Employing Batch Reinforcement Learning to Control Gene Regulation without Explicitly Constructing Gene Regulatory Networks

Utku Sirin[1], Faruk Polat[1] and Reda Alhajj[2]
Computer Eng. Dept., Middle East Technical University, Ankara, Turkey[1]
{usirin,polat}@ceng.metu.edu.tr
Computer Science Dept., University of Calgary, Calgary, Alberta, Canada[2]
alhajj@ucalgary.ca

Abstract

The goal of controlling a gene regulatory network (GRN) is to generate an intervention strategy, i.e., a control policy, such that by applying the policy the system will avoid undesirable states. In this work, we propose a method to control GRNs by using Batch Mode Reinforcement Learning (Batch RL). Our idea is based on the fact that time series gene expression data can actually be interpreted as a sequence of experience tuples collected from the environment. Existing studies on this control task try to infer a model using gene expression data and then calculate a control policy over the constructed model. However, we propose a method that can directly use the available gene expression data to obtain an approximated control policy for gene regulation that avoids the time consuming model building phase. Results show that we can obtain policies for gene regulation systems of several thousands of genes just in several seconds while existing solutions get stuck for even tens of genes. Interestingly, the reported results also show that our method produces policies that are almost as good as the ones generated by existing model dependent methods.

1 Introduction

Gene Regulatory Networks (GRNs) model gene regulation in terms of multivariate interactions among the genes. One of the major goals of building a GRN for a set of genes is to predict and control the behavior of the cellular system for several reasons such as developing therapies or drugs that would prevent the system from falling into an undesired absorbing state.

The control problem for GRNs is defined as controlling the state of the regulation system through interventions of a set of genes. That is, we apply a series of actions to some pre-selected set of genes, and expect the regulation system not to fall into undesirable states. There are several studies in the area of controlling GRNs. They all model gene regulation as a Probabilistic Boolean Network (PBN) and attempt to identify the best intervention strategy over the constructed PBN [Shmulevich *et al.*, 2002]. The study described in [Datta *et al.*, 2003] tries to find an optimal finite horizon intervention strategy for PBNs so that at the ultimate horizon, the system achieves the highest probability of being in a desirable state. The study in [Pal *et al.*, 2006] finds optimal infinite horizon intervention strategy by formulating the control problem as a Markov Decision Process (MDP) and then solving the problem with the help of the Value Iteration algorithm. They showed that the optimal policy can shift the probability mass from undesirable states to desirable ones in the constructed PBN. The study in [Faryabi *et al.*, 2007a; 2007b] applies Reinforcement Learning (RL) techniques to obtain an approximate control policy of the constructed PBN.

The basic and most important problem with the existing solutions for controlling GRNs is that none of them can solve the control problem for systems with more than several tens of genes due to their exponential time and space requirements. The finite horizon study of [Datta *et al.*, 2003] builds the Markov chain of the PBN, and the infinite horizon study of [Pal *et al.*, 2006] builds an MDP over the PBN. Both require exponential time and space in terms of the number of genes. A 30-gene system, for example, requires dealing with more than 1 billion states which is highly infeasible even to keep in the memory. Although the study in [Faryabi *et al.*, 2007a] proposes an approximated algorithm that runs in polynomial time, it again requires exponential space since they explicitly keep a state-action function Q. Besides, their approximated solution still requires to construct a PBN, which already takes $O(d^k \times n^{k+1})$ time and space, where n is the number of genes, k is the maximum number of predictor genes and d is the discretization level [Shmulevich *et al.*, 2002]. The study in [Faryabi *et al.*, 2007a] reports that the construction of the PBN for their 10-gene system takes more than 3 days for $k = 3$ and $d = 2$.

In this paper, we propose a novel method to control GRNs making use of Batch Mode Reinforcement Learning (Batch RL) approach. Batch RL provides approximated infinite horizon control policies without requiring the model of the environment. That is, it tries to obtain a generalized control policy based on limitedly available experience tuples collected from the environment. Our idea is that time series gene expression data can actually be interpreted as a sequence of experience tuples collected from the environment. Each sample represents the state of the system, and successive state transitions demonstrate system dynamics. Therefore, instead of modeling gene regulation as a PBN and obtaining a control policy

over the constructed PBN, we directly use the available gene expression samples to obtain an approximated control policy for gene regulation using Batch RL. Using this method, we are able to benefit from the great reduction on both time and space since we get rid of the most time consuming phase, inferring a PBN out of gene expression data. Figure 1 depicts the two alternative solution methods, where the flow in the box summarizes our proposal. The results show that our method can find policies for regulatory systems of several thousands of genes just in several seconds. Interestingly, the results also show that our approximate control policies have almost the same solution quality compared to that of the existing PBN-based optimal solutions. This means our method is not only much faster than the previous solution alternatives but also provides almost the same solution quality.

