2021 IEEE International Conference on Systems, Man, and Cybernetics (SMC)
17-20 October, 2021. Melbourne, Australia

2021 IEEE International Conference on Systems, Man, and Cybernetics (SMC) | 978-1-6654-4207-7/21/$31.00 ©2021 IEEE | DOI: 10.1109/SMC52423.2021.9658853

Random Forest Q-Learning for Feedback Stabilization of Probabilistic Boolean Control Networks

Pratik Bajaria, Amol Yerudkar and Carmen Del Vecchio

***Abstract*— In this paper, we propose a novel random forest (RF) Q-learning hybrid with experience replay for feedback stabilization of probabilistic Boolean control networks (PBCNs). In particular, by resorting to a model-free reinforcement learning (RL) framework, we present a random forest Q-learning (QLRF) algorithm to design optimal state feedback controllers, thereby stabilizing PBCNs to a given equilibrium point. In reference to better the process of learning the Q-table by replacing it with a function approximator, we substitute the existent neural network (NN) architecture by a RF. We provide insights on the overall computational complexity between the two ways of solving the same problem, proving RF better than its NN counterparts for such applications. The simulations performed on some of the standard examples in the literature demonstrates the effectiveness of the proposed idea.**

I. Introduction

Markov decision processes (MDPs) model sequential decision-making problems arising in manufacturing, robotics, automatic control, sensor networks, and gene regulatory networks (GRNs). These models are characterized by the state-evolution dynamics, a control policy that allocates an action to each state, and an associated transition cost from the current state to the next state. A control problem of an MDP is to devise the optimal control policy which leads to the desired path, a sequence of actions and states at minimum cumulative cost. A reinforcement learning (RL) problem [1] is a well-known problem of this kind wherein the cost at each step is revealed at the end of each transition. Most common methodologies that address RL problem with MDP as underlying structure include dynamic programming (DP), value iteration, policy iteration [2], and linear programming [3]. Furthermore, RL provides a model-free framework such as Q-learning (QL) [4] and solves a discrete-time optimal control problem cast as an MDP.

In this paper, we utilize model-free RL framework and aim to design the optimal control policy to feedback stabilize probabilistic Boolean control networks (PBCNs) [5] modeled as an MDP. PBCNs are a collection of Boolean control networks (BCNs) switching randomly between constituent BCNs with certain probability distribution. With the probabilistic switching law, PBCNs provide a natural framework to study the dynamics of GRNs. Many control problems of PBCNs (and BCNs) have been studied in the literature, including but not limited to controllability and observability [6], stabilization [7], [8], output regulation [9]–[11] and disturbance decoupling [12]. Stabilization is one of the crucial issues of PBCNs. Indeed, the use of control inputs to manipulate system behavior in order to make it desirable or steady-state is of utmost importance in medical practices. This can help deciding therapeutic intervention strategies to achieve a healthy and robust state in specific GRNs. For example, modeling the development of a disease in terms of tumor (diseased cell) growth and developing therapeutic strategies to eradicate disease cells [13].

The feedback stabilization problem of GRNs modeled as PBCNs has been extensively investigated by resorting to model-based framework developed using the semi-tensor product (STP) [14] of matrices. For example, see [8], [9], [15]–[20], and the references therein. Despite the fact that model-based approaches are very efficient, exponential computational complexity is a major obstacle to their application to large-scale GRNs. Moreover, due to the huge structure of GRNs, oftentimes, it is not possible to build the appropriate PBCN model. To mitigate these issues, some model-free control techniques have been recently presented wherein PBCNs are presented in the MDP framework. For example, Acernese *et al.* [21], [22] and Bajaria *et al.* [23] presented some algorithms based on Q-learning and deep Q-learning, respectively, to solve the stabilization problem of PBCNs. Further, authors in [24] and [25] explored double deep Q-learning approach to solve controllability and output tracking problem of PBCNs, respectively.

In this paper, we further explore Q-learning algorithm to feedback stabilize GRNs (modeled as PBCNs) at a given equilibrium point and design an optimal state feedback controller. With the direction of research set to understand ways to find a computationally better approach, it was pointed out in [26], [27] that, the values in a Q-table can be replaced using a deep neural network to work as a function approximator. It must be noted that, this observation is quite interesting with modern neural network (NN) architecture and implies that the Q-values are just relative indicators and a rough approximation can prove sufficient. Inspired by this idea, we replace neural network architecture with ensemble learning methods or decision trees. The notion behind choosing a decision tree to deep neural network is the fact that decision trees like random forests (RFs) make use of lesser levels of tree structures to predict the outcomes and have limited feature learning characteristics which proves sufficient as compared to end-to-end training in deep Q-learning networks. Thus, we propose a new algorithm, namely Q-learning random forest (QLRF), to

P. Bajaria is with the Department of Electrical Engineering, Veermata Jijabai Technological Institute, Mumbai 400019, India. pkbajaria_p15@ee.vjti.ac.in.

A. Yerudkar and C. Del Vecchio are with the Department of Engineering, University of Sannio, Benevento 82100, Italy. {ayerudkar, c.delvecchio} @unisannio.it.

control large-scale MDPs. To the best of our knowledge, a QLRF hybrid is one of its kind and proves to be effective compared to its deep neural network counterparts.

