative implementation of optimization method and the implementation of AO* method share similar components and only differ in the way they expand nodes of the search graph. This implementation is designated as AO*.

For our POMDP formulation method we used Symbolic Perseus POMDP solver for solving the constructed POMDP problem. We designated our implementation as UHP which is an acronym for Unbounded Horizon POMDP.

All four methods are evaluated with a gene regulatory network based on metastatic melanoma data [8]. Kim et.al. developed a PBN model based on this gene expression data [28]. Datta et.al formulated a control problem on a 7-gene version of this PBN formulation. The number of observed genes, intervened genes and target genes vary in the different settings of the control problem. This control problem is also used by Bryce et.al. for comparison. In this experimental evaluation, we also used this problem for comparing the performance of our POMDP formulation with these related work.

For evaluating the alternative algorithms we first solved the control problem and measured processing time and memory used in solution process. Then we measured the quality of the solution by running 100 simulations of the solution. Total simulation time and average reward are calculated after this simulation.

Since our UHP method works on an unbounded horizon, we did not solve the control problem for different horizon values as we did with other methods. For each problem setting with given number of actions, observations and target genes, we run our solu-

Table 4.3: Running time and memory usage for alternative methods with $|A| = 1$

$\|O\|$	Horizon	Solution Time (s)				Memory Used			
		OPT-1	OPT-2	AO*	UHP	OPT-1	OPT-2	AO*	UHP
1	$h = 3$	0.16	1.94	0.44	8.34	395 K	943 K	527 K	26 K
	$h = 6$	0.38	-	19.14		570 K	-	2 M	
	$h = 9$	13.72	-	-		17 M	-	-	
	$h = 12$	853.60	-	-		1 G	-	-	
	$h = 15$	-	-	-		-	-	-	
	$h = 18$	-	-	-		-	-	-	
	$h = 21$	-	-	-		-	-	-	
2	$h = 3$	0.17	46.62	6.86	8.68	404 K	5M	1 M	64 K
	$h = 6$	6.81	-	-		11 M	-	-	
	$h = 9$	3577.26	-	-		8 G	-	-	
	$h = 12$	-	-	-		-	-		
	$h = 15$	-	-	-		-	-		
	$h = 18$	-	-	-		-	-		
	$h = 21$	-	-	-		-	-		
3	$h = 3$	0.30	-	625.35	8.79	475 K	-	10 M	66 K
	$h = 6$	205.06	-	-		704 M	-	-	
	$h = 9$	-	-	-		-	-	-	
	$h = 12$	-	-	-		-	-	-	
	$h = 15$	-	-	-		-	-	-	
	$h = 18$	-	-	-		-	-	-	
	$h = 21$	-	-	-		-	-	-	
4	$h = 3$	0.41	-	-	11.03	1 M	-	-	68 K
	$h = 6$	-	-	-		-	-	-	
	$h = 9$	-	-	-		-	-	-	
	$h = 12$	-	-	-		-	-	-	
	$h = 15$	-	-	-		-	-	-	
	$h = 18$	-	-	-		-	-	-	
	$h = 21$	-	-	-		-	-	-	

tion method only once. Then the generated policy is used in simulations for different horizon values.

All software components except the POMDP formulation part of our own method work on MATLAB. We used Symbolic Perseus for solving our POMDP formulations and Probabilistic Boolean Network Toolbox for formulations of other competing methods. POMDP formulation of our method is implemented on Ruby. All experiments are carried on Intel Core i7 740QM 1.73 Ghz Processor and 6 GB memory. We used the default memory limitations of MATLAB as memory limit in our experiments. For each experimental run, we used 30 minutes as time limit.

4.3.1 Experimental Results for Evaluation of POMDP Formulation and Data Analysis

Tables 4.3-4.7 present the results of the experiments we run on four approaches. Tables 4.3-4.6 present solution time and total memory used for four competing ap-

Table 4.4: Running time and memory usage for alternative methods with $|A| = 2$

$\|O\|$	Horizon	Solution Time (s)				Memory Used			
		OPT-1	OPT-2	AO*	UHP	OPT-1	OPT-2	AO*	UHP
	$h = 3$	0.25	8.79	0.63		535 K	2 M	732 K	
	$h = 6$	1.79	-	25.91		1 M	-	3 M	
	$h = 9$	325.41	-	-		436 M	-	-	
1	$h = 12$	-	-	-	9.32	-	-	-	27 K
	$h = 15$	-	-	-		-	-	-	
	$h = 18$	-	-	-		-	-	-	
	$h = 21$	-	-	-		-	-	-	
	$h = 3$	0.24	539.92	10.41		545 K	15 M	2 M	
	$h = 6$	45.23	-	-		84 M	-	-	
	$h = 9$	-	-	-		-	-	-	
2	$h = 12$	-	-	-	8.42	-	-	-	67 K
	$h = 15$	-	-	-		-	-	-	
	$h = 18$	-	-	-		-	-	-	
	$h = 21$	-	-	-		-	-	-	
	$h = 3$	0.35	-	980.69		703 K	-	13 M	
	$h = 6$	1630.95	-	-		5 G	-	-	
	$h = 9$	-	-	-		-	-	-	
3	$h = 12$	-	-	-	10.35	-	-	-	68 K
	$h = 15$	-	-	-		-	-	-	
	$h = 18$	-	-	-		-	-	-	
	$h = 21$	-	-	-		-	-	-	
	$h = 3$	0.93	-	-		2 M	-	-	
	$h = 6$	-	-	-		-	-	-	
	$h = 9$	-	-	-		-	-	-	
4	$h = 12$	-	-	-	23.74	-	-	-	61 K
	$h = 15$	-	-	-		-	-	-	
	$h = 18$	-	-	-		-	-	-	
	$h = 21$	-	-	-		-	-	-	

Table 4.5: Running time and memory usage for alternative methods with $|A| = 3$

$\|O\|$	Horizon	Solution Time (s)				Memory Used			
		OPT-1	OPT-2	AO*	UHP	OPT-1	OPT-2	AO*	UHP
	$h = 3$	0.32	44.63	0.81		655 K	5 M	936 K	
	$h = 6$	6.55	-	25.91		6 M	-	3 M	
	$h = 9$	-	-	-		-	-	-	
1	$h = 12$	-	-	-	10.32	-	-	-	30 K
	$h = 15$	-	-	-		-	-	-	
	$h = 18$	-	-	-		-	-	-	
	$h = 21$	-	-	-		-	-	-	
	$h = 3$	0.36		13.28		690 K	-	2 M	
	$h = 6$	190.53	-	-		352 M	-	-	
	$h = 9$	-	-	-		-	-	-	
2	$h = 12$	-	-	-	10.06	-	-	-	68 K
	$h = 15$	-	-	-		-	-	-	
	$h = 18$	-	-	-		-	-	-	
	$h = 21$	-	-	-		-	-	-	
	$h = 3$	0.66	-	1121.24		971 K	-	16 M	
	$h = 6$	-	-	-		-	-	-	
	$h = 9$	-	-	-		-	-	-	
3	$h = 12$	-	-	-	13.51	-	-	-	70 K
	$h = 15$	-	-	-		-	-	-	
	$h = 18$	-	-	-		-	-	-	
	$h = 21$	-	-	-		-	-	-	
	$h = 3$	-	-	-		-	-	-	
	$h = 6$	-	-	-		-	-	-	
	$h = 9$	-	-	-		-	-	-	
4	$h = 12$	-	-	-	15.35	-	-	-	72 K
	$h = 15$	-	-	-		-	-	-	
	$h = 18$	-	-	-		-	-	-	
	$h = 21$	-	-	-		-	-	-	

Table 4.6: Running time and memory usage for alternative methods with $|A| = 4$

$\|O\|$	Horizon	Solution Time (s)				Memory Used			
		OPT-1	OPT-2	AO*	UHP	OPT-1	OPT-2	AO*	UHP
1	$h = 3$	0.49	177.07	1.02	11.61	1 M	9 M	1 M	31 K
	$h = 6$	-	-	44.10		-	-	4 M	
	$h = 9$	-	-	-		-	-	-	
	$h = 12$	-	-	-		-	-	-	
	$h = 15$	-	-	-		-	-	-	
	$h = 18$	-	-	-		-	-	-	
	$h = 21$	-	-	-		-	-	-	
2	$h = 3$	0.93	-	19.82	12.12	2 M	-	3 M	70 K
	$h = 6$	190.53	-	-		352 M	-	-	
	$h = 9$	-	-	-		-	-	-	
	$h = 12$	-	-	-		-	-	-	
	$h = 15$	-	-	-		-	-	-	
	$h = 18$	-	-	-		-	-	-	
	$h = 21$	-	-	-		-	-	-	
3	$h = 3$	1.47	-	-	12.67	3 M	-	-	71 K
	$h = 6$	-	-	-		-	-	-	
	$h = 9$	-	-	-		-	-	-	
	$h = 12$	-	-	-		-	-	-	
	$h = 15$	-	-	-		-	-	-	
	$h = 18$	-	-	-		-	-	-	
	$h = 21$	-	-	-		-	-	-	

proaches, for action sets of different cardinalities (denoted by $|A|$). Each table for an action set presents solution time and memory values for different cardinalities of observation set (denoted by $|O$) and different horizon values (denoted by h). Table 4.7 present the average reward collected, when the policy generated by each approach is tested on a simulation run. This table again presents results for different cardinalities of action set and observation set, and different horizon values.

In all experiments, solution time is the time to build the problem description and generate the policy. As the number of actions and observations increase, the problem gets more complicated and solution time typically increases for all methods in our experiment. This increase is very fast for both optimization implementations and AO* algorithm. None of these algorithms produce policies in given time with $h > 12$, even in the simple setting of single action and single observation. However, UHP method calculates the policy without considering the horizon value, thus the solution time does not exceed 30 seconds even in the most complex problem setting and high horizon values ($h = 21$)

Figure 4.2 shows the solution times as the horizon increases. UHP method have a constant solution time w.r.t. horizon and it's insignificant compared to other methods. Other methods exceed execution time limit for horizon values larger than 3-5.

Table 4.7: Average Reward Gained for alternative methods

$\|A\|, \|O\|$	Horizon	Average Reward				$\|A\|, \|O\|$	Average Reward			
		OPT-1	OPT-2	AO*	UHP		OPT-1	OPT-2	AO*	UHP
$\|A\| = 1$ $\|O\| = 1$	$h = 3$	8.7	14.9	12.6	13.28	$\|A\| = 3$ $\|O\| = 1$	9.2	12.3	12.2	13.28
	$h = 6$	19.5	-	23.94	21.65		24.9	-	32.83	21.65
	$h = 9$	46.4	-	-	28.12		-	-	-	28.12
	$h = 12$	64.5	-	-	33.22		-	-	-	33.22
	$h = 15$	-	-	-	37.65		-	-	-	37.65
	$h = 18$	-	-	-	40.80		-	-	-	40.80
	$h = 21$	-	-	-	42.79		-	-	-	42.79
$\|A\| = 1$ $\|O\| = 2$	$h = 3$	9.8	12.3	12.54	13.44	$\|A\| = 3$ $\|O\| = 2$	15	-	14.7	13.44
	$h = 6$	21.6	-	-	23.50		30.9	-	-	23.50
	$h = 9$	40.3	-	-	31.34		-	-	-	31.34
	$h = 12$	-	-	-	36.94		-	-	-	36.94
	$h = 15$	-	-	-	40.83		-	-	-	40.83
	$h = 18$	-	-	-	43.86		-	-	-	43.86
	$h = 21$	-	-	-	45.90		-	-	-	45.90
$\|A\| = 1$ $\|O\| = 3$	$h = 3$	13.6	-	12.08	13.28	$\|A\| = 3$ $\|O\| = 3$	15.8	-	15.82	13.28
	$h = 6$	33.4	-	-	23.40		-	-	-	23.40
	$h = 9$	-	-	-	30.93		-	-	-	30.94
	$h = 12$	-	-	-	37.39		-	-	-	37.39
	$h = 15$	-	-	-	41.64		-	-	-	41.63
	$h = 18$	-	-	-	44.47		-	-	-	44.47
	$h = 21$	-	-	-	46.60		-	-	-	46.60
$\|A\| = 1$ $\|O\| = 4$	$h = 3$	11.4	-	-	13.28	$\|A\| = 3$ $\|O\| = 4$	-	-	-	13.28
	$h = 6$	-	-	-	23.15		-	-	-	23.15
	$h = 9$	-	-	-	30.10		-	-	-	30.10
	$h = 12$	-	-	-	35.45		-	-	-	35.45
	$h = 15$	-	-	-	39.12		-	-	-	39.12
	$h = 18$	-	-	-	40.32		-	-	-	40.31
	$h = 21$	-	-	-	41.27		-	-	-	41.27
$\|A\| = 2$ $\|O\| = 1$	$h = 3$	13.9	13.2	11.4	13.28	$\|A\| = 4$ $\|O\| = 1$	14.5	13.1	11.24	13.28
	$h = 6$	20.9	-	24.12	21.65		-	-	18.75	21.65
	$h = 9$	42.9	-	-	28.12		-	-	-	28.12
	$h = 12$	-	-	-	33.22		-	-	-	33.22
	$h = 15$	-	-	-	47.65		-	-	-	37.65
	$h = 18$	-	-	-	40.80		-	-	-	40.80
	$h = 21$	-	-	-	42.79		-	-	-	42.79
$\|A\| = 2$ $\|O\| = 2$	$h = 3$	13	5.12	6.52	13.44	$\|A\| = 4$ $\|O\| = 2$	10.5	-	10.6	13.44
	$h = 6$	-0.4	-	-	23.50		-	-	-	23.50
	$h = 9$	-	-	-	31.34		-	-	-	31.34
	$h = 12$	-	-	-	36.94		-	-	-	36.94
	$h = 15$	-	-	-	40.83		-	-	-	40.83
	$h = 18$	-	-	-	43.86		-	-	-	43.86
	$h = 21$	-	-	-	45.90		-	-	-	45.90
$\|A\| = 2$ $\|O\| = 3$	$h = 3$	14.2	-	12.3	13.28	$\|A\| = 4$ $\|O\| = 3$	12	-	-	13.28
	$h = 6$	28	-	-	23.40		-	-	-	23.40
	$h = 9$	-	-	-	30.94		-	-	-	30.94
	$h = 12$	-	-	-	37.39		-	-	-	37.39
	$h = 15$	-	-	-	41.64		-	-	-	41.64
	$h = 18$	-	-	-	44.47		-	-	-	44.47
	$h = 21$	-	-	-	46.60		-	-	-	46.60
$\|A\| = 2$ $\|O\| = 4$	$h = 3$	13.1	-	-	13.28	$\|A\| = 5$ $\|O\| = 1$	14.6	13.2	11.18	13.28
	$h = 6$	-	-	-	23.15		-	-	36.8	21.64
	$h = 9$	-	-	-	30.10		-	-	-	28.12
	$h = 12$	-	-	-	35.45		-	-	-	33.22
	$h = 15$	-	-	-	39.12		-	-	-	37.65
	$h = 18$	-	-	-	40.32		-	-	-	40.80
	$h = 21$	-	-	-	41.27		-	-	-	42.79
						$\|A\| = 5$ $\|O\| = 2$	13.3		12.98	13.44
							-	-	-	23.50
							-	-	-	31.34
							-	-	-	36.94
							-	-	-	40.83
							-	-	-	43.86
							-	-	-	45.90

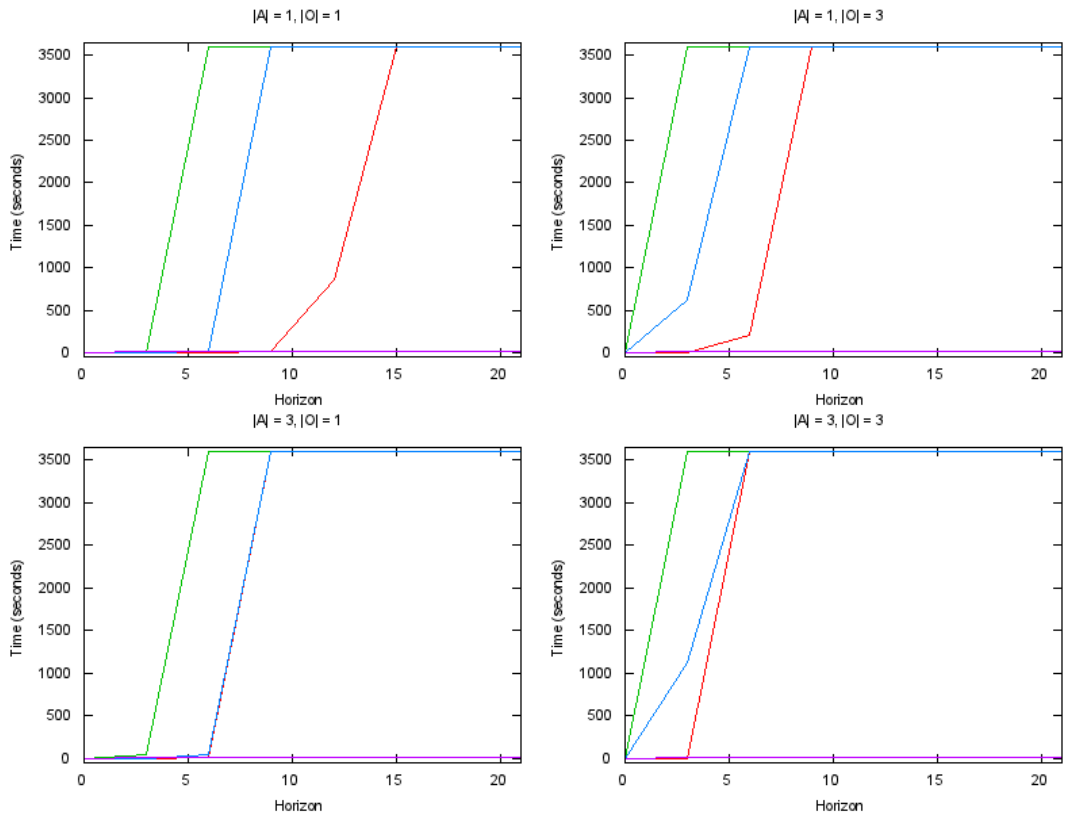

Figure 4.2: Change of solution times for different implementations when horizon value increases. Graphs are plotted for different $|A|$ and $|O|$ values.

