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

SEPTEMBER 2011

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 Department, University 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

September 2011, 79 pages

The need to analyze and closely study the genes related mechanism motivated for 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. The GRN control problem has been studied mostly with the aid of probabilistic boolean networks, and corresponding control policies have been devised. Though 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 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). The method proposed in this work is a POMDP formulation

for the infinite horizon version of the problem. This formulation first decomposes the problem by isolating different unrelated parts of the problem, and then makes use of existing POMDP solvers to solve the obtained subproblems; the final outcome is a control mechanism for the main problem. The reported test results using both synthetic and real GRNs are promising in demonstrating the applicability, effectiveness and efficiency of the proposed approach.

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

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

Eylül 2011, 79 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. GDA kontrol problemi en çok olasılıksal boole ağları yardımıyla çalışılmaktadır ve bu model yardımıyla kontrol problemlerine çözümler üretilmektedir. 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 önerilen

metot problemin sonsuz adımdan oluşan bir halinin POMDP ile tanımlanmasıdır. Bu tanımlama öncelikle problemin birbirinden bağımsız kısımlarını soyutlamakta ve daha sonra elde edilen alt-problemleri varolan bir POMDP çözücü yardımıyla çözmektedir; sonuçta ana problem için bir çözüm elde edilmektedir. Sunulan test sonuçları önerilen metodun hem doğal hem de yapay GDA'lar için uygulanabilirliğini, etkisini ve başarımını göstermek konusunda başarılıdır.

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

Dedication Goes Here

ACKNOWLEDGMENTS

Acknowledgements Goes Here

TABLE OF CONTENTS

LIST OF TABLES

TABLES

LIST OF FIGURES

FIGURES

CHAPTER 1

Introduction

It is true that each cell in a living organism contain the same DNA sequence; but except primitive organisms, each living organism has different kinds of cells specialized in some functions. The mechanism behind this differentiation is the existence of factors effecting the transcription process.

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, 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 intention of our research described in this paper is to help in identifying the malfunctioning molecules by developing a more natural approach capable of deriving policies for the gene regulatory network (GRN) control problem.

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 paper, 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 expression network [14]. 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 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

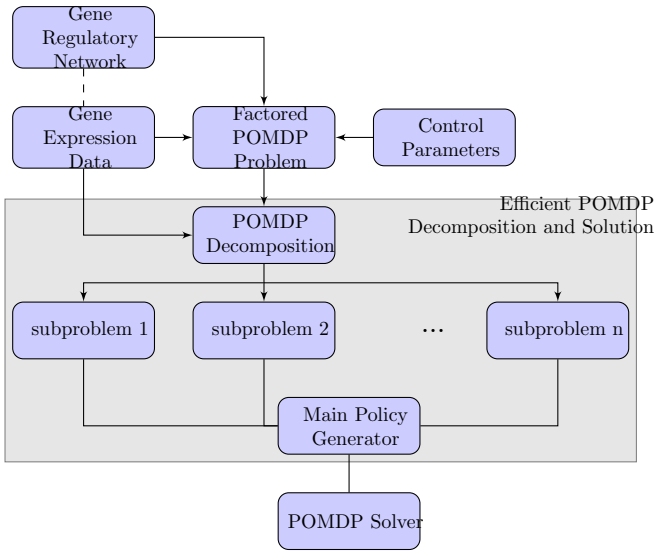

Figure 1.1: Block Diagram of the proposed POMDP Formulation and Solution Framework

because of partial observability of real states.

In this paper, 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.

An abstract block diagram of the approach 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 expression 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 expression networks inferred from real data. Also, 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 summary, contributions of the problem solving method described in this paper 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.

2. 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.

3. 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.

4. We conducted experiments using both synthetics and real GRNs; we compared the proposed approach with the other existing approaches by considering space and time utilization.

The rest of this paper is organized as follows. Chapter 2 briefly overviews the related work and Chapter 3 introduces some background material. Chapter 4 presents the proposed method of using POMDP in handling the GRN control problem. Chapter 5 covers gene expression data analysis. Chapter 6 describes the process developed for the decomposition of POMDP problem. Chapter 7 describes the conducted experiments and discusses the results. Chapter 8 is conclusions and future research directions

CHAPTER 2

Related Work

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 TFs 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 [11, 20, 19]. 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 [12]. 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 [14]. PBN model network dynamics and different approaches can be used for solving control problems defined on this representation. Expressing the problem in terms of an 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. [23] and the work of Datta et al. [11] who studied the dynamics of PBN model in the probabilistic context

of Markov chains. The work of Pal et al. [18] explores an alternative model, namely context-sensitive probabilistic Boolean network 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 network and different formulations of the intervention problem, including MDP formulation [2, 1, 25, 26, 27]. 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 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 our techniques have been effective and well received by the research community [2, 1, 25, 26, 27], we realized the need for incorporating partial observability in the problem definition; and accordingly, the target of our approach described in this paper 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. [11] developed a finite horizon control algorithms for GRNs; their algorithms can work with imperfect information. They modeled the gene regulatory network as a PBN and used Markovian probabilities for their control algorithm, which is basically an optimization algorithm based on dynamic programming. Bryce et al. [7] were first to model the GRN control problem as a POMDP. Their approach makes use of PBN representation of the problem and their POMDP formulation is similar to the PBN formulation. Their focus was on solving finite horizon problems, and obviously they did not use a POMDP solver for solving the derived POMDP problem. They applied a probabilistic planning algorithm for extracting an intervention mechanism for a finite horizon. Our approach described in this paper 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

Partially Observable Markov Decision Processes and Factored Representations

In this chapter, we presented the fundementatl concepts on Partially Observable Markov Decision Processes and factored represenation schemas used to represent and efficiently solve POMDP problems.

3.1 Markov Decision Processes

Markov Decision Process (MDP) is a framework for studying decision making problems in probabilistic domains.

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.

Also, the Markovian property reduces the solution complexity, since a 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

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

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.

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 considering execution histories of 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*).

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

$$V_0^\delta(s) = \sum_{s'} T(s, s', \delta(s))[R(s, s', \delta(s))]$$

$V_t^\delta(s)$ is the expected cumulative reward of applying policy δ for $t+1$ steps, starting at state s.

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. The pseudocode for the algorithm is given in Figure 3.1.

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. 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.

Below formulation is 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 finding the real value function, V^*.

Then we find the V^*, $\delta(s)$ 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^*. Algoritm starts with an initial random guess of V^*, called V^0 [1].A refined version of V^0, V^1 can be constructed by using the cummulative 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 preffered to use a discount factor $\gamma(0 < \gamma < 1)$ to speed up the convergence to V^*:

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

[1] It's important not to confuse this superscript with the numerical subscript used to indicate the horizon of the function.

Thus the actual reward values influence V^1 more then 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 continously 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 $|S|$.

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

Figure 3.1: Value Iteration Algorithm

3.1.6 Q-Learning Algorithm

It's sometimes preferable to refine the value function and define it from $S \times A$. So instead of $V(s)$ we have $Q(s, a)$. A similar iterative algorithm can update Q values.

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

At each iteration of the formula, we should apply it to all "state-action pairs".

The advantage of Q-Learning Algorithm is, we know the exact value of each action at each state.

Q-Learning algorithm has the same time complexity with the Simple Value Iteration Algorithm. However if we are using dynamic programming, the size of the Q-table is $\mathcal{O}(|S||A|)$ which is much bigger then an ordinary value table. And more importantly the extra information that the Q values supply is redundant for solving an MDP problem. It is possible to deduce a policy without this extra knowledge.