To summarize, we set our objectives as follows. We utilize the MDP framework to represent GRNs modeled as PBCNs. We present QLRF algorithm to: i) make sure that the overall learning is computationally cost-effective; ii) design an optimal state feedback controller to stabilize the PBCNs at a given equilibrium point. Further, we provide a discussion on computational complexity involved, and make use of standard PBCN examples available in the literature to demonstrate the capability of the QLRF algorithm to serve the defined objectives.

II. Preliminaries and Problem Formulation

Notation. $\mathbb{R}$ and $\mathbb{Z}_+$ denote the sets of real numbers and nonnegative integers, respectively. We denote by $[k,\, n]$, the integer set $\{k, k+1, \ldots, n\}$, $k,\, n \in \mathbb{Z}_+$, $k \leq n$. $\mathcal{B} := \{0, 1\}$, and $\mathcal{B}^n := \underbrace{\mathcal{B} \times \ldots \times \mathcal{B}}_{n}$. The basic logical operators Negation, And, Or are denoted by $\neg$, $\wedge$, $\vee$, respectively. δ_n^i denotes the ith canonical vector of size n, and $\mathcal{L}_n$ the set of all n-dimensional canonical vectors. A matrix $L = [\delta_m^{i_1} \;\ldots\; \delta_m^{i_n}]$ is called as logical matrix, for suitable indices $i_1, i_2, \ldots, i_n \in [1, m]$, and $\mathcal{L}_{m\times n}$ is the set of all $m \times n$ logical matrices. A bijective correspondence between a Boolean variable $x \in \mathcal{B}$ and a vector $\mathbf{x} \in \mathcal{L}_2$ is defined by the relationship $\mathbf{x} = \begin{bmatrix} x \\ \neg x \end{bmatrix}$. The STP between $A \in \mathbb{R}_{m\times n}$ and $B \in \mathbb{R}_{p\times q}$ is defined as $A \ltimes B := (A \otimes I_{\frac{g}{n}})(B \otimes I_{\frac{g}{p}})$, where g is the least common multiple of n, p, and $\otimes$ is the Kronecker product of matrices. Then, we can get a bijective correspondence between $\mathcal{B}^n$ and $\mathcal{L}_{2^n}$ as: given $x = [x^1\; x^2\; \cdots\; x^n]^\top \in \mathcal{B}^n$, we have

$$\mathbf{x} := \begin{bmatrix} x^1 \\ \neg x^1 \end{bmatrix} \ltimes \begin{bmatrix} x^2 \\ \neg x^2 \end{bmatrix} \ltimes \cdots \ltimes \begin{bmatrix} x^n \\ \neg x^n \end{bmatrix}.$$

A. Probabilistic Boolean Control Networks

A PBCN with n nodes and m control inputs is defined as

$$s^i(t+1) = f_i^{\sigma(t)}\left(a(t),\; s(t)\right),\; i = 1, \ldots, n, \qquad (1)$$

where $s_i(t) = (s^1(t),\; s^2(t),\; \ldots,\; s^n(t))$ and $a_j(t) = (a^1(t),\; a^2(t),\; \ldots,\; a^m(t))$ denote the n-dimensional state variable and m-dimensional control input at time $t \in \mathbb{Z}_+$, taking values in $\mathcal{B}^n$ and $\mathcal{B}^m$, respectively. $f_i \in \mathcal{F}_i = \{f_i^1,\; f_i^2,\; \ldots,\; f_i^{l_i}\} : \mathcal{B}^{n+m} \to \mathcal{B}$, $i = 1, \ldots, n$, are logical functions chosen randomly with probability $\{\mathsf{P}_i^1,\; \mathsf{P}_i^2,\; \ldots,\; \mathsf{P}_i^{l_i}\}$, where $\sum_{j=1}^{l_i} \mathsf{P}_i^j = 1$ and $\mathsf{P}_i^j \geq 0$. We assume that the selection of a sub-network for each logical function f_i is independent of time and current state. The switching signal $\sigma(t) \in \Phi = [1,\, \phi]$ is an independently and identically distributed (i.i.d.) process, where $\phi = \prod_{i=1}^n l_i$ is the total number of sub-networks. Given an initial state $s(0) \in \mathcal{B}^n$ and a control sequence $\mathbf{A}_t := \{a(\cdot)_{[0,\, t-1]}\}$ in the discrete time interval $[0,\, t-1]$, denote the solution to PBCN (1) by $s\,(t;\; s(0),\; \mathbf{A}_t)$. A state $s(0) = s_0 \in \mathcal{B}^n$ is called an equilibrium point if there exists a control $a(0) \in \mathcal{B}^m$ such that $\mathsf{P}\{s\,(1;\; s(0),\; a(0)) = s(0)\} = 1$. We denote by $\bar{s}_j,\; j \in [1,\, 2^n]$, the j-th state in $\mathcal{B}^n$.

Definition 1: A PBCN (1) is said to be asymptotically stabilizable in probability at a given equilibrium point $\bar{s}_e \in \mathcal{B}^n$ in distribution, if for every $s_0 \in \mathcal{B}^n$ there exists $\mathbf{A}_t$ such that $\lim_{t\to\infty} \mathsf{P}\{s(t;\, s_0,\, \mathbf{A}_t) = \bar{s}_e\} = 1$.