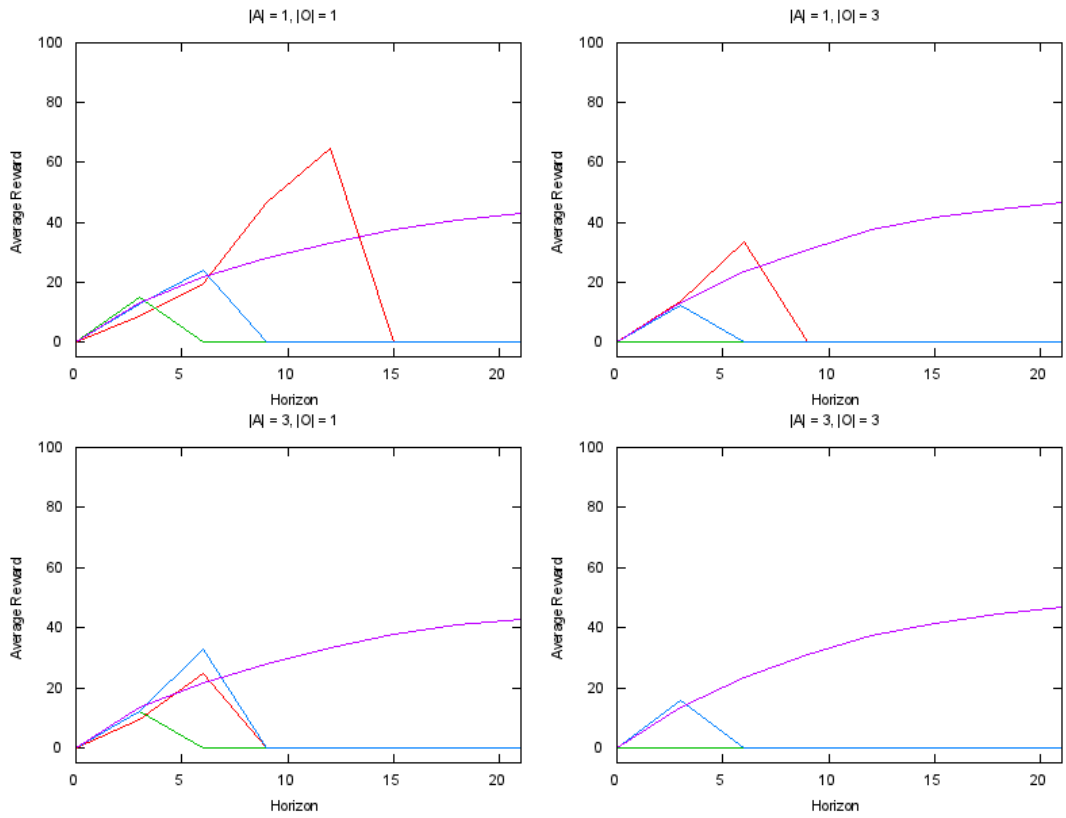

Figure 4.3: Change of average reward for different implementations when horizon value increases. Graphs are plotted for different $|A|$ and $|O|$ values.

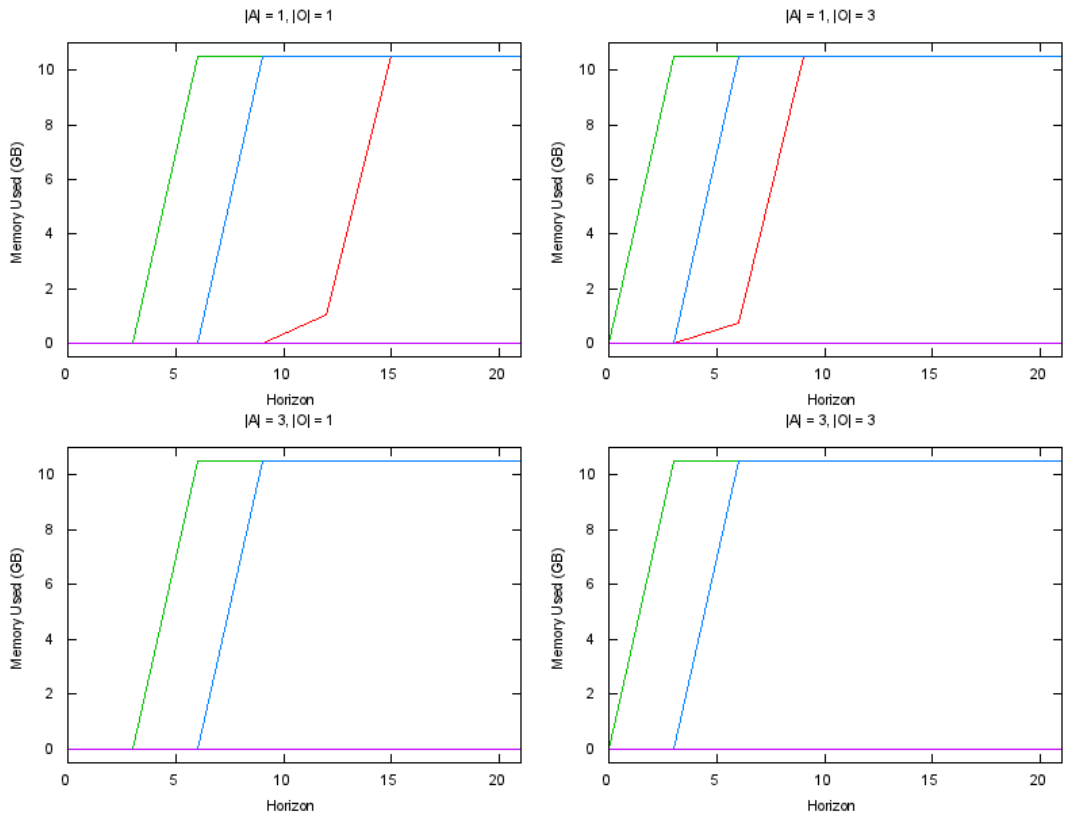

Figure 4.4: Change of memory used for different implementations when horizon value increases. Graphs are plotted for different $|A|$ and $|O|$ values.

Memory column in experimental results also give hints on the scalability of the approaches. OPTIMIZATION-1 implementation scales better than OPTIMIZATION-2 implementation, because it extensively uses dynamic programming tables that organize the solution space more efficiently. These dynamic programming tables might get as big as 8 GBs. When the same algorithm is implemented with search graphs, the utilization of memory decreases and the scalability drops significantly. AO* algorithm is similar to the OPTIMIZATION-2 implementation in memory use. However, the most important aspect of this algorithm is the pruning parts of the state space, which can be observed in memory usage values. UHP method uses significantly less memory than all other methods considered.

Figure 4.4 shows the memory usage of the methods. In this figure, cases where time limit is exceeded are also shown on maximum memory usage for smoother graphs. Reader can refer to Tables 4.3-4.6 to see the actual memory usage in details, which is consistent with this choice. Behavior of memory usage is very similar to execution time. Our method uses minimal memory, while for larger horizon values all other methods either exceed memory limit, or failed to terminate in time limit.

The average reward gained at simulation is similar for all approaches. Note that the average is taken on the number of runs, not on the length of horizon and thus as the horizon length increases the average reward increases too.

Figure 4.3 shows the reward collected by each method. For cases where a method exceeds time of memory limit no reward is collected. All methods collect similar amounts of rewards for small horizon values. However UHP method collect consistently increasing amount of reward for larger horizons, while other methods fail to complete in time or memory limit.

Most significant outcome of the experiments is the comparison of the scalability of methods. Optimization methods and AO* algorithm scale poorly as the horizon increases. Especially the optimization methods fail to produce any results as $A \cup O$ gets bigger. Figure 4.5 displays scalability of tested algorithms with different values of $|A|$ and $|O|$. The scalability value shown is the maximum horizon value for which each method were able to produce an answer without violating time or memory limit.

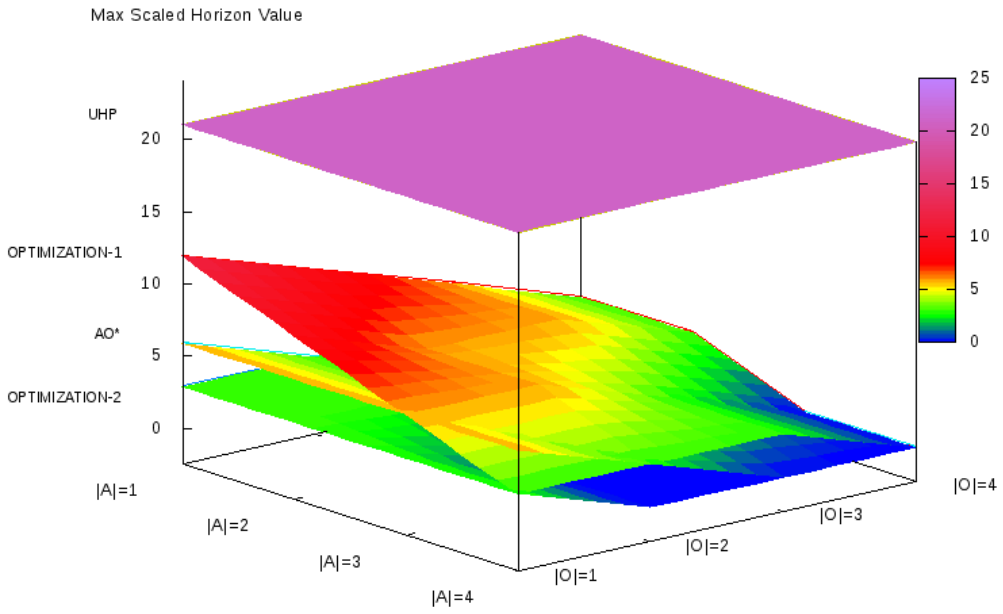

Figure 4.5: A surface plot of scalability of four alternative approaches. Surfaces are also plotted as color map to improve visual distinction. XY-axis indicates $|A|$ and $|O|$ values. Z-axis indicates the horizon value. Each surface are drawn at maximum horizon value it scales to.

For small values of $|A|$ and $|O|$ all algorithms scale around horizon values of 5 or 10. AO* method scales better than OPTIMIZATION-2 method. However OPTIMIZATION-1 implementation of the same algorithm outperforms AO*. This is due to our choice of implementation environment, MATLAB. On MATLAB OPTIMIZATION-1 implementation uses arrays for dynamic programming tables and linear algebra routines for calculating necessary probability distributions. Since these are the programming tools that MATLAB can handle very efficiently scalability of OPTIMIZATION-1 increases drastically. Conversely, MATLAB does not offer any efficient implementation for general purpose computing tools, such as graphs that are used extensively by OPTIMIZATION-2 and AO*. It is safe to predict that on a general purpose programming environment, OPTIMIZATION-1 and OPTIMIZATION-2 implementations would scale to close horizon values, while AO* would scale better than both of them.

Our UHP method is indifferent to the horizon value of the problem so it always scales up to maximum horizon value used, which is 21 for this experimental setting. However our UHP approach does not sacrifice from the solution quality as average rewards clearly depicts. The average reward received is not significantly less than the other three approaches even for high horizon values.This is a very important result, since other approaches, especially OPTIMIZATION-1 and OPTIMIZATION-1 theoretically generate the best policy for the GRN control problem. The actual average rewards still do not converge to a single value due to uncertainties involved in the problem. However average reward collected by these approaches is the expected average reward to be collected when GRN is controlled optimally. Our approach makes use of point-based value iteration algorithm for solving POMDP problems. This algorithm is based on approximating the whole value function by sampling points on the belief state space, and theoretically is a sub-optimal solution algorithm for finite horizon. However experimental results show that our formulation and solution method collects rewards close to other optimal solution methods.

4.3.2 Experiments on the Influence of Gene Expression Data and Gene Regulatory Network

This section presents two sets of experimental results for evaluating the impact of variations in gene expression data and gene regulatory network to data analysis method we proposed. In order to understand and improve the gene expression data analysis, we felt the necessity of exploring the impact of fundamental features of the gene expression data and gene regulatory network to the data analysis algorithm. These test are designed to fulfill this need.

4.3.2.1 Experiments on Data Order

As Section 4.2 discusses, the order of samples in our data set has an impact on our window based gene expression data analysis algorithm. A trivial way to reduce the impact of ordering, one can theoretically carry on the analysis for all permutations of the data and combine the results as we combined the local results to obtain global similarity values. However this approach is computationally intractable. It is impossible to carry on even the simplest analysis on all different permutations, since the number of permutations of the data grows with the order of factorial function.

Thus we designed a simpler test that analyzes an arbitrary number of permutations of our data. We used the metastatic melanoma gene expression data we used in previous experiments. We reordered or data samples 10000 times randomly and carried on our gene expression analysis for each of the 10000 permutations. We obtained a partitioning of genes for each case.

For our original analysis, which is based on the original ordering of the data, we calculated the average Hamming Distance between our analysis result and the analysis results from 10000 permutations.

Then we chose 50 permutations among 10000. Each of the 50 permutations are used as an alternative ordering of samples on which gene expression analysis can be applied. For each of the 50 samples we calculated the average Hamming Distance between the analysis result of the permutation and all 10000 analysis results. The results of all 51

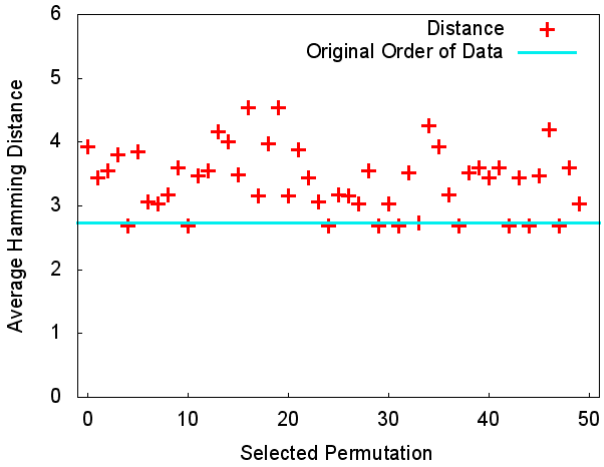

Figure 4.6: Comparison of solution similarity for different permutations of data samples

cases are shown in Figure 4.6

Since our data set contains 7 genes and we relabel all the analysis results, maximum Hamming Distance between two analysis results are 6 (partition of the first gene is always labeled as the first group).

The results show that for most of the possible ordering of data, An average Hamming Distance between 3 and 4 exists between the analysis result and all possible analysis results. Only a minority of permutations lead to Hamming Distances greater than 4 or less than 3.

This shows that a different permutation of data samples might produce different analysis results. However different permutations only show slight deviation from an average case where half of the other possible analysis results are equivalent or very close to the selected permutation and the remaining other half gives different analysis results.

Moreover our original analysis based on the original ordering of data has an average Hamming Distance of less than 3 to all 10000 permutations (shown with the horizontal line in the Figure). Thus we experimentally verified that analysis result we obtained using the original ordering of the data has the similar statistical properties to a different analysis result produced with a different ordering of data. We can conclude that using the original ordering of the data does not deviate the analysis results much, in fact the analysis results in this case are closer to the other possible results above

average.

4.3.2.2 Experiments on Network Connectivity

Another experiment set we conducted attempts to measure the effect of network connectivity to our method.

The main focus of our method is to analyze the gene expression data, explore and group genes according to their impact on the control problem to be solved. Genes that have little influence on the control problem are eliminated totally. According to this perspective it is expectable for our approach to identify such genes more frequently in a sparse network than a dense one. In a dense network each gene is connected to more genes and thus eliminating each gene has a bigger impact on the whole network.

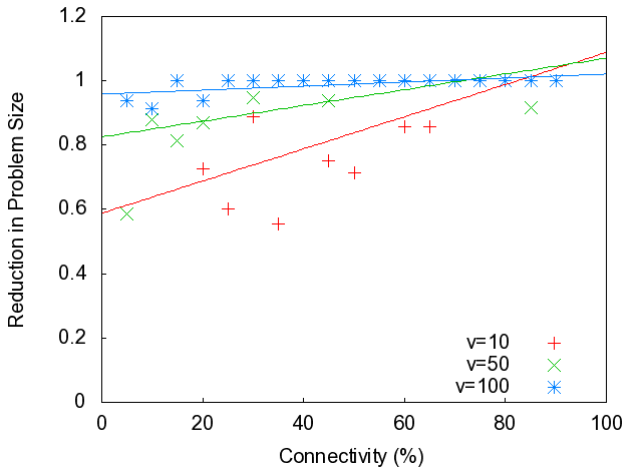

Figure 4.7: Comparison of reduction in decomposed problem size for networks with different connectivity and size

To identify how our approach behaves with different levels of connectivity we generated a separate set of random gene regulatory networks. These networks are created by modifying randomly generated graphs. They are processed identically with the synthetic gene regulatory networks we used in the experiments. First synthetic gene expression data is created by using these networks, this data is analyzed and the analysis results are used for creating decomposed version of the POMDP problems at hand.