Q-values becomes important if the problem is a learning problem, where state transition function is unknown to agent (as the name Q-Learning suggests). For such problems, it is not possible to maximize over all actions, unless the agent has not executed them and perceived the actual next state. So it is not easy to apply Value Iteration algorithm. Since Q-Learning does not have a maximization over actions, traces of agent's actions can be fed into Q-update rule as (s, s', a) triplets.

3.1.7 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.

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:

1. Value Calculation Phase

2. Policy Improvement Phase:

3.1.7.1 Value Calculation Phase

In this 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.

3.1.7.2 Policy Improvement Phase

It is possible to create a refined version of the policy by using the value function. This phase is identical to policy extraction step of Value Iteration algorithm.

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

3.1.7.3 Termination Condition

When policy 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 optimal policy again.

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 original one, the algorithm terminates, converging to optimal policy.

3.1.7.4 Computational Complexity of the Algorithm

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[17].

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 [13, 8, 9].

The model is formulated as

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

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, however worthless, to construct the policy as a state-action mapping since state information is not accessible. Then the policy should also contain a way to determine the current state to use this state-action pair.

It is not possible 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.

Since these approaches should not work, it is necessary to construct a more sophisticated policy 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.

Transition function of the new problem is defined as probability of perceiving an observation of POMDP problem, given a history-state and an action. New state is just the history-state, concatanated 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 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 new 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.

3.2.1.2 Belief State Based Algorithms

History-state based approach has a huge state space and a simpler approach is possible if the system dynamics are well known. When the transition function and and 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 can be formulated as a vector.

The value function over belief states is a piecewise linear function. So similarly value function can be represented as a set of n dimension 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.

It is possible to use Value Iteration and find the vectors that form value function.

However there is one important obstacle, the belief state space is the set of all $n-$ *dimensional* probability distribution vectors, and this state space is a continuous one. It is known that the problem is inherently a discrete problem, however discritizing the continuous belief state space is a difficult problem.

Different algorithms are proposed for maintaining a discrete view by using the Value Iteration.

- Sondik/Monahan's Enumeration Algorithm tries to enumerate all possible useful belief states and apply Value Iteration algorithm only considering these belief states.
- 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.
- Cheng's Linear Support Algorithm tries to build the Value Function directly. The approach is similar to the One Pass Algorithm, however Linear Support Algorithm relies of geometric properties of Piecewise Linear Value Function, rather then 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.

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 based POMDP algorithms. A belief based algorithm has to find a way to deal with the continuous space as a first task. Luckily a value function of a PODMP problem is a piecewise linear and we have a simple finite mean to represent it. This also allows overcoming the challenge of the continuity.

- *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 simples possible form. Thus huge number of states and actions increases both the size of value function and time/space necessary to compute it.

- *Curse of Horizon* is an effect of Curse of Dimensionality. Most of the problems proposed for POMDP problems are iterative. 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. However one of the underlying reasons of these difficulties is POMDP models use representations more inefficient then 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 [22].

Factored POMDPs could be seen as the most important attempt made to exhibit the structure of POMDP problems [5]. 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 [24, 16]. Our intention is to use the factored PODMP in tackling the GRN control problem.

Factored POMDP can be demostrated by using an example problem. For this purpose we used RockSample problem. 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 rocks, which makes the problem partially

observable. The robot can use its long range sensor to determine the value of any rock. The sensor reading is noisy proportional to the distance to the rock sensed. The robot have specific start and goal locations.

In our examples a 4x4 environment RockSample problem is used with 4 rocks in the environment.

For RockSample domain, a possible representation is

- $S = \{L, RQ1, RQ2, RQ3, RQ4\}$ 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.

The transition function is deterministic and the observation function has probabilities associated with sense actions dependent to the distance between the agent and the sensed rock.

The reward function maps +10 reward to sampling a rock with value, -10 reward to sampling a rock without value. Also the reward function maps +10 to reaching the goal location.

This representation is previously used for algorithms working on Factored representation of POMDP problems. There is only single modification we made . We did not used extra observations or extra rewards for reinforcing the agent not to execute illegal actions. Generally to provide homogeneity in problem description, such minor modifications are embedded as extra observations or extra rewards. However our aim

is not homogeneity, on the contrary, to partition the problem using the structure it exhibits. So we were fine with omitting such modifications for the sake of homogeneity.

Last element of the factored representation is determining the relations between state variables in the given problem description. It is possible to use Dynamic Bayesian Networks for this purpose.

Figure 3.2 depicts the factored representation of RockSample problem. Each part of the figure presents the world and observation dynamics for a single action. For any action that involves moving (in particular ML for the figure), only the state variable L changes. For any action that involves sensing rock quality (in particular $SR2$ for the figure) the value of the onservation perceived is determined by the actual quality of the rock and the location of the agent. No state variable changes for this action. Finally grab action possibly changes all the rock quality state variables, depending to the location of the agent. If the agent is adjacent to a certain rock, after grabbing the rock quality variable related to that rock will change while other variables remain the same; if the agent is not adjacent to any rock, no rock quality variable will change.

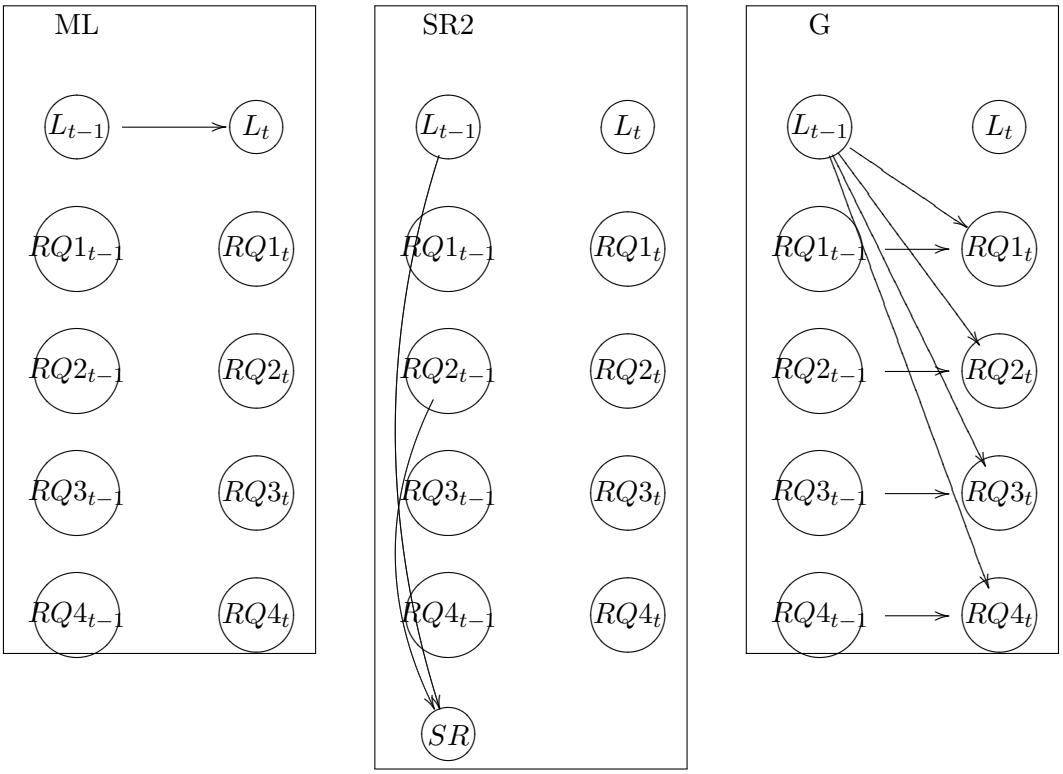

Figure 3.2: Factored dependencies of the problem

CHAPTER 4

POMDP Model Formulation for GRN Control Problem

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 paper constructs a factored POMDP.