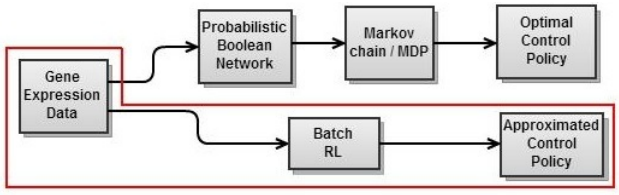

Figure 1: Flowchart of control solutions

The rest of this paper is organized as follows. Section 2 introduces Batch RL. Section 3 describes our proposed method for controlling GRNs. Section 4 shows the experimental evaluations of our method and Section 5 concludes our work.

2 Batch Mode Reinforcement Learning

A Markov Decision Process (MDP) is a framework composed of a 4-tuple (S, A, T, R), where S is the set of states; A is the set of actions; $T(s'|s,a)$ is the transition function which defines the probability of observing state s' by firing action a at state s; and $R(s,a)$ is the expected immediate reward received in state s firing action a. The transition function defines the dynamics of the environment, and the reward function specifies the rewards with respect to the state and action configurations [Bellman, 1957]. The optimal policy of an MDP framework can be defined based on optimal state-action function, Q^*, of Bellman equation:

$$Q^*(s,a) = R(s,a) + \gamma \sum_{s'} T(s'|s,a) \max_{a'} Q^*(s',a') \quad (1)$$

where γ is the discount factor [Bellman, 1957]. The optimal state-action function can be found by iterating over the Bellman equation. The optimal policy can then be found as shown in Equation 2 [Sutton and Barto, 1998].

$$\pi(s) = \arg\max_a Q^*(s,a) \quad (2)$$

Reinforcement Learning (RL) is a framework to find the optimal state-action function, Q^*, without using the model of the environment, which is $T(s'|s,a)$, the transition function, and $R(s,a)$, the reward function [Sutton and Barto, 1998]. As the learner interacts with the environment, RL incrementally updates the state-action function. One of the RL methods, Q-Learning, applies the update equation below [Watkins, 1989]:

$$Q(s,a) = (1-\alpha)Q(s,a) + \alpha(R(s,a) + \gamma \max_{a'} Q(s',a')) \quad (3)$$

where α is the learning rate, γ is the discount factor. Here, the agent applies the action a at the state s and observes the immediate reward $R(s,a)$ and the next state s'. It has been proven that for sufficiently large number of iterations, Equation 3 converges to the optimal state-action function Q^* as the learning rate, α, gradually lower downs to zero [Sutton and Barto, 1998].

Batch Mode Reinforcement Learning (Batch RL), on the other hand, is an extension of classical RL. In Batch RL, the learner directly takes several number of experience tuples that are already collected. The experience tuples can be collected arbitrarily, even randomly in the exploration stages of the learner [Lange *et al.*, 2011]. The main idea is to use these limitedly available experience tuples in batch to obtain an approximated and generalized state-action function [Lange *et al.*, 2011; Busoniu *et al.*, 2010; Ernst *et al.*, 2005].

There are several Batch RL algorithms. The Experience Replay algorithm presented in [Lin, 1992] assumes the learner experiences again and again what it had experienced before. The Kernel-Based RL in [Ormoneit and Sen, 2002] introduces kernel functions to get an approximated and generalized state-action function. The Fitted Q Iteration (FQI) algorithm presented in [Ernst *et al.*, 2005], on the other hand, converts the Batch RL problem into a supervised learning problem. The main idea is that it is actually a supervised learning problem to find an approximated and generalized state-action function mapping all possible state action pairs into their state-action values.