Figure 4.7 illustrates the ratio of decomposed problem and original POMDP problem

size. For dense graphs of 80% connectivity decomposition preserves nearly all of the states. However it is possible to eliminate 10%-15% percentage of the genes when the network is denser.

This result verifies the expectation that, in sparse networks there are more isolated genes that can be identified and removed from problem description by our method.

CHAPTER 5

EFFICIENT POLICY GENERATION BY AUTOMATIC DECOMPOSITION OF POMDP MODEL OF GRN

The previous chapters presented the POMDP problem formulation of the GRN control problem. As we have stated above, after this formulation, it is possible to use any POMDP solver to generate a policy and intervene with the GRN accordingly. In order to efficiently solve the POMDP problem, we also developed a method to pre-process the POMDP problem. The goal of this method is decomposing the POMDP problem into subproblems. Each subproblem will contain genes closely related and exhibiting similar behavior. The motivation behind this decomposition is separating different parts of the problem from each other.

One of the important problems of the existing POMDP solution methods is curse of dimensionality [6]. The complexity of solving the problem is dominated by the number of possible states and for a flat representation, POMDP problems have huge state space (exponential in terms of state variables). Fortunately, factored representation is a way to represent the problem in a more compact form. However, without taking advantage from this compact representation, the computational cost of the solution methods does not decrease. Decomposing the problem into subproblems produces a number of subproblems; each subproblem can be solved with a very small computational cost compared to the cost of solving complete large problem. Moreover, separating different parts of the problem provides the chance to identify and remove the unrelated subproblems from the solution process.

The gene expression data analysis explained in Section 4.2 is the method we used for identifying classes of genes. The next sections present how the subproblems are created for each class and how these subproblems are coordinated for the solution. Algorithm 4 gives a brief outline of POMDP decomposition.

Algorithm 4: POMDP Decomposition and Execution

Input: M: POMDP Problem Formulation of GRN Control Task, G: Gene Regulatory Network, P: Partitioning of genes

foreach *partition p in P* **do**

  Formulate POMDP problem for p using M and add it to DEC-POMDP

end

Eliminate Redundant Subproblems in DEC-POMDP

foreach *problem dp in DEC-POMDP* **do**

  Determine Goal Description of dp

  Postulate Action Set of dp

end

Construct Execution Graph EG using DEC-POMDP

Solve subproblems in DEC-POMDP

Execute Policy using solutions of DEC-POMDP and EG

5.1 POMDP Formulation of subproblems

It is possible to apply the POMDP formulation presented in Chapter 4 to each class of genes in order to produce POMDP formulation for the subproblems. However, we did not apply the same method, instead we partitioned the POMDP problem into subproblems by using some elements of the POMDP formulation in the process.

The main motivation for not using the same POMDP formulation as a constructive manner is the fact that the main problem is something more than the union of the subproblems. It is possible to use the POMDP formulation for constructing every component of every subproblem. However, some of the subproblems may contain genes that are influenced by genes that belong to other subproblems. Similarly, some of the subproblems may contain genes that influence genes that belong to other subproblems.

We call these relations *dependencies* and these dependencies are not components of any single subproblem. If we use this approach, we should also regenerate these dependencies by repeating some of the steps used in constructing the main problem. Moreover, generating components of the subproblems (e.g., transition functions) also require multiple repetition of the already carried POMDP formulation. So, using the POMDP formulation for subproblems brings a lot of excessive computation if we want the subproblems to fully express the main problem. Thus, instead of repeating the same process, we adapted a decomposition approach which basically uses projections or subsets of POMDP components of the main problem.

Another motivation underlying not using the POMDP formulation is to keep the formulation and solving methods independent from each other as much as possible. In this work, POMDP formulation and POMDP solving methods have only a single common element, which is the gene expression data analysis. The POMDP solving method can be adapted to any other partially observable probabilistic control problem by replacing the gene expression data analysis with an appropriate method particular to the investigated problem (possibly a similar data analysis component). However, if we had used the POMDP formulation for defining the subproblems, the solving method would have been dependent on the POMDP problem formulation, which would have been customized for the GRN control problem. This would severely affect the portability of the POMDP solving method.

In order to partition a POMDP problem, it is sufficient to partition each problem element. Partitioning the state space, action space, observation space and observation function are trivial, since all of them are made up of elements related to a single gene. For each existing partition, we identify the states, actions and observations related to the genes in the partition. For each observation, the function can be defined based on the same genes and by obeying the specifications given in Section 4.1.4.

The reward function for each subproblem is the same as the reward function defined for the main problem, however the goal description might change. The goal of the main problem mentions a number of genes. For each subproblem, we modify the goal description, such that it only mentions genes (i.e., state variables) related to the subproblem.

Note that, some of the subproblems might not be related to any gene (state variable) mentioned in the goal description. Thus, their goal descriptions become empty and the reward function becomes meaningless after decomposition. Similarly, some of the subproblems might not be related to any input gene. Thus, these subproblems would not contain any action after decomposition. We will enumerate and process these cases further when coordinating the subproblems in the next subsection.

The transition function is derived as explained in Section 4.2.2. The conditional probability values extracted from the gene expression data are used for constructing the transition function for each subproblem. However, for each subproblem, the transition function only contains probability distributions for expression levels of related genes.

Note that a gene g might be influenced by gene g' from another partition. In this case, the similarity value between the two genes should be used when constructing the transition function. However, using this dependency in the transition functions of both subproblems is redundant. We only add the influenced gene (g) to the subproblem containing influencing gene (g'), and we use the similarity value to calculate the transition function value regarding g and g'. Influenced gene (g), which now exists in both subproblems, is called *subproblem input* for the original subproblem in which it exists; and it is called *subproblem output* for the other subproblem to which it is added.

This is the outline for producing each subproblem from the main POMDP problem components and gene expression data analysis. When all the subproblems are constructed, what is left is to postprocess them in order to fill any remaining detail in their formulation, and coordinating the subproblems to produce a policy for the main problem.

5.2 Coordinating Subproblems

5.2.1 Idea Behind Coordination

By decomposing the POMDP problem, we obtained a number of subproblems. It is possible to solve these problems by a POMDP solver; however, we need to organize

the solution process by coordinating these subproblems. The idea behind this organization is to establish how each subproblem contributes to the main problem. Each subproblem contains three kinds of genes (or state variables in a more general form):

1. **Influencing Genes:** It is possible to influence the expression levels of these genes. This group contains:
   (a) **Input genes:** These are genes that are in the input gene set of the main control problem
   (b) **Subproblem input genes:** These are genes that are not in the input gene set of the main control problem (i.e., can not be influenced directly), however their expression level is influenced by gene(s) that belong to another subproblem.
2. **Influenced Genes:** The expression levels of the genes in this group are important because this group contains:
   (a) **Target genes:** These are genes that are in the target gene set of the main problem
   (b) **Subproblem output genes:** These are genes that are not in the target gene set, however their expression levels influence gene(s) that belong to another subproblem.
3. **Abstract Genes:** It is not possible to influence the expression levels of these genes, and furthermore their expression levels are not important because they are neither influencing nor influenced genes.

Note that a gene can be a member of more than one group. For example, an input gene might also be a subproblem output gene.

Each subproblem can be viewed as a small portion of the main control problem. Each subproblem has two purposes:

1. If all of the subproblems have little or no dependency (i.e., there is only a small number of subproblem input and output genes) then solving each subproblem

is enough for deducing a policy to control target genes that belong to this subproblem. In this case, we can see each subproblem as a portion of the main control problem; it is concentrated on a group of target genes and all unrelated problem elements are discarded.

2. If subproblems have dependencies among themselves, beside solving each subproblem for controlling target genes, we should also take into consideration effects of each subproblem on others. Subproblem input and output genes are formulated for this purpose. They maintain the dependencies between subproblems and can be used for controlling how one subproblem influences another.

By solving subproblems it is possible to control the influenced genes, which contains target genes and subproblem output genes. Controlling target genes is the main purpose, and controlling subproblem output genes allows us to control subproblem input genes. Subproblem input genes, with input genes, are used as control inputs of the sub problems when solving them. Abstract genes acts as hidden state elements that effect the system dynamics, but can not be controlled directly and their expression values are not important for our purposes.

As an example. assume a gene regulatory network of genes labeled from 1 to 8 as shown in Figure 5.1. Genes 1, 2, 3, 4, and 5 are observable. Genes 1 is input genes and gene 2 is the target gene. Assume a subproblem set generated by our approach is $\{\{1,3\},\{2,4,8\},\{5,6,7\}\}$. For the first subproblem, gene 1 is both an input gene and a subproblem output gene; gene 3 is a subproblem output gene. For the second subproblem genes 4 and 8 are subproblem input genes since all of them are influenced by gene 1; gene 2 is a target gene. For the third subproblem gene 5 is influenced by gene 3; genes 6 and 7 are influenced by gene 1 so all of them are subproblem input genes.

5.2.2 Eliminating Redundant Subproblems

The first step in coordinating the subproblems is to decide for each subproblem, whether we should solve it or not. It is obvious that we should solve the subproblems with a goal description (i.e., subproblems related to at least one of the target genes).

However, we mentioned before that some of the subproblems might not have a goal description (i.e., not related to any of the target genes). We should decide which one of them should be solved.

We use a simple algorithm for this purpose. First, we mark all subproblems with a goal description to be solved. Then, we iterate over the remaining subproblems. If any of these subproblems contain a subproblem output which is related to a subproblem previously marked "to be solved", then the subproblem containing the output is also marked "to be solved". We repeat this iteration until no more subproblem could be marked "to be solved".

By using this simple algorithm, we identify all subproblems directly or indirectly related to the controlling target genes. It is sufficient to solve only these subproblems and the other subproblems are discarded.

Note that there is a slight possibility that subproblems marked "to be solved" do not contain any of the input genes. In this case, we can conclude that the input genes are not influential on the target genes and the control problem has no possible solution, regardless of the formulation used.

Considering the example of the previous subsection, subproblem $\{5, 6, 7\}$ can safely be eliminated since all subproblem input genes of the subproblem $\{2, 4, 8\}$ are related to subproblem $\{1, 3\}$ and this subproblem does not have any subproblem inputs.

5.2.3 Determining Goal Descriptions

For the second step, we should formulate goal descriptions for subproblems marked "to be solved". It is guaranteed that these subproblems have subproblem outputs (If they did not have, then either they would not be marked "to be solved", or they would already have a goal description). The outputs of these subproblems should be used as goal description. However, it is not possible to automatically specify whether it is desirable for these genes to be expressed or not.And also there is a non-trivial relationship between these subproblem output genes and related subproblem input genes. In order to postulate a complete problem description, we add the related subproblem input genes to the problem and mark them as goal of the subproblem.

However, it is still not possible to state whether it is desirable for these genes to be satisfied, or not. Thus, we produce multiple copies of these subproblems, one for each expression level of the goal genes. If there are k goal genes for a given subproblem, we produce 2^k copies of the problem, assuming a binary expression level. In the worst case, this approach produces exponential number of copies of each problem and the total performance of the method suffers from this step. However, for most cases, if the gene expression data properly partitions the genes, then each subproblem would have very few number of (typically 1 or 2) goal genes. Thus, we predict that this approach will on average contribute a constant factor to the computational complexity.

Continuing with our example, for the first subproblem $1, 3$ we extend the problem to $1, 3, 4, 8$ and the goal description includes genes 4 and 8. For the second subproblem, goal description only includes gene 2. This means we have to solve 4 copies of problem 1 for each possible goal description regarding genes 4 and 8; we only need to solve a single copy of the second subproblem.

5.2.4 Postulating Action Sets

For the third step, we formulate actions for all subproblems. We have already mentioned above that some of the subproblems might not contain any action at all. If these subproblems contain subproblem inputs, these genes are treated as input genes and action descriptions for controlling these genes are provided. If these subproblems do not contain any input gene or subproblem input, then they are discarded and marked as "not to be solved". We repeat this process until no subproblem is discarded.

Note that there is also a slight possibility that the subproblems discarded in this step contain target genes. In this case, we can conclude that these target genes can not be controlled by any input gene. If all of the target genes are removed in this fashion, we can conclude that the problem has no possible solution, regardless of the formulation used.

For our example since both subproblems contains input genes or subproblem input genes, none of them can be eliminated. Action set for the subproblem $\{1, 3, 4, 8\}$ only contains gene 1; action set of subproblem $\{2, 4, 8\}$ contains genes 4 and 8.

5.2.5 Construction of Execution Graph

For the fourth step, we construct a directed graph of subproblems. Vertices of the graph are subproblems and there is a directed edge between two subproblems connected with a subproblem input-output pair. This graph structure is our scheme for solving subproblems. For simplicity, we process this graph and remove all loops. We detect loops from short to longer, and for each loop found, all subproblems in the loop are merged into a single problem. The merging process can be carried out as the inverse of decomposition. State, action, and observation spaces are merged; observation functions are merged; new reward functions and transition functions are formulated. Finally, as a binding element, all subproblems on the directed graph are solved using POMDP solver. Each subproblem generates a policy for intervening its input genes. Note that these input genes might be actual input genes, or subproblem inputs. The policy elements with actions intervening actual input genes can be realized; however, it is not possible to realize the policy elements with actions intervening subproblem inputs. So, we apply a controlled execution scheme.

We select the subproblems with target genes as main execution subproblems. At each instance, the policies generated by these subproblems are always executed. For input genes related to these subproblems, intervening actions are executed directly. However, for actions intervening subproblem inputs, we do not execute any action; we just select the related subproblem's policy for execution. For example, assume that SP_a is a subproblem containing a target gene as state variable and gene i as subproblem input. There exists another subproblem SP_b with subproblem output gene i (there should be an edge in the binding graph from SP_b to SP_a). Then if the policy generated by SP_a tells us to intervene with gene i and set its expression level to *on*, then we execute the policy related to SP_b. At each instance, we can just select the policies to be executed in this fashion. Since each subproblem contains different genes and different action, no inconsistency arises from executing multiple policies.

For our example, the execution graph is a simple 2-node graph. There is a directed edge from subproblem $\{1, 3, 4, 8\}$ to subproblem $\{2, 4, 8\}$. Execution is based on subproblem $\{2, 4, 8\}$. Genes 4 and 8 are input to this subproblem. A policy is based on these input genes. Four copies of subproblem $\{1, 3, 4, 8\}$ are solved for each possible

expression level of genes 4 and 8. Whenever the policy for the subproblem $\{2, 4, 8\}$ requires genes 4 and 8 are set to specific expression levels, the related policy is fetched and input gene 1 is set to the expression level designated in the fetched policy.

5.3 Experimental Evaluation of Decomposition Method

In this section we present experimental results of the comparisons of plain POMDP formulation and POMDP decomposition models. The goal of this experiment set is to understand the overhead of POMDP decomposition method and compare this overhead with the benefit gained.

5.3.1 Experiments on POMDP Decomposition

In this subsection, we evaluate the proposed decomposition method by conducting a sequence of experiments to demonstrate its applicability, effectiveness and efficiency. We start by describing the implementation and the testing environment. Then we present an illustrative example. Finally we report and analyze the test result.

We used two synthetic networks and one real network in the experiments. For each of the two networks, the gene expression data is generated using the method proposed by Yu et al. [62]. For solving the POMDP problems, we used symbolic Perseus. The main policy generator uses symbolic Perseus extensively; it has been coded partially in Matlab and Ruby. All other modules of the system are coded in Ruby.

The execution times presented in the reported results have two components, problem preparation phase and execution phase. The problem preparation phase was run on AMD Turion x2 1.8Ghz CPU/1GB RAM and Intel Core 2 Duo E4600 2.4 Ghz CPU/4GB RAM configuration computer; and the problem solution phase was run on Intel Core 2 Duo E4600 2.4 Ghz CPU/4GB RAM configuration computer.

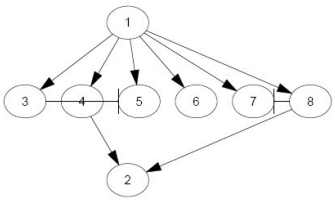

Figure 5.1: Example gene regulatory network

5.3.1.1 An Example Decomposition and Execution

Shown in Figure 5.1 is one of the synthetic GRNs used in the experiments. In this subsection, we will use this network to illustrate and demonstrate the decomposition and execution phases.We typically defined gene 2 to be a target gene in our experiments. Note that genes $5, 6$ and 7 in this network do not effect the expression level of gene 2. So, we predicted our approach could decompose the problem such that these genes could be eliminated.

Assume that in the network shown in Figure 5.1 the control problem is defined such that gene 1 is the input gene and gene 2 is the target gene to be promoted. In Section 4.2, we presented an example of similarity analysis based on gene expression data. Here we will not go into the details of the gene expression data analysis, instead we will consider two different scenarios with two possible partitioning schemes extracted after the data analysis.

For the first scenario, consider a partition of genes, say P_1, extracted via gene expression data analysis, where $P_1 = \{\{1, 2, 3, 4, 6, 8\}, \{5, 7\}\}$. Then the problem will be divided into two subproblems, one subproblem for genes $\{1, 2, 3, 4, 6, 8\}$ and another subproblem for genes $\{5, 7\}$. For the first subproblem, the goal gene is 2 and the input gene is 1. These genes are from the input and goal sets of the main problem. For the second subproblem, there is no goal gene or input gene, since genes 5 and 7 are neither in the goal nor in the input set of the main problem.