B. Reinforcement Learning (RL)

RL is a smart extension to dynamic programming, whereby optimal substructures are solved by mere experiences gained from interactions. Briefly, the RL architecture comprises two fundamental units, 'the agent' and 'the environment'. The learner or the decision-maker is called as the agent. The system that the agent wants to explore or control is called the environment. In the course of interacting with the environment, new set of actions are made available to the agent while providing rewards, which stands as the quanta of approaching towards the desired objective. The agent's only goal is to maximize these rewards at the end of successive transitions (termed as an episode)[1] before reaching the desired state.

In the RL jargon, state and control actions are denoted as follows. States $s \in S$ are representation of environment variables with S being a set of all possible states accessible by the agent. Agent interacts with the environment by taking action $a \in A(s)$, the set of all possible actions that are available while being in state s. In reference to state-action pairs (s, a), the agent transits to a newer state s' recording reward $r \subseteq \mathcal{R}$ offered by the environment, where $\mathcal{R} \in \mathbb{R}$ is the set of all the possible rewards. At each time step, the agent devises a policy $\pi(a|s)$ assigning action probabilities to all allowable actions. A common rule to define these policies depends on agent's ability to explore the environment (in case the agent is starting afresh and knows nothing) or exploit (when the agent has gathered enough experiences through interactions) with a certain probability. The agent tries to operate optimally between exploration-exploitation phases so as to finally reach the desired state.

C. Q-Learning

An RL formulation resorts to the MP, whereby a state signal summarizes past experiences compactly, yet retaining all the information. This can be formally written as

$$\mathsf{P}(s',\, r|s, a) = \mathsf{P}\{r_{t+1} = r,\; s_{t+1} = s'|s_t = s, a_t = a\}, \qquad (2)$$

for all r, s', s_t, a_t. It must be noted that, using (2) one can compute underlying expected rewards of the environment as follows

$$r(s,\, a) = \mathbb{E}[r|s,\, a] = \sum_{r\in\mathcal{R}} r \sum_{s'\in S} \mathsf{P}(s',\, r|s,\, a), \qquad (3)$$

where $\mathbb{E}[\,\cdot\,]$ is the expected value operator.

[1]An episodic framework is an organized agent-environment interaction process of imparting information (in the form of reward) to the agent with the aim of improving the agent's knowledge about the environment. This is also called an online training/learning session in the RL jargon.

Also, the state transition probabilities and expected reward can be computed using state-action-next-state triples as,

$$\begin{aligned} \mathsf{P}(s'|s,\, a) &= \sum_{r\in\mathcal{R}} \mathsf{P}(s',\, r|s,\, a), \\ r(s,\, a,\, s') &= \mathbb{E}[r|s,\, a,\, s'] = \frac{\sum_{r\in\mathcal{R}} r\mathsf{P}(s',\, r|s,\, a)}{\mathsf{P}(s'|s,\, a)}. \end{aligned} \tag{4}$$

We make extensive use of Markovian property (2) to model a PBCN as an MDP thereby using RL formulation to achieve stabilization objectives. Considering a single episode comprising of successive Markovian links and (3) as mentioned earlier, the agent tries to maximize cumulative rewards at the end of each episode. In particular, it chooses an action a to maximize the expected discounted return as $G_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$, where $\gamma \in \mathbb{R}, 0 \leq \gamma \leq 1$ is the discount rate and G_t is the performance index computed as discounted sum of future rewards.

Q-learning is a technique that resorts to G_t and defines a map $Q(s,a)$ between state-action pairs and its corresponding discounted rewards/returns. The goal of an agent in Q-learning is to maximize these discounted rewards received at the end of each episode. This can be achieved by approximating an optimal $Q(s,a)$ and greedily selecting an action using a temporal difference policy as [4], [28],

$$Q(s,a) \leftarrow Q(s,a) + \alpha(r + \gamma \max_{a'\in A} Q(s',a') - Q(s,a)), \tag{5}$$

where $\alpha \in \mathbb{R}, 0 < \alpha \leq 1$ is the learning rate, a parameter that is responsible for how much the learning process is affected by newly acquired information. Q-learning generates outputs $Q_t(s,a)$ that, as $t \to \infty$ asymptotically approaches the optimal value $Q^*(s,a)$. During t iterations[2], $Q^*(s,a)$ can be approximated by selecting at each time step action a with random action probability ϵ and maximum expected reward with probability $(1-\epsilon)$. The technique is called as ϵ-greedy policy [28] given by,

$$\pi(a|s) = \begin{cases} 1 - \epsilon + \epsilon/|A(s)|, & \text{if } a = \arg\max_{a\in A(s)} Q(s,a) \\ \epsilon/|A(s)|, & \text{for all other actions,} \end{cases} \tag{6}$$

where $|A(s)|$ denotes number of available actions and $\epsilon \in \mathbb{R}, 0 < \epsilon \leq 1$ is the exploration probability.