2.1 Least-Squares Fitted Q Iteration

In our method, we choose to use the Least-Squares Fitted Q Iteration (FQI) algorithm presented in [Busoniu *et al.*, 2010; Ernst *et al.*, 2005]. Least-Squares FQI employs parametric approximation to obtain an approximated and generalized state-action function $\hat{Q}$. $\hat{Q}$ is parameterized by an n-dimensional vector Θ, where every parameter vector Θ corresponds to a compact representation of the approximate state-action function $\hat{Q}$ as Equation 4 shows.

$$\hat{Q}(s,a) = [F(\Theta)](s,a) \quad (4)$$

The calculation of $[F(\Theta)](s,a)$, on the other hand, is done by utilization of feature values as Equation 5 shows.

$$[F(\Theta)](s,a) = [\phi_1(s,a), ..., \phi_n(s,a)]^T \cdot [\Theta_1, ..., \Theta_n] \quad (5)$$

where ϕ_i stands for a single feature specific for the state s, and the action a, Θ_i stands for the parameter corresponding to the i^{th} feature, and Θ is the parameter vector of Θ_i's. Least-Squares FQI algorithm, iteratively train the parameter vector Θ with respect to the defined feature values and calculated state-action values for each experience tuples by using least-squares linear regression. Note that the number of parameters in Least-Squares FQI is same as the number of features defined. The overall algorithm is shown in Algorithm 1.

Algorithm 1: Least-Squares Fitted Q Iteration

Input: discount factor γ,
experience tuples $\{(s_i, a_i, r_i, s'_i)|i = 1, ..., N\}$
$j \leftarrow 0$, $\Theta_j \leftarrow 0$, $\Theta_{j+1} \leftarrow \epsilon$
while $|\Theta_{j+1} - \Theta_j| \geq \epsilon$ **do**
  for $i = 1...N$ **do**
    $T_i \leftarrow r_i + \gamma \max_{a'}[F(\Theta_j)](s'_i, a')$
  end
  $\Theta_{j+1} \leftarrow \Theta^*$,
  where $\Theta^* \in \arg\min_{\Theta} \sum_{i=1}^{N} (T_i - [F(\Theta)](s_i, a_i))^2$
  $j \leftarrow j + 1$
end
Output: Θ_{j+1}

We begin with an initial parameter vector Θ_0. At each iteration, Least-Squares FQI firstly assigns a target state-action value T_i for each experience tuple. This is achieved by summing immediate reward r_i and the discounted future reward $\gamma \max_{a'}[F(\Theta_j)](s'_i, a')$ for each experience tuple. Note that this equation is same as Equation 3 with learning rate (α) as 1. Then, those target values and available feature values are used to train the parameter vector Θ by using least squares linear regression, which provides the next parameter vector Θ_{j+1}. The algorithm continues with the next iteration by using the same feature values but the refined parameter vector. Once the parameter vector is converged, the algorithm outputs the parameter vector which can be used to find the approximate state-action function based on the Equation 5. Then, it is easy to find the approximate infinite horizon control policy by taking the minimizing action for each state as Equation 6 shows.

$$\pi(s) = \arg\max_{a'}[F(\Theta_{j+1})](s, a') \quad (6)$$

3 Batch RL for Controlling GRNs

This section describes our proposed method for solving the GRN control problem. We have used the Least-Squares FQI algorithm explained in Section 2.1. Figure 2 shows the block diagram of our proposed method. First, we convert gene expression data, which is sampled from the gene regulation system that we want to control, into a series of experience tuples. Then, we calculate feature values for each experience tuple to use them in the Least-Squares FQI algorithm. Note that as Equation 5 shows, feature values depend only on the current state and action values of each experience tuples. Therefore, they do not change over the iterations of Least-Squares FQI shown in Algorithm 1. Hence, we calculate them once for each available experience tuples beforehand, and let Least-Squares FQI use them. Lastly, we invoke the Least-Squares FQI algorithm given in Algorithm 1, and obtain the approximated control policy. Thereby, we obtain a control policy for a gene regulation system directly from the gene expression data without making use of any computational model. In the following subsections, we will describe how we convert the gene expression samples into experience tuples and what features we used with the Batch RL algorithm.

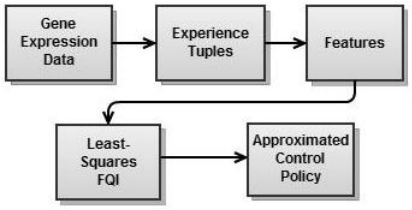

Figure 2: Batch RL for controlling GRNs

3.1 Experience Tuples

This section describes how we converted gene expression data into a series of experience tuples, which is one of the most critical steps of our solution. An experience tuple is a 4-tuple (s, a, s', c), where s is the current state, a is the current action, s' is the next state and c is the immediate cost. It represents one-step state transition in the environment. Here, we explain how each of the four elements of each experience tuple is obtained from the gene expression data. Note that instead of associating reward values for the desirable states, in our method we have associated cost values for undesirable states as presumed by previous studies [Datta *et al.*, 2003; Pal *et al.*, 2006]. Instead of maximizing reward, this time we will try to minimize the cost [Sutton and Barto, 1998].

