Optimizing Production Manufacturing using Reinforcement Learning

Sridhar Mahadevan and **Georgios Theocharous**
Department of Computer Science
Michigan State University
East Lansing, MI 48824
(mahadeva,theochar)@cps.msu.edu

Abstract

Many industrial processes involve making parts with an assembly of machines, where each machine carries out an operation on a part, and the finished product requires a whole series of operations. A well-studied example of such a factory structure is the transfer line, which involves a sequence of machines. Optimizing transfer lines has been a subject of much study in the industrial engineering and operations research fields. A desirable goal of a *lean* manufacturing system is to maximize demand, while keeping inventory levels of unfinished product as low as possible. This problem is intractable since the number of states is usually very large, and the underlying models are stochastic. In this paper we present an artificial intelligence approach to optimization based on a simulation-based dynamic programming method called reinforcement learning. We describe a reinforcement learning algorithm called SMART, and compare its performance on optimizing manufacturing systems with that of standard heuristics used in industry.

Introduction

Industrial manufacturing usually involves making parts with an assembly of flexible machines. The machines are programmable in some way in that their operations can be selected from a repertoire of basic operations, including doing one of several operations on a part, or doing maintenance. Optimizing manufacturing requires making parts with the lowest cost, which is usually a function of the number of parts stored in inventory (not yet finished), maintenance and failure costs of the machines involved etc. Although there are well-known stochastic models for optimization of such machine assemblies, these models are intractable to solve for large numbers of machines (usually three or more) (Gershwin 1994).

An alternative approach to optimization is through the use of simulation models, which are a time-honored approach to modeling complex systems (Law & Kelton 1991). Many software tools are available to simulate a wide range of systems including manufacturing, process control systems, and robotic control. These tools, however, rely on a human decision-maker to supply a fixed procedure or policy, and only provide a statistical profile on the quality of the policy. In this paper, we describe a new approach to automatically find good policies to intelligently control a complex simulation model. Our approach is based on a machine learning framework for autonomous agents called *reinforcement learning* (RL) (Mahadevan & Kaelbling 1996; Kaelbling, Littman, & Moore 1996; Sutton & Barto 1998). This framework models the sequential decision making problem faced by an agent (i.e., what action should the agent do in a particular state) as a Markov Decision Process (MDP) (Puterman 1994). The aim of the agent is to learn a policy (mapping states to actions) that maximizes its performance on the task.

Reinforcement learning is an ideal approach to optimize simulation models, since these generate sample trajectories through the state space as the agent experiments with its current policy. Classical optimization methods, such as dynamic programming (Bertsekas 1995), cannot be applied here because they require a *transition* model to predict the next state distribution, given the current state and action. Simulation models can generate sample next states, but cannot directly provide the information needed by dynamic programming.

In this paper, we are interested in studying a broad class of optimization problems in the general area of manufacturing, such as flexible product manufacturing, product scheduling, maintenance, inventory control, and transfer line optimization. In particular, we will focus on the problem of optimizing a transfer line (as shown in Figure 1). Transfer line optimization is a well-studied problem (Gershwin 1994), but the analytical models are intractable to analyze for lines with more than 2-3 machines. Transfer lines with 20 machines are standard in industry, but generate state spaces with 10^{20} states or more, which seems out of the realm of possibility for optimization using analytical models (as noted by (Gershwin 1994)). We will present empirical results comparing the performance of an average reward reinforcement learning algorithm called SMART (Mahadevan *et al.* 1997) on optimizing such a transfer line, and compare it with a well-known heuristic from the literature called Kanban, originally developed by Toyota Motor company (Shingo 1989; Bonvik & Gershwin 1996).

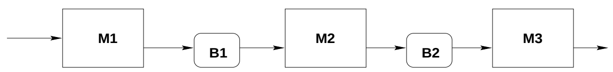

Figure 1: A transfer line models an assembly line operation in many factories, where a sequence of machines (M) make parts, that are isolated by product buffers (B). The machines are unreliable, and may fail occasionally. The problem is to optimize the throughput of the transfer line, while minimizing the in-product inventory, and minimizing failures.

