FLEXIBLE STATE-DEPENDANT MACHINE SCHEDULING PROBLEMS USING REINFORCEMENT LEARNING

Carlos D. Paternina-Arboleda
Laboratorio de Robótica y Automatización de la Producción
Universidad del Norte
Barranquilla, Colombia
(*cpaterni@uninorte.edu.co*)

Abstract

This paper presents a simulation-based optimization methodology called reinforcement learning (RL) and suggests a neural approach to approximate the values when the systems under study are complex and involve large-scale decision-making sequential tasks. Computer simulation based reinforcement learning (RL) methods of stochastic approximation have been proposed in recent years as viable alternatives for obtaining near optimal policies for large scale MDPs with considerably less computational effort than that required for DP algorithms. RL does not require computation of the transition probability and reward matrices, and can handle problems with very large state spaces since its computational burden is related only to value function estimation.

Keywords: scheduling, simulation, optimization, reinforcement learning.

1. INTRODUCTION

Reinforcement learning (RL) is a way of teaching agents (decision-makers) optimal control policies. This is accomplished by assigning rewards and punishments for their actions based on the temporal feedback obtained during active interactions of the learning agents with dynamic systems. The agent's behavior should choose actions that tend to increase the long run average reward (Kaelbling et al. [8]). Such an incremental learning procedure specialized for prediction and control problems was developed by Sutton [14] and is referred to as *temporal-difference (TD)* methods.

A typical learning model (as depicted in Figure 1) contains 4 elements, which are the environment, the learning agent, a set of actions, and the environmental response (sensory input). The learning agent selects an action for the system, which leads the system evolution along a unique path till the system encounters another decision-making state. At that time, the system consults with the learning agent for the next action. After a state transition, the learning agent gathers sensory inputs from the environment, and from it derives information about the new state, immediate reward, and the time spent during the state-transition. Using this information and the algorithm, the agent updates its knowledge base and selects the next action. This completes one step in the iteration process. As this process repeats, the learning agent continues to improve its performance. A simulation model of the system provides the environment component of the model.

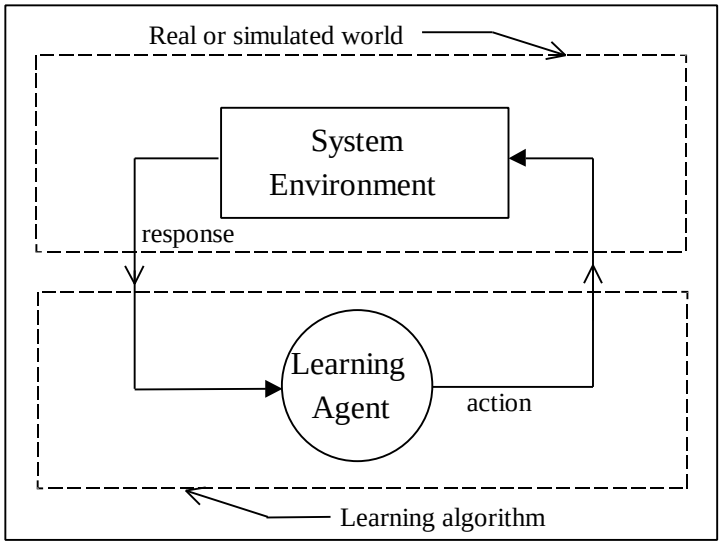

Figure 1. Typical reinforcement learning scheme

There are two different factors that determine the utility of an action. These are the immediate reward, and the action value of the state to which a transition occurs as a result of that action. When a system visits a state, the decision maker chooses an action with the highest (or lowest for minimization) action value (greedy policy). Initially, the action values for all state-action pairs are assigned arbitrary equal values (e.g., zeros). When a system visits a state for the first time, and several other times during the learning phase, the decision maker explores the environment by taking random actions. As the systems revisits the state, the learning agent selects the action based on the current action values. As good actions are rewarded and bad actions are punished over time, for every state, the action values of a smaller subset (one or more) of the actions tend to grow and other diminish. The learning phase ends when a clear trend appears with one or more actions in every state being dominant. These actions constitute the decision policy vector. For further explanation of the working of a reinforcement learning model, refer to Das et al. [**4**].