States: As all previous studies for controlling GRNs, the state of a GRN is defined by the discretized form of the gene expression sample itself [Datta *et al.*, 2003; Pal *et al.*, 2006; Faryabi *et al.*, 2007a]. Hence, the i^{th} and $(i+1)^{th}$ gene expression samples constitute the current state s and the next state s' values for the i^{th} experience tuple. Similar to most of the previous studies, we have used binary discretization [Datta *et al.*, 2003; Pal *et al.*, 2006; Faryabi *et al.*, 2007a]. Note that there are 2^n possible states where n is the number of genes in the regulation system.

Actions: The action semantics for a gene regulation system is mostly implemented through reversing the value of a specific gene or a set of genes, i.e., changing its value from 0 to 1 or 1 to 0 [Datta *et al.*, 2003; Pal *et al.*, 2006; Faryabi *et al.*, 2007a]. Those reversed genes are named as *input genes* and should be specified in the context of the control problem. If the value of an input gene is reversed, the action is assumed as 1, if it is left as it is the action is assumed to be 0. Hence, there are 2^k distinct actions given k input genes. In order to obtain the action values from the gene expression samples, we have checked the absolute value of the difference between the values of the input genes in the successive gene expression samples. For a regulation system of six genes, for example, let the gene expression sample at time t be 101001 and at time $t+1$ be 011000. If the input genes are the 2^{nd} and 5^{th} genes, the action value, i.e., a in the experience tuple, at time t is 10 in binary representation since the 2^{nd} gene has changed its value while the 5^{th} gene has not.

Costs: The only remaining values to be extracted from the gene expression samples is the cost values, i.e., c in the experience tuples. Costs are associated with the goal of the con-

trol problem. The goal can be defined as having the value of a specific gene as 0 as in [Pal *et al.*, 2006], or as reaching to a specific basin of attractors in the state space as in [Bryce *et al.*, 2010]. If the state of the regulation system does not satisfy the goal, it is penalized by a constant value. Moreover, applying an action for each input gene also has a relatively small cost to realize it. So, the cost function can be defined as follows:

$$cost(s,a) = \begin{cases} 0 + n \times c & \text{if goal(s)} \\ \alpha + n \times c & \text{if } \neg \text{ goal(s)} \end{cases} \quad (7)$$

where α is the penalty of being in an undesirable state, n is the number of input genes whose action value is 1, and c is the cost of action to apply for each input gene.

3.2 Features

This section describes the features that are built from the experience tuples obtained from the gene expression samples. Although there may be different types of features, in our study we used straightforward but strong features. In the GRN domain, state values are composed of the discretized forms of gene expression samples, hence they provide a deep insight and rich information about the characteristics of the GRN. Based on this fact, we decided to use the current state values of the experience tuples, i.e., the discretized gene expression samples itself, directly as features. That is, for each experience tuple, there are exactly as many features as the number of genes and feature values are equal to the binary discretized gene expression values, which can be formulated as below.

$$\phi_i(s) = \begin{cases} 0 & \text{if } s(i) == 0 \\ 1 & \text{if } s(i) == 1 \end{cases} \quad (8)$$

where $0 \leq i \leq n$, n is the number of genes and ϕ_i is the i^{th} feature in the feature vector and it is equal to the expression value of the i^{th} gene in the discretized gene expression sample. So, for a 6-gene regulation system, a state having binary value as 101011 has its feature vector same as 101011. Note that as suggested in [Busoniu *et al.*, 2010], we have used different parameter vectors for each possible action, therefore the action does not affect the feature values in Equation 8.

4 Experimental Evaluation

4.1 Melanoma Application

This section describes the experimental evaluation of our proposed method for controlling gene regulation systems in terms of its solution quality. We have applied our algorithm to the melanoma dataset presented in [Bittner *et al.*, 2000]; it is composed of 8067 genes and 31 samples. It is reported that the WNT5A gene is highly discriminating factor for metastasizing of melanoma, and deactivating the WNT5A significantly reduces the metastatic effect of WNT5A [Bittner *et al.*, 2000; Datta *et al.*, 2003; Pal *et al.*, 2006; Faryabi *et al.*, 2007a]. Hence, a control strategy for keeping the WNT5A deactivated may mitigate the metastasis of melanoma [Datta *et al.*, 2003].