The approach proposed in this paper can be applied to many other problems, including flexible manufacturing, product switching, ATM optimization, admission control in queues, and so on.

Discrete-Event Sequential Decision-Making Model

Decision-making in many domains can be abstractly viewed as follows. At each step, the agent perceives (perhaps imperfectly) the underlying environment as being in one of a (possibly very large, but finite) set of states. The agent can choose one of a set of finite actions in a given state, and carry it out. The action modifies the environment in some way (or transports the agent around), thereby modifying the perceived state into a new state. Much recent work in AI on autonomous agents, ranging from reinforcement learning (Sutton & Barto 1998), to robotics (Simmons & Koenig 1995), has adopted this framework.

For the purpose of optimizing simulation, we need to extend the discrete-time framework to a discrete-event framework, thereby generalizing it in two ways. Time is explicitly modeled as a continuous variable, but the agent observes the environment and makes decisions only at certain discrete points (or decision epochs). In between these epochs, the state of the system can be changing in some complex way, but these changes may not provide the agent with any additional information. Furthermore, actions take non-constant time periods, modeled by some arbitrary time distribution. It is not possible to model the optimization of factory processes as a discrete-time MDP, without drastic loss of information (for example, Poisson demand processes or failure distributions require using real-valued stochastic time distributions). This is well known in the simulation community, since discrete-event models have been used as the basis for simulation studies in a multitude of domains, ranging from manufacturing, queueing, networking, and transportation (Law & Kelton 1991).

Formally, the evolution of the environment at decision epochs can be modeled as a semi-Markov decision process (SMDP) (Puterman 1994). It is termed a semi-Markov model, because the transition depends not only on the current state and action, but also on how long the system has been in the current state. An SMDP is defined as a five tuple (S, A, P, R, F), where S is a finite set of states, A is a set of actions, P is a set of state and action dependent transition probabilities, R is a reward function, and F specifies the probability distribution of transition times for each state-action pair. $P(y \mid x, a)$ denotes the probability that action a will cause the system to transition from state $x \in S$ to $y \in S$. This transition probability function describes the transitions at decision epochs only.

The reward function for SMDP's is more complex than in the MDP model. In addition to the fixed reward $k(x, a)$, accrued due to an action taken at a decision epoch, an additional reward may be accumulated at *rate* $c(y, x, a)$ for the time the natural process remains in state y between the decision epochs. Note that the natural process may change state several times between decision epochs, and therefore, the rate at which the rewards are accumulated between decision epochs may vary. For example, in our experiments, the average size of the inventory level is translated into a rate cost that is computed as the area under the buffer size curve. We will describe the cost model in more detail below.

Reinforcement learning infers the optimal policy indirectly by inferring instead a *value function* (mapping states or state-action pairs to real numbers). Policies determine a value function based on an optimality metric, which is usually either a discounted model, or an average-reward model. We will primarily use the average-reward model in our work.

Value Functions for Average Reward SMDPs

The expected reward between two decision epochs, given that the system in state x, and a is chosen at the first decision epoch, may be expressed as

$$r(x, a) = k(x, a) + E_x^a \left\{ \int_0^\tau c(W_t, x, a) dt \right\} \quad (1)$$

where τ is the transition time to the second decision epoch, and W_t denotes the state of the natural process. Now, starting from state x at time 0, and using a policy π for n decision epochs until time t, the expected total reward can be expressed as

$$V_t^\pi(x) = E_x^\pi \left\{ \int_0^t c(W_u, x_{n_u}, a_{n_u}) du + \sum_{q=0}^{n_t - 1} k(x_n, a_n) \right\} \tag{2}$$

where $k(x_n, a_n)$ is the fixed reward earned at the nth decision epoch, and $c(y, x_n, a_n)$ is the rate at which reward is accumulated from the nth to the $(n+1)$th decision epoch. The average reward ρ^π for a policy π can be defined by taking the limit inferior of the ratio of the expected total reward up until the nth decision epoch to the expected total time until the nth epoch. So the average reward of a policy $\rho^\pi(x)$ can be expressed as the ratio

$$\lim_{n \to \infty} \frac{E_x^\pi \left\{ \sum_{i=0}^n \left[k(x_i, a_i) + \int_{\sigma_i}^{\sigma_{i+1}} c(W_t, x_i, a_i) dt \right] \right\}}{E_a^\pi \left\{ \sum_{i=0}^n \tau_i \right\}} \tag{3}$$