Note that gene 5 is controlled by genes 1 and 3; and gene 7 is controlled by gene 8. So, the second subproblem is controlled by some genes in the first subproblem; however, the inverse is not true. Genes 5 and 7 do not influence any gene in the first subproblem. For the first step in coordinating the subproblems, we enumerate the subproblems to

be solved. The first subproblem has some goal description (promoting gene 2), so it is marked "to be solved". The second subproblem does not have any goal description; so it is not marked "to be solved". Then, we explore whether the first subproblem is controlled by the second subproblem; because the result of the check is negative, no other subproblem is marked "to be solved". Therefore, the first subproblem is the only subproblem we should solve. In the execution part, we simply solve the first subproblem and use the policy returned.

As a more complex scenario, consider another partition of genes P_2 extracted via gene expression data analysis, where $P_2 = \{\{1, 3, 5, 6\}, \{2, 4, 7, 8\}\}$. Then the problem will be divided into two subproblems, one subproblem for genes $\{1, 3, 5, 6\}$ and another subproblem for genes $\{2, 4, 7, 8\}$. Note that genes $4, 7$ and 8 are controlled by gene 1. So the second subproblem is controlled by some genes in the first subproblem. However, the inverse is not true. Genes $2, 4, 7$ and 8 do not influence any gene from the first subproblem. For the first step in coordinating the two subproblems, we enumerate the subproblems that should be solved. The first subproblem does not have any goal description, so it is not marked "to be solved". The second subproblem has some goal description (promoting gene 2) so it is marked "to be solved". Then, we explore whether the second subproblem is controlled by the first subproblem. Since the first subproblem controls the second subproblem, the first subproblem is also marked "to be solved".

Now both subproblems should be solved. The influenced genes $4, 7$ and 8 are added to the first subproblem as target genes. The first subproblem now contains genes $1, 3, 4, 5, 6, 7, 8$. The three genes $4, 7$ and 8 are also input genes for the second subproblem. The second subproblem is solved for a policy governing genes $4, 7$ and 8. Since in this work we do not make use of joint actions, each action in this policy will govern the expression level of a single gene and there are six possible actions, namely $gene_4 = on$, $gene_4 = off$, $gene_7 = on$, $gene_7 = off$, $gene_8 = on$ and $gene_8 = off$. Each of these six case is a goal description for the first subproblem; however some of the actions might not be used in the policy. Assume these three actions are used in the policy $gene_4 = on$, $gene_7 = off$ and $gene_8 = on$; accordingly, we solve three copies of the first problem, each with a single goal description corresponding to one of these actions.

Finally, our policy is simply a glued version of policies from the two problems. When we need to execute the action $gene_4 = on$, instead we execute the action in the policy of the first subproblem (with goal description $gene_4 = on$), we use observations from the second problem to decide on an action in the policy of the second problem and observations from the first problem to decide on an action in the policy of the first problem.

5.3.1.2 General Outline of the Quantitative Experiments

In the following subsection, we present and discuss the results of the conducted experiments. We have conducted two sets of experiments for analyzing the performance of our method. The first group of experiments are performed using two random synthetic GRNs and the gene expression data sampled from these networks. We defined a control problem on each network, and we used gene expression data samples of different sizes to solve the control problems by using our method. For each result set, the outcome from the two control problems defined on the two GRNs are presented together.

The second group of experiments are based on gene profiling data produced from a study of metastatic melanoma by Bittner et al. [8]. This data contains 31 samples of 587 genes. Kim et al. [28] first studied this data for finding a PBN representation of the GRN. It is computationally intractable to use all the genes for the control problem. So, they detected the 10 most relevant genes and built a PBN of these 10 genes. Datta et al. [18] and Bryce et al. [11] separately used this same data in their studies; they separately formulated partially observable control problems. Datta et al. used a seven genes GRN. We used both the ten genes and the seven gene networks in our experiments. These networks are actually wiring diagrams of the PBNs, where each gene is influenced by 3-4 other genes. We also used sparser versions of these networks, where all bidirectional connections are dropped. We present the results from the second group of experiments in the next subsection where we discuss the performance of our method relative to the works of Datta et al. [18] and Bryce et al. [11].

For all the experimental cases, we present two sets of results from two problems. The

first problem uses only the POMDP formulation method and a policy is generated by solving a single POMDP. The second problem applies the whole POMDP decomposition method outlined. In the first problem, the policy generated by our method is identical to the policy generated from the original POMDP problem. Thus, the results related to this problem can be interpreted as worst case scenario. In the second problem, we observed smaller policies generated in less time. Thus, this problem can be interpreted as a best case scenario, where maximum benefit from our approach is reported.

There are also two sets of separate experiments used to explore the impact of network connectivity and data order to our approach. For experiments on data order, we used the metastatic melanoma data and generated permutations of this data as explained in the corresponding subsection. For experiments related to network connectivity we generated randomly connected and directed graphs and constructed gene regulatory networks by slightly modifying these graphs.

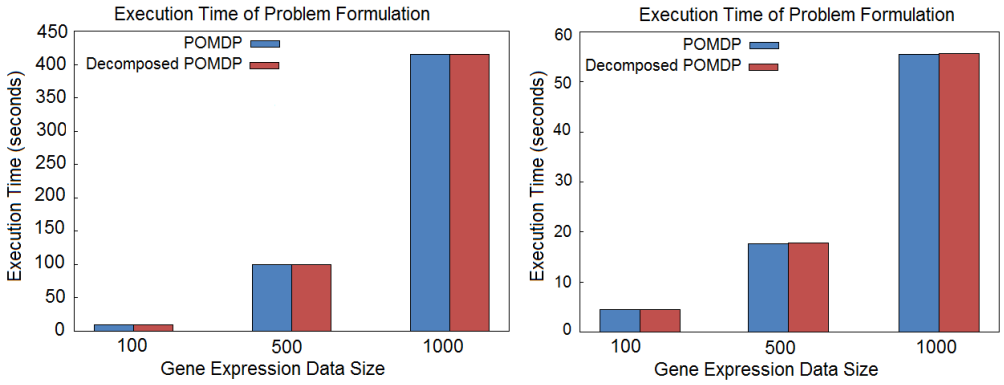

Figure 5.2: Problem formulation module execution times for two different networks

5.3.1.3 Experiments on Synthetic GRNs

The problem formulation costs are shown in Figure 5.2. POMDP result set is the computational cost of constructing the main POMDP problem in seconds. Decomposed POMDP result set is the computational cost of formulating the control problem in decomposed POMDP format. Note that the latter process requires first formulating the main problem, thus the actual cost of decomposing the main problem is the

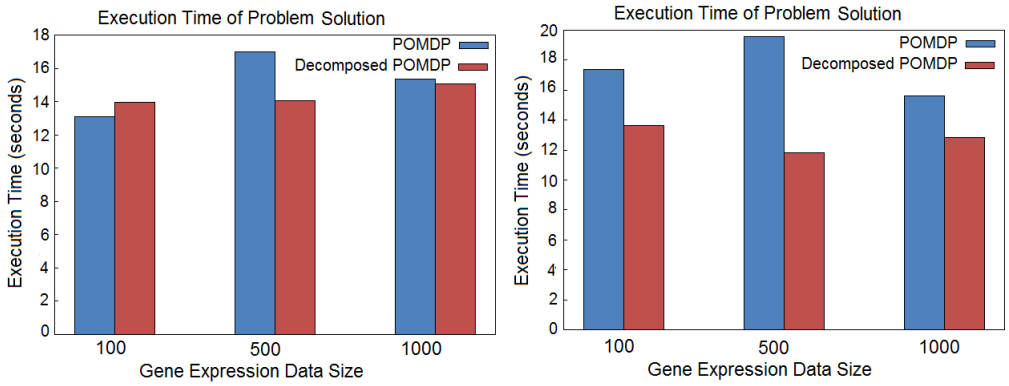

Figure 5.3: Execution times for policy generation

difference between the two bars, which is quite small. The dominating factor in both formulations is the gene expression data analysis. For a data set of 1000 samples, due to the cost of this analysis, the execution time goes up to 6 minutes. However, on average it is possible to conclude that the analysis and the problem formulation are completed in a reasonable amount of time. These result sets clearly show that the overhead of decomposing the POMDP problem is very small.

Figure 5.3 shows the execution times used for constructing the policies. This result set may be considered as the most important result because the main goal of our approach is to solve the GRN control problem more efficiently. Note that for the first problem, the execution times are close; our approach is performing slightly better for larger gene expression data sets. However, for the second problem, the execution cost of our approach is clearly much less than the plain POMDP method. The structure of the decomposed problem in the second network is very similar to the first example presented in Section 5.3.1.1. The problem is decomposed into two pieces; we could completely omit one of the pieces, and this leads to a smaller single POMDP problem. However, the first problem is decomposed into four pieces, which are closely related to each other (thus forming loops in the subproblem dependency graph). Thus all the subproblems are re-combined again and the performance is very similar to the original problem. This result set clearly shows that applying our method could possibly lead to significant gain in performance for problems with loosely coupled state variables.

For both control problems, 500 runs have been carried out, mainly for determining the quality of the policies generated. The length of the simulations are also recorded

in order to understand any performance gain or loss introduced by our approach.

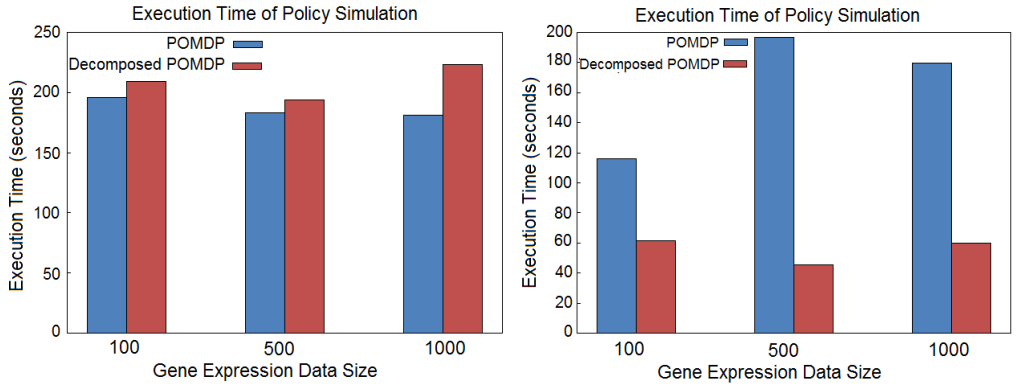

Figure 5.4: Execution times for policy execution

Figure 5.4 presents the lengths of the simulations. For the first network, the lengths

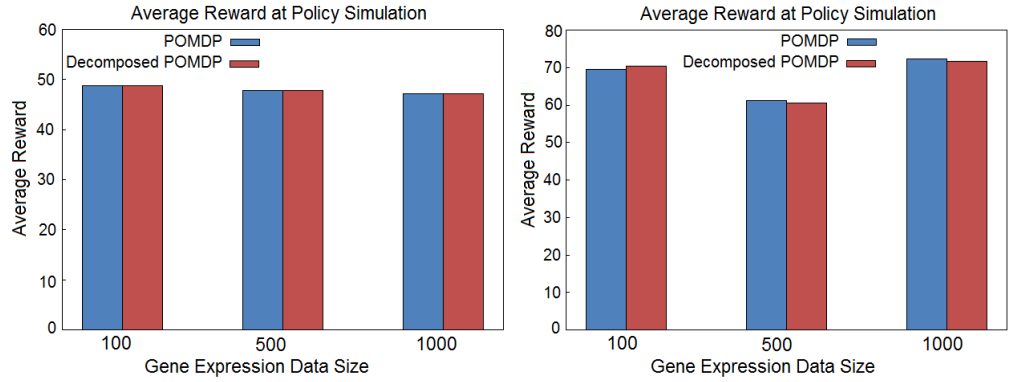

Figure 5.5: Average Rewards

of the simulations are close; our approach is performing slightly worse. This slight decrease in performance is expected since similar policies are generated for this problem. The difference in the performance can be explained by the overhead of our method, which is not so significant compared to the simulation length. However for the second network, the simulation time drastically decreased to 80%. The most important reason behind this increase in performance is the fact that our approach generates a smaller policy for the problem.

Figure 5.5 presents the average reward gained in these simulations. The amount of reward gained by our method is nearly identical to the amount of reward gained by the single POMDP problem. By examining these results, we can conclude that our

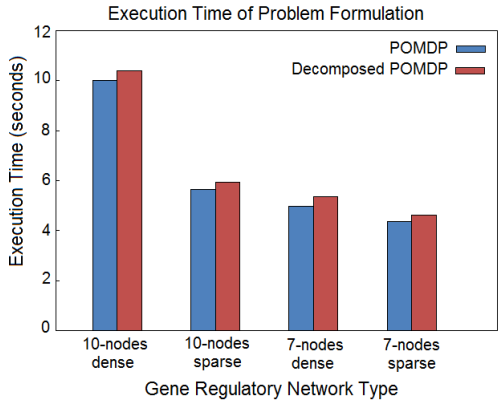

Figure 5.6: Problem formulation module execution times for four different networks.

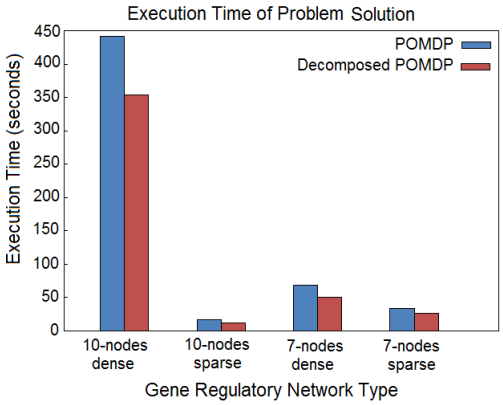

Figure 5.7: Execution times for policy generation

method is producing possibly smaller policies without any loss in the policy quality. This characteristic of our approach leads to simpler and effective policies.

By combining all the results, we can conclude that our approach is successful in achieving the performance goals we established beforehand for the synthetic data we used. Our method produces structured problem formulations with little overhead. These formulations can be solved in less time with standard POMDP solvers. The optimality of the resulting policies could be classified as close to that of the generated plain POMDP policies; however our approach has the capability of producing more compact policies that can be executed more efficiently.

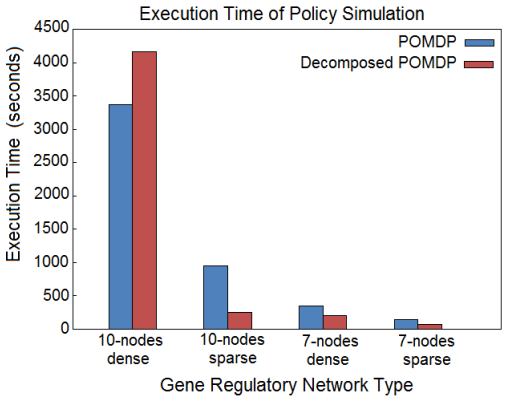

Figure 5.8: Execution times for policy execution

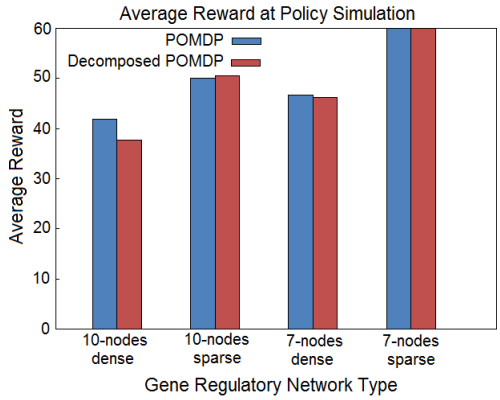

Figure 5.9: Average Rewards

5.3.1.4 Experiments on Real Biological Gene Expression Data

Figures 5.6 - 5.9 present the execution times for different stages of the control problem and the average reward collected. The execution times for problem formulation module are similar to the results of the previous subsection. There is a problem formulation time around 5 seconds before attempting to solve the problem and the overhead of decomposing the POMDP problem is small compared to the formulation overhead.

The execution times for policy generation clearly show the benefits of POMDP decomposition as in the previous subsection. For all networks based on the gene profile data, decomposing the POMDP problem helped in solving the problem faster. For the sparse networks, using its full potential, our method generates a policy in around 10 to 20 seconds, even for a 10-genes GRN. Also the average rewards clearly shows that the generated policies are still close to optimal, even closer than the plain POMDP approach, for all cases. Finally, our method also succeeded in reducing the time necessary for executing the generated policy. Especially for the sparse network of 10 genes, our approach successfully reduces the simulation time by nearly 70%.

CHAPTER 6

GENE EXPRESSION DATA ENRICHMENT PROCESS

This chapter presents the gene expression data enrichment process we are proposing as a potential preprocessing step for any computational method on gene expression data. Motivation of this data enrichment process is solely to increase the number of samples in a gene expression data set. This process should not be interpreted as an intelligent method that finds new data points related to a data set. Our data enrichment approach does not claim to broaden the data set or interpolate (or extrapolate) new data points that are missing in the original data set.

This data enrichment approach targets to increase the number of samples in a data set consistently. In order to *consistently* increase the samples, we defined some metrics to be used in data enrichment process itself, and for evaluating the data produced. These metrics are designed with a unique goal in mind: This data enrichment method should introduce new samples that are correlated with the existing data set without blindly replicating existing samples. We would like our data set to maintain its original qualifications in order to consider this data enrichment process to be successful.