It can be noted that structure G_t in (5) forms the basis of Q-learning and thus can be mathematically expressed as the outcome of Bellman's optimality principle as follows,

$$\Psi^*(Q(s,a)) = Q^*(s,a) = \mathbb{E}[r + \gamma \max_{a'} Q(s',a')|s,a], \tag{7}$$

where $\Psi^* : \mathbb{R}^{|S|} \to \mathbb{R}^{|S|}$ is the Bellman operator. The optimal policy can hence be chosen as $\pi^*(a|s) = \arg\max_{a\in A(s)} Q^*(s,a)$. As the given state-action pairs are visited infinitely often, the expected rewards converge to an optimal value as well as the stochastic greedy policy $\pi(a|s)$ transforms into deterministic optimal policy $\pi^*(a|s)$ [29].

[2]For notational brevity, we exclude explicit use of t with $Q(s,a)$, unless required.

It is worthwhile noticing from (5) that, Q-learning is a model-free approach, and hence can be an efficient tool in terms of minimizing the cost of computing an accurate dynamical model of large-scale PBCNs. Additionally, Q-learning method can be used to compute state feedback controllers $\big($i.e., $a = \pi^*(a|s)\big)$ [28], [30] for the stabilization objectives mentioned in Definition 1.

III. PBCN Stabilization using QLRF

In this section, we study the stabilization problem of PBCNs and present the main results in this paper. The idea behind devising a control to achieve the stabilization objectives in a model-free framework relies on Markov property (3) and the ability of discounted returns Q to converge after certain episodes due to underlying Bellman's principle. Precisely, the agent tries to explore the environment using Q-learning and generates discounted reward values corresponding to the state-action pairs visited. This tuple $(s,a,Q(s,a))$ then serves as a data-set D to train a RF function approximator. We follow a top-down approach of proposing a novel Q-learning random forest (QLRF) first, followed by mathematical proofs on its convergence properties as well as related stabilization criteria.

A. Q-Learning Random Forest (QLRF)

RF is an ensemble learning method which constructs multitude of decision trees while learning and outputs mode of classes (for classification problem) or mean of predictions of individual trees (for regression problem) [31]. We choose a sample training data-set D divided into $D_i \subset D,\ i \in \mathbb{Z}_+$, subsets finding predictors or function approximators ($\Theta_i,\ i \in [1,N]; \forall N \in \widetilde{N}$, $\widetilde{N}$ being maximum number of trees that can be added for learning) and averaging the outcomes of all the predictors to find an effective model. This is commonly referred to as bootstrapping, aggregating or bagging in decision trees [31]. RFs are a special case of decision trees, where sample spaces D_i contain random features in order to increase the accuracy. Being one of the machine learning techniques to train the data, we choose RFs for its ability to perform faster yet provide fairly accurate predictions for regression problems as compared to its neural network counterparts. The complete procedure is as shown in Algorithm 1 with additional comments at the end.

We setup the stabilization problem of PBCNs, whilst stating assumptions as follows.

Assumption 1: The PBCN (the environment unknown to the agent) is stabilizable, i.e.,

1) $\bar{s}_e$ is an equilibrium point.
2) $\bar{s}_e$ is reachable from every initial state s_0, i.e., $\exists T \in \mathbb{Z}_+$ and $\mathbf{A}_t : \mathsf{P}\{s(t;\, s_0,\, \mathbf{A}_t) = \bar{s}_e\} > 0,\ \forall s_0 \in \mathcal{B}^n$, $\forall t \geq T$.

We generate the tuple $(s,a,Q(s,a))$ that serves as a data-set D to train and aid the agent's learning process using a RF predictor $\frac{1}{N}\sum_{i=1}^{N} \Theta_i(s,a) = Q(s,a)$. Specifically, the agent interacts with the environment for a sufficiently large number of episodes in pursuit of an optimal path from each initial state $s_0 \in \mathcal{B}^n$ to $\bar{s}_e$ while maximizing the

cumulative rewards. This predictor then serves as a function approximator to predict the optimal action a given a state s in turn stabilizing the system to desired/terminal state $\bar{s}_e$.

Briefly, Algorithm 1 takes into account the underlying Bellman property (7) of Q-learning as outputs to be learnt and trains RF trees using the states as features. At the end, we store the state-action pair values in a Q-table to analyze convergence graphs of learning and to attain optimal controller at the end. Although, in a function approximator approach, the tabular part can be easily skipped and the learnt values can be extracted using the predictor function $\Theta(\cdot)$. Additionally, QLRF implements 'warm-start' (i.e., train RFs only for unique data-sets encountered) of RF trees so as to use previous trees for repeated features and add ensembles only when new features arrive, thereby making the learning more effective and faster. Hyper-parameters used include replay buffer (RB) capacity given by $RB_{cap} \in \mathbb{Z}_+$, for which the algorithm waits until it starts training RFs. Thus, the data is first stored in staging area called RB, and once the capacity reaches a threshold, random batches (Batch of size $\text{Batch}_{cap} \leq RB_{cap} \in \mathbb{Z}_+$) are drawn for training. This idea of storing experiences and reusing them when needed is termed as "experience replay" [27]. Target(s, a) is a matrix of discounted returns for a set of given state-action pairs recorded in RB (used as output feature) that QLRF replays to learn $\hat{Q}(s, a)$ minimizing a tractable loss. Eventually, these $\hat{Q}(s, a)$ values can be used by the agent so as to reach terminal state ($\bar{s}_e$) ensuring maximization of rewards. Also, along Algorithm 1 one can make use of (4) in order to compute the underlying transition probabilities. This can be achieved by taking $\mathbb{E}[r(s, a, s')]$ for all the observations made, thereby normalizing it by a factor $\max_{s' \in S} r(s, a, s')$.