We have compared our method with the previous infinite horizon solution for the GRN control problem presented in [Pal *et al.*, 2006] in terms of their solution qualities. Hence, our experimental settings are the same as that of [Pal *et al.*, 2006]. We considered a seven-gene subset of the complete melanoma dataset, which are WNT5A, pirin, S100P, RET1, MART1, HADHB, and STC2, in order, i.e., WNT5A is the most significant bit and STC2 is the least significant bit in the state values. We have selected the 2^{nd} gene, pirin, as the input gene. Therefore, there are two possible actions defined in the control problem, reversing the value of pirin, $a = 1$, or not reversing it, $a = 0$. We also set the cost of applying an action as 1. The goal objective is to have WNT5A deactivated, its expression value as 0, and set penalty of not satisfying the goal as 5. Hence, the cost formulation mentioned in Section 3.1 can be realized as follows:

$$cost(s,a) = \begin{cases} 0 & \text{if goal(s) and a = 0} \\ 1 & \text{if goal(s) and a = 1} \\ 5 & \text{if } \neg \text{ goal(s) and a = 0} \\ 6 & \text{if } \neg \text{ goal(s) and a = 1} \end{cases} \quad (9)$$

Note that for states $[0-63]$ WNT5A has the value of 0, and for the remaining states $[64-127]$ WNT5A is 1 since WNT5A is the most significant bit in the state values. Hence, the states $[0-63]$ are desirable while states $[64-127]$ are undesirable. We also set our discount factor in the Algorithm 1 as 0.9.

Based on these experimental settings, we obtained an approximate control policy from our proposed method and compared it with the optimal policy obtained by the method proposed in [Pal *et al.*, 2006]. Remember that [Pal *et al.*, 2006] models gene regulation as a PBN and then tries to obtain the infinite horizon optimal control policy of the constructed PBN. They formulate the control problem as an MDP and apply the Value Iteration algorithm. In our comparative experiment, we do the following. First, we have constructed a PBN from the seven-gene melanoma dataset. It is done based on the PBN construction algorithm presented in [Shmulevich *et al.*, 2002]. Then, we obtained a control policy from the method presented in [Pal *et al.*, 2006] and from our proposed method separately. Lastly, we applied both of the policies to the same constructed PBN separately and checked the steady-state probability distributions of the controlled PBNs. Note that the method proposed by [Pal *et al.*, 2006] used the constructed PBN to obtain a control policy. Whereas, our method did not use the constructed PBN, but obtained a control policy directly from the gene expression data as explained in Section 3. In fact, our aim is not to control the constructed PBN, but to control directly the gene regulation system that produced the real gene expression data. However, to be able to compare the quality of the two produced policies, we applied the two policies separately to the same constructed PBN and checked the steady-state probability distributions of the controlled PBNs, which is also the way the other studies, e.g., [Pal *et al.*, 2006; Faryabi *et al.*, 2007a; Datta *et al.*, 2004] follow. Figure 3 shows the results.

We see that the steady-state probability distributions are shifted from undesirable states to the desirable states in the controlled PBNs with respect to the uncontrolled PBN. Note that uncontrolled PBN means to run the PBN without any intervention, i.e., the action value is always 0. If we compare the two probability shifts provided by the policy of our Batch

