Value Function Based Production Scheduling

Jeff G. Schneider[1] and Justin A. Boyan and Andrew W. Moore[1]
Robotics Institute and Computer Science Department
Carnegie Mellon University
Pittsburgh, PA 15213
{schneide,jab,awm}@cs.cmu.edu

Abstract

Production scheduling, the problem of sequentially configuring a factory to meet forecasted demands, is a critical problem throughout the manufacturing industry. The requirement of maintaining product inventories in the face of unpredictable demand and stochastic factory output makes standard scheduling models, such as job-shop, inadequate. Currently applied algorithms, such as simulated annealing and constraint propagation, must employ ad-hoc methods such as frequent replanning to cope with uncertainty.

In this paper, we describe a Markov Decision Process (MDP) formulation of production scheduling which captures stochasticity in both production and demands. The solution to this MDP is a value function which can be used to generate optimal scheduling decisions online. A simple example illustrates the theoretical superiority of this approach over replanning-based methods. We then describe an industrial application and two reinforcement learning methods for generating an approximate value function on this domain. Our results demonstrate that in both deterministic and noisy scenarios, value function approximation is an effective technique.

Keywords: Reinforcement learning, Applications, Production scheduling, Optimization, Memory-based learning.

[1]Also at Schenley Park Research, Inc.

1 Introduction

Production scheduling is a critical problem throughout the manufacturing industry. In this paper, we argue that in order to deal with uncertainty in factory production and demands, a Markov Decision Process (MDP) formulation is superior to the approaches currently in use. Our paper is organized as follows:

- Section 2 describes the abstract task of production scheduling and the sources of uncertainty which make the task difficult for current approaches. It also gives details of the particular scheduling instance we have worked on in collaboration with a major U.S. food manufacturer.

- Section 3 introduces the MDP model of the scheduling task and its solution based on value functions. A simple example illustrates that in the presence of uncertainty, the MDP model produces the optimal solution where both open-loop and closed-loop planners do not. We then discuss two reinforcement learning algorithms, Memory-Based RTDP and ROUT, which are applicable for solving large-scale MDPs by value function approximation.

- Section 4 presents experimental results with ROUT and RTDP on two somewhat simplified versions of the real-world manufacturing task. The results compare favorably to greedy and simulated annealing algorithms in both noisy and (surprisingly) deterministic scheduling scenarios.

- Finally, Section 5 discusses our results, related work, and promising future directions.

2 Production Scheduling

2.1 Problem Specification

Production scheduling is the problem of deciding how to configure a factory sequentially to meet demands. We restrict our attention here to a type of production scheduling called "make to stock." We assume we have a modest number of products (2–100) and must produce enough of each to keep warehouse stocks high enough to satisfy customer requests for bulk shipments. This production model is common, for example, for most goods

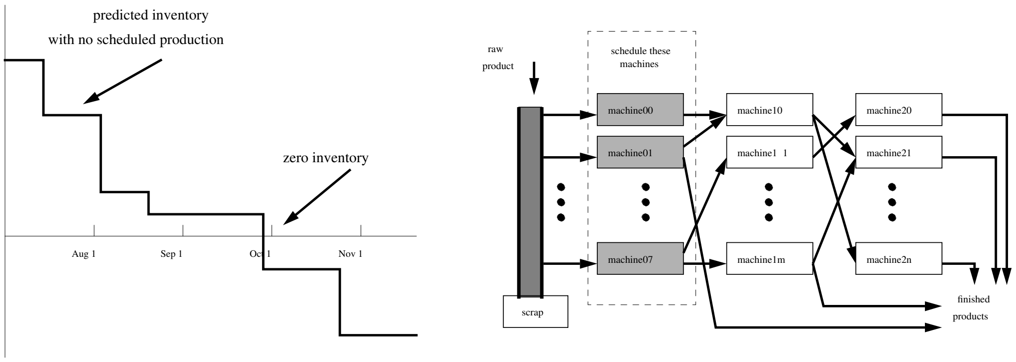

Figure 1: Demand inventory curves (left) and factory layout (right). See text for further explanation.

found in a supermarket. Automobile production, by contrast, is typically not scheduled under this model since cars are assembled individually with different options depending on specific customer orders.