Algorithm 1 QLRF algorithm for PBCN stabilization

Input: $(s, a, \text{Target}(s, a))$, RB_{cap}, $\text{Batch}_{cap} \in \mathbb{Z}_+$
Output: $\hat{Q}(s, a), \Theta_i(s, a)$

Initialize: $\hat{Q}(s, a) \leftarrow 0$, RB$\leftarrow 0$,
  $\Theta_i(s, a)$, $i \in [1, N]$, $\forall N \in \widetilde{N}$ with random coefficients
for Maximum Iterations $\mathcal{T}$ **do**
  Sample $s \in S$ and store in RB
  if Replay Buffer $== RB_{cap}$ **then**
    Batch $\leftarrow$ Sample random states from RB
    for Every state in Batch **do**
      for Every $a \in A$ **do**
        if $s' == s_e$ **then** $\text{Target}(s, a) = r(s, a)$
        else
          $\text{Target}(s, a) = r(s, a) + \gamma \max_{a' \in A} \left(\frac{1}{N} \sum_i^N \Theta_i(s', a') \right)$
        end if
        Store data-set $D = (s, A, \text{Target}(s, A))$
      end for
    end for
    Update $\Theta_i(s, a)$ using 'warm-start' RF using D
    Empty RB for newer samples
  end if
end for
Repeat until convergence
$Q^*(s, a) \leftarrow \hat{Q}(s, a) = \frac{1}{N} \sum_i^N \Theta_i(s, a)$

Remark 1: It must be noted that, Algorithm 1 makes use of deliberately choosing all the actions being in a single state to generate complete Target(s, a) feature space at once. This is made possible through multi-output regression solvers, thereby reducing number of iterations required for convergence. Alternatively, the actions can also be chosen using the ϵ-greedy policy mentioned in (6) pertaining to the basic idea of visiting a given state-action pair infinitely often. This would be effectively slower but in conjunction with conventional workflow, and still consume less memory space compared to tabular approaches.

B. Q-Learning and Contraction Mappings

Next, we look into convergence properties of the proposed QLRF algorithm. One of the important factors to be noted is the capability of $Q(s, a)$ to converge after certain episodes due to the underlying Bellman property (7). As the number of iterations increases, the temporal difference policy (5) tries to minimize the difference between optimal $Q^*(s, a)$ and the iterating value update of $Q(s, a)$ at every time step.

Thus, an obvious choice is to make RF learn the Bellman term (7) and attain an optimal $Q^*(s, a)$. Therefore, using the Markovian property it can be shown that all the points in $Q(s, a)$ converge to an optimal $Q^*(s, a)$ with probability 1 and is a contraction mapping [32]. Making use of [33, Th. 7.12], we state as under.

Theorem 1: For every state-action pairs (s, a), the discounted reward values $Q(s, a)$ in Algorithm 1 converge to the optimal value $Q^*(s, a)$ via a contraction mapping operator $\Psi^*(\cdot)$ given by (7).

Proof: For proof, reader can refer to [23]. ■

Using the outcomes of QLRF algorithm discussed above, we present the following straightforward result on the design of stabilizing state feedback control law [21].

Assume that Algorithm 1 converges to Q^* for all $\bar{s}_j$, $j \in [1, 2^n]$, then there exists an optimal policy $\pi^*(a|\bar{s}_j) = \arg\max_{a \in A(\bar{s}_j)} Q^*(\bar{s}_j, a)$, for all $\bar{s}_j$. From $\pi^*(a|\bar{s}_e)$, an optimal state feedback control can be computed as $\mathbf{a}_t = \mathcal{K}\mathbf{s}_t$, where $\mathbf{a}_t$, $\mathbf{s}_t$ are in the canonical vector form, and the static state feedback gain matrix $\mathcal{K} \in \mathcal{L}_{2^m \times 2^n}$ (not necessarily unique) is given as $\mathcal{K} = [\delta_{2^m}^{\pi^*(a|1)} \ \delta_{2^m}^{\pi^*(a|2)} \ \dots \ \delta_{2^m}^{\pi^*(a|2^n)}]$.

Remark 2: It is worth highlighting that, in our work, the system under considerations is PBCN which is stabilizable according to Assumption 1. Thus, by resorting to the Bellman equation's convergence property as discussed in Theorem 1, the proposed QLRF algorithm converges and provides a state feedback law, thereby stabilizing the PBCN to a given equilibrium point. Hence, there always exists a solution to the considered system. Further, under the current setting, multiple state feedback laws exist, not necessarily unique, rendering the system stable at a given equilibrium point.