Data enrichment method we propose uses three computational models for modeling gene expression data. Probabilistic Boolean Network model is used to model the GRN structure that fits the gene expression data; Hierarchical Markov models are used to formulate generative structures that fit gene expression data; and Genetic Algorithm model is used to search for new samples by using evolutionary approach.

This chapter presents the data enrichment process in detail, introduce metrics used

in the process and present the experiment result for evaluating the data enrichment method introduced.

6.1 General Outline of the Sample Generation Process

Data enrichment process has five distinct steps:

1. Discretization
2. Model Building
3. Sample Generation
4. Producing Continuous Samples
5. Evaluation

6.1.1 Discretization

Since our method relies on computational models of gene expression, and most of these models are discrete, we chose to carry on most of the steps on discrete data. As a first step we apply a discretization. As a result, our method can work on any continuous data set or any binary data set by skipping the discretization.

In this work we assumed all the data used are discretized into a binary alphabet. The generative models also assume that the alphabet is binary. The PBN model explicitly relies on the binary alphabet. The Hierarchical Markov model does not rely on any restriction on the alphabet, so might work for different discrete data sets easily. Genetic algorithm model uses a binary encoding for generating new samples, however it is possible to formulate a different encoding schema to use different discrete data sets.

6.1.2 Model Building

Using the discretized data, three models for the data are produced : Probabilistic Boolean network, hierarchical Markov model and genetic algorithm. Background ma-

terial related to these models are presented in Chapter 2. Details on how there models are constructed is given in Section 6.2.

6.1.3 Sample Generation and Production of Continuous Samples

After constructing the generative models, each generative model is used to produce new samples. The exact mechanism of sample generation is different for each generative model and discussed in Section 6.2 with model details.

In this work all generative models we used produce binary samples. Since our data space is continuous we need to build continuous samples based on these discrete samples. It might be possible to use different generative models and this step can be skipped if the model used generates continuous samples. However in this work we generated continuous samples as a separate step.

For producing continuous samples we used the ranges of the original data set and applied the discretization process in reverse. The ranges used in the discretization step for values 0 and 1 are our target ranges for generating continuous data. We build a normal distribution on these ranges and sample from the appropriate range for each gene.

Assume that the data range for $gene_i$ is $[l, h]$ and in the discretization step we further divided this interval to $[l, m]$ for values discretized to 0 and $[m, h]$ for values discretized to 1. Then the normal distribution we build is $N(m, max(m-l, h-m)/2)$ and it is truncated into interval $[l, m]$ for values 0, $[m, h]$ for values 1. These two truncated normal distributions are used for sampling data points corresponding to discrete components of the new samples.

6.1.4 Evaluation

The last step of our method is evaluating all the samples produced by generative models, selecting the desired number of samples according to the evaluation results and outputting the new samples together with the evaluation results. In order to produce k new samples, we collect k samples from each generative model and evaluate

all $3k$ candidate samples. Evaluation is very important in deciding the final output of the method and quality of the output.

We've defined three criteria for evaluating the samples and there is an evaluation metric for each criteria. The total evaluation score of the sample is calculated as summation of these separate metrics. Each of these metrics are normalized before summation.

Compatibility Newly generated samples should resemble the existing samples. This metric measures "how close a sample is, to the closest neighbor in the existing data set" We use Euclidean distance to measure the proximity here, and the compatibility metric is the sum of Euclidean distances to closest sample in original data set.

Diversity Although we desire the newly generated samples to resemble existing samples, we do not desire the new samples to be duplicates of existing ones. This metric measure "how different a sample is, from the data points in the existing data set". We use Minkowski distance with $p = 1$ to measure the difference between samples, and the compatibility metric is the sum of Minkowski distances to all samples in original data set.

Coverage Newly generated samples should also cover the sample space as much as possible. This metric measures "how apart a sample is, from the other samples created by our method" We use Euclidean distance to measure the distance between sample points. For each new sample generated, the coverage metric is the sum of Euclidean distances to all other new samples. If a single sample is created, the value of the coverage metric is set to maximum of the normalization interval.

The results of all metrics are normalized and added up to produce a single evaluation result for each sample. Desired number of samples are output according to their evaluation result. We also output the scores separately for the user since giving more information is helpful to the user.

6.2 Generative Models Used

The main component responsible for data generation is the array of generative models. We used three generative models in this work:

1. Probabilistic Boolean Network model
2. Hierarchical Markov Model
3. Genetic Algorithm model

The details of these models are discussed in the following subsections.

The common idea behind all these models is using the existing data for building up the model and using the model for data generation. So details of each model is mainly consists in the method used for building the model and configuring the model as a generative data source.

6.2.1 Probabilistic Boolean Network Model

Probabilistic Boolean networks are proposed for specifically modeling gene regulatory networks. They are probabilistic versions of the Boolean networks. PBNs present a first order Markovian view of the system they model. Each node in the network is associated with multiple Boolean functions and a specific wiring diagram for each function. The value of each node at time $t + 1$ is calculated by randomly selecting one of the Boolean functions associated with it. The variables of the functions are values of nodes at time t and the wiring diagram specifies the the node each variable is associated with. Section 2.2 gives brief information about PBNs and on inferring a PBN from gene expression data.

With this working principals, PBNs are suitable computational tools for modeling gene regulatory networks. So, we chose the PBNs as one of our generative models. In order to generate samples similar to the original sample set, a PBN that estimates the original data set should be constructed. After constructing the PBN, it is sufficient to run the PBN further and use the values of nodes as a binary vector of gene expression data.

In order to infer a PBN from the gene expression data, we use the coefficient of determination method proposed in [30] and outlined in Section 2.2. To compute the parameters of PBN, we first adjust the s existing samples as time series. We mark samples $[1, s-1]$ as inputs of the PBN from $t = 1$ to $t = s-1$, and mark samples $[2, s]$ as outputs of the PBN from $t = 1$ to $t = s-1$. Thus we try to find a PBN that approximately generates our existing samples as time series. Each sample of the data is an input to the PBN and the next sample is the expected output. We've restricted the Boolean estimators to have ternary functions and each gene to be estimated by three estimators at most. Within these restrictions it is possible to evaluate all possible estimators and find the probabilistic Boolean network that fits the gene expression data at most.

Once the parameters of Probabilistic Boolean Network are computed, the network can be used for generating new data. The approach we used for generating new data is simply using the PBN for computing the continuation of the time series that the existing data forms.

Remember that when computing the parameters of the PBN, we used the existing data as time series from $t = 1$ to $t = s-1$. The PBN constructed is used for extending this time series from $t = s$ to $t = s + k - 1$ assuming we are asked to generate k samples.

The nodes of the PBN are initialized as if PBN is at state $t = s-1$. The last sample of the existing data is used for this initialization. Then the PBN is run for k steps. At each step, new values of the nodes of PBN are updated and new values are recorded. Finally for each of the n nodes in the PBN, we have k values corresponding to k steps of execution. These values are our generated k samples from the PBN generative model, a n element vector for each sample.

Also in order to ascertain diversity in the data produced, we use perturbation. When running the network, there is a small probability that the value of a node will change. Perturbation ensures that even the PBN overlearns the existing data, we will still be able to produce samples different from the exiting ones.

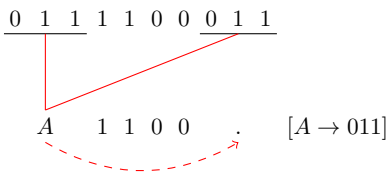

Figure 6.1: An example rule derivation step

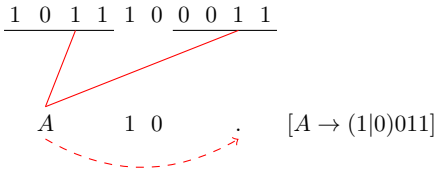

Figure 6.2: An example rule derivation step with an uncertain rule derived

6.2.2 Hierarchical Markov Model

The second generative model we used in our work is inspired from music generation. There is a considerable amount of research on inferring parameters, structures and models of the musical compositions and using them for recomposing new pieces with similar characteristics.

This problem is very similar to our data generation problem. A musical piece is simply a time series of notes and producing a similar musical piece is done by extending the time series. The criteria for correctness in our work also resembles the criteria used in music generation. In music generation, the piece produced should contain same features with the original piece. However it is also important to produce an original piece, thus the produced piece should divert from the original one to some extend.

The method we use for phrase hierarchy construction is a custom method that is designed for offline processing and concentrates on formulating rules that increase the information content of the representation. The idea behind our phrase structure algorithm is iteratively inserting new rules and reducing the string to a simple representation with non-terminal symbols. Two example rule derivations are shown in Figures 6.1 and 6.2.

In Figure 6.1 the substring 011 is replaced by a new unique non-terminal A. The reason for choosing 011 is that its the longest repeating substring in the string 0111100011. The string reduces to $A1100A$ with this rule and the process is repeated.

In Figure 6.2 an uncertain rule is introduced by replacing both strings 1011 and 0011 by a new unique non-terminal A. The string reduces to $A10A$ with this rule and the process is repeated.

For simplicity, we assumed that uncertain rules only contain uncertain symbols at the beginning of the rule body, at the end of the rule body or at both positions. Since we work on binary alphabet, uncertain rules occur more frequently and we derive them as much as we can, only with this limitation.

We also calculate the ratio of each uncertain symbol in the rule to the total occurrence of uncertain symbols and store these probabilities with the rules. Using these probabilities lead to application of rules in a more controlled way. For the rule $A \rightarrow (1|0)011$ at Figure 6.2, the probabilities of both 0 and 1 would be 0.5, since both of these symbols are observed once, as part of pattern $(1|0)011$ in the string 1011100011.

In theory, this phrase hierarchy derivation mechanism should be applied for each gene and a separate hierarchical model should be constructed for each gene in the original data set. However we estimated that this would have a very high computational cost if we consider the number of genes can perfectly reach thousands. Thus we form clusters of the original data w.r.t genes and we derive a single phrase structure for each cluster.

For clustering the data set for genes, Hamming distance is used for grouping more similar genes together and a cut off value of 0.5 for used for building clusters. This implies that, if two genes are in the same cluster, then for at least half of the samples these two genes had same expression level in data set. For each cluster we use the disjunction of all data sets related to this cluster and a phrase hierarchy is constructed for it.

Algorithm 5 summarizes all the phases of constructing Hierarchical Markov Model.

After constructing a models for each of the clusters of original data, we can use these models to generate new data samples. Each model generates m strings as new data

Algorithm 5: Construction Hierarchical Markov Model

Input: $n \times s$ gene expression data matrix D (n genes, s samples)

Result: k Hierarchical Markov Models for the data M

Clusters ⟵ clusterData (D, *hammingdistance*, 0.5)

k ⟵ number of clusters(Clusters)

foreach *c in Clusters* **do**

    str ⟵ $\bigvee(c)$

    while $|str| > 1$ **do**

        s ⟵ longest repeating substring in str

        NT ⟵ a new non-terminal symbol

        if *s is not a prefix in str* **then**

            pre_uncertain ⟵ occurrences of symbol 1 preceding s / occurrences of s

        else pre_uncertain = nil

        if *s is not a suffix in str* **then**

            post_uncertain ⟵ occurrences of symbol 1 following s / occurrences of s

        else post_uncertain = nil

        R ⟵ rule(NT, s, pre_uncertain, post_uncertain)

        add R to M(c)

    end

    add rule(S, str, nil, nil) to M(c)

end

samples, where m is the number of genes that belong to that cluster. These m strings are randomly assigned to each gene in the cluster. Each of the strings generated has k symbols, where k is the number of new samples to be produced. Thus we have n strings, each of length k which constitute our new sample set.

One important thing in this process is deriving a fixed length string from the feature hierarchies should be handled efficiently. Since the phase hierarchies we build introduce brand new non-terminals at each step, there is no recursive rules involved. However still there are multiple alternatives at each derivation step, since we have uncertain rules. Thus generating a fixed length string involves applying a search algorithm.

It is possible to use the length of shortest possible string that can be derived from each non-terminal, as a heuristic value in this search. By knowing these values we can prevent applying some of the derivation steps and we can generate a string without going through all the search tree.

However this string is not guaranteed to have the desired length. Since our phrase hierarchies do not have recursive or empty rules, there are limits on the length of the derivable strings. As searching through the possible derivation, by using the heuristic of shortest derivable string length, we try to generate a string with the closest possible length to our target length value.

At the end of the search we come up with a string of length k'. There are three cases to be considered here:

1. $k = k'$, then the derived string has the desired length;
2. $k > k'$, then the derived string is shorter than expected. In this case the derivation is repeated to derive a string of length $k - k'$ and two strings are concatenated;
3. $k < k'$, then the derived string is longer than expected. In this case the randomly selected $k'-k$ symbols of the string are deleted and the new string has the desired length.

Algorithms 6 and 7 summarizes the generation of new samples from the phrase hierarchies.

6.2.3 Genetic Algorithm Model

The third method we used for generating new samples is applying a genetic algorithm. This method differs from the other two in the sense that the explicit model is constructed. The existing samples we have are represented in the classical genetic algorithm framework and new samples are generated by applying steps of selection and mutation.

We present each step of applying genetic algorithm separately.

Algorithm 6: Generating strings from Hierarchical Markov Model

Input: m clusters of original data C, m phrase hierarchies M, number of samples to be produced k, number of genes n

Result: $n \times k$ sized new data set S

$S \longleftarrow \{\}$

foreach *cluster c in C* **do**

  G $\longleftarrow$ genes that belong to c

  foreach *gene g in G* **do**

    $s \longleftarrow$ `generateString(`$M(c)$, k`)`

    add(s, S)

  end

end

Parent 1:	0	1	1	1	1	0	0	0	1	1
Parent 2:	1	0	0	1	1	1	1	0	1	0
Offspring 1:	0_1	1_1	0_2	1_1	1_2	1_2	1_2	0_2	1_1	1_1
Offspring 2:	1_2	0_2	1_1	1_2	1_2	0_1	0_1	0_1	1_2	0_2

Figure 6.3: An example crossover

6.2.3.1 Representation and Initialization

Since we use binary discretized data, the representation step of genetic algorithm is trivial. Each sample of the existing data of length n is represented as a binary chromosome as a member of the initial population.

6.2.3.2 Crossover Mechanism

The crossover strategy we use in the genetic algorithm model is the uniform crossover [34, 52]. In the uniform crossover, for each position it is independently decided which parent will contribute to the offspring chromosome for that position. Thus the chromosomes of parents are mixed in the offspring chromosome. Figure 6.3 presents an example crossover by applying uniform crossover strategy and mixing ratio of 0.5. In

Algorithm 7: generateString(FH, l)

$s \longleftarrow S$

while *s contains non-terminals* **do**

  $nt \longleftarrow$ select a non-terminal in s

  maxdl $= l - \Sigma$ minimum derivation lengths of all non-terminals in s - # of terminals in s

  $r \longleftarrow$ the rule in FH, such that $r.lefthand = nt$

  $r' \longleftarrow$ resolve uncertainties in r randomly, provided $|r'.righthand| - maxdl$ is minimal and prefer $|r'.righthand| \geq maxdl$ on ties

  $s \longleftarrow$ replace nt with $r'.righthand$ in s

end

if $|s| < l$ **then**

  $s \longleftarrow s$.`generateString(`*FH,* $l - |s|$`)`

else if $|s| > l$ **then**

  $s \longleftarrow$ delete $|s| - l$ random symbols of s

end

return s

the figure, the subscripts indicate which parent's chromosome is used for indicated position. Approximately half of the positions in the chromosomes of each offspring are filled from first parent and the remaining half is filled from the second parent.

6.2.3.3 Selection and Fitness Function

We apply the crossover step to chromosomes selected randomly from the population. For selecting samples, and applying the crossover we use a selective aging policy. The idea behind this selective aging process is to maintain a uniform age in the population. This policy can be summarized as follows:

1. Each chromosome in the initial population is embedded with the generation information 0^{th} generation.

2. After each crossover, the generation of offspring is set to the generation of the

younger parent, incremented by one. For example if the parents are 5^{th} and 7^{th} generations respectively, the offspring are 8^{th} generation.

3. When a new generation is introduced first time, the eldest generation is removed from the population. This process is done starting with the introduction of 4^{th} generation.

By employing this policy we allow the population to reach to k^{th} generation, where k is the number of samples we should produce. A single chromosome is selected from each generation and this set of k binary strings becomes the new samples generated by genetic algorithm model.

When selecting parents for applying crossover, we use a fitness function and a generic selection procedure. Our fitness function is calculated for each chromosome in population and a cumulative probability function is constructed. Sampling two points from this function gives us two chromosomes to apply crossover.

Our fitness function is basically the first evaluation criteria, *compatibility*, we specified in section 6.1.4. We believe that the dynamics of the genetic algorithm will also improve the other two criteria, however in order to produce compatible new samples, we need to use this criteria in our fitness function.

In section 6.1.4, it was suggested that we use Euclidean distance for measuring compatibility however it was appropriate to use it after converting the discrete data to continuous samples. Here, since we are working with the binary discrete data we are using the Hamming distance as fitness function. The fitness function for each chromosome is calculated as summation of the hamming distances between the chromosome and all the 0^{th} generation chromosomes (i.e. original samples).

6.2.3.4 Mutation

At each crossover, there is a chance that offspring chromosomes might change. The PBN model we used also has a similar perturbation probability, such that the value of each node might change at each time step. We used the same probability value for mutation probability in genetic algorithm model, so that two models exhibit same

random behavior.

6.3 Discussion on Three Models Used

Section 6.4 gives a quantitative analysis of the data generated by each model and execution times. However we think it is appropriate to outline the characteristics of each model before going into a quantitative analysis.

