Simulation-Based Approximate Dynamic Programming for Near-Optimal Control of Re-entrant Line Manufacturing Models

José A. Ramírez-Hernández, and Emmanuel Fernandez

Abstract—This paper presents the comparison of four different simulation-based Approximate Dynamic Programming (ADP) approaches for the optimization of job sequencing and job releasing operations in two simple Re-entrant Line Manufacturing (RLM) models. The ADP approaches utilized included Q-Learning, Q-Learning with a state aggregation approach, SARSA(\(\lambda\)), and an Actor-Critic architecture. Results from numerical experiments demonstrated the efficacy of these ADP methods in approximating optimal control solutions. In general, the Actor-Critic approach, which utilizes one linear parametric approximation structure, a temporal difference learning algorithm, and a gradient descent method, consistently generated close approximations to the optimal control policies.

Index Terms—Re-entrant lines, approximate dynamic programming, optimal control, queueing networks.

I. INTRODUCTION

The control of queueing networks [1] with re-entrant lines [2] has been a topic of interest for the research community, e.g., see [2], [3]. These systems differ from other queueing networks because of the inclusion of feedback loops in the system; thus, jobs are allowed to return to previous processing steps before leaving the system. Research on the control problem of re-entrant lines is of relevance in various fields but especially in industrial applications such as semiconductor manufacturing [4], [5]. Hereafter we refer to these systems as re-entrant line manufacturing (RLM) models.

As a result, these systems are usually characterized by large state and action spaces that make its control difficult. In particular, semiconductor fabrication facilities can be modeled as RLM systems with hundreds of processing steps and dozens of queues and workstations.

Two of the main control operations in RLM systems are job releasing and job sequencing [5]. Job sequencing deals with selecting one out of several jobs competing for service, at a given station or machine, to be processed next. Job releasing deals with the decision of when to release a new job into the factory. In addition, because of the complexity of real RLM systems which are also commonly characterized by both large state and action spaces, analytical models and optimal control solutions for these systems are usually intractable. Thus, e.g., in semiconductor manufacturing, extensive and sophisticated simulation models [6] are utilized instead for assessing overall system performance.

An emergent approach that could be utilized to obtain near-optimal control solutions for realistic RLM models is Approximate Dynamic Programming (ADP) [7], [8], [9], also known as Reinforcement Learning [10] or Neuro-Dynamic Programming [11]. Although it is well known that Bellman’s curse of dimensionality would preclude the direct application of the Dynamic Programming (DP) algorithm [12], ADP approaches can provide simulation-based optimization algorithms conceived to deal with large state and action spaces.

As illustrated in Figure 1, a main feature of these algorithms is the utilization of simulation models of the system instead of analytical ones for obtaining near-optimal control policies. Such features can thus be advantageously exploited in, e.g., semiconductor manufacturing.

![Fig. 1. Typical simulation-based ADP architecture.](image-url)

The objective of this paper is then to present results from numerical experiments that compare the performance of four simulation-based ADP approaches in the optimization of control operations of two simple RLM models. The ADP approaches utilized in this research are part of a wide spectrum of ADP algorithms available (e.g., see [11]) and also represent a progression from those considered in the literature as basic to those more advanced. Moreover, the approaches utilized include methods with lookup table representations and parametric approximation structures. These methods are among the most commonly utilized techniques to represent approximations of optimal quantities (e.g., optimal cost functions) and are considered as core methods that are utilized in many variations of ADP algorithms [8], [9], [10], [11].

In previous work [13], [14], [15], we presented results from
the application of three simulation-based ADP approaches, namely, Q-Learning [16], SARSA(λ) [10], and an ADP Actor-Critic architecture [11] in the optimization of job sequencing and job releasing operations in two simple RLM models and under a discounted cost (DC) criterion [12]. The two RLM models utilized have been previously reported in e.g., [3], [14]. Although the models utilized in this research study are simple, these are also useful to evaluate and compare the performance of the simulation-based ADP approaches utilized. That in turn provide additional insight regarding which of the ADP approaches employed could be more suitable for other RLM models. One of the models studied represents a two-machine and three finite-capacity buffers system with job sequencing control only. The other model is an extension of the first one by including a job releasing control and one additional finite buffer at the entrance of the system. Thus, these models offer two levels of complexity with respect to both the control and state space. Results reported in [13], [14], [15] demonstrated that the ADP approaches utilized provided good approximations to the optimal policies. As such, these results also served to validate the applicability of ADP in this type of problems. However, the research conducted did not provide a comparison between the performances obtained by each ADP approach utilized. Moreover, the ADP approaches employed were applied in either one or the other RLM model but not in both.

Thus, in this paper we present new results from numerical experiments specifically designed to compare the performance of the ADP approaches reported in [13], [14], [15]. Such comparison also involved the numerical computation of both optimal control policies and performances via a modified policy iteration (MPI) algorithm [12]. In contrast to previous work, in this research study all the ADP approaches were applied for the control of the two RLM models of interest and under uniform conditions in terms of costs functions and system’s parameters. In addition, in this research study we incorporated an new ADP approach which is based on Q-Learning and a state aggregation approach [8], [10], [11]. In this paper we show that such ADP approach provided implementation advantages over the classical version of Q-Learning. Also, we provide some insight regarding the advantages and disadvantages of the ADP approaches studied to guide the selection of ADP algorithms that could be more suitable in applications with more complex RLM models. The work presented here is also a continuation of research work on the application of Markov Decision Processes [12] in semiconductor manufacturing systems as presented in [17], [18].