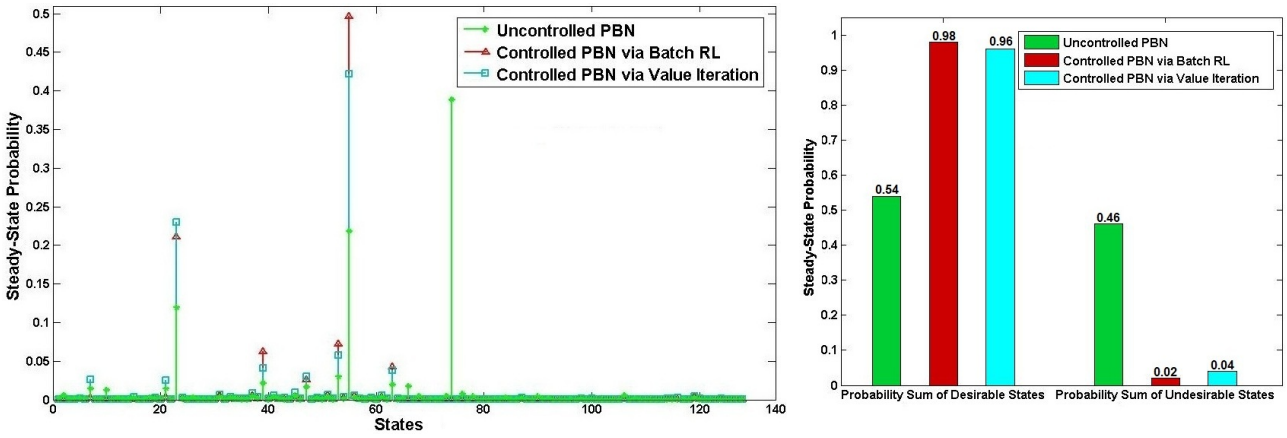

Figure 3: Steady-state probability distribution

RL based method and provided by the policy of the existing Value Iteration based method, we see that our method is able to shift the probability mass better than the existing method. The probability values of being in the states 39, 53 and 55, which are among the desirable states, are significantly larger for the policy of our method while the other probability values are almost the same. When we sum the probability values of the desirable states for both policies, we get 0.96 for the probability distribution of the PBN controlled by the existing method and 0.98 for the probability distribution of the PBN controlled by our method. The probability sum of the desirable states for the uncontrolled PBN, on the other hand, is 0.54. Right-hand side of the Figure 3 compares the probability sums.

If we compare the expected costs for the two control solutions, we see that our method is able to reduce the expected cost almost optimally. The expected cost for a control solution is calculated as the dot product of the steady-state probability distribution of the controlled PBN and the cost function, which is given below (Equation 10).

$$E[C(\pi)] = \sum_{s=1}^{N} p(s) \cdot cost(s, \pi(s)) \tag{10}$$

where N is the number of states, $p(s)$ is the steady-state probability value of the state s, π is the produced control policy, and $cost$ is the cost function defined in Equation 9. Remember that uncontrolled PBN refers to the PBN without any intervention, i.e., $\pi(s) = 0$ for all the states of the uncontrolled PBN. The expected cost for the uncontrolled PBN is 2.29, for the controlled PBN via Value Iteration is 0.21 and for the controlled PBN via Batch RL is 0.40. Note that Value Iteration provably produces the optimal solution and the expected cost of the control policy obtained from the Value Iteration is 0.21, the minimum possible expected cost [Bellman, 1957]. Our method is able to produce a near-optimal solution requiring much less time.

It is indeed a very interesting result. Because our method has nothing to do with PBN, but its produced policy works almost as successful as the optimal policy obtained over the constructed PBN itself with the Value Iteration algorithm. Hence, it can be said that instead of building a gene regulation model such as PBN and dealing with complex details of the constructed computational model, a control problem can be solved by using directly the state transitions available in the gene expression data with almost the same solution quality. Moreover, since PBN is the most time consuming part of the available control solutions, it reduces the time requirements of the problem greatly as presented in detail in Section 4.3.

4.2 Large Scale Melanoma Application

This section describes the results of our method on a large scale gene regulation system. We have again used a subset of the melanoma dataset presented in [Bittner *et al.*, 2000]. We combined the 10 interacting genes presented in Table 3 of [Kim *et al.*, 2002] and the 22 highly weighted genes that form the major melanoma cluster presented in Figure 2b of [Bittner *et al.*, 2000]. Since the four genes WNT5A, pirin, MART1 and HADHB, are available in both sets, we have 28 genes in total and 31 samples that the melanoma dataset already provides. Our objective is again to have WNT5A deactivated, its expression value as 0, and use the same cost formulation shown in Equation 9. We again set the WNT5A as the most significant bit and STC2 as the least significant bit. Hence, the desirable states are $[0 - 2^{27})$ since WNT5A has its expression value as 0, and undesirable states are $[2^{27} - 2^{28})$ since WNT5A has its expression value as 1. The input gene is also the 2^{nd} gene, pirin. The order of the genes in the state value representation is WNT5A, pirin, MART1, HADHB, CD63, EDNRB, PGAM1, HXB, RXRA, ESTs, integrin b1, ESTs, syndecan4, tropomyosin1, AXL, EphA2, GAP43, PFKL, synuclein a, annexin A2, CD20, RAB2, S100P, RET1, MMP3, PHOC, synuclein and STC2.

We have applied our proposed method to the extended 28-gene subset of the melanoma dataset. As in Section 4.1, we have again measured the quality of the policy produced by our method with respect to the shift of the probability mass from undesirable states to the desirable states in a controlled PBN. Hence, firstly we applied our proposed method to the 28-gene melanoma dataset and obtained an approximated control pol-

icy for the 28-gene regulation system. Then, we constructed the PBN of the 28-gene regulation system with the algorithm presented in [Shmulevich *et al.*, 2002]. We applied the policy produced by our method to the constructed PBN and checked the steady-state probability distribution of the controlled PBN with respect to the steady-state probability distribution of the uncontrolled PBN. Since it is almost impossible to plot the probability value of each possible 2^{28} states, here, we sum the probability values in the desirable states and in the undesirable states. Then, we compared the probability sums of the desirable states and the undesirable states in the controlled and uncontrolled PBNs. Figure 4 shows the results.