Although there are several different types of reinforcement learning models in the open literature, this study focuses on the implementation of infinite horizon models with the average reward metric. Following is a brief description of the average reward metric.

2. MATHEMATICAL MODEL

First, we identify the stochastic processes that govern the system, in particular the production and the demand processes. The reader may notice that the equations for the one-step transition probability structure are not shown. The system is represented by an (n+1)-dimension vector Θ, which is given by:

$$\Theta = \{\omega, b_1, b_2, \ldots, b_n\}, \qquad \text{(equation 1)}$$

where ω is the state of the flexible manufacturing station at the decision-making epoch by the agent, and b_i is the buffer level for each product-type at the same decision-making epoch.

The system changes states every time that anyone of the vector components is modified. However, the object of interest under the semi-Markov assumption is to emphasize those epochs in which a decision has to be made, in search for a near-optimal sequencing policy for the flexible station. The

decision-making states are only seen after a production completion event or at a new demand arrival event (if the machine is idle at the time).

After an action is executed by the agent the system reaches state *j* (decision-making epoch). If X_m is defined as the system-state at time epoch *m* and T_m is the time required to reach such epoch then *X* follows certain property and it is considered that the process is associated to a Markov chain,

$$P\{X_{m+1} = j \mid X_0, \ldots, X_m; T_0, \ldots, T_m\} = P\{X_{m+1} = j \mid X_m; T_{m-1} - T_m\}, \quad \text{(equation 2)}$$

Furthermore, if Y_t indicates the state of the system at time *t*, then it is obvious that Y_t is a semi-Markov process and the time between the m-th and the (m+1)-th time epochs is a *random* variable [12].

In order to measure the agent's development, a cost (reward) structure function is defined as follows. If the system is run for *t* time units under a fixed sequencing policy and Cp_i is the reward obtained after a demand satisfaction event for the i-th product. $Cs_{i,j}$ is the set-up cost when the station switches from product i to product j. Then the average reward function is defined as (with no handling or backlog costs):

$$r = \frac{1}{t}\left(\sum_i Cp_i d_i - \sum_i \sum_j Cs_{i,j} n_{i,j}\right), \quad \text{(equation 3)}$$

where d_i represents the number of demand entities for the *i*-th product that were satisfied during *t* time units and $n_{i,j}$ represents the number of times the system switch from product *i* to product *j* during *t* time units.

3. AVERAGE REWARD RL

In most manufacturing systems, the optimal total expected reward is finite, either due to the effects of discounting or because of a reward-free termination state that the system eventually enters. In many situations, however, discounting is inappropriate and there is no natural reward-free state. This encourages the search for an optimal average reward per stage starting from stage *i* which is defined by a policy $\pi = (\pi_0, \pi_1, \ldots)$ by

$$R^p(i) = lim_{N\to\infty} \frac{1}{N} E\left[\sum_{k=0}^{K} \left(r(i_k, p_k, i_{k+1}) \mid i_0 = i\right)\right], \quad \text{(equation 4)}$$

assuming that the limit exists, where *r(i, a, j)* is the reward received by taking action *a* in state *i* and going to state *j*. The average reward per stage of policy is actually the reward received in the long term. Hence the rewards received in the early stages do not matter because their contribution to the average reward per stage is reduced to zero as $N \to \infty$, i.e.,

$$lim_{N\to\infty} \frac{1}{N} E\left[\sum_{k=0}^{K} \left(r(i_k, p_k, i_{k+1}) \mid i_0 = i\right)\right] = 0, \quad \text{(equation 5)}$$

for any *K* that is either fixed or is random and has a finite expected value.

4. MODEL-FREE RL

This section discusses the kind of reinforcement learning algorithms that do not require the assumption of an underlying model to optimize the utility/value function. As with their model-based counterparts, these methods are iterative procedures that search for an optimal cost-to-go. Both discounted and average reward methods have been developed, and some have been proven to have convergence properties (Bertsekas and Tsitsiklis [**2**], Gosavi [**6**]).

The model-based RL algorithms estimate the transition probabilities using simulation. Hence, a strong disadvantage of DP (i.e., the need for computing the transition probabilities) is not avoided in model-based RL. The algorithms that obviate this need are referred to as model-free algorithms. Model-free algorithms can infer action values directly from sample paths generated by simulation. For problems with large state spaces, the action values need to be represented by some standard function approximator, such as neural networks, multivariate regression analysis, or some other novel approaches such as kernel regression or wavelet analysis.