The remaining of this paper is organized as follows. Section II provides a brief description of the RLM models utilized. Section III presents details of the control optimization problem and as well as an overview and a qualitative comparison of the ADP approaches studied. Section IV present the results from numerical experiments, and conclusions are given in section V.

II. SIMPLE RLM MODELS WITH JOB SEQUENCING AND RELEASING CONTROL

Figure 2 depicts the system studied in this paper. As illustrated in the figure, two simple RLM models can be obtained by fixing the position of the selector $d_{IR}$. When $d_{IR} = 1$, the model includes a control for releasing new jobs into system. If $d_{IR} = 0$, then another simple RLM model that does not regulate the entrance of new jobs is obtained. Additional details of the models can be found in [5], [15], [19]. Because each model can be represented by a continuous-time controlled Markov chain, these can also be modeled as Semi-Markov Decision Processes (SMDP) [12] when an optimization control problem is considered.

Fig. 2. A simple RLM model, adapted from [2], [14].

When the model includes the job releasing control, i.e., $d_{IR} = 1$, the state of the system is given by the tuple $s(t) := (w(t), i(t), j(t), l(t))$ corresponding to the buffer levels at time $t$, with $s(t) \in S$, where $S := \{(w, i, j, l) \mid w \leq L_0, 1 \leq L_1, j \leq L_2, l \leq L_3; \text{ and } L_q \in \mathbb{Z}^*, q = 0, 1, 2, 3\}$ is the state space, $\mathbb{Z}^* := \mathbb{Z}^+ \cup \{0\}$, $L_0$, $L_1$, $L_2$, and $L_3$ represents the buffer capacities in Buffer 0, 1, 2, and 3, respectively. If a buffer reaches its maximum capacity, then a blocking-before-service mechanism is activated and no jobs are allowed to be received in that buffer. Similarly, when the model does not include the job releasing control, i.e., $d_{IR} = 0$, the state is given by the tuple $s(t) := (i(t), j(t), l(t))$, and the state space by $S := \{(i, j, l) \mid i \leq L_1, j \leq L_2, l \leq L_3; \text{ and } L_q \in \mathbb{Z}^*, q = 1, 2, 3\}$.

The RLM models described above have simple production sequences. For instance, let $d_{IR} = 1$, then new job arrivals join Buffer 0 waiting for a job releasing action when $u_R = 1$; from Buffer 0 the jobs move to Buffer 1 at a rate $\mu_{R_1}$, waiting for service by Machine 1; from Machine 1 jobs move to Buffer 2, waiting for service by machine 2; from Machine 2 jobs move to Buffer 3, waiting for rework at Machine 1 before exiting the system. Thus, jobs in both Buffer 1 and 3 compete for service by Machine 1. When $d_{IR} = 0$, no job releasing control is performed and new job arrivals are immediately moved to
Buffer 1. In addition, both the arrival times of new jobs and the processing times at each machine are exponentially distributed with means $\frac{1}{\lambda_1}, \frac{1}{\mu_1}, \frac{1}{\mu_2}, \frac{1}{\mu_3}$, respectively.

The control decisions in this system deal with both job releasing and sequencing into and within the RML model, respectively. When $d_{IR} = 1$, new jobs are released into the RLM only when $u_R = 1$, i.e., a job is taken from Buffer 0 and sent into the RLM system at rate $\mu_R$. For the job sequencing control, jobs waiting in Buffers 1 and 3 are chosen to be served in Machine 1 when $u_s = 1$ and $u_a = 0$, respectively.

III. CONTROL OPTIMIZATION PROBLEM & SIMULATION-BASED APPROXIMATE DYNAMIC PROGRAMMING APPROACHES

In this section we first describe the optimization control problem for which a DP formulation can be obtained. In the remaining of this section we provide an overview of the four simulation-based ADP approaches utilized to approximate the optimal control policies for the simple RLM models described above. Most of the ADP approaches reviewed here are presented in more detail in [13], [14], [15]. However, for clarity in the presentation we provide in this section a brief overview of such approaches as well as additional comments regarding the implementation and scalability of the methods for the control of RLM models.

A. Optimal Control Problem: Dynamic Programming Formulation

The control optimization problem considers the minimization of the infinite horizon discounted cost criterion defined as follows:

**Definition 1:** Given a discount factor $\beta > 0$, with $\beta \in \mathbb{R}$, then

$$J_\beta^\pi(s_0) := \lim_{N \to \infty} E_\pi \left\{ \int_0^{t_N} e^{-\beta t} g(s(t)) \, dt \mid s(0) = s_0 \right\},$$

(1)

is the $\beta$-discounted cost under policy $\pi \in \Pi_{ad}$, where $\Pi_{ad}$ is the set of admissible policies, $t_N$ is the time for the $N$-th state transition, $g(s(t))$ is the cost function, and $s_0$ is the initial state, with $s(t) \in S$. For simplicity we assume that there are no costs associated with the control actions. Then, the cost function $g(s(t))$ is a function of the state only.