Remark 3: Considering that the QLRF predictor tries to minimize a mean squared error (MSE) at all times adjusting

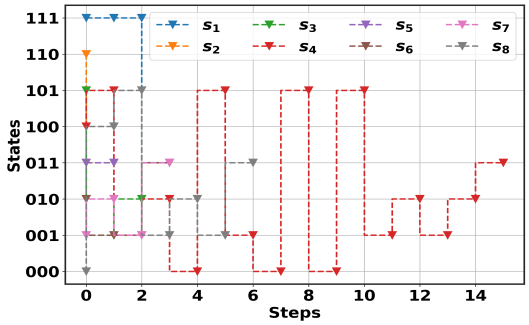

Fig. 1: Feedback stabilization using QLRF for Example 1.

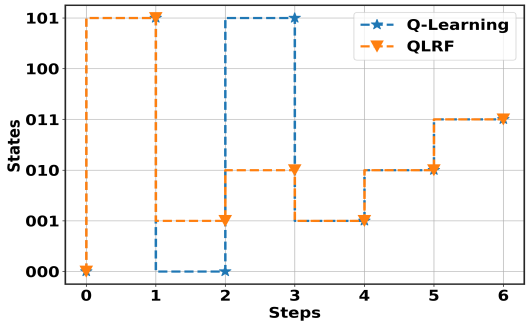

Fig. 2: State-of-the-art comparison of feedback stabilization control using different set of controls for Example 1.

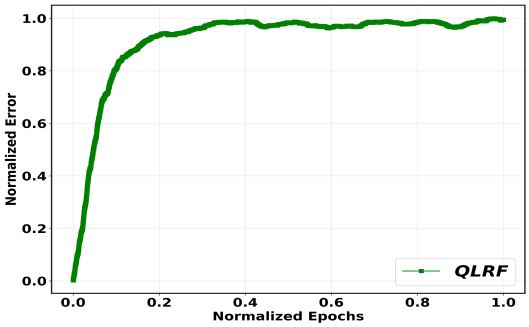

Fig. 3: Normalized MDP score using QLRF for Example 2.

coefficients of each and every predictor, thus next can be inferred. The total computational complexity of building an RF model is $O(\mathcal{T}\cdot\text{Batch}_{cap}\cdot N\cdot\mathcal{O}\cdot\log\mathcal{O})$, with N predictors, Batch_{cap} samples and $\mathcal{O}$ features [34], [35]. In terms of peer-to-peer comparison, a neural network based technique would take exponentially higher computations compared to logarithmic burden for RF. As far as space complexity is concerned, the computational burden of a QLRF can be considered to be $O(2^d)$, where $d \in \mathbb{Z}_+$ is the depth of the designed RF. Additionally, RB and Batch sizes differ as per complexity of the network. $\hat{Q}(s,a)$ can be computed on an adhoc basis and can be flushed out of memory similar to clearing the RB and Batch vectors. Again the control of PBCNs or BCNs is NP-hard [36] when solved using model-based approaches and a direct comparison to model-free techniques seems unjust. Although, the results for stabilization objectives obtained via both model-based and model-free approaches are comparable.

IV. Illustrative Examples

To illustrate effectiveness of the proposed results, we consider two PBCN examples. Without loss of generality, we construct the environment's rewarding scheme (as seen by the agent) to attain better inferential results [28]. A unique rewarding scheme is considered wherein the desired state is assigned a maximum positive reward, and a negative reward is given when the agent explores undesired attractors. It must be noted that, while the approach we use doesn't involve any prior knowledge of the PBCN's structure in order to determine a control policy, we have included the system dynamics so as to demonstrate the complexity of the networks.

All the computer simulations in this work are carried out in Python 3.0 on a desktop computer (16GB RAM, 8GB GPU) or a hosted instance ($\leq$ 16GB RAM). It must be noted though that none of the illustrations in this work made use of an explicit GPU, except for the inbuilt ones in packages.

Example 1: We consider the following PBCN model of apoptosis network with $n = 3$ and $m = 1$ [21]:

$$s_+^1 = \begin{cases} \neg s^2 \wedge a, \ \mathsf{P} = 0.6 \\ a, \ \mathsf{P} = 0.4 \end{cases}; \ s_+^2 = \begin{cases} \neg s^1 \wedge s^3, \ \mathsf{P} = 0.7 \\ s^2, \ \mathsf{P} = 0.3 \end{cases}$$

$$s_+^3 = \begin{cases} s^2 \vee a, \ \mathsf{P} = 0.8 \\ s^3, \ \mathsf{P} = 0.2 \end{cases}$$

Here, s_+^i indicates the value of the ith node at next time step. The equilibrium state $\bar{s}_e$ is given as $(0,\ 1,\ 1)$.

By following Algorithm 1, we obtain state feedback gain matrix as $\mathcal{K}_{Q\text{LRF}} = [\delta_2^1\ \delta_2^1\ \delta_2^2\ \delta_2^1\ \delta_2^2\ \delta_2^2\ \delta_2^2\ \delta_2^1]$. Fig. 1 depicts the closed-loop evolution of system states towards the equilibrium point $(0,\ 1,\ 1)$. We further compare (in Fig. 2) the state feedback controller obtained using QLRF algorithm with the one obtained using Q-learning algorithm presented in [21]. It can be clearly seen that the devised controller provides shortest path (as in [21]) to steer the state $(0,\ 0,\ 0)$ towards the equilibrium.