PBN model is specially developed for modeling genes, gene regulatory networks and gene expression data. This model is proven to be a suitable model for gene expression data. However there are a number of factors that affect how close is the PBN derived from gene expression data to a perfect model for the data.

The most important factor is the number of existing samples. As any other model constructed or estimated with any structural learning method, PBN rely heavily on the data used to construct them. The reflection of this fact to our problem is, if we have enough existing data that describes the actual behavior of the genes with little noise, then we can expect that the PBN to successfully model the existing data and generate new samples very similar to the existing ones. As we discussed in Section 6.2.1, overlearning the data is a potential problem for generating different new samples, however by using perturbation we are always certain that some different samples will be generated randomly.

Another important factor that affects the PBN constructed is the PBN parameters. The computational power of the PBN is determined by how strongly connected the nodes are and how many Boolean functions are used for estimating each node. However it is not possible to use large values for any of these parameters, because the computational cost of building the PBN is increases greatly when the number of Boolean functions used increases and when the nodes are more strongly connected. We tried to establish a balance for the computational cost and model richness as discussed in Section 2.2 and analyzed in Section 6.4 quantitatively.

Hierarchical Markov Model is designed mainly for music generation. As long as gene expression data for each gene behaves as a string of information with some pattern, we are ascertain that HIMM constructed would be successful in modeling the string

and producing variations of it.

We are processing the data as strings of samples and thus at first sight we do not capture or use the relationship between genes. Our method treats each cluster of genes independently. In this sense, it is clear that the HIMM can not model the gene expression data as precisely as PBN alternative. However it is important to note that we also generate the samples in the saw way and the generated samples should reflect patterns observable on the gene expression data. Thus it is safe to say that the samples we generate using HIMM would behave as if the dynamics effecting the existing data are at work and similar samples are being produced. The uncertainties in the feature hierarchies would provide random behavior and the samples produced would meet our success criteria.

The works inspired us into using HIMM always try to maintain minimal representation, or fastest processing online or offline. Since our goal was not building persistent models for the data we only focused on building models in a fast offline way. We do not believe that this would affect the quality of the samples generated in any way.

Finally considering the computational power of HIMMs, the class of languages HIMMs generate are subsets of regular languages (note that HIMMs we use do not even have recursive rules, this means all possible strings are enumerable given the model) This proved challenging when generating new samples, since we needed to run the model multiple times or cut some of the data. However we believe that using a computationally richer model (like regular or context free languages) would make the construction of the model a real challenging task and would not be so appropriate. We believe that, that's the reason this model is preferred in music generation domain and for the same reasons we applied this model here.

The genetic algorithm model, this model is the most general purpose one in three. When using the genetic algorithm model we do not get an explicit model of the data in any way, however we use the existing data to produce new samples.

Considering the three criteria we established before, we believe that genetic algorithm model would produce successful samples as the other two methods. The diversity and coverage criteria are met in the way the genetic algorithm works. We apply crossovers

that produce offspring different from its parents. Mutations also act in favor of diversity. Applying both these concepts in iteration after a small number of iterations we also cover different parts of the sample space. The last criteria compatibility is explicitly favored by our fitness function and the new generations are produced from applying crossover to more compatible parents.

If we consider our choice of uniform crossover method here, we will see that this method is the most appropriate one since the order of the values on a chromosome is not significant in any way and just depends on the order of genes in the sample. So applying a fixed point or two point crossover would not have any different effect on the crossover process.

Finally, if we consider all three models together, we can safely say that each of them exhibit some features that don't exist at the other models. By implementing these three different models, the limitations of each model can be overcome by the other models and we can guarantee that the new samples always meet the success criteria we defined.

6.4 Experimental Evaluation of Gene Expression Data Enrichment Process

In this chapter we present experimental results of the gene expression data enrichment process proposed. In order to quantitatively measure the performance and quality of our method, we performed sample generation experiments with some real life biological data. The data we used is the gene expression profile of metastatic melanoma cells [8]. The data originally contains around 8000 genes and 31 samples.

We implemented our sample generation system on Matlab. We used PBN toolbox for PBN generative model. Other models do not use any toolbox. The platform we run our experiments has an Intel Core2Duo E4600 2.4Ghz processor with 4GB of memory.

In the experiments we generated new samples by using our biological data in different settings. Our comparison criteria are the evaluation metrics defined before. We used these criteria to measure the quality of the new data samples generated.

For each experiment setting, each generative model generates required number of new samples, i.e. if k new samples are to be produced, each of the three generative models we used produce k new samples for a total of $3k$. All these $3k$ samples are evaluated by our criteria and select best k ones among. These k samples are the final output of sample generation process. These evaluation results of these k samples are reported with the samples, so that the quality of the new samples are reported to the user.

For each setting, we also analyzed the execution times of the all runs to have an idea on the factors that affect the performance of our method. The effect of experiments settings to the execution time are analyzed so that we can identify the factors dominating the execution of our sample generation methods.

The details of the evaluation criteria are discussed in Section 6.1.4. The results are presented without normalization in order to observe the similarities between the evaluation criteria. The higher values of diversity and coverage are desirable, while the lower values of compatibility and execution time are desirable.

Each step of each experiment is repeated 10 times to decrease the effects of highly randomization nature of our method. The reported metrics are the average values over repetions and the reported execution times are the total execution time.

6.4.1 Experiments on Number of Genes

The goal of our first experiment is to analyze the effect of number of genes (i.e. dimension) of the data to our method. For this purpose we randomly selected 10,20,30,40 and 50 genes among the 8000 genes of the original data and generated new samples by using the expression profiles of the selected genes only. 5 new samples are generated on each run.

The reason we did not explore the whole 8000 gene range is the probabilistic Boolean network model we have used has practical restrictions. The PBN toolbox can work on arbitrarily large PBNs. However the PBN construction method is restricted to around 50 genes.

The compatibility, diversity,coverage values and execution times are shown in Figures

6.4 and 6.5.

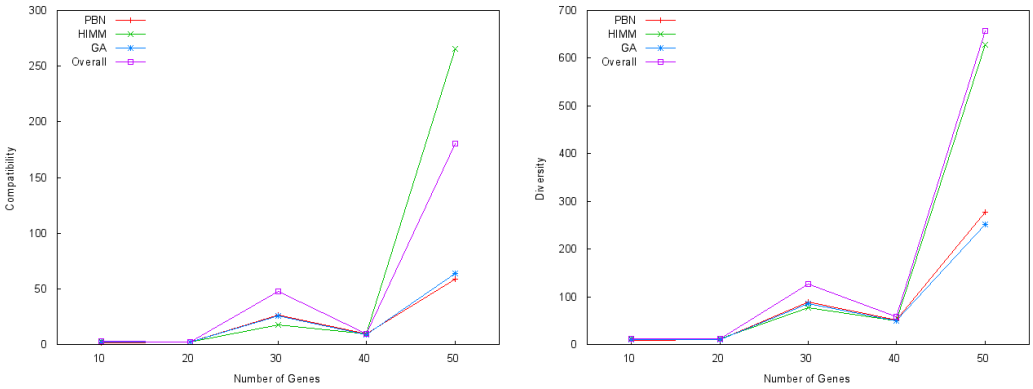

Figure 6.4: Compatibility and diversity values with different number of genes

In this experiment the evaluation metrics of all methods behave very similarly for all cases, except the 50 gene case. The 50 gene part of the input used clearly exhibit different characteristics than the other cases. In this case the diversity and the coverage values definitely increased for all methods. We can safely say that all of the methods we used were more focused on coverage and diversity for the 50 input case and thus sacrificed from compatibility.

However for less genes, the average compatibility metric of the samples we generated is very small. This lead to the conclusion that the 10,20,30 and 40 gene data sets were successfully modeled, even overlearned by our method. Thus the new samples generated for these cases were very close to existing ones (low compatibility, low diversity and coverage). For the 50 gene input the opposite was true. Our methods, especially the HIMM method generated new samples that divert from the existing

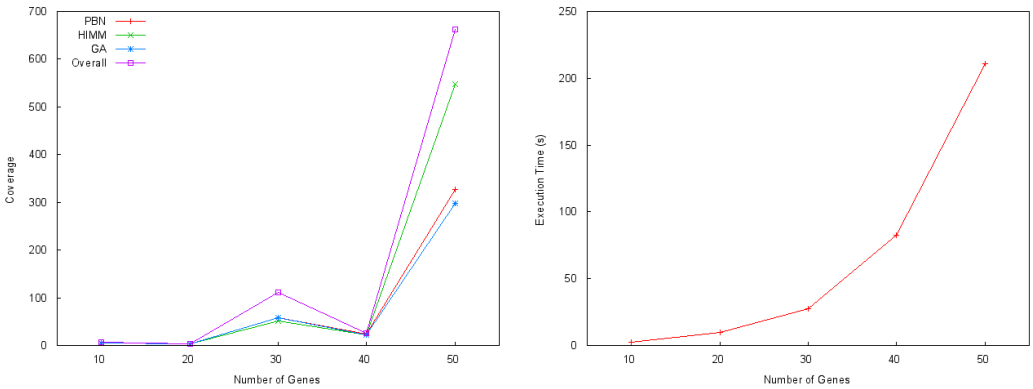

Figure 6.5: Coverage and execution times for different number of genes

ones highly.

If we consider the dynamics of the HIMM method, it is easy to see the reason behind its failing to generate more compatible samples. HIMM models are constructed for each sample in the original data set. As the number of genes increase, the length of each sample vector increases. HIMM treats each sample vector as a string and builds a phrase hierarchy on it. As the string size increases more uncertain rules are introduced. When there are a lot of uncertain rules in phrase hierarchy, the strings derived from the models gets more diverted from the original string.

Genetic algorithm and PBN methods are not much effected by the increase in the number of genes. The number of genes affect the size of the PBN network constructed linearly, however since we do not allow the PBN parameters (arity and number of Boolean functions for each node) increase likewise, the general performance of the network is not affected. For genetic algorithm, the increase in the size of the chromosome only affects the likelihood of mutations, slightly. Since mutations are rare, there is no significant change in the evolution of our solution population.

For the 50 gene case, coverage and diversity increased with a little increase in compatibility, for genetic algorithm and HIMM. These two methods continued to perform well for this case, since the number of genes is not so critical for them.

The combined results of the three methods is close to the average of the methods in all cases except the 50 gene case. In this case since the samples generated by the HIMM model show high diversity and coverage, these samples are ranked high in overall evaluation and selected in the combined results.

Finally if we analyze the execution time of the experiment the execution time grows like an exponential curve. The main reason behind this fast growth is the computational complexity of constructing the PBN network. Constructing the PBN involves evaluating the estimators for all possible Boolean functions over nodes of network. Thus as the number of genes increase the number of possible Boolean functions increases nearly in factorial rate. Thus more that 50 genes limit the construction of PBN. However even for the limit case, a single run of our approach only took around 20 sec., which is appropriate.

6.4.2 Experiments on Number of Samples

The goal of the second experiment is to analyze the effect of the existing sample size to our method. For this purpose we randomly selected 10,20 and 30 samples out of our data set. In all cases the data contained 20 genes and 5 samples are produced.

The compatibility, diversity,coverage values and execution times are shown in figures 6.6 and 6.7.

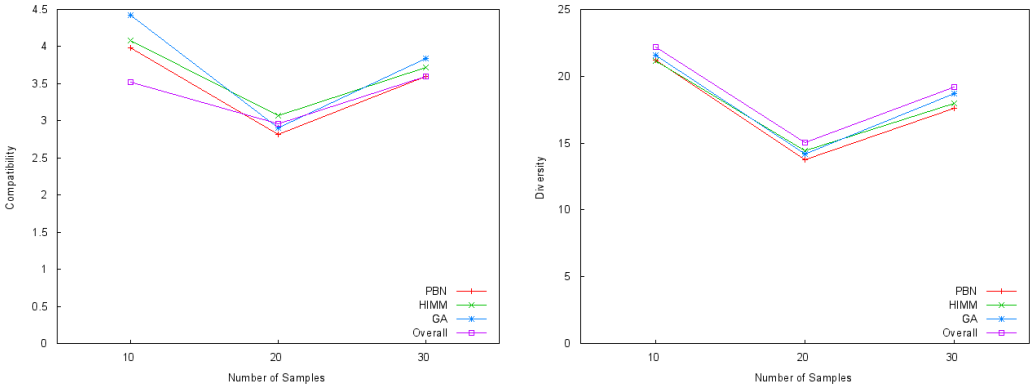

Figure 6.6: Compatibility and diversity values with different sample sizes

In this experiment the evaluation metrics and the execution time is not disturbed at all by the changes in the original sample size. The evaluation metrics of all methods and combined samples are very close to each other and the execution time only increases slightly with the increase in the sample size.

Note that this is perfectly expected if we consider the genetic algorithm and HIMM. These success of these two models are not related to the number of samples at all.

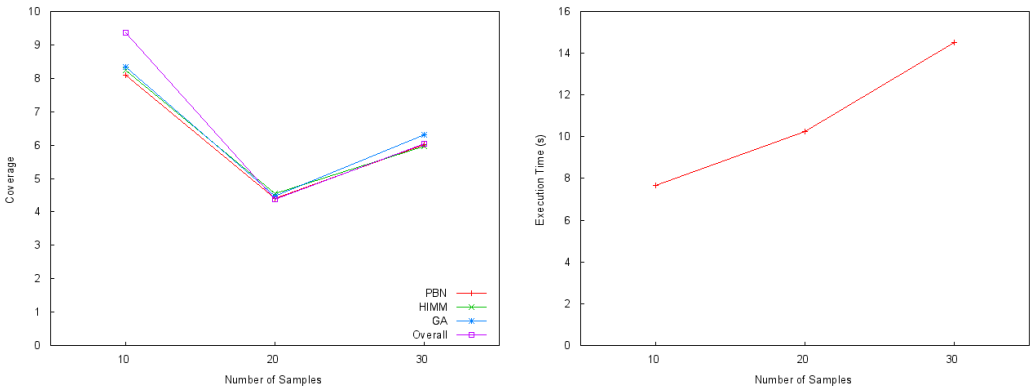

Figure 6.7: Coverage and execution times for different sample sizes

For the genetic algorithm method, the number of samples increases the initial population size. The size of the initial population has an impact on the coverage and diversity of the solution space. However since we measure coverage and diversity measures by calculating distances of new samples to the original sample set and since the changes in the original sample set are reflected in the new samples, the diversity and coverage values does not change drastically.

For the HIMM method, each original sample is treated as a separate string and the number of samples only affect the number of models produced. Since the uncertainty of each model are independent of the other models, the new samples are not much affected by the increase in the number of models.

However it is possible for the PBN model to build unsuccessful models with little number of samples. With little number of initial samples it might not be possible to properly derive the network parameters and the PBN network constructed might not model the gene expression data properly. Then the new samples produced might not be compatible with the original sample space at all. However this was not observed at the experiment. The compatibility values of the produced samples are reasonable. We can conclude that even with 10 samples, PBN parameters are successfully learned and PBN model correctly reflected the original sample space.

The execution time of the sample generation increases slightly with the increasing number of original samples. The slight increase in the execution time is expected since the number of samples is only a multiplier factor at most for the complexity of each method. The PBN construction method iterates over the original samples, so increasing the number of samples increases the execution time linearly. For HIMM models, construction of each model only depends on the number of genes. The number of samples affect the number of HIMM models constructed and this contributes to the execution time linearly. For genetic algorithm, the original population size only effects parent selection procedure slightly at the initial generations and does not have an important effect on the execution time at all.

Note that in this experiment the combined samples produced performs close to the average of each method for each three evaluation metrics, even slightly better than the individual methods. We can say that this experiment clearly shows the success of

using a multi-model approach.

6.4.3 Experiments on Number of Samples Produced

For the next experiment our goal is to analyze how our method performs when it is producing different number of samples. For this purpose by using a 30 sample, 20 gene data subset of our data set, we run our method multiple times. In the runs we generated 5, 15, 30, 45 and 60 new samples.

The compatibility, diversity,coverage values and execution times are shown in figures 6.8 and 6.9.

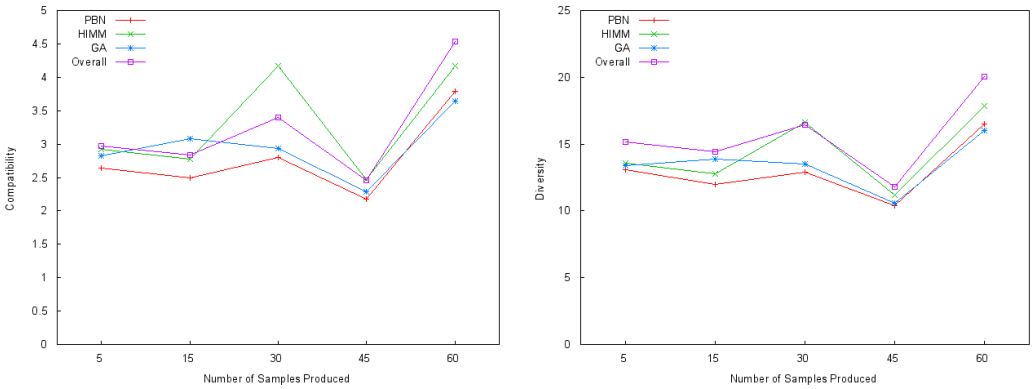

Figure 6.8: Compatibility and diversity values with different number of samples produced

In this experiment the results are similar to the previous one. We see that the evaluation metrics are slightly disturbed when we generate more samples. When the number

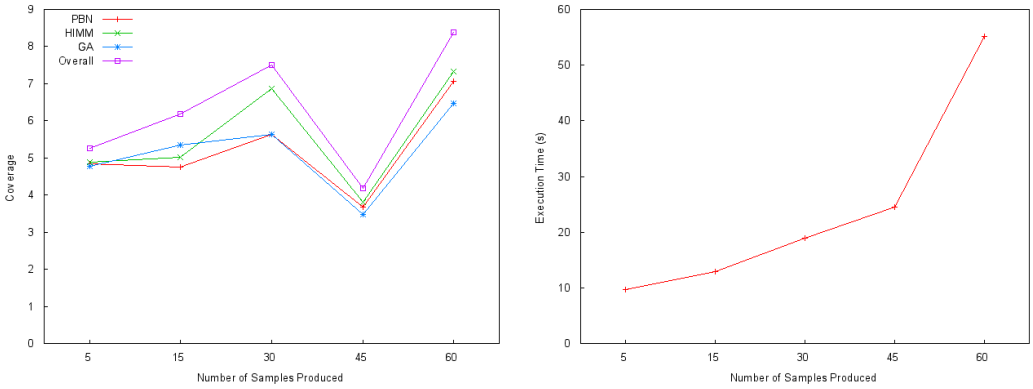

Figure 6.9: Coverage and execution times for different number of samples produced

of samples produced is 60, all evaluation metrics increase. We can conclude that when more samples are produced, they cover the whole sample space more successfully and are diverse, with a slight decrease in compatibility.