From (1), the optimal $\beta$-discounted cost is then defined as $J_\beta^\pi(s_0) := \min_\pi J_\beta^\pi(s_0)$. Moreover, if $J_\beta^\pi(s_0) = J_\beta^\pi(s_0)$, then $\pi^* \in \Pi_{ad}$ is said to be an optimal policy.

Given that the RLM model studied has exponentially distributed transition times, an uniformization procedure [12] is utilized to obtain a discrete-time and statistically equivalent model, i.e., an optimal policy in the discrete-time model is also optimal in the continuous-time one. Such discrete-time optimization model is described next.

**Definition 2:** Given a uniformized version of a SMIP under the discounted cost criterion (1), then

$$J_\alpha^\pi(s) := \lim_{N \to \infty} \frac{1}{\beta + \nu} E_\pi \left\{ \sum_{k=0}^{N} \alpha^k g(s_k) \mid s_0 = s \right\}$$

(2)

is the $\alpha$-discounted cost under policy $\pi \in \Pi_{ad}$, where $\alpha := \frac{1}{\beta + \nu}$, and $g(s_k)$ are, namely, the discount factor and the one-stage cost for the discrete-time model. In addition, $k = 0, 1, 2...$ is the state-transition index and $\nu$ is the uniform transition rate [12].

The optimal $\alpha$-discounted cost $J_\alpha^\pi(s)$ is defined as $J_\alpha^\pi(s) := \min_\pi J_\alpha^\pi(s)$. In addition, in (2) it is assumed that the one-stage cost function $g(s)$ is nonnegative and monotonically nondecreasing w.r.t. the usual componentwise partial order.

In general, Bellman’s optimality equation [12] for this problem can be expressed as follows:

$$J_\alpha^\pi(s) = \frac{1}{\beta + \nu} \min_{\pi \in \Pi_{ad}} \left\{ g(s) + \nu \sum_{s'} p(s \mid s', \pi) J_\alpha^\pi(s') \right\},$$

(3)

where $s, s' \in S$, and $s'$ represents the next state, with $s' = s(t_{n+1}) = f(s, u)$, and $\{p(s \mid s', \pi)\}$ are the state transition probabilities for the uniform version of the continuous-time Markov chain. In addition, $u$ is a vector of control actions and $U$ is a set of constraints for $u$. For instance, when $d_{IR} = 1$ we have that $u := [u_R, u_s]$, and when $d_{IR} = 0$, then $u := [u_a]$. It is worth mentioning that obtaining exact optimal solutions of control problems for this type of queuing networks is usually a challenging task. There are only a few and specific cases for which exact optimal solutions are known, e.g., the $\mu c$-rule [12]. However, in some cases it is possible to obtain sufficient conditions for optimality. For instance, for the RLM model given in section II with $d_{IR} = 0$, sufficient conditions for optimality and structural properties of the job sequencing policy are provided in [20] but under strong constraints in the system’s parameters. Moreover, in the case of the models described in section II, classical numerical methods such as value iteration and policy iteration [12] can be utilized to numerically solve the problem under an appropriate selection of the buffers capacities.

In the next subsection we provide an overview of the four simulation-based ADP approaches utilized to obtain near-optimal solutions for the control optimization problem described above.

B. Simulation-Based ADP Approaches

For this research we selected four different simulation-based ADP approaches to approximate the optimal control policies for the two RLM models described in section II. The approaches selected are Q-Learning, Q-Learning with state aggregation, SARSA(\lambda), and an Actor-Critic architecture. These approaches represent basic and more advanced approaches that ranges from those based on lookup table representations to those that use parametric approximation structures to estimate optimal cost functions. Moreover, these approaches are core methods utilized in different variations of ADP algorithms [8], [9], [10], [11].

In the next subsections we provide a brief overview of each of these approaches as well as the key aspects for the application in the optimization of control operations of the RLM models from section II.
1) **Q-Learning**: The first simulation-based ADP approach employed was the well known Q-Learning algorithm [10], [11], [16]. Q-learning has the advantage of both being easy to implement and also does not require explicit information from the system’s model. It only requires information of the costs and state transitions which can be easily generated with a simulation model. Q-learning has been extensively studied and solid results on convergence have been established, e.g., see [11]. However, given that the method is based on lookup table representations, the approach may be impractical when the control problem needs to deal with large action and state spaces as it will be illustrated in the sequel. Nevertheless, in the experiments conducted with the simple RLM models, Q-Learning provided good performance and as such it deserved to be included in the final comparison of the ADP approaches employed.