Example 2: Consider the following dynamics of 9-gene PBCN model ($n = 9,\ m = 2$) of the lactose operon in the *E. Coli* bacteria [37].

$s_+^1 = \neg s^7 \wedge s^3$; $s_+^2 = s^1$; $s_+^3 = \neg a^1$; $s_+^4 = s^5 \wedge s^6$; $s_+^5 = \neg a^1 \wedge s^2 \wedge a^2, \mathsf{P} = 0.7$ and $s_+^5 = s^5, \mathsf{P} = 0.3$; $s_+^6 = s^1$; $s_+^7 = \neg s^4 \wedge \neg s^8$; $s_+^8 = s_4 \vee s^5 \vee s^9$; $s_+^9 = \neg a^1 \wedge (s^5 \vee a^2), \mathsf{P} = 0.6$ and $s_+^9 = s^9, \mathsf{P} = 0.4$.

Here, $s_+^i,\ i = 1,2,3,4,6,7,8$ have deterministic update rule. The *E. Coli* network is responsible for the transport and metabolism of lactose in the absence of glucose. We aim to design an optimal state feedback controller rendering the closed-loop system to stabilize at the equilibrium $\bar{s}_e = (1,1,1,1,1,1,0,1,1)$ corresponding to ON operation of lactose operon [37].

Convergence of QLRF (Algorithm 1) is shown using normalized MDP score in Fig. 3. It can be seen that the Q-table values attain a steady-state after certain set of episodes. Further, using the attained controller after the learning phase (or training) we choose a random initial state to check the evolution of the PBCN under the obtained controller

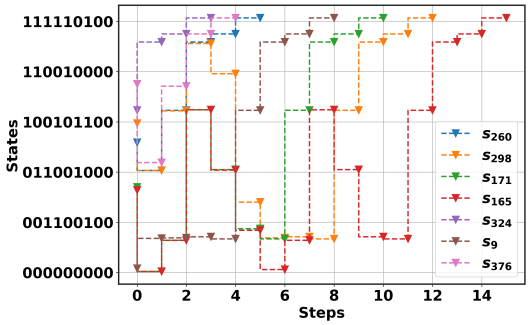

Fig. 4: Feedback stabilization using QLRF for Example 2.

using QLRF in Fig. 4. It can be observed that the closed-loop system is stabilized at the desired equilibrium $\bar{s}_e = (1,1,1,1,1,1,0,1,1)$. Due to space limit, the detailed $\mathcal{K} \in \mathcal{L}_{4\times 512}$, $\lambda \in \mathbb{R}^{512}$ is not given in the paper, which is available on the GitHub repository mentioned in the footnote[3].

REFERENCES

[1] R. S. Sutton, D. Precup, and S. Singh, "Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning," *Artificial Intelligence*, vol. 112, no. 1-2, pp. 181–211, 1999.

[2] D. P. Bertsekas and J. N. Tsitsiklis, *Neuro-dynamic programming*. Athena Scientific, 1996.

[3] D. P. De Farias and B. Van Roy, "The linear programming approach to approximate dynamic programming," *Operations Research*, vol. 51, no. 6, pp. 850–865, 2003.

[4] C. J. Watkins and P. Dayan, "Q-learning," *Machine learning*, vol. 8, no. 3-4, pp. 279–292, 1992.

[5] I. Shmulevich, E. R. Dougherty, S. Kim, and W. Zhang, "Probabilistic Boolean networks: a rule-based uncertainty model for gene regulatory networks," *Bioinformatics*, vol. 18, no. 2, pp. 261–274, 2002.

[6] Y. Liu, H. Chen, J. Lu, and B. Wu, "Controllability of probabilistic Boolean control networks based on transition probability matrices," *Automatica*, vol. 52, pp. 340–345, 2015.

[7] R. Zhou, Y. Guo, Y. Wu, and W. Gui, "Asymptotical feedback set stabilization of probabilistic Boolean control networks," *IEEE transactions on neural networks and learning systems*, vol. 31, no. 11, pp. 4524–4537, 2019.

[8] C. Huang, J. Lu, G. Zhai, J. Cao, G. Lu, and M. Perc, "Stability and stabilization in probability of probabilistic Boolean networks," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 32, no. 1, pp. 241–251, 2021.

[9] A. Yerudkar, C. Del Vecchio, and L. Glielmo, "Output tracking control of probabilistic Boolean control networks," in *2019 IEEE International Conference on Systems, Man and Cybernetics (SMC)*. IEEE, 2019, pp. 2109–2114.

[10] S. Zhu, J. Lu, Y. Liu, T. Huang, and J. Kurths, "Output tracking of probabilistic Boolean networks by output feedback control," *Information Sciences*, vol. 483, pp. 96–105, 2019.

[11] A. Yerudkar, C. Del Vecchio, and L. Glielmo, "Output tracking control design of switched Boolean control networks," *IEEE Control Systems Letters*, vol. 4, no. 2, pp. 355–360, 2019.

[12] K. Sarda, A. Yerudkar, and C. Del Vecchio, "Disturbance decoupling control design for Boolean control networks: a boolean algebra approach," *IET Control Theory & Applications*, vol. 14, no. 16, pp. 2339–2347, 2020.