4.1 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 expression level space. However, for a factored representation, it is easy to represent each gene as a state variable.

The most important question about accuracy of this state representation is whether all the necessary genes are included in this expression level network. In the ideal sense, since we do not 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 problem. 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 the gene expression control problem, but the assumption is strong in the generalized case.

Another important point to consider is related to components of the gene expression vector. The source 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 expression, there is not much distinction between minor differences in the expression of a gene. 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 used a binary set for discretizing gene expression levels. 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.

4.2 Actions

Actions in the gene expression control 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.

4.3 Transition Function

The transition function of the gene expression control problem expresses the interaction between genes. The gene expression problem itself does not dictate a transition function. Given a graph of the gene expression network, it is possible to formulate a

transition function using the graph structure.

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 Chapter 5.

In the ideal case, this transition function forumlation 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 graph of the GRN or the gene expression data) as close as possible. Assuming our construction actually builds a closed 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.

4.4 Observations and Observation Function

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

- Gene expression data that describes 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; some possible scenarios are described next:

1. 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.

2. 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 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.

If we concentrate on the second factor mentioned here, 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 this observation 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.

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

4.5 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, provided that two genes are expressed inclusively). This condition can be expressed as a boolean expression. 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.

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

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

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 to be specified.

CHAPTER 5

Gene Expression Data Analysis and Formulation Steps Based on This Analysis

In this chapter, we present details of the gene expression data analysis method used in our work. The goal of this method is exploring similarities between state variables, and then use these similarities in formulating a more compact POMDP that utilizes actual behavior of the system.

5.1 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 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 labelled 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. If the frequency of samples where the two genes have same

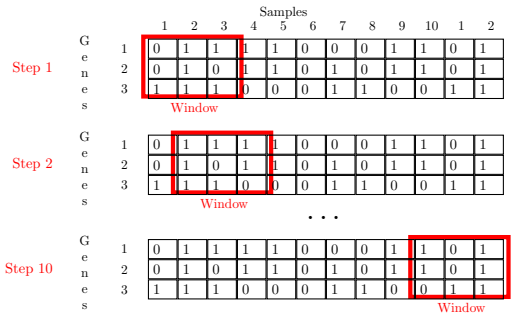

Figure 5.1: An example of shifting window for similarity analysis. Each position of the windows is a step in the algorithm. At each step, expression levels of all samples in the window are compared for each pair of genes. Window is shifter circularly.

expression levels is above some threshold, the two genes are labelled as similar for the specific position of the window. The window is shifted all through the data and the comparison is performed for each window position. If the frequency of the window positions with similar expression levels is above some threshold, the two genes are labelled 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 5.1, 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 5.1 presents the results of the analysis. Each row in the table corresponds to a step of the algorithm, and each + or - sign is a positive or negative result of the local similarity test. For example, the entry + at *Step 5* and *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 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 5.1 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

Table 5.1: Results of the example similarity analysis carried on in Figure 5.1

	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	+	-	-

are calculated as following:

$$\gamma^p(g_i, g_j) = \sum_{k=p}^{p+WS} \begin{cases} 1 & \text{if} EL_k(g_i) = EL_k(g_j) \\ 0 & \text{else} \end{cases} \tag{5.1}$$

$$SML^p(g_i, g_j) = \frac{\gamma^p(g_i, g_j)}{WS} \tag{5.2}$$

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 similarity measure as following:

$$\delta(g_i, g_j) = \sum_{k=1}^{\#ofsamples} \begin{cases} 1 & \text{if} SML^k(g_i, g_j) > \theta_l \\ 0 & \text{else} \end{cases} \tag{5.3}$$

$$SM(g_i, g_j) = \frac{\delta(g_i, g_j)}{\#ofsamples} \tag{5.4}$$

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 similar-

ities exceeding this threshold value, we obtain similarity measure function SM which maps a pair of genes to $[0, 1]$.

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 all the gene expression data (i.e., for all placements of the window) for marking them similar. When this threshold is low, only 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. 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, it is only possible to consider 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 windows 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.

In experimental evaluation chapter we also 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.

We have also improved this simple comparison algorithm in order to deal with noise in the data, namely shifted windows. For each window position, the comparison is actually performed multiple times, each time shifting the expression data for one of the genes. The genes are labelled similar for the actual window position if any of these comparisons succeeds. The improved version of the process is given as Algorithm 5.2.

The result of Algorithm 5.2 is used to serve two purposes in this work, namely identifying classes of genes and constructing the transition function; details are discussed

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, 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

Figure 5.2: Gene Expression Data Analysis

in the sequel.

5.2 Identifying Classes of Genes

The similarity relation calculated in the previous subsection 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 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 clearly an equivalence relation by definition. Thus, the 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 Algorithm 5.3. Chapter 6 presents how subproblems are created by using these classes.

5.3 Constructing the 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 transition function for the factored representation, we do not

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

Figure 5.3: Forming classes of genes

Table 5.2: 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

need a joint probability distribution of state changes. 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 a 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 5.1. 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 identically 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:

$$Pr(g_i = on, g_i^1, g_i^2, ...g_i^j) = \prod_{x=1}^{j} \begin{cases} SM(g_i, g_i^x) & \text{if } g_j^x = on, \\ 1 - SM(g_i, g_i^x) & \text{if } g_j^x = off. \end{cases} \tag{5.5}$$

$$Pr(g_i = off, g_i^1, g_i^2, ...g_i^j) = \prod_{x=1}^{j} \begin{cases} SM(g_i, g_i^x) & \text{if } g_j^x = off, \\ 1 - SM(g_i, g_i^x) & \text{if } g_j^x = on. \end{cases} \tag{5.6}$$

For factored representation, the state space is expressed in terms of each state variable seperately. Thus it is possible to directly use these probability values as descriptors of state variables. The POMDP planer 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 compatible with our previous assumption, where each gene is only influenced by genes it is connected to in th 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:

$$T(s, noop, s') = T'(s, s') \tag{5.7}$$

$$T(s, a, s') = T'(A(s), s') \tag{5.8}$$

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 5.2.

Also gene expression levels do not change instantly; we assumed that the expression level of each gene depends on the expression level of the previous state. To reflect this, all the probabilities are multiplied by 0.9 on the condition that the expression level does not change, and they are multiplied by 0.1 on the condition that the expression level changes. For example, the value 0.47 that reflects $Pr(gene_i^t = On | gene_j^{t-1} = On, gene_k^{t-1} = On)$ is further refined as $Pr(gene_i^t = On | gene_j^{t-1} = On, gene_k^{t-1} = On, gene_i^{t-1} = On) = 0.9 \times 0.47 = 0.42$ and $Pr(gene_i^t = On | gene_j^{t-1} = On, gene_k^{t-1} =$

$On, gene_i^{t-1} = Off) = 0.1 \times 0.47 = 0.04.$

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 5.2 are normalized values. Note that values in a row add up to 1.

CHAPTER 6

POMDP Decomposition

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 subproblem. 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 [3]. 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 5.1 is the method we used for identifying classes of genes. The next subsections present how the subproblems are created for each class and how these subproblems are coordinated for the solution.

Algorithm 6.1 gives a brief outline of POMDP decomposition.

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

Figure 6.1: POMDP Decomposition and Execution

6.0.1 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 4.4.