The Q-learning algorithm departs from the concept of Q-factors [10], [11], [16] of a state-control pair \((s, u)\) and a stationary policy \(\pi\), which are defined as follows:

\[
Q^\pi(s, u) := \sum_{s'} p_{ss'}(u)(g(s, u, s') + \alpha J^\pi_\alpha(s')) ,
\]

where \(Q^\pi(s, u)\) is the expected discounted cost starting from state \(s\) and by applying the control \(u\) under the policy \(\pi\). In addition, \(g(s, u, s')\) is the cost per stage resulting from the transition from \(s\) (current state) to \(s'\) (next state) and under the control \(u\), and \(\{ p_{ss'}(u)\} \) represents the transition probabilities. If \(\pi\) is optimal, then we obtain the optimal Q-factors \(Q^*(s, u)\), and Bellman’s equation can be expressed as

\[
J^*_{\alpha}(s) = \min_{u \in U(s)} Q^*(s, u) ,
\]

where the optimal action \(u^* \in U\) is given by \(u^*(s) = \arg \min_{u \in U(s)} Q^*(s, u)\). In [11] it is shown that by assuming that the cost function \(g(s)\) is nonnegative and bounded, the optimal Q-factors can be obtained by visiting the state-control pairs \((s, u)\) an infinite number of times, and by using the following iterative equation

\[
Q_{k+1}(s, u) = (1 - \gamma_k(s, u))Q_k(s, u) + \gamma_k(s, u) \left[ g(s, u, s') + \alpha \min_{\ell \in U} Q_k(s', \ell) \right] ,
\]

where \(\gamma_k(s, u)\) is a nonnegative step size that is reduced to zero as \(k \rightarrow \infty\) in order to obtain convergence of the Q-factors to \(Q^*(s, u)\) [11]. Q-learning can also include additional methods for exploring the control space. For instance, the so-called \(\epsilon\)-greedy approach [10], [11] is commonly utilized to select greedy actions w.p. \(1 - \epsilon\) and a random action w.p. \(\epsilon\).

In terms of implementation, Q-learning requires a lookup table with space to store the total number of Q-factors associated with the state-action pairs, \((s, u)\). Thus, in the case of the RLM models presented here, the dimension of the state space is determined by the buffer’s capacities. For instance, consider the RLM model with \(L_1 = L_2 = L_3 = 100\) jobs and \(d_{LR} = 0\). Given that there are two combinations for the control action \(u_a \in \{0, 1\}\), for this problem a lookup table with \(100 \times 100 \times 100 \times 2 = 2 \times 10^6\) entries would be required to store all the Q-factors.

2) **Q-Learning with a State Aggregation Approach**: As an alternative to both reduce the size of the lookup table and to facilitate the implementation of a Q-Learning-based algorithm, we utilized a compact state representation via a state aggregation approach (SA) [8], [10], [11]. We will refer to this approach as Q-Learning & SA.

The proposed ADP approach utilizes a compact representation of the state space that is based on the aggregation of states according to the relative value of the buffer levels in the system. Figure 3 depicts how the aggregation of the states is performed for the case of the RLM model with job sequency control only (i.e., \(d_{LR} = 0\)), where the state components (i.e., buffer levels) are \(i, j, l\). As shown in Figure 3, the proposed method aggregates the states \(s = (i, j, l) \in S\) in 13 “super” states \(S_0, S_1, ..., S_{12}\). For example, the super state \(S_0\) aggregates all the states \((i, j, l) \in S\) such that \(i = l\) AND \(i = j\) AND \(j = l\). Thus, if the set of super states is denoted by \(\Sigma := \{S_0, S_1, ..., S_{12}\}\), and \(S_{agg} \in \Sigma\), then the following equation, based on Q-learning, is utilized to update the Q-factors after each state transition obtained through simulation:

\[
\tilde{Q}_{k+1}(S_{agg}, u) = (1 - \gamma_k(S_{agg}, u))\tilde{Q}_k(S_{agg}, u) + \gamma_k(S_{agg}, u) \left[ G(S_{agg}, u, S'_{agg}) + \alpha \min_{u' \in U} \tilde{Q}_k(S'_{agg}, u') \right] ,
\]

where \(G(S_{agg}, u, S'_{agg})\) is the one stage cost \(g(s, u, s')\) evaluated in the states \(s, s' \in S\) that are aggregated into \(S_{agg}\) and \(S'_{agg}\), respectively.
The proposed state aggregation approach can be extended for a general system with \( n \) buffers. However, as the number of buffers in the system is increased, the implementation also requires a large amount of entries for the lookup table. For instance, Table I below lists the exact number of aggregated states \( N_{\Sigma, n} \) for a RLM model with \( n \) buffers. Notice that the table lists the values of \( N_{\Sigma, n} \), obtained numerically, for the cases \( n = 2, \ldots, 6 \). Higher values of \( n \) require extensive computations and are not presented. However, by using combinatorial arguments it is possible to obtain an upper bound for \( N_{\Sigma, n} \). Thus, the table also lists this upper bound, \( \Omega_n \), where \( N_{\Sigma, n} \leq \Omega_n \). As a reference, the table includes the number of states in a system with a fixed capacity of 100 jobs per buffer.

**TABLE I**  
**Comparison of Number of States Obtained with and without State Aggregation**

<table>
<thead>
<tr>
<th>( n )</th>
<th>( N_{\Sigma, n} )</th>
<th>( \Omega_n )</th>
<th>( 100^n )</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>3</td>
<td>3</td>
<td>( 1 \times 4 )</td>
</tr>
<tr>
<td>3</td>
<td>13</td>
<td>13</td>
<td>( 1 \times 6 )</td>
</tr>
<tr>
<td>4</td>
<td>75</td>
<td>79</td>
<td>( 1 \times 8 )</td>
</tr>
<tr>
<td>5</td>
<td>541</td>
<td>621</td>
<td>( 1 \times 10 )</td>
</tr>
<tr>
<td>6</td>
<td>4683</td>
<td>5971</td>
<td>( 1 \times 12 )</td>
</tr>
<tr>
<td>50</td>
<td>-</td>
<td>( 3.57 \times 10^{77} )</td>
<td>( 1 \times 100 )</td>
</tr>
<tr>
<td>100</td>
<td>-</td>
<td>( 6.04 \times 10^{185} )</td>
<td>( 1 \times 200 )</td>
</tr>
</tbody>
</table>

*The number of states resulting when no state aggregation was utilized assumed buffer’s capacities of 100 jobs.