Model-free algorithms belong to a class of stochastic iterative algorithms of which an usual updating scheme for action values can be described as follows. Assume that when an action a is taken in state i, and it results in an immediate reward of $r_{imm}(i, a)$ and a system transition to state j. Then, the action value for the state-action pair (i, a) is updated as follows,

$$R_{m+1}(i, a) \leftarrow (1 - \alpha_m) R_m(i, a) + \alpha_m\{r_{imm}(i, j, a) - r_m t(i, j, a) + \max_{b \in A} R_m(j, b)\}, \qquad \text{(equation 6)}$$

where α_m is the learning rate at decision epoch m, and $r_{imm}(i, j, a) - r_m t(i, j, a)$ is an estimate of $R(i, a)$ calculated from the feedback obtained during the system simulation. Q-Learning (Watkins [**15**]), and SMART (Das et al. [**4**]) are examples of model-free RL. In the next section, the relationship between RL and DP, and RL and stochastic iterative algorithms are discussed in brief. For a more detailed discussion, the reader is referred to Bertsekas and Tsitsiklis [**2**].

5. SMART: SEMI-MARKOV AVERAGE REWARD TECHNIQUE

SMART (Das et al. [**4**]) is the first algorithm that deals with Semi-Markov decision-making problems. Suppose a system is in state i, and action a is selected, then the system moves to state j. Let $r_{imm}(i, a, j)$ be the reward generated by going from state i to state j under action a, r be the average reward, and $t(i, a, j)$ be the time spent during the system transition.

- Step 1. Initialize
  Let time step $m = 0$. Initialize action values $R_{old}(i, a) = R_{new}(i, a) = 0$, " i Î E, and a Î A. Choose the current state i arbitrarily. Set the cumulative reward $c_m = 0$, $t_m = 0$, $r_m = 0$. Choose initial values of the rates for exploration (p_m) and learning (α_m).
- Step 2. While $m < MaxSteps$ do

  (a) With high probability (1- p_m), choose an action a that minimizes $R_{new}(i, a)$, otherwise choose a the other action.
  (b) Let the state at the next decision epoch be j, $t(i, j, a)$ be the transition time due to action a, and $r_{imm}(i, j, a)$ be the immediate reward earned as a result of taking action a in state i. Then update the action value for (i, a) as follows:

  $R_{m+1}(i, a) \neg (1 - \alpha_m) R_m(i, a) + \alpha_m\{r_{imm}(i, j, a) - r_m t(i, j, a) + \max_{b \in A} R_m(j, b)\}$

(c) In case a nonrandom action was chosen in step 2(a)

1. Update total time: $t_m \neg t_m + t(i, j, a)$
2. Update total reward for the agent: $c_m \neg c_m + r_{imm}(i, j, a)$
3. Update average reward: $r_m \neg c_m/t_m$

(d) Step 3. Set $i \leftarrow j$, $m \leftarrow m + 1$, decrease a_m, and p_m.

5.1 Relaxed-SMART

Gosavi [**6**] demonstrated that for certain conditions, the original version of the SMART algorithm may not converge. Such conditions are mainly related to the choice of the reinforcement values. If these values are significantly different, the algorithm actually diverges from the optimal. To avoid this problem, a set of stochastic approximation equations is introduced in the algorithm. These equations are known as the Robbins-Monroe stochastic updating scheme,

$$r^{k+1} = (1 - b(k))r^k + b(k)\,\frac{\left[T(k)\,r^k + g(i,u,e_{iu}^k)\right]}{T(k+1)}. \qquad \text{(equation 7)}$$

Relaxed-SMART can be shown to be equivalent to the Bellman equation value iteration. It achieves asynchronously what the Bellman equation value iteration would achieve in a synchronous fashion. The value iteration algorithm requires in every iteration the average cost of the current policy. In *Relaxed-SMART* an estimate of this quantity is supplied by the value of r. The Robbins-Monroe equation of *Relaxed-SMART* ensures that the average cost of the policy keeps changing and an estimate of the current cost can be obtained by r.