For each transition, the expected transition time is defined as:

$$y(x, a) = E_s^a \tau = \int_0^\infty t \sum_{z \in S} Q(dt, z \mid x, a) \tag{4}$$

for each $x \in S$, and $a \in A$.

The Bellman optimality equation for *unichain* average reward SMDP's is analogous to that for the discrete-time model, and can be written as

$$V^*(x) = \max_a \left(r(x, a) - \rho^* y(x, a) + \sum_{z \in S} P_{xz}(a) V^*(z) \right) \tag{5}$$

Here, V^* is the optimal value function and ρ^* is the optimal average reward. Note that we are assuming unichain SMDP's, where the average reward is constant across states. Many real-world SMDP's, including the elevator task (Crites & Barto 1996) and the production inventory task and transfer line problems described below, are unichain.

SMART: A Simulation-based Average-Reward Reinforcement Learning Algorithm

We now describe an average-reward algorithm called SMART (for Semi-Markov Average Reward Technique), which was originally described in (Mahadevan *et al.* 1997). This algorithm is based on representing the value function using action values $R(x, a)$, which is the utility of doing action a in state x. These action values can be learned by running a simulation model of the manufacturing domain, and using a feedforward neural net to approximate the action values.

The SMART algorithm can be derived straightforwardly from the Bellman equation for SMDP's (Equation 5). First, we reformulate Equation 5 using the concept of action-values. The average-adjusted sum of rewards received for the non-stationary policy of doing action a once, and then subsequently following a stationary policy π can be defined as

$$R^\pi(x, a) = r(x, a) - \rho^\pi y(x, a) + \sum_{z \in S} P_{xz}(a) \max_b R^\pi(z, b) \tag{6}$$

The temporal difference between the action-values of the current state and the actual next state is used to update the action values. In this case, the expected transition time is replaced by the actual transition time, and the expected reward is replaced by the actual immediate reward. Therefore, the action values are updated as follows:[1]

$$R(x, a) \stackrel{\alpha_n}{\leftarrow} \left(r_{imm}(x, a) - \rho\tau + \max_b R(z, b) \right) \tag{7}$$

where $r_{imm}(x, a)$ is the actual cumulative reward earned between decision epochs due to taking action a in state x, z is the successor state, ρ is the average reward, and α_n is a learning rate parameter. Note that ρ is actually the reward rate, and it is estimated by taking the ratio of the total reward so far, to the total simulation time.

$$\rho = \frac{\sum_{i=0}^n r_{imm}(x_i, a_i)}{\sum_{i=0}^n \tau_i} \tag{8}$$

where $r_{imm}(x_n, a_n)$ is the total reward accumulated between the nth, and $(n+1)$th decision epochs, and τ_n is the corresponding transition time. Details of the algorithm are given in Figure 2. The learning rate α_n and the exploration rate p_n are both decayed slowly to 0 (we used a Moody-Darken search-then-converge procedure).

Value Function Approximation in SMDP's

In most interesting problems, such as the transfer line problem in Figure 1, the number of states is quite large, and rules out tabular representation of the action values. One standard approach, which we followed, is to use a feedforward net to represent the action value

[1] We use the notation $u \stackrel{\alpha}{\leftarrow} v$ as an abbreviation for the stochastic approximation update rule $u \leftarrow (1 - \alpha)u + \alpha v$.

1. Set decision epoch $n = 0$, and initialize action values $R_n(x, a) = 0$. Choose the current state x arbitrarily. Set the total reward c_n and total time t_n to 0.