As presented in Table I, although the proposed state aggregation can provide substantial reduction in computational loads when the number of buffers is small, when dozens of buffers are considered the number of aggregated states may result as cumbersome to handle as the number of states without using state aggregation.

Thus, it is clear that the application of ADP methods based on lookup tables for near-optimal control of RLM models could be restricted to small systems only. As it will be discussed next, an alternative to deal with larger state spaces is that based on the utilization of parametric structures for approximating either the optimal Q-factors or the optimal cost function.

3) **SARSA(\( \lambda \)): parametric approximation of the optimal Q-factors:** An effective alternative to deal with large state spaces while applying ADP in control problems is to utilize parametric structures to estimate the optimal Q-factors or optimal cost functions. An example of such approaches is the denominated SARSA(\( \lambda \)) algorithm with parametric approximation [10], [11]. Similar methods also include Q(\( \lambda \)), Watkin’s Q(\( \lambda \)), and Peng’s Q(\( \lambda \)) [10]. Among them, SARSA(\( \lambda \)) has shown advantages for being computationally least expensive and for providing faster convergence [10].

For this application we utilized a linear approximation structure to estimate the optimal Q-factors as follows:

\[
\hat{Q}(s, ru) := \psi(s) \cdot ru^T \approx Q^*(s, u), \tag{7}
\]

where \( \hat{Q}(s, ru) \) is the parametric approximation of the optimal Q-factor given \( s \in S \) and \( u \in U \), \( \psi(s) \) is a vector of basis functions [10], [11], and \( ru \) is the vector of parameters to be adjusted by the ADP algorithm that depends on the control action \( u \).

An important aspect in this ADP approach is that there are as much \( \hat{Q}(s, ru) \) functions as combinations of the control components of \( u \). For instance, in the case of the RLM model with \( d_{IR} = 1 \) we have the control actions \( u_R \) and \( u_s \). Therefore, in the application of SARS(\( \lambda \)) for optimization of control actions in such model we utilized one parametric structure to approximate the Q-factors for each combination of the control actions \( (u_s, u_R) \), i.e., \( \{(0,0), (0,1), (1,0), (1,1)\} \). We utilized a gradient-descent approach with temporal differences (TD) learning [10], [11] for tuning each of the parametric structures. Such approach seeks to minimize the mean-squared error between the estimated and the optimal value of the Q-factors. In addition, an \( \epsilon \)-greedy policy approach is utilized within the algorithm. Therefore, exploration on the control space is allowed by selecting a random control action with a small probability \( \epsilon \). Additional details regarding the Algorithm and the application in the case of the RLM model with both job sequencing and job releasing (i.e., \( d_{IR} = 1 \)) can be found in [14].

It is worth noting that, although results suggest that this particular ADP approach can deal with large state spaces and shares similar features with Q-learning (e.g., does not need explicit information of the model), a clear disadvantage stands. In this ADP approach, the complexity in the implementation and computational work increases with the number of control actions in the RLM model. An alternative to deal with large action spaces is to directly estimate the optimal cost function \( J^*_s(s) \). Such approach will be explained in the next subsection.

4) **A Simulation-Based ADP Approach Based on an Actor-Critic Architecture:** Results indicate that, given the difficulties of large action spaces mentioned above for the ADP approach based on SARSA(\( \lambda \)), a more suitable alternative for dealing with such conditions is to directly approximate the optimal discounted cost \( J^*_s(s) \). In this case, the architecture utilized by the ADP approach follows a type of actor-critic architecture [10], [11] as illustrated in Figure 4.

As shown in the figure, a simulation model is utilized to generate both traces of the state \( s \) and the one-stage cost values \( g(s) \). These are in turn utilized by an ADP agent to tune a vector of parameters \( r \) used by an estimator of the optimal cost \( J^*_s(s, r) \approx J^*_s(s) \). The estimations of the optimal cost are then utilized to generate approximations to the optimal control \( u^* \approx u^* \). The application of this approach for the control of the RLM model with job releasing and sequencing control is
presented in detail in [15]. However, here we discuss some of the key aspects of the implementation in the case of the RLM model with job sequencing only.

Unlike SARSAR(λ), this approach has the advantage of only utilizing one approximation structure. Therefore, this ADP approach is able to provide a better handling of large action spaces. Another difference with respect to the approach presented previously for approximating optimal Q-factors is that some additional information of the system model could be required for defining the controller or actor.

As an example we present next some details on the design of the actor in the case of the RLM model with job sequencing control only (i.e., \( d_{IR} = 0 \)). The definition of the actor for the RLM model with both job sequencing and job releasing is detailed in [15].