An instance of the production scheduling problem is composed of five parts:

Machines and products. This is a list of what machines are present in the factory, and what products can be made on the machines. There may be complex constraints such as "machine A can only make product 1 when machine B is not making product 3." A complete, legal assignment of products onto the set of machines is called a *configuration*. There is also a special "closed" configuration which represents a decision to shut the factory down.

Changeover times. It generally takes a certain amount of time to switch the factory from one configuration to another. During that time, there is no production. The problem definition includes a (possibly stochastic) estimate of how long it takes to change each configuration to each other configuration.

Production rates. Each configuration produces a set of products at a certain rate. There may be dependencies between the machines. For example, machine B may produce product 2 faster if machine A is also producing product 2. The actual production rates in the factory may be very stochastic; for example, some machines may jam frequently, causing irregular delays on the production line.

Inventory demand curves. At the time a schedule is created, a demand curve for each product is available from a corporate marketing and forecasting group. As shown in Fig. 1, each curve starts at the left with the current inventory of that product. The inventory decreases over time as future product shipments are made and eventually goes below zero if no new production occurs. To avoid penalties, the scheduler should call for more production before the demand curve falls below zero. These curves may also change over time as new information about future product demand becomes available.

Schedule costs. Running a schedule generates a dollar measure of net profit or loss. This includes the costs of running the factory, paying the workers, purchasing the raw materials, and carrying inventory at the warehouse. It also includes heuristic costs such as an estimate of the damage done by failing to fill a customer request when the warehouse inventory goes to zero. Finally, it includes the revenue generated from selling product to a customer.

Given this problem description, the task of production scheduling is to maximize expected profit by selecting factory configurations over a period of time. In cases where the production rates and demand curves are assumed deterministic, the problem reduces to finding the optimal *open-loop* schedule: that is, find a fixed sequence of configurations that maximizes profit. In the general stochastic case, the optimal choice of configuration at time t will depend on the outcomes of earlier configurations, so the optimal solution has the form of a *closed-loop* scheduling policy.

2.2 A Real Production Scheduling Problem

We have devoted considerable effort to optimizing the production scheduling of a particular U.S. factory. The physical layout of one production line in the factory is shown in Fig. 1. Raw materials enter the factory and are processed using a (proprietary) system that creates up to twelve output streams of finished products simultaneously. Depending on how numerous machines and links between machines are configured, the rate of production of each of the twelve kinds of products varies. Production costs (caused by fuel uses, personnel costs, and wasted material) also vary according to the factory configuration.

Taking into account all the constraints between machines in the factory, there are about 100,000 different possible configurations. Factories of this type typically produce in the order of \$50 million up to \$2 billion worth of product annually, so the opportunities for cost savings via improved scheduling are large.

2.3 Conventional Solution Methods

Production scheduling is difficult to model within the standard job-shop scheduling paradigm. In job-shop scheduling, the problem is to complete a batch of atomic jobs under ordering constraints and constraints on which machines can handle which jobs, and at what speeds and costs. This model cannot readily be adapted to handle production rate interdependencies among machines, the desire to keep inventory levels above zero at all times (rather than just completing jobs by their deadlines), and stochasticity of demand forecasts and production.

Constraint propagation methods (e.g. [14]) are commonly used to solve industrial problems. They operate by efficiently managing constraints on production deadlines and machine capabilities. Solution methods tend to search by iteratively fixing violated constraints, and use heuristics to guide the fixes. Constraint propagation focuses primarily on generating *feasible* schedules, and only secondarily on *cost optimality*. This is appropriate when feasibility is difficult, but not as good in "make to stock" scenarios where feasibility is easy and cost reduction is the main goal. Constraint propagation will not receive further consideration here for that reason.