2. While $n <$ MAX_STEPS do

   (a) With high probability p_n, choose an action a that maximizes $R_n(x, a)$, otherwise choose a random action.

   (b) Perform action a. Let the state at the next decision epoch be z, the transition time be τ, and r_{imm} be the cumulative reward earned in this epoch as a result of taking action a in state x.

   (c) Update $R_n(x, a)$ using :

$$R_{n+1}(x,a) \stackrel{\alpha_n}{\leftarrow} \left(r_{imm} - \rho_n \tau + \max_b R_n(z,b) \right)$$

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

   - Update total reward $c_n \longleftarrow c_n + r_{imm}$
   - Update total time $t_n \longleftarrow t_n + \tau$
   - Update average reward $\rho_n \longleftarrow \frac{c_n}{t_n}$

   (e) Set current state x to new state z, and $n \leftarrow n + 1$.

Figure 2: The SMART algorithm for unichain SMDP's.

function (Crites & Barto 1996). Equation 7 used in SMART is replaced by a step which involves updating the the weights of the net. So after each action choice, in step 2(c) of the algorithm, the weights of the corresponding action net is updated according to:

$$\Delta\phi = \alpha_n \epsilon(x, z, a, r_{imm}, \phi) \bigtriangledown_\phi R_n(x, a, \phi) \qquad (9)$$

where $\epsilon(x, z, a, r_{imm}, \phi)$ is the temporal difference error

$$\left[r_{imm}(x,a) - \rho_n \tau + \max_b R_n(z,b,\phi) - R_n(x,a,\phi) \right]$$

and ϕ is the vector of weights of the net, α_n is the learning rate, and $\bigtriangledown_\phi R_n(x, a, \phi)$ is the vector of partial derivatives of $R_n(x, a, \phi)$ with respect to each component of ϕ.

Experimental Results

Single machine optimization

We now present detailed results of the SMART algorithm for optimization of manufacturing processes. To set the context, we begin by reviewing our earlier work on optimizing single machines with SMART (Mahadevan *et al.* 1997). A single flexible manufacturing machine can make several parts, each of which go into a separate buffer. The transfer line generalizes this example to several interconnected machines. The system we consider consists of a machine capable of producing 5 different products. The main parameters for the system are estimated as follows:

- Demand arrival process for each product i is Poisson (γ_i)
- Production time for each product has a Gamma (d_i, λ_i) distribution
- Time between failures has a Gamma (k, μ) distribution
- Time required for maintenance has a Uniform (a, b) distribution
- Time for repair has a Gamma (r, δ) distribution

The reinforcement learning algorithm is implemented using CSIM, a commercially available discrete-event simulator (see Figure 3). An example result of using the reinforcement learning algorithm to optimize a single flexible manufacturing machine is shown in Figure 4. As the figure shows, the production/maintenance policy learned by reinforcement learning is much better than the ones derived from two heuristic procedures.

Transfer Line Optimization: Multiple machines

We now present results of optimizing a 3 machine transfer line, as shown in Figure 1, and compare it to the performance of a standard well-accepted heuristic called Kanban. The goal of optimizing a transfer line is to maximize satisfied demand (which means delivering finished product to customers, any time a such a demand occurs), while keeping the inventory levels as low as possible.

The Kanban heuristic: This heuristic was developed at the Toyota Motor company, which is usually credited as an originator of the concept of lean manufacturing (Shingo 1989). The basic idea behind the heuristic is illustrated in Figure 5. The heuristic uses the concept of cards (or Kanbans) that circulate between a buffer and the immediate upstream machine, and are a signal for a machine to stop or start production. Essentially, in this method a machine produces until its output buffer is full. If a unit of finished product is consumed by demand, the last machine receives a Kanban card authorizing it to produce an additional