In order to define the actor for the ADP algorithm, we depart from the underlying formulation of the optimal control problem. From (3) and the description of the model given in section II, we obtain the following Bellman’s optimality for problem. From (3) and the description of the model given in section II, we obtain the following Bellman’s optimality for problem. From (3) and the description of the model given in section II, we obtain the following Bellman’s optimality for problem.

From (10) and (11), notice that, for defining the actor, it is necessary to incorporate information from the system’s dynamics in terms of the processing rates \( \mu_1, \mu_2, \) and the mappings from \( S \) to \( S, B_1 \) and \( B_3 \). However, by adding knowledge from the system’s dynamics it is possible to reduce the additional computational tasks derived from the application of algorithms based on approximation of optimal Q-factors such as SARSAR(λ). Also, this approach seems to better scale with the dimension of the control space by requiring only one approximation structure for the optimal cost function.

The next section provides results from numerical experiments on which the four ADP approaches described above were evaluated in the approximation of optimal control policies for the RLM models from section II.

IV. NUMERICAL EXPERIMENTS: COMPARING THE PERFORMANCE OF THE ADP APPROACHES

In order to evaluate and compare the performance of the simulation-based ADP approaches described in section III, different numerical experiments were conducted. These experiments consisted in obtaining approximations to the optimal control policies for the two RLM models given in section II and for two types of cost functions \( g(s) \), a linear and a quadratic cost function.

These experiments were conducted in three steps. In the first step, each of the four ADP approaches utilized generated several approximations of the optimal policy under the same set of system’s parameters and cost function \( g(s) \). Different approximations to the optimal policies were obtained by varying the corresponding parameters for each algorithm. In a second step of the experiments, the best policies obtained via ADP were identified by numerically evaluating them to compute the discounted cost under such policies. Finally, the best policies obtained were compared against optimal solutions obtained numerically with a modified policy iteration (MPI) algorithm [12].

The following are specific conditions considered for these experiments:

- Both the simulation models and the ADP algorithms were implemented with the simulation software ARENA [21].
- Each ADP approach utilized generated approximations to the optimal policies with a “learning process” that consisted of between 50 and 400 simulation replications with lengths between 2000 and 10000 time units each.
- The parametric approximation structures utilized in the SARSAR(λ) and the Actor-Critic architecture were quadratic structures in the state \( s \) such as those described in [14], [15]. For example, when the model considered the job releasing control, the vector of basis functions utilized was \( \vec{\psi}(\mu, i, j, l) := [\mu^2 \ i^2 \ j^2 \ \mu \ i \ j \ l \ 1] \), and then the vector of parameters \( \mu \) included eight parameters.
- An ε-greedy policy approach was utilized in all the ADP approaches with values of \( \epsilon \in \{0, 0.01, 0.1\} \).
We utilized values for the parameter $\lambda$ for the TD($\lambda$) algorithm in the set $\lambda \in \{0.6, 0.7, 0.8, 0.9\}$. However, the best results were obtained for $\lambda = 0.7$ and $\lambda = 0.9$. This agrees with experimental observations reported in [22] for which best approximations with TD($\lambda$) algorithms are generally obtained around the value of $\lambda = 0.7$. Moreover, as indicated in [11, Proposition 6.5, pp. 305], when linear approximation structures are utilized with a TD($\lambda$) algorithm, the estimation error is reduced as $\lambda \to 1$.

We utilized the same values for the system’s parameters reported in [3] for the simple RLM model with job sequency only, where $\lambda_a = 0.1429$, $\mu_1 = \mu_3 = 0.3492$, $\mu_2 = 0.1587$. In addition, when the model consider the job releasing control, we assumed that $\lambda_R = 0.4492$ such that the releasing process is faster when compared with the other processings in the system.

All buffer capacities were set to 20 jobs which facilitated the numerical computation of the optimal cost and policy via a MPI algorithm.

We tested values for the discount factor $\beta \in \{0.1, 0.2, 0.5, 0.7\}$. However, here we present results when $\beta = 0.1$ for the case of the model with job sequency only, and $\beta = 0.2$ when the model incorporates both job sequency and releasing controls. These values of $\beta$ imply $\alpha \approx 0.91$ and $\alpha \approx 0.88$, respectively. Notice from (2) that as $\alpha \to 1$ the portion of the planning horizon for which the control actions have more impact in the discounted cost is extended. As a result, the approximation of the optimal policy via ADP is more challenging in such cases when compared to the cases with smaller values of $\alpha$.

For both RLM models studied, we utilized a linear and quadratic cost functions $g(s)$. When $d_{IR} = 0$ we used $g(i, j, l) = c_1 i + c_2 j + c_3 l$ and $g(i, j, l) = c_1 i^2 + c_2 j^2 + c_3 l^2$, and for $d_{IR} = 1$ $g(w, i, j, l) = c_0 w + c_1 i + c_2 j + c_3 l$ and $g(w, i, j, l) = c_0 w^2 + c_1 i^2 + c_2 j^2 + c_3 l^2$. Also, we assumed that the system starts with one job in Buffer 1 such that the initial states for each model were $s_0 = (1, 0, 0)$ and $s_0 = (1, 0, 0, 0)$, respectively.