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 section.

The transition function is derived as explained in Section 5.3. 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.

6.0.2 Coordinating Subproblems

6.0.2.1 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 labelled from 1 to 8 as shown in Figure 7.1. 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.

6.0.2.2 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.

6.0.2.3 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.

6.0.2.4 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.

6.0.2.5 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.

CHAPTER 7

Experimental Evaluation

In this chapter we present experimental results of the method proposed. We designed three sets of experiments for evaluation.

In the first set of experiments we compared 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.

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

In the third 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.

7.1 Experiments on POMDP Decomposition

In this section, 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

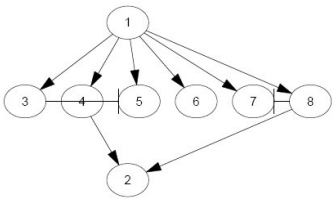

Figure 7.1: Example gene regulatory network

the two networks, the gene expression data is generated using the method proposed by Yu et al. [28]. For solving the POMDP problems, we used symbolic Perseus, a factored POMDP solver developed in Matlab by Pascal Poupart [21]. 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 (jruby 1.4.0).

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.

7.1.1 An Example Decomposition and Execution

Shown in Figure 7.1 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 7.1 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 5.1, 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.

7.1.2 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. [4]. This data contains 31 samples of 587 genes. Kim et al. [15] 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. [11] and Bryce et al. [7] 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. [11] and Bryce et al. [7].

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 outlines in the paper. 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.

7.1.3 Experiments on Synthetic GRNs

The problem formulation costs are shown in Figure 7.2. 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

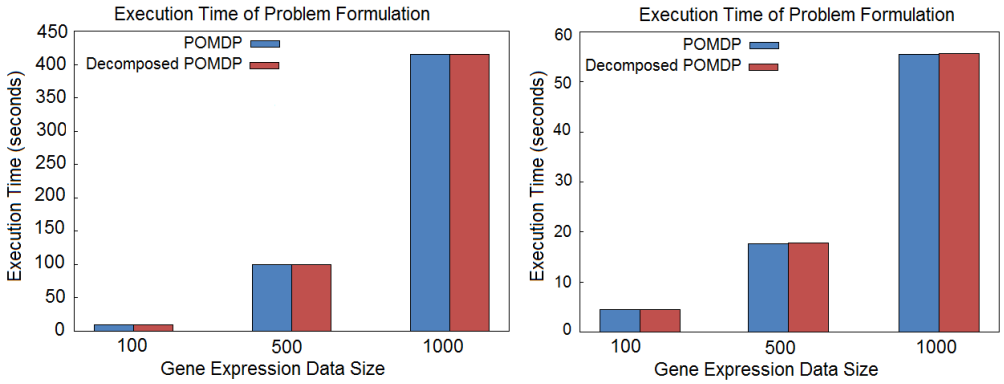

Figure 7.2: Problem formulation module execution times for two different networks

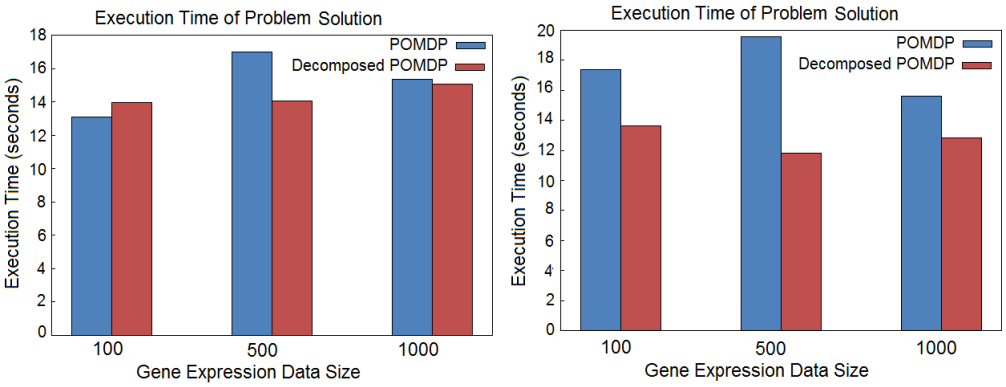

Figure 7.3: Execution times for policy generation

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 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 7.3 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 7.1.1. 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.

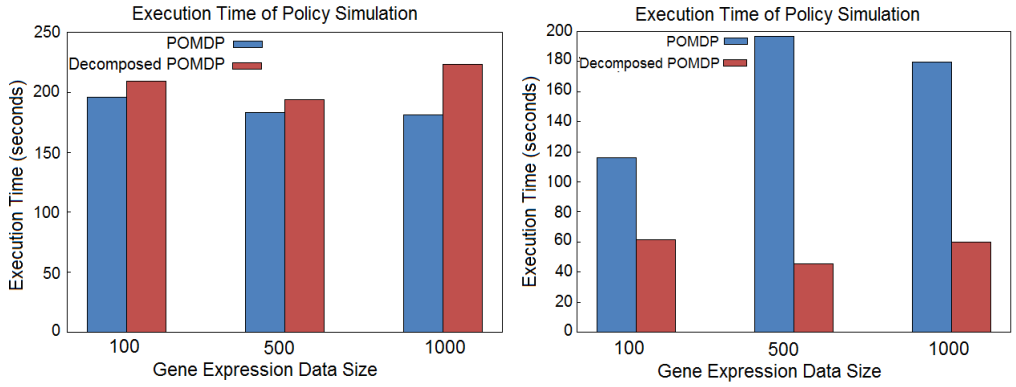

Figure 7.4: Execution times for policy execution

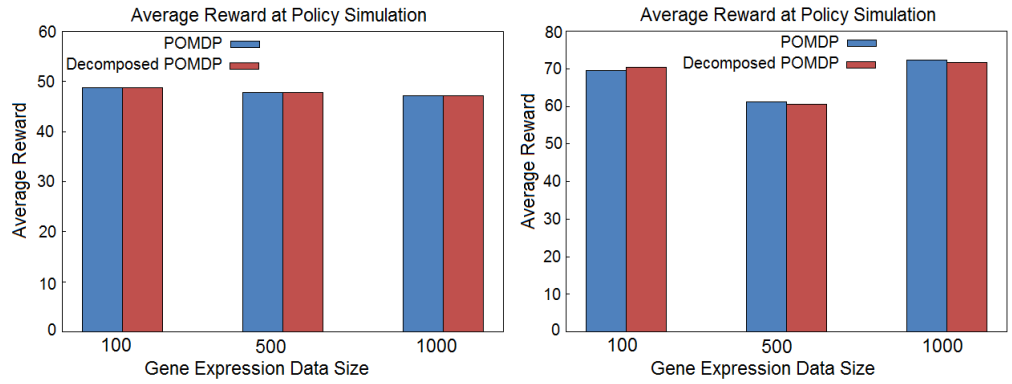

Figure 7.5: Average Rewards

Figure 7.4 presents the lengths of the simulations. For the first network, the lengths 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 7.5 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 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.

7.1.4 Experiments on Real Biological Gene Expression Data

Figures 7.6 - 7.9 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

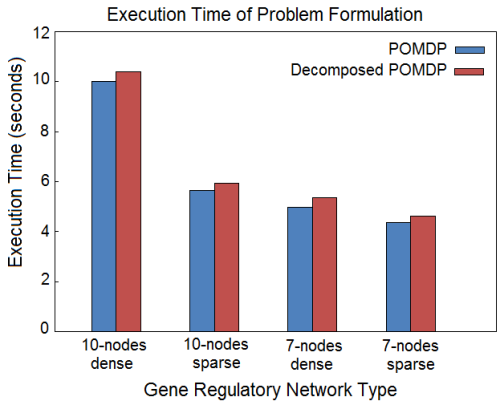

Figure 7.6: Problem formulation module execution times for four different networks.