For the uncontrolled PBN, the probability sum of desirable states is 0.01 and the undesirable states is 0.99. For the controlled PBN, on the other hand, the probability sum of desirable states is 0.8 and the undesirable states is 0.2. This means, by applying the policy obtained by our method to the constructed PBN, we can significantly shift the probability mass from undesirable states to desirable states. It was 0.99 probability to be in one of the undesirable states in the uncontrolled PBN. However, this value reduces to 0.2 if we control the PBN with respect to the control policy produced by our method. Therefore, we can say that our method not only works for small regulatory systems, but also solves large scale control problems; this is a good justification for verifying its robustness and effectiveness.

Note that, 28-gene regulatory system may not seem as large enough since our method can easily produce control policies for regulation systems composed of several thousands of genes. However, here, we verify our method with respect to a constructed PBN as existing methods have done, and PBN construction algorithm limits our experiments due to its $O(d^k \times n^{k+1})$ time and space complexities, where n is the number of genes, k is the maximum number of predictor genes and d is the discretization level [Shmulevich *et al.*, 2002]. Actually, PBN construction algorithm does not work for regulation systems larger than 50 genes for $k = 3$ and $d = 2$ with our current hardware configuration, Intel i7 processor and 8-GB memory, especially due to its space requirement. Still, to the best of our knowledge, it is the first study successfully producing a control solution for a gene regulation system with more than 15 genes.

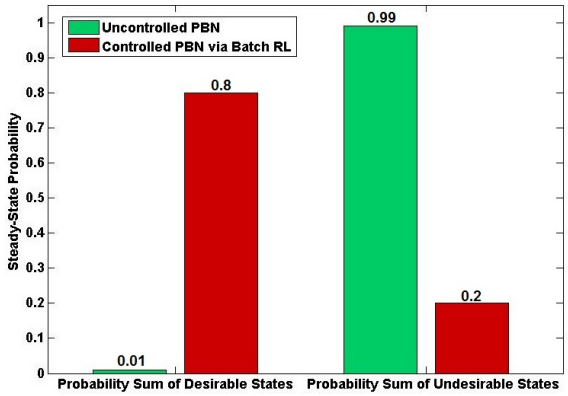

Figure 4: Large scale melanoma steady-state probability shift

4.3 Time Requirements

This section describes the time requirement of our proposed solution for controlling GRNs. We again used the gene expression data presented in [Bittner *et al.*, 2000]. This time, we gradually increased the number of genes in the dataset from 10 genes to 8067 genes, applied our method and checked the elapsed time to obtain a controlling policy. Figure 5 shows the results. As it is shown, time increases linearly with the number of genes in the dataset and the maximum required time to obtain a control policy for the complete gene regulation system of 8067 genes is just about 6 seconds [1]. It is a great improvement since existing PBN-based studies cannot solve control problems even for several tens of genes. Moreover, to our best knowledge, it is the first solution that can produce policies for regulation systems with several thousands of genes.

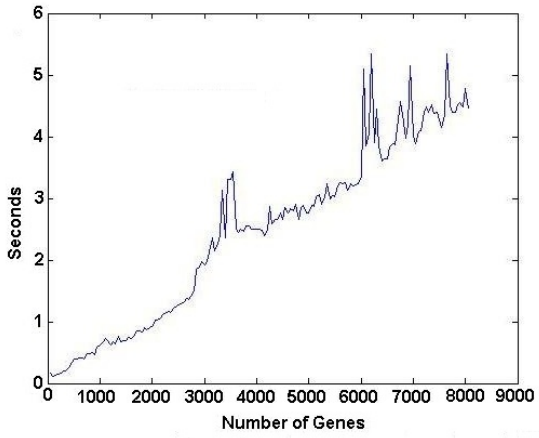

Figure 5: Execution time for our method

5 Conclusion

In this study, we have proposed a novel method for controlling GRNs. Our algorithm makes use of Batch Mode Reinforcement Learning to produce a control policy directly from the gene expression data. The idea is to treat the time series gene expression samples as a sequence of experience tuples and calculate an approximated policy over those experience tuples without explicitly generating any computational model for gene regulation such as Probabilistic Boolean Network. The reported results show that our proposed method is successful in producing control policies with almost the same solution qualities compared to the previous control solutions. Moreover, it can solve control problems with several thousands of genes just in seconds, whereas existing methods cannot solve the control problem even for several tens of genes. To the best of our knowledge, it is the first study that can generate solutions for gene regulation systems with several thousands of genes.