The model construction phases of PBN and HIMM methods are completely independent of samples to be produced. Thus the model quality is not affected by number of samples to be produced at all. Our genetic algorithm approach produces as many generation as the number of samples to be produced. In this sense, the number of samples to be produced has an impact on the method. However the compatibility, diversity and coverage values are not significantly affected in this experiment. We can conclude that the crossover and mutation mechanisms used has a more significance impact on these metrics than the number of generations. Thus even the number of generations increase, the quality of the new samples do not change much accordingly.

The execution times again rise slightly, however faster than the previous experiment. We can conclude that the number of samples produced is a larger multiplier than the original sample size. However even this increase is slow. When 10 times more samples are produced, the execution only increases 5 times.

The number of samples to be produced only affects the sample generation phases of PBN and HIMM methods. However genetic algorithm method does not have a distinction of model construction and sample generation, so when the number of new samples to be produced increases, the execution time of the model increases linearly. The evaluation and selection of the final samples are also have computational complexity linear in terms of number of samples to be produced. Thus the overall execution time curve increases linearly with the increase in the number of new samples.

Also note that the combined samples has a better diversity and coverage than each model at each run. The compatibility of the combined samples is close to the average compatibility of each model too.

6.4.4 Crosschecking the Partitions of the Original Sample Space

As a last experiment, we tried to design an experiment that will crosscheck whether our method can predict some of the missing samples in a data set. We prepared a 30

sample, 20 gene subset of our original data. We split this 30 samples into 10 groups. We executed 10 runs which involves producing 3 samples from 9 of the 10 groups of the data set. After each run we evaluated the produced samples by comparing them with the omitted group.

Only compatibility values for the combined data is reported for this experiment. The coverage and diversity metrics are not beneficial for evaluating our method in this prediction task.

The compatibility values of each run is shown in figure 6.10.

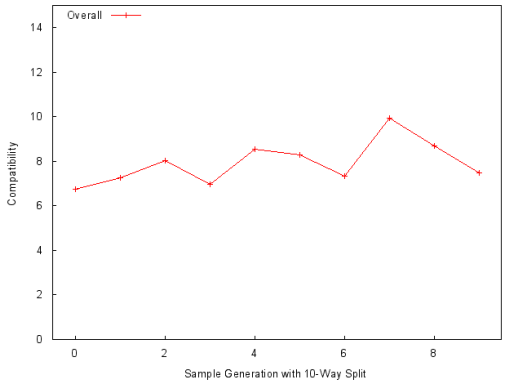

Figure 6.10: Compatibility values for crosscheck experiment

The compatibility values of each run is consistent. However the values are higher than the compatibility values from the previous experiments. The main reason is our method is not specially designed for predicting missing data points. All the models we use try to produce samples similar to the original sample space. Since samples of the gene expression data are not close to each other generally the compatibility values of our approach are higher than the previous results.

However we want to note that these compatibility values are slightly lower than the average distance between two 20-dimension random vectors.

The consistency in the compatibility values clearly shows that our method does not responds to slight changes in the original data. If that were the case we could see significantly higher and lower compatibility values than the average trend. But all the instances of the experiment produce similar compatibility values. Thus we can conclude that our approach is robust enough.

6.4.5 Evaluating the Generated Data via Biomarker Analysis

In this subsection we are presenting an alternative approach for evaluating our data generation method. For this evaluation we use a clustering based biomarker method [8]. Our goal is to analyze whether extending a dataset with out data generation method effects the success of biomarker analysis positively or negatively.

The biomarker method we used as a reference [8] analyses gene expression data and tries to classify tissue samples as cancerous and healthy. Authors used a classical clustering method to divide the samples according to their gene expression values.

We applied the same clustering method in three datasets:

1. Original dataset
2. Original dataset + Data generated with our method
3. Set (2) divided into two groups (3a and 3b)

For each setting we used three methods for clustering dataset:

1. Clustering based on Pearson distance, using average as a linkage method and a fixed threshold for cut off.
2. Clustering based on Euclidean distance and producing 2 classes.
3. Clustering based on Pearson distance, using average as a linkage method and producing 2 classes.

First clustering method we used is the exact clustering method used in [8]. Other two methods were introduced to observe the effects of different clustering parameters to our approach.

The results of the clustering process are class labels for each of the samples. We try to measure how much does the distribution of these labels change for sets 2, 3a and 3b compared to original dataset 1. To measure the change in clusters we carry on a simple analysis over means of clusters and number of points in each cluster. We

try to match the clusters of each dataset by using a greedy approximation algorithm. After matching the clusters and sorting values for number of points in each cluster accordingly, we have a distribution vector for each dataset. Normalized distance between these vectors is used as a similarity metric.

We applied this cluster similarity analysis to metastatic melanoma data we used in the previous subsections. Figure 6.11 presents the results of the analysis. Most of the results are in 80%-100% boundary, which means that data enriched by our approach lead to clusters similarity to the original data. The results are especially successful for the original clustering method proposed in [8], however for the variations of the clustering method, data generated by our method leads to clusters deviating from the original. This deviation is only observed in the first result set. Other result sets that show the cluster similarities for combined data and its partitions report similar clusters though.

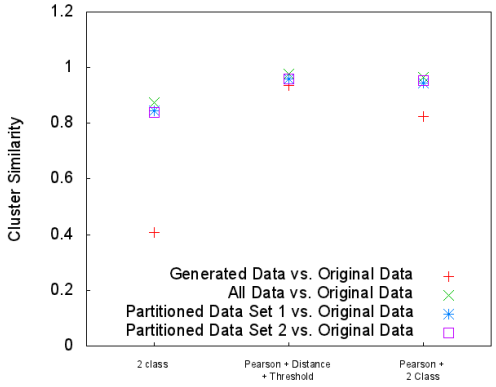

Figure 6.11: Cluster Similarities for original approach

In this experimental setting we also applied two approaches for reducing the randomness in our approach. In order to produce more stable results we

1. produced excessive amounts data and report averages

2. produced excessive amounts of candidate data from each data generation method

These results match the previous figure. For the clustering method proposed in [8] clustering similarity is still near 100%. For 2-class clustering variation, first stabilization method increases the cluster similarity drastically. For the last variation

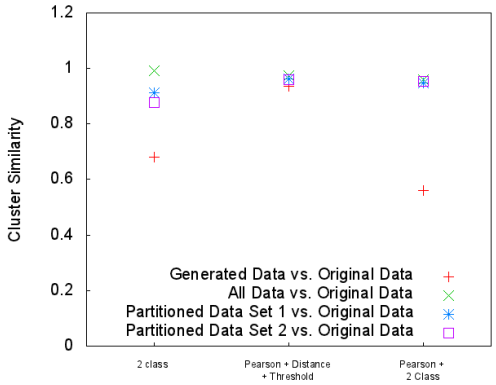

Figure 6.12: Cluster Similarities for stable approach 1

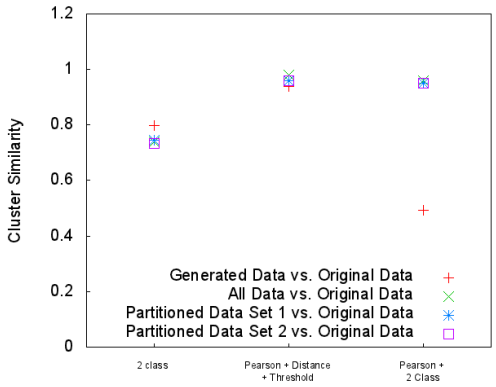

Figure 6.13: Cluster Similarities for stable approach 2

stabilization methods does not effect the cluster similarity values of combined data, however cluster similarity of generated data to original data drops for that variation as we force stability.

As a result of this analysis, we can conclude that our data generation approach has the ability to enrich the data without changing the general behavior of data sets with respect to the biomarker based clustering method we used and its variations. This supports that our method produces new data samples that are compatible with the original dataset.

CHAPTER 7

DISCUSSION, CONCLUSION AND FUTURE WORK

This research focuses on different aspects of gene regulatory network control problem. By controlling regulation of genes, our eventual goal is to control certain biological processes and activities in human cell. With detailed exploration of human genome and recent advances in bioinformatics, today we are much closer to this goal than ever. It's everyone's hope that in the near future biology and medical sciences will make vast progress on understanding and controlling genome based activities in humans. For example this would make personalized drugs possible. It is not hard to see that such advances would change medical science, biology and human society drastically.

Today, there is a research community actively working on analyzing, examining and modeling the dynamics of genome and gene expression. Progress in this area is a result of joint work of subject matter experts and computational methods of the bioinformatics. However, it is not wrong to say that most of the research in this area is directed to understanding whole dynamics of gene expression. Bits and pieces learned from studies in this wide area of research are put into practical applications as much as possible and we are still far away from revealing all the secrets of DNA.

Since the puzzle is far from complete, computational efforts on gene expression mechanism focus on incomplete and maybe incorrect views on genes and specifics of gene regulation mechanism. Due to our incomplete and inaccurate knowledge it is very hard to capture the overall gene regulation apply computational methods successfully.

GRN control problem is an important research problem in this picture. With little information of gene regulation dynamics, the control problem is overly simple. Traditionally gene regulation has been explained by gene pathways that provide a simple

view of dependencies between genes. With this approach, understanding the dependency between two genes also provides a simple control mechanism. Today we have scarce knowledge about the actual interaction between genes, so it is not a challenging task to devise control mechanisms out of this limited knowledge. In the future, with our improved knowledge on interactions between genes, simple views will not yield satisfying results for control tasks and we will find ourselves trying to solve real life control problems similar to ones being solved by computer science for decades.

This work aims to provide a contribution in the concrete realization of gene regulation control problem. Partially observability is an inherent feature of the GRN control problem. However, it is ignored mostly in the research community, due to challenges introduced by partially observability. It's been the goal of the research in this field to devise simple policies appropriate for our limited knowledge and intervention capabilities. Thus, generally a fully observed version of the problem is solved in relevant literature. Numerous works have correctly solved the problem in partially observable setting, but their formulation was simple and the solutions they are able to produce were only appropriate for a *limited* horizon setting.

In this work we presented an infinite horizon, partially observable formulation of the GRN control problem. To the best of our knowledge, our formulation is the most realistic formulation of this control problem presented in the literature. Moreover our method aims to establish first real connection between real partially observable problem domain and GRN control research by using POMDP model as a tool.

In our study, we've shown that GRN control problem can be formulated as a POMDP. We made use of some classical problem formulation approaches together with gene expression data analysis method we've introduced. By using this data analysis method we tried to establish a bond between the control problem formulation and gene expression data. Considering gene expression data is our primary source of information in the realistic case, we believe that this approach has its merits.

Our experimental studies has shown that POMDP based formulation of GRN control problem is as successful as the other similar partially observable formulations. In a finite horizon comparison setting, our POMDP based formulation managed to collect comparable amounts of reward. However the computational cost of solving our

POMDP formulation is shown to be much less than the alternative methods. The use of infinite horizon and POMDP framework in our formulation not only provided a more realistic formulation but made whole POMDP solution methods available and by careful selection of an approximation algorithm it is possible to solve the control problem much more efficiently. Alternative partially observable finite horizon formulations of GRN control problem solve the problem by iterating over horizon. Since each of these iterations over horizon have nearly exponential cost that increases with horizon, these methods all fail to produce any solution when horizon exceeds even small values.

Our formulation clearly outperforms alternative formulations in solution cost. There are different alternative formulations for POMDP model elements, and some of them are very apparent. In order to provide a richer model these alternative formulations should be studied. Most significant research topics on these alternative formulations would be using a richer action and observation model, exploring effects of alternative reward mechanism to our formulation and most importantly comparing the robustness of our data analysis based formulation of transition function with the alternatives.

In our POMDP formulation we've assumed that both gene regulatory network and gene expression data are available, however in reality the source of our information is mostly gene expression data and gene regulatory network is inferred from gene expression data via analysis by an expert or computational methods. There are important problems about the abundance of gene expression data and in this work we've also proposed a data enrichment process to increase the number of samples in a gene expression data set.

Our data enrichment process uses different models and combine data generated by them. The goal of using a hybrid approach is to prevent one model to dominate the data generated and to have an enrichment process that is independent of any property of any specific model used. The data is evaluated and selected according to some statistical metrics we presented. Our objective in formulating these statistical metrics is to provide simple means to measure statistical similarities of the original data set and generated data.

In our experiments on data enrichment process, we analyzed the statistical properties

of data generated in different settings. Assuming that the statistical metrics are able to represent our objective in enriching the data precisely, this study guides us on how to use the process presented to achieve better results. We also presented an independent study that applies a biomarker analysis on the generated data. This study aims to show that enriched data exhibits similar results with the original data with respect to the biomarker analysis carried out.

The data enrichment process we proposed has huge potential since the scarcity of data is a common problem for computational and clinic methods. Providing a trusted process for enriching the biological datasets is an invaluable asset the gene expression research. However, in order to fulfill this role, success of the data enrichment should be validated further. It is important to improve the process by introducing more relevant models to gene expression profiles, so that data generated would fit the gene expression data better. Also, in order to validate the process further it is important to define more convincing metrics and also provide a mechanism to express the objectives of the data enrichment process. Thus, an improved version of the data fusion and evaluation mechanism can be introduced. In fact this task is a recent research problem our group working on.

Finally this work also aims to solve the POMDP formulation of the GRN control problem more efficiently. Thus, we also contribute to the POMDP solving phase of the method by introducing the decomposition process. Despite the fact that they are an important problem class, difficulty in solving POMDP problems is notorious and emphasized in this work. One of the trending views in research community is that our flat and unstructured view of problems leads to poor solution performance. more efficiently and in order to scale the state spaces sizes of solvable

In this work we used a decomposition method that tries to explore weak dependencies between groups of states in the factored representation form. These weak dependencies are explored by the gene expression data analysis method. By directly using the gene expression data to explore the dependencies between genes we are able to evaluate the strength of dependencies without the effect of any model built on data. Since our POMDP formulation also depends on gene expression data analysis, we can consistently identify the problem parts that have weak linkage.

These weak dependencies allow us to decompose the problem into subproblems and our method ensures that in significant amount of the cases this decomposition leads to either discovery of components unrelated to the solution of the problem, or formulation of smaller subproblems that can be solved more efficiently or both.

The experimental results we presented for our decomposition method shows that decomposition of the problem has little overhead to the problem formulation we introduced. Most expensive phase of the method is the gene expression data analysis we performed and this analysis is essential for both POMDP formulation and decomposition. Aside from the gene expression data analysis, the decomposition costs a little overhead to the POMDP formulation cost. Our experiments showed that when decomposed subproblems were solved independently or some of the subproblems are eliminated the solution cost of the whole method can drop significantly. And more importantly, this decrease in solution cost does not cause any significant drop in the reward collected.

The experimental results shows promising results for the decomposition method we produced. However, it is vital to assess the actual success of the method in different real and synthetic problem settings. This way we can improve and fine tune the decomposition method to better utilize the problem structure. For this purposes an improved prototype of the decomposed policy generator is to be completed soon. This improved policy generator will allow us for rapid and precise application of decomposition process. It would be an important contribution to POMDP framework to improve this decomposition tool further and evolve it into a general purpose decomposed POMDP solver tool.

An important point to note is that possible improvements on POMDP formulation and POMDP decomposition shall proceed side by side. An improved and semantically richer formulation of POMDP problem would possibly yield higher chances to discover weak dependencies in the problem and similarly if we are able to improve the POMDP decomposition method and we can gain maximum benefit from the structural properties of the problem, we can explore problem formulations that would exploit the benefit of decomposition.

REFERENCES

[1] O. Abul, R. Alhajj, and F. Polat. Markov decision processes based optimal control policies for probabilistic boolean network. In *IEEE BIBE*, pages 337–344, 2004.

[2] O. Abul, R. Alhajj, and F. Polat. Optimal Multi-Objective Control Method for Discrete Genetic Regulatory Networks. In *BioInformatics and BioEngineering, 2006. BIBE 2006. Sixth IEEE Symposium on*, pages 281–284. IEEE, 2006.

[3] T. Akutsu, S. Miyano, S. Kuhara, et al. Identification of genetic networks from a small number of gene expression patterns under the boolean network model. In *Pacific Symposium on Biocomputing*, volume 4, pages 17–28. World Scientific Maui, Hawaii, 1999.