Three different cases for the values of the cost coefficients $c_0, c_1, c_2, c_3 \in \mathbb{R}^+$ were utilized in the experiments.

The values of the cost coefficients in each of the three cases considered for the linear and quadratic cost functions are listed in Tables II and III, respectively. In these tables JS stands for the RLM model with job sequencing control only, and JS & IR stands for the model with both job sequencing and job releasing control.

### Table II: Cost Coefficients Cases: Linear Cost

<table>
<thead>
<tr>
<th>RLM Model</th>
<th>Case (a)</th>
<th>Case (b)</th>
<th>Case (c)</th>
</tr>
</thead>
<tbody>
<tr>
<td>JS</td>
<td>$c_1 = 1$</td>
<td>$c_1 = 1$</td>
<td>$c_1 = 1$</td>
</tr>
<tr>
<td></td>
<td>$c_2 = 1$</td>
<td>$c_2 = 1$</td>
<td>$c_2 = 1$</td>
</tr>
<tr>
<td></td>
<td>$c_3 = 1$</td>
<td>$c_3 = 3$</td>
<td>$c_3 = 1$</td>
</tr>
<tr>
<td>JS &amp; IR</td>
<td>$c_0 = 1$</td>
<td>$c_0 = 3$</td>
<td>$c_0 = 4$</td>
</tr>
<tr>
<td></td>
<td>$c_1 = 1$</td>
<td>$c_1 = 1$</td>
<td>$c_1 = 3$</td>
</tr>
<tr>
<td></td>
<td>$c_2 = 1$</td>
<td>$c_2 = 2$</td>
<td>$c_2 = 2$</td>
</tr>
<tr>
<td></td>
<td>$c_3 = 1$</td>
<td>$c_3 = 3$</td>
<td>$c_3 = 1$</td>
</tr>
</tbody>
</table>

The selection of the values for the cost coefficients was based on experimental observations of numerical computations of the optimal policy. Thus, the cases listed in Tables II and III provided different variations of the optimal policy for which it was possible to test the capabilities of each ADP algorithm utilized in the estimation of the corresponding optimal policy. Case (a) is considered a neutral case on which each buffer has the same holding costs and priorities in certain control decisions are not necessarily biased due to the value of the cost coefficients. In Case (b), the holding costs are higher in Buffer 3 (and Buffer 0) which yields an optimal policy that provides higher priority of service to Buffer 3 (and the action of releasing new jobs into the system). For Case (c), there is a difference in the type of bias that was given to the optimal policies. On the one hand, for both type of cost functions we provide values to the cost coefficients such that the optimal job sequency policy gives priority to the service of Buffer 1. On the other hand, in the case of the optimal job releasing policy, we provide cost coefficients that yielded priority to release jobs in the case of linear costs, and not to release jobs in the case of quadratic costs. The reason for this difference is that experimentally we observed that for linear costs the optimal policies are static while for quadratic costs the optimal policy follows a switching-over function [1] on the state $s$. Therefore, in the case of linear costs we avoided the the selection of cost coefficients that yields the optimal policy of not releasing jobs at all, which is not interesting.

The evaluation of the policies obtained via ADP, say $\pi_{ADP}$, was performed numerically via successive approximations to the discounted cost under such policy. This type of numerical evaluation follows from Corollary 1.2.1.1 reported in [12, pp. 11] which can be expressed as follows: for any stationary policy $\pi$ in discounted cost problems we have that $\lim_{k \to \infty} (T^k J^\pi_\alpha(s_0)) = J^\pi_\alpha(s_0) \forall s_0 \in S$, where $(T^k J^\pi_\alpha)(s_0)$ is viewed as the cost-to-go for the $k$-th state transition, for the $\alpha$-discounted cost problem, with initial state $s_0$, under the policy $\pi$, with the one-stage cost function $g(s)$, and terminal cost function $\alpha^k J^\pi_\alpha$. For the results presented in this section, the computation of the discounted cost $J^\pi_\alpha(s_0)$ was performed with a tolerance of $1 \times 10^{-9}$ w.r.t. the exact value.

For comparison purposes, the corresponding optimal control policies were also computed numerically with a MPI algorithm. The values for the optimal discounted cost function departing from the initial state $s_0$, $J^\pi_\alpha(s_0)$, was computed with a tolerance of $1 \times 10^{-9}$ w.r.t. the exact value. These optimal performances under the linear and quadratic cost functions.
considered, and according to the cases of cost coefficients indicated above, are listed in Tables IV and V, respectively.

The results for the experiments are listed in Tables VI to IX below, where QL, QL & SA, and AC, stands for Q-Learning, Q-Learning with State Aggregation, and Actor-Critic, respectively. These tables list the percentual difference between the performance yielded by policies obtained via ADP and the corresponding optimal values listed in Tables IV and V. Such percentual difference is defined as % Error and it is computed using the following formula

\[ \text{% Error} := \frac{J^\text{ADP}(s_0) - J^\ast(s_0)}{J^\ast(s_0)} \cdot 100\%. \]  

(12)

The results in each table are also listed according to the probability of exploration \( \epsilon \) utilized within the \( \epsilon \)-greedy policy approach in all the ADP algorithms.