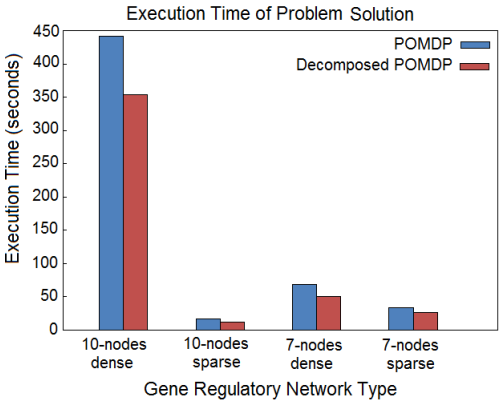

Figure 7.7: Execution times for policy generation

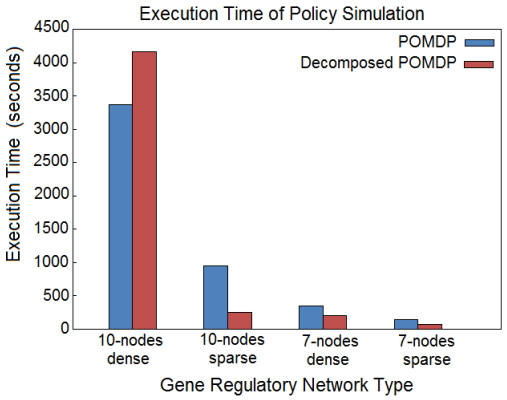

Figure 7.8: Execution times for policy execution

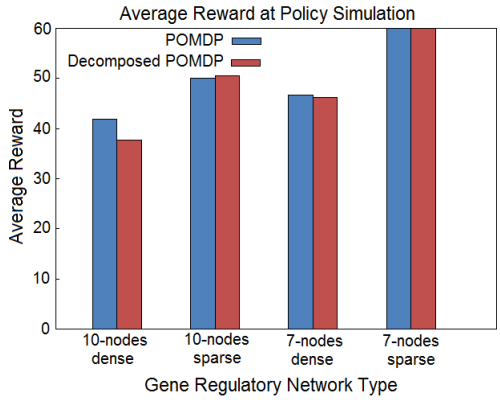

Figure 7.9: Average Rewards

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%.

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

7.2.1 Experiments on Data Order

Independent from other experimental studies, we also conducted a test for understanding the impact of the order of data. As Section 5.1 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 intractible. It is even impossible to test the impact of all different permutations by designing a one shot test.

Thus we designed a simpler test that considers an arbitrary number of permutations of or data. We used the biological 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 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.

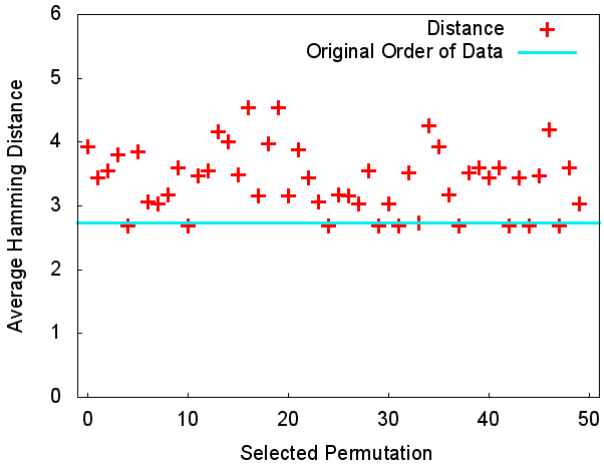

Figure 7.10: Comparison of solution similarity for different permutations of data samples

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 cases are shown in Figure 7.10

Since our data set contains 7 genes and we relabel all the analysis results, maxiumum Hamming Distanve between two analysis results are 6 (first gene is always partitioned to 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 then 4 or less then 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 then 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.

7.2.2 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 impack on the control problem to be solved. Genes that have little influnce 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 then 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.

To identify how our approach behaves with different levels of connectivity we generated a seperate 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 7.11 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.

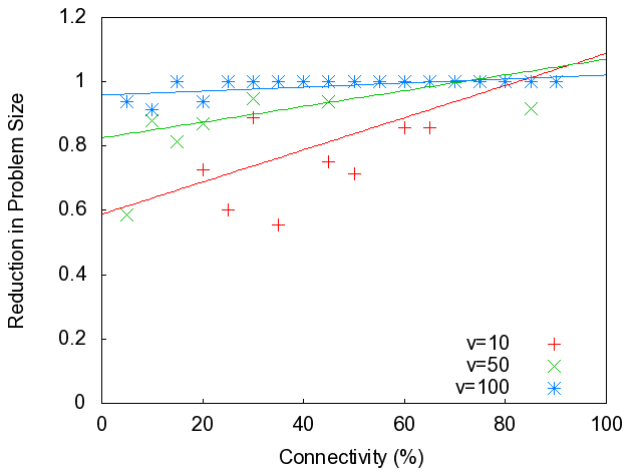

Figure 7.11: Comparison of reduction in decomposed problem size for networks with different connectiviy and size

7.3 Experiments on Comparison of POMDP Formulation with Finite Horizon Models

For the experimental tests, we build 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 [10]. 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 [6] 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. Symbolic Perseus is a POMDP solver that makes use of a point-based value iteration algorithm [21]. It can solve large POMDP problems and the POMDP solving algorithm it uses is an infinite horizon algorithm. It also uses factored POMDP representation which provides gains in performance and ease of defining problems. For working with Symbolic Perseus, our implementation prepares a problem description file that is compatible with Symbolic Perseus. Symbolic Perseus can input the problem and produce a policy. We designated our approach as UHFP which is an acronym for **U**nbounded **H**orizon **F**actored **P**OMDP.

All four methods are evaluated with a gene regulatory network based on metastatic melanoma data [4]. Kim et.al. developed a PBN model based on this gene expression data [15]. 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 UHFP method works on an unbounded horizon, we did not solve the control problem for different horizon values as other methods did. For each problem setting with given number of actions, oberservations and target genes, we run our solution 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.

7.3.1 Experimental Results

Tables 7.1-7.5 present the results of the experiments we run on four approaches. Each table present four metric for the performance of an algorithm. These metrics are solution time for the problem, memory used while solving the problem, simulation time and average reward received at simulation.

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, UHFP 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 ($h = 21$)

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 might get as big as 8 GBs. When the same algorithm is implemented with less storage usage 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. UHFP method uses significantly less memory then all other methods considered.