6. NEED FOR A FUNCTION APPROXIMATION SCHEME

Notice that the updating scheme of the RL algorithms presented before require the storing of action values $R(i, a)$ for each state-action pair. For a problem with a small state space, the action values for each state-action pair can possibly be stored. The optimal action to be taken in each state can therefore be decided based upon which action value is higher (lower for minimization). But a reasonably sized manufacturing problem, there could be millions of states. This huge state space makes it infeasible to store all the action values explicitly. Hence there is a need for a function approximation scheme, which can be used to determine the action values for each state-action pair.

6.1 Neural Networks

Artificial Neural Networks (ANN) provide a general and practical method for learning real-valued and discrete-valued functions from examples. A Neural Network can be defined as an *interconnected assembly of simple processing elements, units or nodes, whose functionality is loosely based on the brain neuron* (Haykin [**7**]). A primitive class of ANN consisting of a single neuron and operating under the assumption of linearity is presented next. This class of network is known as the *Least Mean Square* (LMS) algorithm, the delta-rule or the *Widrow-Hoff* rule. This algorithm is based on the use of instantaneous estimates of the environment (simulation) response to the learning agent. The method is very simple and allows for incremental learning, which makes it perfect for the kind of problems that are solved with the RL methodology. A summary of the LMS algorithm follows,

- Step 1. Initialization. Set
  $\hat{w}_k(1) = 0$ for $k = 1, 2, ..., p$
- Step 2. Filtering. For time $n = 1, 2, ...,$ compute

$$y(n) = \sum_{j=1}^{p} \hat{w}_j(n) \cdot x_j(n)$$

$$e(n) = d(n) - y(n)$$

$$\hat{w}_k(n+1) = w_k(n) + he(n)x_k(n) \qquad \text{for } k = 1, 2, ..., p$$

where $w_k(n)$ are estimates of the neuron weights at time step n, $y(n)$ is the actual output of the neuron, $d(n)$ is the desired output, $e(n)$ is the error, and h is the learning rate.

The main difficulties encountered with the LMS algorithm may be attributed to the fact that the learning rate parameter is maintained constant throughout the computation, as shown by $h\ (n) = h_0$, for all n. Darken and Moody (1992 [**3**]) proposed the use of a so-called *search-then-converge* schedule, defined by,

$$h\ (n) = \frac{h_0}{1 + (n/t)}, \qquad \text{(equation 8)}$$

where h_0 and t are constants. In the early stages of adaptation, the learning rate parameter is approximately same as the constant h_0 and the algorithm mainly explores. As the number of time steps approaches the constant t, the algorithm rather converges. For a number of iterations n sufficiently large compared to the search time constant t, the learning rate parameter operates as a traditional stochastic approximation algorithm.

After each action choice in step 2 of the *Relaxed-SMART* algorithm presented in chapter 4, the weights of the corresponding action LMS neuron are updated as follows,

$$\Delta w_x = h(m)\ a(m)\ e(m)\ X_i, \qquad \text{(equation 9)}$$

where $h(m)$ is the neuron convergence parameter at decision making epoch m, $a(m)$ is the learning rate of the *Relaxed-SMART* algorithm, x_i is the variable for which the weights are being updated, $e(n)$ is the temporal difference error

$$[r_{imm}(i, j, a) - r_m t(i, j, a) + \min_{a \in A} R_m(j, a, w) - R_m(i, a, w)], \qquad \text{(equation 10)}$$

where w is the vector of weights of the neuron.

7. NUMERICAL EXAMPLE

Consider a production system that consists of a flexible manufacturing station with capacity for 3 products. The system has the following operating characteristics during 240,000 time units with a warm-up period of 9,600 time units:

- Production times Gamma distributed for each product with parameters (n, l) = (3, 1/8),
- the demand process is Poisson with rate 1/60 for each product,
- earnings for demand satisfaction are 10, 12 and 15 for products 1, 2 and 3 respectively,

- set-up costs are 2, 1.5 and 2 for products 1, 2 and 3 respectively (it does not matter the machine setting for the previous operation).