[4] R. Bellman. A markovian decision process. *Indiana Univ. Math. J.*, 6:679–684, 1957.

[5] R. E. Bellman. *Dynamic Programming*. Princeton University Press, Princeton, NJ, 1957.

[6] R. E. Bellman. *Adaptive control processes - A guided tour*. Princeton University Press, 1961.

[7] A. Bernard, A. Hartemink, et al. Informative structure priors: joint learning of dynamic regulatory networks from multiple types of data. In *Pac Symp Biocomput*, volume 10, pages 459–470, 2005.

[8] M. Bittner, P. Meltzer, Y. Chen, Y. Jiang, E. Seftor, M. Hendrix, and . et al. Molecular classification of cutaneous malignant melanoma by gene expression profiling. *Nature*, 406(6795):536–40, 2000.

[9] C. Boutilier, R. Dearden, and M. Goldszmidt. Exploiting structure in policy construction. In *IJCAI*, pages 1104–1111, 1995.

[10] D. Bryce and S. Kim. Planning for gene regulatory network intervention. In *Life Science Systems and Applications Workshop, 2006. IEEE/NLM*, pages 1–2. IEEE, 2006.

[11] D. Bryce and S. Kim. Planning for gene regulatory network intervention. In *IJCAI*, pages 1834–1839, 2007.

[12] A. R. Cassandra. Optimal policies for partially observable markov decision processes. Technical report, Brown University, Providence, RI, USA, 1994.

[13] A. R. Cassandra. *Exact and approximate algorithms for partially observable markov decision processes*. PhD thesis, Brown University, Providence, RI, USA, 1998. AAI9830418.

[14] W. Chen, H. Lu, M. Wang, and C. Fang. Gene expression data classification using artificial neural network ensembles based on samples filtering. *Artificial Intelligence and Computational Intelligence, International Conference on*, 1:626–628, 2009.

[15] H.-T. Cheng. *Algorithms for partially observable markov decision processes*. PhD thesis, 1989. AAI0567543.

[16] F. Collins, E. Lander, J. Rogers, R. Waterston, and I. Conso. Finishing the euchromatic sequence of the human genome. *Nature*, 431(7011):931–945, 2004.

[17] A. Datta, A. Choudhary, M. Bittner, and E. Dougherty. External control in markovian genetic regulatory networks: the imperfect information case. *Bioinformatics*, 20(6):924, 2004.

[18] A. Datta, A. Choudhary, M. L. Bittner, and E. R. Dougherty. External control in markovian genetic regulatory networks. *Machine Learning*, 52(1-2):169–191, 2003.

[19] E. Dougherty, S. Kim, and Y. Chen. Coefficient of determination in nonlinear signal processing. *Signal Processing*, 80(10):2219–2235, 2000.

[20] H. Franz, C. Ullmann, A. Becker, M. Ryan, S. Bahn, T. Arendt, M. Simon, S. Paabo, and P. Khaitovich. Systematic analysis of gene expression in human brains before and after death. *Genome Biology*, 6(13):R112, 2005.

[21] M. R. Green. Targeting Targeted Therapy. *N Engl J Med*, 350(21):2191–2193, 2004.

[22] J. Holland. *Adaptation in natural and artificial systems*. University of Michigan Press, 1975.

[23] L. Kaelbling, M. Littman, and A. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101(1-2):99–134, 1998.

[24] L. Kaelbling, M. Littman, and A. Moore. Reinforcement learning: A survey. *Arxiv preprint cs/9605103*, 1996.

[25] M. Kathleen Kerr and G. A. Churchill. Statistical design and the analysis of gene expression microarray data. *Genetics Research*, 77(02):123–128, 2001.

[26] S. A. Kauffman. *The Origins of Order: Self-Organization and Selection in Evolution*, chapter 5, pages 182–235. Oxford University Press, USA, 1 edition, June 1993.

[27] M. Kim, S. B. Cho, and J. H. Kim. Mixture-model based estimation of gene expression variance from public database improves identification of differentially expressed genes in small sized microarray data. *Bioinformatics*, 26(4):486–492, 2010.

[28] S. Kim, H. Li, E. R. Dougherty, N. Cao, Y. Chen, M. Bittner, and E. B. Suh. Can markov chain models mimic biological regulation?. *Journal of Biological Systems*, 10(4):337, 2002.

[29] B. Kveton, M. Hauskrecht, and C. Guestrin. Solving factored mdps with hybrid state and action variables. *Journal of Artificial Intelligence Research*, 27:2006, 2006.

[30] H. Lähdesmäki, I. Shmulevich, and O. Yli-Harja. On learning gene regulatory networks under the boolean network model. *Machine Learning*, 52(1):147–167, 2003.

[31] E. Lander, L. Linton, B. Birren, C. Nusbaum, M. Zody, J. Baldwin, K. Devon, K. Dewar, M. Doyle, W. FitzHugh, et al. Initial sequencing and analysis of the human genome. *Nature*, 409(6822):860–921, 2001.

[32] M. Lee and G. Whitmore. Power and sample size for DNA microarray studies. *Statistics in Medicine*, 21(23):3543–3570, 2002.

[33] M. L. Littman, T. L. Dean, and L. P. Kaelbling. On the complexity of solving markov decision problems. In *Proceedings of the Eleventh Annual Conference on Uncertainty in Artificial Intelligence (UAI–95)*, pages 394–402, Montreal, Québec, Canada, 1995.

[34] M. Mitchell. *An introduction to genetic algorithms*. The MIT press, 1998.

[35] G. Monahan. A survey of partially observable markov decision processes: Theory, models, and algorithms. *Management Science*, pages 1–16, 1982.

[36] R. Pal, A. Datta, M. L. Bittner, and E. R. Dougherty. Intervention in context-sensitive probabilistic Boolean networks. *Bioinformatics*, 21(7):1211–1218, 2005.

[37] R. Pal, A. Datta, and E. R. Dougherty. Optimal infinite-horizon control for probabilistic boolean networks. *IEEE Transactions on Signal Processing*, 54(6-2):2375–2387, 2006.

[38] R. Pal, A. Datta, and E. R. Dougherty. Robust intervention in probabilistic boolean networks. *IEEE Transactions on Signal Processing*, 56(3):1280–1294, 2008.

[39] R. Pal, I. Ivanov, A. Datta, M. Bittner, and E. Dougherty. Generating boolean networks with a prescribed attractor structure. *Bioinformatics*, 21(21):4021–4025, 2005.

[40] W. Pan, J. Lin, and C. Le. How many replicates of arrays are required to detect gene expression changes in microarray experiments? A mixture model approach. *Genome Biol*, 3(5):0022–1, 2002.

[41] R. Parsons, S. Forrest, and C. Burks. Genetic Algorithms for DNA Sequence Assembly. In *Proceedings of the 1st International Conference on Intelligent Systems for Molecular Biology*, page 318. AAAI Press, 1993.

[42] J. T. Pedersen and J. Moult. Genetic algorithms for protein structure prediction. *Current Opinion in Structural Biology*, 6(2):227 – 231, 1996.

[43] G. Piatetsky-Shapiro, T. Khabaza, and S. Ramaswamy. Capturing best practice for microarray gene expression data analysis. In *KDD '03: Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 407–415, New York, NY, USA, 2003. ACM.

[44] P. Poupart. *Exploiting structure to efficiently solve large scale partially observable markov decision processes.* PhD thesis, University of Toronto, Toronto, Ont., Canada, Canada, 2005.

[45] M. Schut, M. Wooldridge, and S. Parsons. On partially observable mdps and bdi models. In *Selected papers from the UKMAS Workshop on Foundations and Applications of Multi-Agent Systems*, pages 243–260, London, UK, 2002. Springer-Verlag.

[46] I. Shmulevich, E. R. Dougherty, and W. Zhang. Gene perturbation and intervention in probabilistic Boolean networks. *Bioinformatics*, 18(10):1319–1331, 2002.

[47] I. Shmulevich, A. Saarinen, O. Yli-Harja, and J. Astola. Inference of genetic regulatory networks via best-fit extensions. *Computational and statistical approaches to genomics*, pages 197–210, 2003.

[48] T. Smith, D. R. Thompson, and D. S. Wettergreen. Generating exponentially smaller POMDP models using conditionally irrelevant variable abstraction. In *Int. Conf. on Applied Planning and Scheduling*, 2007.

[49] E. J. Sondik. *The optimal control of partially observable Markov processes.* PhD thesis, Stanford University, 1971.

[50] M. Spaan and N. Vlassis. Perseus: Randomized point-based value iteration for pomdps. *Journal of artificial intelligence research*, 24(1):195–220, 2005.

[51] R. Sutton and A. Barto. *Reinforcement learning: An introduction*, volume 1. Cambridge Univ Press, 1998.

[52] G. Sywerda. Uniform crossover in genetic algorithms. In *Proceedings of the third international conference on Geneticalgorithms*, pages 2–9, San Francisco, CA, USA, 1989. Morgan Kaufmann Publishers Inc.

[53] M. Tan, R. Alhajj, and F. Polat. Large-scale approximate intervention strategies for probabilistic Boolean networks as models of gene regulation. In *BioInformatics and BioEngineering, 2008. BIBE 2008. 8th IEEE International Conference on*, pages 1–6. IEEE, 2008.

[54] M. Tan, R. Alhajj, and F. Polat. Automated large-scale control of gene regulatory networks. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, 40(2):286 –297, April 2010.

[55] M. Tan, R. Alhajj, and F. Polat. Scalable approach for effective control of gene regulatory networks. *Artif. Intell. Med.*, 48(1):51–59, 2010.

[56] M. Tan, M. AlShalalfa, R. Alhajj, and F. Polat. Combining multiple types of biological data in constraint-based learning of gene regulatory networks. In *Computational Intelligence in Bioinformatics and Computational Biology, 2008. CIBCB '08. IEEE Symposium on*, pages 90 –97, sept. 2008.

[57] C. Thornton. Hierarchical markov modeling for generative music. In *Proceedings of The Eighth International Conference onMachine Learning and Applications (ICMLA 2009)*, pages 49 – 52, 2009.

[58] M. van Iterson, P. 't Hoen, P. Pedotti, G. Hooiveld, J. den Dunnen, G. van Ommen, J. Boer, and R. Menezes. Relative power and sample size analysis on gene expression profili ng data. *BMC Genomics*, 10(1):439, 2009.

[59] J. C. Venter, M. D. Adams, E. W. Myers, P. W. Li, R. J. Mural, G. G. Sutton, H. O. Smith, M. Yandell, C. A. Evans, R. A. Holt, J. D. Gocayne, P. Amanatides, R. M. Ballew, D. H. Huson, J. R. Wortman, Q. Zhang, C. D. Kodira, X. H. Zheng, L. Chen, M. Skupski, G. Subramanian, P. D. Thomas, J. Zhang, G. L. Gabor Miklos, C. Nelson, S. Broder, A. G. Clark, J. Nadeau, V. A. McKusick, N. Zinder, A. J. Levine, R. J. Roberts, M. Simon, C. Slayman, M. Hunkapiller, R. Bolanos, A. Delcher, I. Dew, D. Fasulo, M. Flanigan, L. Florea, A. Halpern, S. Hannenhalli, S. Kravitz, S. Levy, C. Mobarry, K. Reinert, K. Remington, J. Abu-Threideh, E. Beasley, K. Biddick, V. Bonazzi, R. Brandon, M. Cargill, I. Chandramouliswaran, R. Charlab, K. Chaturvedi, Z. Deng, V. D. Francesco, P. Dunn, K. Eilbeck, C. Evangelista, A. E. Gabrielian, W. Gan, W. Ge, F. Gong, Z. Gu, P. Guan, T. J. Heiman, M. E. Higgins, R.-R. Ji, Z. Ke, K. A. Ketchum, Z. Lai, Y. Lei, Z. Li, J. Li, Y. Liang, X. Lin, F. Lu, G. V. Merkulov, N. Milshina, H. M. Moore, A. K. Naik, V. A. Narayan, B. Neelam, D. Nusskern, D. B. Rusch, S. Salzberg, W. Shao, B. Shue, J. Sun, Z. Y. Wang, A. Wang, X. Wang, J. Wang, M.-H. Wei, R. Wides, C. Xiao, C. Yan, A. Yao, J. Ye, M. Zhan, W. Zhang, H. Zhang, Q. Zhao, L. Zheng, F. Zhong, W. Zhong, S. C. Zhu, S. Zhao, D. Gilbert, S. Baumhueter, G. Spier, C. Carter, A. Cravchik, T. Woodage, F. Ali, H. An, A. Awe, D. Baldwin, H. Baden, M. Barnstead, I. Barrow, K. Beeson, D. Busam, A. Carver, A. Center, M. L. Cheng, L. Curry, S. Danaher, L. Davenport, R. Desilets, S. Dietz, K. Dodson, L. Doup, S. Ferriera, N. Garg, A. Gluecksmann, B. Hart, J. Haynes, C. Haynes, C. Heiner, S. Hladun, D. Hostin, J. Houck, T. Howland, C. Ibegwam, J. Johnson, F. Kalush, L. Kline, S. Koduru, A. Love, F. Mann, D. May, S. McCawley, T. McIntosh, I. McMullen, M. Moy, L. Moy, B. Murphy, K. Nelson, C. Pfannkoch, E. Pratts, V. Puri, H. Qureshi, M. Reardon, R. Rodriguez, Y.-H. Rogers, D. Romblad, B. Ruhfel, R. Scott, C. Sitter, M. Smallwood, E. Stewart, R. Strong, E. Suh, R. Thomas, N. N. Tint, S. Tse, C. Vech, G. Wang, J. Wetter, S. Williams, M. Williams, S. Windsor, E. Winn-Deen, K. Wolfe, J. Zaveri, K. Zaveri, J. F. Abril, R. Guigo, M. J. Campbell, K. V. Sjolander, B. Karlak, A. Kejariwal, H. Mi, B. Lazareva, T. Hatton, A. Narechania, K. Diemer, A. Muruganujan, N. Guo, S. Sato, V. Bafna, S. Istrail, R. Lippert, R. Schwartz, B. Walenz, S. Yooseph, D. Allen, A. Basu, J. Baxendale, L. Blick, M. Caminha, J. Carnes-Stine, P. Caulk, Y.-H. Chiang, M. Coyne, C. Dahlke, A. D. Mays, M. Dombroski, M. Donnelly, D. Ely, S. Esparham, C. Fosler, H. Gire, S. Glanowski, K. Glasser, A. Glodek, M. Gorokhov, K. Graham, B. Gropman, M. Harris, J. Heil, S. Henderson, J. Hoover, D. Jennings, C. Jordan, J. Jordan, J. Kasha, L. Kagan, C. Kraft, A. Levitsky, M. Lewis, X. Liu, J. Lopez, D. Ma, W. Majoros, J. McDaniel, S. Murphy, M. Newman, T. Nguyen, N. Nguyen, M. Nodell, S. Pan, J. Peck, M. Peterson, W. Rowe, R. Sanders, J. Scott, M. Simpson, T. Smith, A. Sprague, T. Stockwell, R. Turner, E. Venter, M. Wang, M. Wen, D. Wu, M. Wu, A. Xia, A. Zandieh, and X. Zhu. The sequence of the human genome. *Science*, 291(5507):1304–1351, 2001.

[60] C. Wei, J. Li, and R. Bumgarner. Sample size for detecting differentially expressed genes in microarray experiments. *BMC genomics*, 5(1):87, 2004.

[61] I. H. Witten. Adaptive text mining: inferring structure from sequences. *Journal of Discrete Algorithms*, 2(2):137 – 159, 2004. Combinatiorial Pattern Matching.

[62] J. Yu, V. A. Smith, P. P. Wang, E. J. Hartemink, and E. D. Jarvis. Using bayesian network inference algorithms to recover molecular genetic regulatory networks. In *International Conference on Systems Biology*, 2002.

CURRICULUM VITAE

PERSONAL INFORMATION

Surname, Name: Erdoğdu, Utku

Nationality: Turkish (TC)

Date and Place of Birth: 9 May 1980, Ankara

Marital Status: Single

Phone: +90 312 2105593

Fax: +90 312 2105544

EDUCATION

Degree	Institution	Year of Graduation
M.S.	METU Computer Engineering	2004
Minor	METU Philosophy (History of Philosophy)	2002
B.S.	METU Computer Engineering	2001
High School	Ankara Atatürk Anatolian High School	1997

PROFESSIONAL EXPERIENCE

Year	Place	Enrollment
2011-2012	METU Computer Engineering	TUBITAK Project Scholarship Student
2009-2010	University of Calgary	Visiting Scholar
2006-2008	METU TSK Modsimmer	Conceptual Model and Software Designer
2001-2008	METU Computer Engineering	Research Assistant
2000	METU Software R&D Center	Internship/Part Time Programmer & Tester
1999	Aselsan Electronics	Internship

PUBLICATIONS

International Conference Publications

Utku Erdogdu, Mehmet Tan, Reda Alhajj, Faruk Polat, Douglas J. Demetrick, Jon G. Rokne: Employing Machine Learning Techniques for Data Enrichment: Increasing the Number of Samples for Effective Gene Expression Data Analysis. BIBM 2011: 238-242

Utku Erdogdu, Reda Alhajj, Faruk Polat: The Benefit of Decomposing POMDP for Control of Gene Regulatory Networks. IAT 2011: 381-385

Utku Erdogdu, Faruk Polat: Dynamic cooperation using resource based planning. AAMAS 2006: 329-331