Horizon	Solution Time	Memory Used	Simulation Time	Average Reward
$\|A\| = 1, \|O\| = 1, \|T\| = 1$				
$h = 3$	0.16 s	395 K	0.03 s	8.7
$h = 6$	0.38 s	570 K	0.05 s	19.5
$h = 9$	13.72 s	17 M	0.05 s	46.4
$h = 12$	853.60 s	1 G	0.06 s	64.5
$h > 12$	Memory Limit Exceeded			
$\|A\| = 1, \|O\| = 2, \|T\| = 1$				
$h = 3$	0.17 s	404 K	0.02 s	9.8
$h = 6$	6.81 s	11 M	0.05 s	21.6
$h = 9$	3577.26 s	8 G	0.53 s	40.3
$h > 9$	Memory Limit Exceeded			
$\|A\| = 1, \|O\| = 3, \|T\| = 1$				
$h = 3$	0.30 s	475 K	0.02 s	13.6
$h = 6$	205.06 s	704 M	0.03 s	33.4
$h > 6$	Memory Limit Exceeded			
$\|A\| = 1, \|O\| = 4, \|T\| = 1$				
$h = 3$	0.41 s	1 M	0.02 s	11.4
$h > 3$	Memory Limit Exceeded			
$\|A\| = 1, \|O\| = 5, \|T\| = 1$				
$h = 3$	1.25 s	5 M	0.02 s	10.2
$h > 3$	Memory Limit Exceeded			
$\|A\| = 2, \|O\| = 1, \|T\| = 1$				
$h = 3$	0.25 s	535 K	0.02 s	13.9
$h = 6$	1.79 s	1 M	0.05 s	20.9
$h = 9$	325.41 s	436 M	0.05 s	42.9
$h > 9$	Memory Limit Exceeded			
$\|A\| = 2, \|O\| = 2, \|T\| = 1$				
$h = 3$	0.24 s	545 K	0.02 s	13
$h = 6$	45.23 s	84 M	0.04 s	-0.4
$h > 6$	Memory Limit Exceeded			
$\|A\| = 2, \|O\| = 3, \|T\| = 1$				
$h = 3$	0.35 s	703 K	0.02 s	14.2
$h = 6$	1630.95 s	5 G	0.07 s	28
$h > 6$	Memory Limit Exceeded			
$\|A\| = 2, \|O\| = 4, \|T\| = 1$				
$h = 3$	0.93 s	2 M	0.02 s	13.1
$h > 3$	Memory Limit Exceeded			
$\|A\| = 2, \|O\| = 5, \|T\| = 1$				
$h = 3$	5.71 s	11 M	0.03 s	11.5
$h > 3$	Memory Limit Exceeded			
$\|A\| = 3, \|O\| = 1, \|T\| = 1$				
$h = 3$	0.32 s	655 K	0.02 s	9.2
$h = 6$	6.55 s	6 M	0.03 s	24.9
$h > 6$	Memory Limit Exceeded			
$\|A\| = 3, \|O\| = 2, \|T\| = 1$				
$h = 3$	0.36 s	690 K	0.02 s	15
$h = 6$	190.53 s	352 M	0.03 s	30.9
$h > 6$	Memory Limit Exceeded			
$\|A\| = 3, \|O\| = 3, \|T\| = 1$				
$h = 3$	0.66 s	971 K	0.01 s	15.8
$h > 3$	Memory Limit Exceeded			
$\|A\| = 4, \|O\| = 1, \|T\| = 1$				
$h = 3$	0.49 s	1 M	0.02 s	14.5
$h > 3$	Memory Limit Exceeded			
$\|A\| = 4, \|O\| = 2, \|T\| = 1$				
$h = 3$	0.93 s	2 M	0.03 s	10.5
$h > 3$	Memory Limit Exceeded			
$\|A\| = 4, \|O\| = 3, \|T\| = 1$				
$h = 3$	1.47 s	3 M	0.02 s	12
$h > 3$	Memory Limit Exceeded			
$\|A\| = 5, \|O\| = 1, \|T\| = 1$				
$h = 3$	1.51 s	5 M	0.02 s	14.6
$h > 3$	Memory Limit Exceeded			
$\|A\| = 5, \|O\| = 2, \|T\| = 1$				
$h = 3$	3.26 s	11 M	0.02 s	13.3
$h > 3$	Memory Limit Exceeded			

Table 7.1: Experimental Results for OPTIMIZATION-1 Implementation

Horizon	Solution Time	Memory Used	Simulation Time	Average Reward
$\|A\| = 1, \|O\| = 1, \|T\| = 1$				
$h = 3$	1.94 s	943 K	0.17 s	14.9
$h > 3$	Time Limit Exceeded			
$\|A\| = 1, \|O\| = 2, \|T\| = 1$				
$h = 3$	46.62 s	5 M	0.42 s	12.3
$h > 3$	Time Limit Exceeded			
$\|A\| = 1, \|O\| = 3, \|T\| = 1$				
Time Limit Exceeded				
$\|A\| = 1, \|O\| = 4, \|T\| = 1$				
Time Limit Exceeded				
$\|A\| = 1, \|O\| = 5, \|T\| = 1$				
Time Limit Exceeded				
$\|A\| = 2, \|O\| = 1, \|T\| = 1$				
$h = 3$	8.79 s	2 M	0.27 s	13.2
$h > 3$	Time Limit Exceeded			
$\|A\| = 2, \|O\| = 2, \|T\| = 1$				
$h = 3$	539.92 s	15 M	0.64 s	5.12
$h > 3$	Time Limit Exceeded			
$\|A\| = 2, \|O\| = 3, \|T\| = 1$				
Time Limit Exceeded				
$\|A\| = 2, \|O\| = 4, \|T\| = 1$				
Time Limit Exceeded				
$\|A\| = 2, \|O\| = 5, \|T\| = 1$				
Time Limit Exceeded				
$\|A\| = 3, \|O\| = 1, \|T\| = 1$				
$h = 3$	44.63 s	5 M	0.30 s	12.3
$h > 3$	Time Limit Exceeded			
$\|A\| = 3, \|O\| = 2, \|T\| = 1$				
Time Limit Exceeded				
$\|A\| = 3, \|O\| = 3, \|T\| = 1$				
Time Limit Exceeded				
$\|A\| = 4, \|O\| = 1, \|T\| = 1$				
$h = 3$	177.07 s	9 M	0.39 s	13.1
$h > 3$	Time Limit Exceeded			
$\|A\| = 4, \|O\| = 2, \|T\| = 1$				
Time Limit Exceeded				
$\|A\| = 4, \|O\| = 3, \|T\| = 1$				
Time Limit Exceeded				
$\|A\| = 5, \|O\| = 1, \|T\| = 1$				
$h = 3$	533.62 s	15 M	0.51 s	13.2
$h > 3$	Time Limit Exceeded			
$\|A\| = 5, \|O\| = 2, \|T\| = 1$				
Time Limit Exceeded				