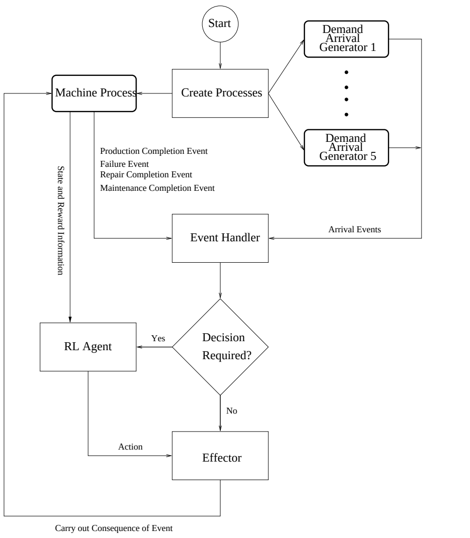

Figure 3: The SMART reinforcement learning algorithm is implemented using CSIM, a commercial discrete-event simulator. CSIM allows construction of simulation models of arbitrary size, since it is directly based on the C language.

part. This machine then picks up its raw materials from the buffer immediately behind it, which releases the Kanban card for that part. This way, information flows back upstream to all the machines for them to produce one additional part.

The following cost model for optimizing a transfer line was adopted. Repairs are modeled as incurring a cost of -1000. Each unit of satisfied demand is worth a positive reward of 10. In addition, there is a continual rate cost based on the average inventory levels, which is scaled by 0.1. In our experiments, each of the three machines receives the same reward.

Table 1 compares the performance of the SMART algorithm with the Kanban heuristic. The table shows the total inventory levels needed by SMART and Kanban for various target demand levels. In each case, the fill rate (percentage of demand satisfied) was about 98%, indicating that most of the demand was satisfied. As the table shows, SMART outperforms the Kanban heuristic in needing much fewer items in the inventory.

Target Demand	SMART	Kanban
0.2	106.7	135.27
0.5	100.69	117.3
0.6	85	101.15

Table 1: Comparison of Total Average Inventory Levels of SMART with Kanban heuristic.

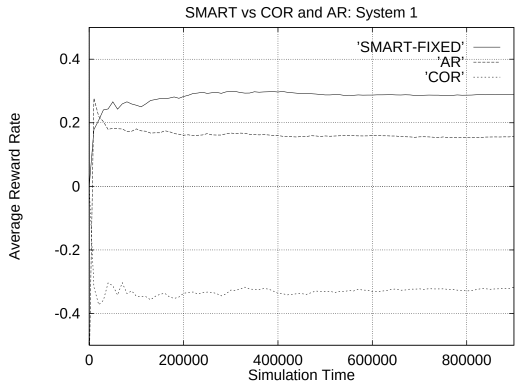

Figure 4: All curves show median-averaged costs of production of 5 parts using an unreliable machine. The top curve corresponds to the policy learned by the reinforcement learning algorithm. The bottom two correspond to fixed policies representing two heuristic methods.

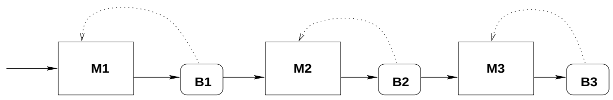

Figure 5: Diagram showing the Kanban heuristic policy for optimizing a transfer line.

Inventory levels are of course, only part of the overall improvement offered by SMART. In particular, because SMART actually learn a maintenance policy in addition to a production policy, the total number of failures is far fewer when using SMART. Table 2 compares the number of failures incurred under SMART and under Kanban. Note that since the Kanban heuristic does not incorporate any maintenance policy, it will obviously incur a far higher number of failures, as the table shows. The higher number of failures implies that the Kanban heuristic will result in a much lower average reward, as shown in Figure 6.

Target Demand	SMART	Kanban
0.2	843	2878
0.5	1880	6458
0.6	2992	8980

Table 2: Comparison of Total Number of Machine Failures for SMART with Kanban heuristic.