[13] C. Del Vecchio, L. Glielmo, and M. Corless, "Equilibrium and stability analysis of x-chromosome linked recessive diseases model," in *2012 IEEE 51st IEEE Conference on Decision and Control (CDC)*. IEEE, 2012, pp. 4936–4941.

[14] D. Cheng, H. Qi, and Z. Li, *Analysis and control of Boolean networks: a semi-tensor product approach*. London, U.K.: Springer, 2011.

[15] Y. Wu and T. Shen, "Policy iteration algorithm for optimal control of stochastic logical dynamical systems," *IEEE transactions on neural networks and learning systems*, vol. 29, no. 5, pp. 2031–2036, 2017.

[16] M. Toyoda and Y. Wu, "On optimal time-varying feedback controllability for probabilistic Boolean control networks," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 31, no. 6, pp. 2202–2208, 2020.

[17] Y. Wu, Y. Guo, and M. Toyoda, "Policy iteration approach to the infinite horizon average optimal control of probabilistic Boolean networks," *IEEE Transactions on Neural Networks and Learning Systems*, 2020, DOI: 10.1109/TNNLS.2020.3008960.

[18] J. Liu, Y. Liu, Y. Guo, and W. Gui, "Sampled-data state-feedback stabilization of probabilistic Boolean control networks: A control lyapunov function approach," *IEEE transactions on cybernetics*, vol. 50, no. 9, pp. 3928–3937, 2019.

[19] L. Lin, J. Cao, and L. Rutkowski, "Robust event-triggered control invariance of probabilistic Boolean control networks," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 31, no. 3, pp. 1060–1065, 2019.

[20] E. Fornasini and M. E. Valcher, "Optimal control of Boolean control networks," *IEEE Transactions on Automatic Control*, vol. 59, no. 5, pp. 1258–1270, 2014.

[21] A. Acernese, A. Yerudkar, L. Glielmo, and C. Del Vecchio, "Reinforcement learning approach to feedback stabilization problem of probabilistic Boolean control networks," *IEEE Control Systems Letters*, vol. 5, no. 1, pp. 337 – 342, 2020.

[22] ——, "Model-free self-triggered control co-design for probabilistic Boolean control networks," *IEEE Control Systems Letters*, vol. 5, no. 5, pp. 1639–1644, 2020.

[23] P. Bajaria, A. Yerudkar, and C. Del Vecchio, "Aperiodic sampled-data stabilization of probabilistic Boolean control networks: Deep q-learning approach with relaxed Bellman operator," in *European Control Conference (ECC)[Accepted]*, 2021.

[24] G. Papagiannis and S. Moschoyiannis, "Deep reinforcement learning for control of probabilistic Boolean networks," in *International Conference on Complex Networks and Their Applications*. Springer, 2020, pp. 361–371.

[25] A. Acernese, A. Yerudkar, L. Glielmo, and C. Del Vecchio, "Double deep-q learning-based output tracking of probabilistic Boolean control networks," *IEEE Access*, vol. 8, pp. 199 254–199 265, 2020.

[26] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[27] M. Lapan, *Deep Reinforcement Learning Hands-On: Apply modern RL methods, with deep Q-networks, value iteration, policy gradients, TRPO, AlphaGo Zero and more*. Packt Publishing Ltd, 2018.

[28] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[29] F. L. Lewis, D. Vrabie, and K. G. Vamvoudakis, "Reinforcement learning and feedback control: Using natural decision methods to design optimal adaptive controllers," *IEEE Control Systems Magazine*, vol. 32, no. 6, pp. 76–105, Dec 2012.

[30] L. Buşoniu, T. de Bruin, D. Tolić, J. Kober, and I. Palunko, "Reinforcement learning for control: Performance, stability, and deep approximators," *Annual Reviews in Control*, vol. 46, pp. 8–28, 2018.

[31] L. Breiman, "Bagging predictors," *Machine learning*, vol. 24, no. 2, pp. 123–140, 1996.

[32] L. Busoniu, R. Babuska, B. De Schutter, and D. Ernst, *Reinforcement learning and dynamic programming using function approximators*. FL USA: CRC press, 2017.

[33] D. V. Anosov, S. K. Aranson, V. I. Arnold, I. Bronshtein, Y. S. Il'yashenko, and V. Grines, *Ordinary differential equations and smooth dynamical systems*. Springer-Verlag, 1997.

[34] I. H. Witten and E. Frank, "Data mining: practical machine learning tools and techniques with java implementations," *Acm Sigmod Record*, vol. 31, no. 1, pp. 76–77, 2002.

[35] G. Biau, "Analysis of a random forests model," *The Journal of Machine Learning Research*, vol. 13, no. 1, pp. 1063–1095, 2012.

[36] T. Akutsu, M. Hayashida, W.-K. Ching, and M. K. Ng, "Control of Boolean networks: Hardness results and algorithms for tree structured networks," *Journal of theoretical biology*, vol. 244, no. 4, pp. 670–679, 2007.

[37] R. Robeva and T. Hodge, *Mathematical concepts and methods in modern biology: using modern discrete models*. Academic Press, 2013.

[3] http://www.github.com/pratikbajaria