Table 7.2: Experimental Results for OPTIMIZATION-2 Implementation

Horizon	Solution Time	Memory Used	Simulation Time	Average Reward
$\|A\| = 1, \|O\| = 1, \|T\| = 1$				
$h = 3$	0.44 s	527 K	0.10 s	12.6
$h = 6$	19.14 s	2 M	0.38 s	23.94
$h > 6$	Time Limit Exceeded			
$\|A\| = 1, \|O\| = 2, \|T\| = 1$				
$h = 3$	6.86 s	1 M	0.12 s	12.54
$h > 3$	Time Limit Exceeded			
$\|A\| = 1, \|O\| = 3, \|T\| = 1$				
$h = 3$	625.35 s	10 M	0.14 s	12.08
$h > 3$	Time Limit Exceeded			
$\|A\| = 1, \|O\| = 4, \|T\| = 1$				
Time Limit Exceeded				
$\|A\| = 1, \|O\| = 5, \|T\| = 1$				
Time Limit Exceeded				
$\|A\| = 2, \|O\| = 1, \|T\| = 1$				
$h = 3$	0.63 s	732 K	0.14 s	11.4
$h = 6$	25.91 s	3 M	0.40 s	24.12
$h > 6$	Time Limit Exceeded			
$\|A\| = 2, \|O\| = 2, \|T\| = 1$				
$h = 3$	10.41 s	2 M	0.11 s	6.52
$h > 3$	Time Limit Exceeded			
$\|A\| = 2, \|O\| = 3, \|T\| = 1$				
$h = 3$	980.69 s	13 M	0.15 s	12.3
$h > 3$	Time Limit Exceeded			
$\|A\| = 2, \|O\| = 4, \|T\| = 1$				
Time Limit Exceeded				
$\|A\| = 2, \|O\| = 5, \|T\| = 1$				
Time Limit Exceeded				
$\|A\| = 3, \|O\| = 1, \|T\| = 1$				
$h = 3$	0.81 s	936 K	0.12 s	12.2
$h = 6$	42.03 s	4 M	0.36 s	32.83
$h > 6$	Time Limit Exceeded			
$\|A\| = 3, \|O\| = 2, \|T\| = 1$				
$h = 3$	13.28 s	2 M	0.11 s	14.7
$h > 3$	Time Limit Exceeded			
$\|A\| = 3, \|O\| = 3, \|T\| = 1$				
$h = 3$	1121.24 s	16 M	0.15 s	15.82
$h > 3$	Time Limit Exceeded			
$\|A\| = 3, \|O\| = 4, \|T\| = 1$				
Time Limit Exceeded				
$\|A\| = 4, \|O\| = 1, \|T\| = 1$				
$h = 3$	1.02 s	1 M	0.12 s	11.24
$h = 6$	44.10 s	4 M	0.37 s	18.75
$h > 6$	Time Limit Exceeded			
$\|A\| = 4, \|O\| = 2, \|T\| = 1$				
$h = 3$	19.82 s	3 M	0.12 s	10.6
$h > 3$	Time Limit Exceeded			
$\|A\| = 4, \|O\| = 3, \|T\| = 1$				
Time Limit Exceeded				
$\|A\| = 5, \|O\| = 1, \|T\| = 1$				
$h = 3$	1.26 s	1 M	0.19 s	11.18
$h = 6$	63.35 s	5 M	0.37 s	36.8
$h > 6$	Time Limit Exceeded			
$\|A\| = 5, \|O\| = 2, \|T\| = 1$				
$h = 3$	28.93 s	4 M	0.13 s	12.98
$h > 3$	Time Limit Exceeded			

Table 7.3: Experimental Results for AO* Implementation

Horizon	Solution Time	Memory Used	Simulation Time	Average Reward
	$\lvert A\rvert = 1, \lvert O\rvert = 1, \lvert T\rvert = 1$			
$h = 3$			0.95 s	13.28
$h = 6$			1.89 s	21.65
$h = 9$			2.64 s	28.12
$h = 12$	8.34 s	26 K	3.42 s	33.22
$h = 15$			4.30 s	37.65
$h = 18$			5.28 s	40.80
$h = 21$			5.86 s	42.79
	$\lvert A\rvert = 1, \lvert O\rvert = 2, \lvert T\rvert = 1$			
$h = 3$			1.16 s	13.44
$h = 6$			2.25 s	23.50
$h = 9$			2.66 s	31.34
$h = 12$	8.68 s	64 K	3.31 s	36.94
$h = 15$			3.88 s	40.83
$h = 18$			4.55 s	43.86
$h = 21$			5.21 s	45.90
	$\lvert A\rvert = 1, \lvert O\rvert = 3, \lvert T\rvert = 1$			
$h = 3$			0.93 s	13.28
$h = 6$			1.49 s	23.40
$h = 9$			2.41 s	30.93
$h = 12$	8.79 s	66 K	3.00 s	37.39
$h = 15$			3.58 s	41.64
$h = 18$			4.15 s	44.47
$h = 21$			4.81 s	46.60
	$\lvert A\rvert = 1, \lvert O\rvert = 4, \lvert T\rvert = 1$			
$h = 3$			0.82 s	13.28
$h = 6$			1.27 s	23.15
$h = 9$			1.80 s	30.10
$h = 12$	11.03 s	68 K	2.39 s	35.45
$h = 15$			2.97 s	39.12
$h = 18$			3.55 s	40.32
$h = 21$			4.00 s	41.27
	$\lvert A\rvert = 1, \lvert O\rvert = 5, \lvert T\rvert = 1$			
$h = 3$			0.86 s	13.28
$h = 6$			1.46 s	22.93
$h = 9$			2.08 s	30.28
$h = 12$	16.80 s	70 K	2.55 s	35.88
$h = 15$			2.99 s	39.40
$h = 18$			3.47 s	41.85
$h = 21$			3.95 s	43.73
	$\lvert A\rvert = 2, \lvert O\rvert = 1, \lvert T\rvert = 1$			
$h = 3$			0.95 s	13.28
$h = 6$			1.96 s	21.65
$h = 9$			2.76 s	28.12
$h = 12$	9.32 s	27 K	3.63 s	33.22
$h = 15$			4.48 s	47.65
$h = 18$			5.38 s	40.80
$h = 21$			6.22 s	42.79
	$\lvert A\rvert = 2, \lvert O\rvert = 2, \lvert T\rvert = 1$			
$h = 3$			1.34 s	13.44
$h = 6$			2.02 s	23.50
$h = 9$			2.77 s	31.34
$h = 12$	8.42 s	67 K	3.46 s	36.94
$h = 15$			4.19 s	40.83
$h = 18$			4.96 s	43.86
$h = 21$			5.73 s	45.90
	$\lvert A\rvert = 2, \lvert O\rvert = 3, \lvert T\rvert = 1$			
$h = 3$			1.05 s	13.28
$h = 6$			1.67 s	23.40
$h = 9$			2.38 s	30.94
$h = 12$	10.35 s	68 K	2.88 s	37.39
$h = 15$			4.33 s	41.64
$h = 18$			4.06 s	44.47
$h = 21$			4.68 s	46.60
	$\lvert A\rvert = 2, \lvert O\rvert = 4, \lvert T\rvert = 1$			
$h = 3$			0.92 s	13.28
$h = 6$			1.64 s	23.15
$h = 9$			2.34 s	30.10
$h = 12$	23.74 s	61 K	3.78 s	35.45
$h = 15$			3.36 s	39.12
$h = 18$			3.92 s	40.32
$h = 21$			4.49 s	41.27
	$\lvert A\rvert = 2, \lvert O\rvert = 5, \lvert T\rvert = 1$			
$h = 3$			0.90 s	13.28
$h = 6$			1.49 s	22.93
$h = 9$			2.21 s	30.28
$h = 12$	20.03 s	71 K	2.81 s	35.88
$h = 15$			3.17 s	39.40
$h = 18$			3.98 s	41.85
$h = 21$			4.32 s	43.73