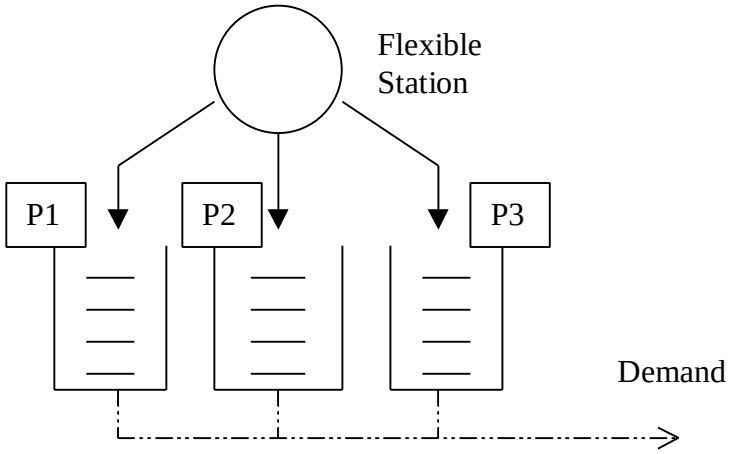

Figure 2 A three-product flexible production station

The total state-space for exploration is approximately 15,000 states. The system evolves and is capable of learning the policy that maximizes the proposed reward function (equation 3). Table 1 shows the results for the optimization problem.

Table 1 Optimization results

Average Reward	Service level		
	u^1	u^2	u^3
58.86	0.91	0.91	0.94

Note that the service levels are always higher than 90%. If the optimization problem is constrained to fulfill at least a pre-determined service level of β (different to 90%), the results would be different. The reader may refer to Paternina and Das [11] for a more detailed discussion on this kind of problems.

Figure 3 shows the agent's behavior during the learning phase. Note that at the beginning of the simulation process, the level of exploration for the agent is higher and it slowly decays until it finally takes the average reward to converge to the optimum value. For a better explanation of the *Relaxed*-SMART behavior and the mathematical convergence proof the reader should refer to Gosavi [6].

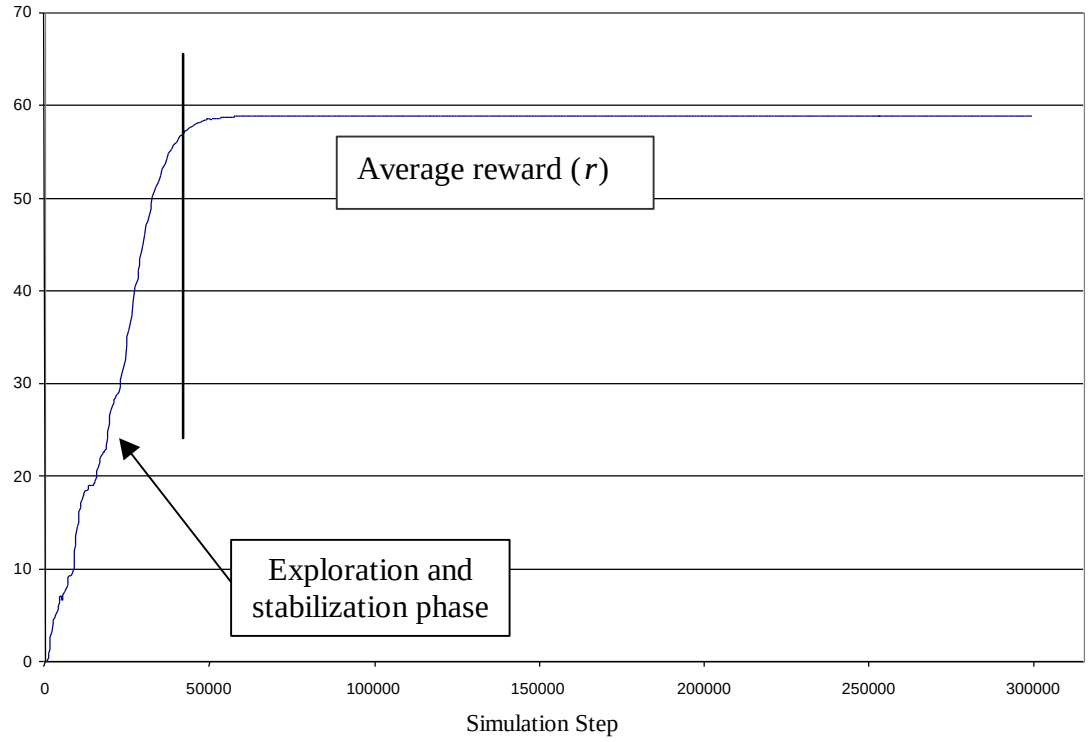

Figure 3 Computational convergence of the agent for the average reward

8. CONCLUSIONS

The application of artificial intelligence procedures to perform simulation-based optimization provides a feasible framework for the analysis of complex production systems in order to improve the operating conditions and, consequently, the productivity of such systems.