[1] We have used MATLAB's pinv (Moore-Penrose pseudoinverse) function for solving the least-squares regression problem in the Least-Squares FQI algorithm, which provides a linear-time solution for regression problems.

References

[Bellman, 1957] Richard Bellman. *Dynamic Programming*. Dover Publications, 1957.

[Bittner *et al.*, 2000] M. Bittner, P. Meltzer, Y. Chen, Y. Jiang, E. Seftor, M. Hendrix, M. Radmacher, R. Simon, Z. Yakhini, A. Ben-Dor, N. Sampas, E. Dougherty, E. Wang, F. Marincola, C. Gooden, J. Lueders, A. Glatfelter, P. Pollock, J. Carpten, E. Gillanders, D. Leja, K. Dietrich, C. Beaudry, M. Berens, D. Alberts, and V. Sondak. Molecular Classification of Cutaneous Malignant Melanoma by Gene Expression Profiling. *Nature*, 406(6795):536–540, August 2000.

[Bryce *et al.*, 2010] Daniel Bryce, Michael Verdicchio, and Seungchan Kim. Planning interventions in biological networks. *ACM Transactions Intelligent Systems Technology*, 1(2):11:1–11:26, 2010.

[Busoniu *et al.*, 2010] Lucian Busoniu, Robert Babuska, Bart De Schutter, and Damien Ernstr. *Reinforcement Learning and Dynamic Programming Using Function Approximators*. CRC Press, 2010.

[Datta *et al.*, 2003] Aniruddha Datta, Ashish Choudhary, Michael L. Bittner, and Edward R. Dougherty. External Control in Markovian Genetic Regulatory Networks. *Machine Learning*, 52:169–191, 2003.

[Datta *et al.*, 2004] Aniruddha Datta, Ashish Choudhary, Michael L. Bittner, and Edward R. Dougherty. External control in Markovian Genetic Regulatory Networks: The Imperfect Information Case. *Bioinformatics*, 20(6):924–930, April 2004.

[Ernst *et al.*, 2005] Damien Ernst, Pierre Geurts, and Louis Wehenkel. Tree-Based Batch Mode Reinforcement Learning. *Journal of Machine Learning Research*, 6:503–556, 2005.

[Faryabi *et al.*, 2007a] B. Faryabi, A. Datta, and E.R. Dougherty. On Approximate Stochastic Control in Genetic Regulatory Networks. *Systems Biology, IET*, 1(6):361 – 368, 2007.

[Faryabi *et al.*, 2007b] Babak Faryabi, Aniruddha Datta, and Edward R. Dougherty. On reinforcement learning in genetic regulatory networks. In *Proceedings of the 2007 IEEE/SP 14th Workshop on Statistical Signal Processing*, SSP '07, pages 11–15, Washington, DC, USA, 2007. IEEE Computer Society.

[Kim *et al.*, 2002] 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*, 10:337–357, 2002.

[Lange *et al.*, 2011] Sacha Lange, Thomas Gabel, and Martin Riedmiller. *Reinforcement Learning: State of the Art*. Springer, 2011.

[Lin, 1992] Long-Ji Lin. Self-Improving Reactive Agents Based on Reinforcement Learning, Planning and Teaching. *Machine Learning*, 8(3-4):293–321, May 1992.

[Ormoneit and Sen, 2002] Dirk Ormoneit and aunak Sen. Kernel-based reinforcement learning. *Machine Learning*, 49:161–178, 2002.

[Pal *et al.*, 2006] Ranadip Pal, Aniruddha Datta, and Edward R. Dougherty. Optimal Infinite Horizon Control for Probabilistic Boolean Networks. *IEEE Transactions on Signal Processing*, 54:2375–2387, 2006.

[Shmulevich *et al.*, 2002] Ilya Shmulevich, Edward R. Dougherty, Seungchan Kim, and Wei Zhang. Probabilistic Boolean Networks: A Rule-Based Uncertainty Model for Gene Regulatory Networks. *Bioinformatics*, 18(2):261–274, 2002.

[Sutton and Barto, 1998] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction (Adaptive Computation and Machine Learning)*. The MIT Press, 1998.

[Watkins, 1989] C. Watkins. *Learning from Delayed Rewards*. PhD thesis, University of Cambridge,England, 1989.