Table 7.4: Experimental Results for UHFP Implementation Part 1

Horizon	Solution Time	Memory Used	Simulation Time	Average Reward
	$\|A\| = 3, \|O\| = 1, \|T\| = 1$			
$h = 3$			0.95 s	13.28
$h = 6$			2.10 s	21.65
$h = 9$			2.86 s	28.12
$h = 12$	10.32 s	30 K	3.62 s	33.22
$h = 15$			4.57 s	37.65
$h = 18$			5.40 s	40.80
$h = 21$			6.12 s	42.79
	$\|A\| = 3, \|O\| = 2, \|T\| = 1$			
$h = 3$			1.11 s	13.44
$h = 6$			2.98 s	23.50
$h = 9$			2.67 s	31.34
$h = 12$	10.06 s	68 K	3.28 s	36.94
$h = 15$			4.06 s	40.83
$h = 18$			4.62 s	43.86
$h = 21$			5.26 s	45.90
	$\|A\| = 3, \|O\| = 3, \|T\| = 1$			
$h = 3$			1.08 s	13.28
$h = 6$			1.91 s	23.40
$h = 9$			2.60 s	30.94
$h = 12$	13.51 s	70 K	3.23 s	37.39
$h = 15$			4.10 s	41.63
$h = 18$			4.47 s	44.47
$h = 21$			5.16 s	46.60
	$\|A\| = 3, \|O\| = 4, \|T\| = 1$			
$h = 3$			1.01 s	13.28
$h = 6$			1.72 s	23.15
$h = 9$			2.46 s	30.10
$h = 12$	15.35 s	72 K	3.11 s	35.45
$h = 15$			3.60 s	39.12
$h = 18$			4.30 s	40.31
$h = 21$			4.83 s	41.27
	$\|A\| = 4, \|O\| = 1, \|T\| = 1$			
$h = 3$			1.09 s	13.28
$h = 6$			2.10 s	21.65
$h = 9$			2.84 s	28.12
$h = 12$	11.61 s	31 K	3.79 s	33.22
$h = 15$			4.61 s	37.65
$h = 18$			5.40 s	40.80
$h = 21$			6.20 s	42.79
	$\|A\| = 4, \|O\| = 2, \|T\| = 1$			
$h = 3$			1.32 s	13.44
$h = 6$			2.49 s	23.50
$h = 9$			3.66 s	31.34
$h = 12$	12.12 s	70 K	4.24 s	36.94
$h = 15$			4.99 s	40.83
$h = 18$			6.06 s	43.86
$h = 21$			6.90 s	45.90
	$\|A\| = 4, \|O\| = 3, \|T\| = 1$			
$h = 3$			1.12 s	13.28
$h = 6$			2.13 s	23.40
$h = 9$			2.95 s	30.94
$h = 12$	12.67 s	71 K	3.81 s	37.39
$h = 15$			4.58 s	41.64
$h = 18$			5.36 s	44.47
$h = 21$			6.23 s	46.60
	$\|A\| = 5, \|O\| = 1, \|T\| = 1$			
$h = 3$			1.10 s	13.28
$h = 6$			2.05 s	21.64
$h = 9$			2.86 s	28.12
$h = 12$	12.50 s	33 K	3.63 s	33.22
$h = 15$			4.52 s	37.65
$h = 18$			5.42 s	40.80
$h = 21$			6.25 s	42.79
	$\|A\| = 5, \|O\| = 2, \|T\| = 1$			
$h = 3$			1.23 s	13.44
$h = 6$			2.16 s	23.50
$h = 9$			2.95 s	31.34
$h = 12$	11.99 s	72 K	3.64 s	36.94
$h = 15$			4.39 s	40.83
$h = 18$			5.14 s	43.86
$h = 21$			5.90 s	45.90

Table 7.5: Experimental Results for UHFP Implementation Part 2

Simulation phase calculates the average reward gained by the policy constructed. The time for this phase is similar for all approaches. Note that for high horizon values, where UHFP can successfully generate policies, this phase takes more time. This is natural since the length of the simulation is directly proportional to the length of horizon. UHFP implementation uses the POMDP planner for the simulation phase too and this also brings a little overhead to the length of simulation.

Similarly the average reward gained at simulation is similar for all approaches. Note that the average is taken on the number of runs and thus as the horizon increases it is expected that average reward increases too.

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.

Our method, named as UHFP (Unbounded Horizon Factored POMDP), is not effected from the horizon value and when horizon is increased it clearly outperforms the alternate approaches.

However our UHFP approach does not sacrifice from the solution quality as average rewards clearly depicts. The average reward received does not decrease even for high horizon values.

CHAPTER 8

Conclusion and Future Work

In this research we developed a POMDP formulation for the GRN control problem and presented an algorithm for solving the problem efficiently by decomposing it to subproblems that exploit the problem structure. In fact, our approach will be beneficial and influential for more realistic GRN interference mechanisms. By considering partial observability, the problem is expressed in a difficult setting and advancements in the settings might also lead to more efficient and successful solutions of the problem.After achieving the successful and encouraging results reported in this paper, we are currently concentrating on advancing this work along two tracks. The setting proposed for the GRN control problem has definitely room for improvements. Different kinds of gene expression data sampling techniques have been developed recently. The gene expression data analysis part can be improved, for using the results more effectively, by employing different gene expression data sampling techniques. Also in this work we did not explore possible correlations or similarities between the GRN and the subproblems. By examining such relationship, it might be possible to guide the decomposition and have more effective decomposition mechanisms that lead to greater improvements in computational cost of the solution. The proposed POMDP formulation of the gene expression control problem assumes that genes influence each other linearly. It is possible to relax this assumption by introducing a more general transition function. Finally, it is worth mentioning that this method has been actually developed for a generic POMDP problem and then adapted to the GRN control problem in hand. Gene expression data has proved to be valuable resource for guiding the decomposition process. Similar problems, where data about the problem is available, can be tackled with our approach. However, for a more general POMDP decomposi-

tion method, we need to analyze the role of the gene expression data and a couple of other components; we should also formalize a more general guidance mechanism.

REFERENCES

[1] 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.

[2] Osman Abul, Reda Alhajj, and Faruk Polat. Markov decision processes based optimal control policies for probabilistic boolean network. In *IEEE BIBE*, pages 337–344, 2004.

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

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

[5] Craig Boutilier, Richard Dearden, and Moises Goldszmidt. Exploiting structure in policy construction. In *IJCAI*, pages 1104–1111, 1995.

[6] 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.

[7] Daniel Bryce and Seungchan Kim. Planning for gene regulatory network intervention. In *IJCAI*, pages 1834–1839, 2007.

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

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

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

[11] Aniruddha Datta, Ashish Choudhary, Michael L. Bittner, and Edward R. Dougherty. External control in markovian genetic regulatory networks. *Machine Learning*, 52(1-2):169–191, 2003.

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

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

[14] Stuart 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.

[15] Seungchan Kim, Huai Li, Edward R. Dougherty, Nanwei Cao, Yidong Chen, Michael Bittner, and Edward B. Suh. Can markov chain models mimic biological regulation?. *Journal of Biological Systems*, 10(4):337, 2002.

[16] Branislav Kveton, Milos Hauskrecht, and Carlos Guestrin. Solving factored mdps with hybrid state and action variables. *Journal of Artificial Intelligence Research*, 27:2006, 2006.

[17] Michael L. Littman, Thomas L. Dean, and Leslie 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.

[18] Ranadip Pal, Aniruddha Datta, Michael L. Bittner, and Edward R. Dougherty. Intervention in context-sensitive probabilistic Boolean networks. *Bioinformatics*, 21(7):1211–1218, 2005.

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

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

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

[22] Martijn Schut, Michael Wooldridge, and Simon 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.

[23] Ilya Shmulevich, Edward R. Dougherty, and Wei Zhang. Gene perturbation and intervention in probabilistic Boolean networks. *Bioinformatics*, 18(10):1319–1331, 2002.

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

[25] 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.

[26] Mehmet Tan, Reda Alhajj, and Faruk 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.

[27] Mehmet Tan, Reda Alhajj, and Faruk Polat. Scalable approach for effective control of gene regulatory networks. *Artif. Intell. Med.*, 48(1):51–59, 2010.

[28] Jing Yu, V. Anne Smith, Paul P. Wang, Er J. Hartemink, and Erich D. Jarvis. Using bayesian network inference algorithms to recover molecular genetic regulatory networks. In *International Conference on Systems Biology*, 2002.