Results show that, in general the Actor-Critic approach provided the best performance when compared with the other ADP approaches studied. The Actor-Critic scheme consistently generated control policies that were close to the optimal performance with a maximum percentage of error of 1.94%. Although the Actor-Critic required additional knowledge of the system when compared to the other methods, the use of such additional data was worthy in terms of consistently obtaining good performance.

From the results it can be also noticed that simple methods such as Q-Learning and Q-Learning \& SA performed quite well in the case of the simplest RLM model which only considered job sequency control. Such approaches even provided convergence to the optimal policy when linear costs were utilized as it is indicated in Table VI for Case (a) and QL \& SA. However, when the control problem becomes more complex by changing the one-stage cost function, adding the job releasing control and additional states, or by changing the value of the cost coefficients, the control policies obtained with such simple ADP approaches yielded the high percentages of error when compared against the optimal policies, e.g., see Table IX, Case (c). As seen in Tables VI to IX, SARS(A(\lambda)) also provided a reasonable good performance throughout the experiments for Case (a) and (b) with a maximum percentage of error of 7.86% w.r.t. to the optimal policy in the case of quadratic costs and the RLM model with both job sequencing and job releasing controls. The approach even provided convergence to the optimal policy in the case of linear costs and the RLM model with job sequency only, such as it is indicated in Tables VI and VIII. However, when the model incorporates both job sequencing and job releasing controls, quadratic costis, and the cost coefficients were set as in Case (c) of Table III, SARS(A(\lambda)) yielded a higher percentual difference with respect to the performance of the optimal policy.

Another observation from the results obtained is that not all the ADP approaches utilized demonstrated benefits by using the \( \epsilon \)-greedy policy approach for exploration purposes. Some of the cases observed correspond to the scenario where Q-Learning was used, with both linear and quadratic costs, \( \epsilon = 0.01 \), and for the RLM model with job sequencing control only. In such case (see Tables VI and VII) the performance improved by including exploration. However, for a higher rate of exploration, the performance is degraded. A similar situation was also observed in the case of SARS(A(\lambda)).

The results presented here also suggest that the selection of the ADP approach to utilize in a given application will be dependent not only of the complexity of the problem but also of the limitations in the implementation of the methods. While for some simple problems Q-Learning will be easy to implement and provide good results, in other applications the utilization of look-up tables may be prohibited due to large state and actions spaces. Similarly, in some simple problems the implementation of Actor-Critic methods could be more costly compared with Q-Learning while providing similar results. Therefore, there is a trade-off in the selection of the most appropriate method that is directly associated with the complexity of the problem and the limitations in the implementation.

V. CONCLUSIONS

We presented a comparison of four different simulation-based ADP approaches for the optimization of control operations in two simple RLM models. Overall, the best performance was obtained from the Actor-Critic approach that utilizes one parametric approximation structure which is tuned via a TD(\( \lambda \)) learning algorithm and a gradient descent method. In contrast with the other approaches utilized, the Actor-Critic scheme requires the incorporation of previous knowledge of the system in terms of processing rates and state transition mappings. However, such inclusion of additional information of the system in the algorithm is worthy given the good performance obtained, and the better scalability features of this approach with respect to the state and action spaces. Thus, the results from this research also suggest that, among the ADP approaches studied, the Actor-Critic architecture would be the most suitable ADP approach for applications in more complex RLM models. Current research is focused in this necessary next step in order to assess such ADP approach in terms of performance, scalability, and handling of the dimensionality difficulties in the state and action spaces when neither an analytical model nor optimal control solutions are available.

REFERENCES

<table>
<thead>
<tr>
<th>Approach</th>
<th>Case (a)</th>
<th>% Error</th>
<th>Case (b)</th>
<th>% Error</th>
<th>Case (c)</th>
<th>% Error</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$e=0$</td>
<td>$e=0.001$</td>
<td>$e=0.1$</td>
<td>$e=0$</td>
<td>$e=0.001$</td>
<td>$e=0.1$</td>
</tr>
<tr>
<td>QL</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>QL &amp; SA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SARS(A)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Approach</th>
<th>Case (a)</th>
<th>% Error</th>
<th>Case (b)</th>
<th>% Error</th>
<th>Case (c)</th>
<th>% Error</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$e=0$</td>
<td>$e=0.001$</td>
<td>$e=0.1$</td>
<td>$e=0$</td>
<td>$e=0.001$</td>
<td>$e=0.1$</td>
</tr>
<tr>
<td>QL</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>QL &amp; SA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SARS(A)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Approach</th>
<th>Case (a)</th>
<th>% Error</th>
<th>Case (b)</th>
<th>% Error</th>
<th>Case (c)</th>
<th>% Error</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$e=0$</td>
<td>$e=0.001$</td>
<td>$e=0.1$</td>
<td>$e=0$</td>
<td>$e=0.001$</td>
<td>$e=0.1$</td>
</tr>
<tr>
<td>QL</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>QL &amp; SA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SARS(A)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

preventive maintenance scheduling in semiconductor manufacturing,”
IEEE Transactions on Semiconductor Manufacturing, vol. 17, no. 3,

queuing system,” in Proceedings of the 28th IEEE Conference on

control in a re-entrant manufacturing line model,” 2009, manuscript in
preparation, University of Cincinnati.


[22] R. S. Sutton, “Learning to predict by the methods of temporal differ-