Limitations and Future Work

The results described in this paper should be viewed as preliminary, particularly those for optimizing a transfer line. We are currently running additional experi-

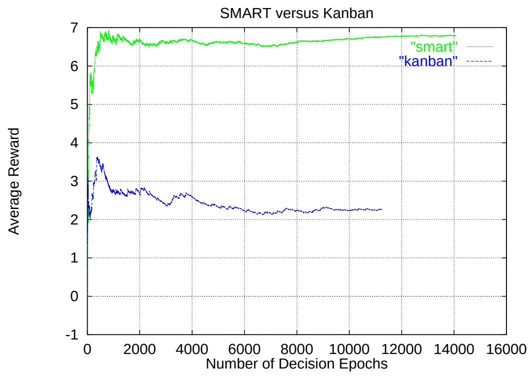

Figure 6: This figure compares the average reward of the policy learned by SMART with that produced by the Kanban heuristic.

ments, which we expect to report during the presentation of this paper. Here, we briefly discuss the nature of the additional experiments being planned.

There are several additional heuristics that have been developed, such as the CONWIP (constant work-in-process) strategy (Spearman, Woodruff, & Hopp 1990), as well as hybrid methods (Bonvik & Gershwin 1996) which combines a kanban with a CONWIP strategy. We plan to compare SMART to these other heuristics.

Another research direction is to investigate hierarchical reinforcement learning methods to scale to large transfer lines, as well as generalizations of transfer lines to job shops and flow shops. Hierarchical optimization of such assemblies has been theoretically investigated in depth (Sethi & Zhang 1994), but however, the computational study of such optimization methods is limited. There is currently much interest in the reinforcement learning literature on hierarchical methods (Parr & Russell 1998), but there has yet been no demonstrable results on interesting large scale problems. We believe the factory optimization domain to be an excellent testbed for hierarchical reinforcement learning algorithms. Finally, each machine is given the same global reward, and we plan to conduct comparative experiments where each machine is given different local rewards.

Acknowledgements

This research is supported in part by an NSF CAREER Award Grant No. IRI-9501852.

References

Bertsekas, D. 1995. *Dynamic Programming and Optimal Control*. Belmont, Massachusetts: Athena Scientific.

Bonvik, A., and Gershwin, S. 1996. Beyond kanban: Creating and analyzing lean shop floor control problems. In *Conference on Manufacturing and Service Operations Management*.

Boutilier, C.; Dearden, R.; and Goldszmidt, M. 1995. Exploiting structure in policy construction. In *Proceedings of the Fourteenth IJCAI*.

Crites, R., and Barto, A. 1996. Improving elevator performance using reinforcement learning. In *Neural Information Processing Systems (NIPS)*.

Gershwin, S. 1994. *Manufacturing Systems Engineering*. Prentice Hall.

Kaelbling, L.; Littman, M.; and Moore, A. 1996. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research* 4.

Law, A., and Kelton, W. 1991. *Simulation Modeling and Analysis*. New York, USA: McGraw-Hill.

Mahadevan, S., and Kaelbling, L. P. 1996. The NSF workshop on reinforcement learning: Summary and observations. *AI Magazine* 1(12):1–37.

Mahadevan, S.; Marchalleck, N.; Das, T.; and Gosavi, A. 1997. Self-improving factory simulation using continuous-time average reward reinforcement learning. In *Proceedings of the Fourteenth International Machine Learning Conference*. Morgan Kaufmann. 202–210.

Parr, R., and Russell, S. 1998. Reinforcement learning with hierarchies of machines. In *Advances in Neural Information Processing Systems (NIPS)*.

Puterman, M. L. 1994. *Markov Decision Processes*. New York, USA: Wiley Interscience.

Sethi, S., and Zhang, Q. 1994. *Hierarchical Decision Making in Stochastic Manufacturing Systems*. Birkhauser.

Shingo, S. 1989. *A Study of the Toyota Production System from an Industrial Engineering Viewpoint*. Productivity Press.

Simmons, R., and Koenig, S. 1995. Probabilistic robot navigation in partially observable environments. In *Proceedings of the IJCAI*, 1080–1087.

Spearman, M.; Woodruff, D.; and Hopp, W. 1990. CONWIP: An alternative to kanban. *International Journal of Production Research* 28(5):879–894.

Sutton, R., and Barto, A. 1998. *Reinforcement Learning: An Introduction*. MIT Press.