The flexibility provided by the off-line optimization analysis allows the implementation of embedded intelligent agents that are capable of responding to conditions in very dynamic production settings, such as logistics (supply chain), capacity planning, and demand processes.

The proposed optimization architecture is generic and can be applied to very diverse production systems that involve dynamic control policies. However, every time a system is modified, it is recommended to update the control policy through a new learning simulation run.

Even for reinforcement learning procedures, the state-space for large complex systems becomes prohibitive for a table look-up approach. To overcome this problem, an incremental regression approach is proposed so as to approximate the values of the payoff matrix. One such approach is the single-adaptive filter network [7]. This can be better seen in [4, 10 y 11].

The reader is referred to [11] for development of an optimization procedure for a function that involves handling and backlog costs.

REFERENCES

1. J. Abounadi. (1998) *Stochastic Approximation for Non-expansive Maps: Applications to Q-learning Algorithms*. Unpublished Ph.D. Thesis, Massachusetts Institute of Technology, Department of Electrical Engineering and Computer Science.

2. D. P. Bertsekas, and J. N. Tsitsiklis. (1996) *Neuro-Dynamic Programming*. 1st Edition, Athena Scientific.

3. C Darken, and J. E. Moody. (1992) Towards Fasters Stochastic Gradient Search. In *Advances in Neural Information Processing Systems 4* (J.E. Moody, S. J. Hanson, and R. P. Lippmann, eds.), pp. 1009-1016. San Mateo, CA: Morgan Kaufmann.

4. T. K. Das, A. Gosavi, S. Mahadevan, and N. Marchellack. (1999) Solving Semi-Markov Decision Problems using Average Reward Reinforcement Learning. *Management Science*, 45, 4, 560-574.

5. T. K. Das, and S. Sarkar. (1999) Optimal Preventive Maintenance in a Production/ Inventory System. *IIE Transactions*, 31, 6, 537-551.

6. A. Gosavi. (1999) *An Algorithm for Solving Semi-Markov Decision Problems Using Reinforcement Learning: Convergence Analysis and Numerical Results*. Unpublished Ph.D. Thesis, University of South Florida, Department of Industrial Engineering.

7. S. Haykin. (1994) *Neural Networks*: A Comprehensive Foundation. McMillan College Publishing Company, Inc. Englewoods Cliffs, NJ, USA.

8. L. P. Kaelbling, M. L. Littman, and A. W. Moore. (1996) Reinforcement Learning: A Survey. *Journal of Artificial Intelligence Research*, 4, 237-285.

9. J. Klir, and B. Yuan. (1995) *Fuzzy Sets and Fuzzy Logic*. Prentice Hall PTR, Prentice Hall, Inc. NJ, USA.

10. S. Mahadevan, and G. Theochaurus. (1998) Optimizing Production Manufacturing using Reinforcement Learning. *Eleventh International FLAIRS Conference*, pp. 372-377, AAAI Press.

11. C. D. Paternina-Arboleda, and T. K. Das. Intelligent Dynamic Control of Single-Product Serial Production Lines. To appear, *IIE Transactions*, Special Issue on Design and Manufacturing.

12. M. L. Putterman. (1994) *Markov Decision Processes*. Wiley Interscience, New York, USA.

13. R. S. Sutton, and A. G. Barto. (1998) *Reinforcement Learning: an Introduction*. A Bradford book. The MIT Press. Cambridge, Massachusetts.

14. R. S. Sutton. (1988) Learning to Predict by the Methods of Temporal Differences. *Machine Learning*, 3, 9-44.

15. C. J. Watkins. (1989) *Learning from Delayed Rewards*. Ph.D. Thesis, Kings College, Cambridge, England, May.