When cost optimality is the primary scheduling objective, global optimization techniques such as simulated annealing (SA) are a good option. These methods search a space of fully-instantiated schedules to find the best ones. However, neither constraint propagation nor simulated annealing is naturally formulated to handle stochastic problems. They can be modified for nondeterminism in two ways:

Optimization open-loop: Search for the fixed schedule s which maximizes the *average* profit over several independent stochastic simulations of s. Here, all the computation is spent at the beginning, and the resulting best schedule is executed without observing actual production statistics along the way. This algorithm suffers because it cannot update the schedule to account for variances in actual production. To compensate for this inadequacy, "replanning" methods are usually adopted.

Replanning closed-loop: When possible, this method starts with the open-loop stochastic evaluation from option 1. For feasibility-based methods it must start with a deterministic version of the problem. In either case, it uses its first schedule only to make some initial scheduling decisions. Then, whenever the output of a stochastic choice is observed, it replans the remainder of the schedule in order to make new decisions.

The closed-loop method can produce good results. However, it is computationally quite expensive. Moreover, although it replans on every step, its policy does not take advantage of the fact that it will be able to replan in the future—and as we show in Section 3.2 below, this dooms it to being unable to attain the optimal profit, no matter how much computation time it is allowed.

3 Production Scheduling with Value Functions

This section describes a principled approach to generating closed-loop production scheduling policies with reinforcement learning methods. The approach is based on representing the problem as an MDP and representing the solution as an approximate value function.

3.1 Production Scheduling as an MDP

Abstractly, a Markov Decision Process (MDP) is defined by a state space X, action set A, immediate reward function $R(x,a)$, and probabilistic transition model $P(x'|x,a)$. The solution to the MDP is a policy $\pi^* : X \to A$ which, if followed by the agent, will maximize the expected long-term sum of rewards attainable starting from any state x. Dynamic programming methods tabulate this optimal cumulative reward in the *value function* $V^*(x)$, which is the unique solution to the *Bellman equations* [3]:

$$V^*(x) = \max_{a \in A} \left(R(x,a) + \sum_{x' \in X} P(x'|x,a) V^*(x') \right) \tag{1}$$

Once V^* is computed, the optimal policy π^* is immediately obtained by choosing any action which instantiates the max in Eq. 1.

The production scheduling problem is modelled very naturally as a Markov Decision Process, as follows:

- The system state is defined by the current time $t \in 0 \ldots T$; the current inventory of each product $p_1 \ldots p_N$; and, if there are configuration-dependent changeover times, the current factory configuration.

- The action set consists of all legal factory configurations. We assume a discrete-time model, so the configuration chosen at time t will run unchanged until time $t+1$.

- The stochastic transition function applies a simulation of the factory to compute the change in all inventory levels realized by running configuration c_t for 1 timestep. This model handles random variations in production rates straightforwardly; it also handles changeover times by simply decreasing production in proportion to the (possibly stochastic) downtime. The time t is incremented on each step, and the process terminates when $t = T$.

- The immediate reward function is computed from the inventory levels, based on the demand curve at time t. It incorporates the revenues from production, penalties from late production, employee costs, operating costs and changeover cost incurred during the period. On the final time period (transition from $t = T - 1$ to T), a terminal "reward" assigns additional penalties for any outstanding unsatisfied demands.

The MDP representation suits this problem very well, for two main reasons. First, in contrast to other trajectory optimization tasks (e.g., the Travelling Salesman Problem), the utility of future decisions does not depend on the entire sequence of previous action choices, but only on a quite compact state description—the current time and inventory levels. Simulated annealing and other global optimization methods do not require this Markov property—nor can they exploit it. Second, the model fully represents uncertainty in production rates and changeover times. As defined here, the model also handles noise in the demands if that noise is time-independent, but it cannot account for the possibility of the demand curves being randomly updated in the middle a schedule, since that would make the MDP transition probabilities nonstationary.

The value function for this MDP specifies a closed-loop scheduling policy which makes optimal decisions with full foresight of the remaining uncertainty in the process. No method based on global optimization can make this claim, even if replanning is allowed, as we now illustrate.

3.2 Illustrative Example

This example illustrates how MDP solutions optimally solve sequential decision problems that methods based on replanning cannot. Suppose we are asked to schedule the production of 12 units of a single product over two days. On each day we can choose one of the following three configurations:

Configuration	with probability	gets production	Cost
1	0.5	3	1
	0.5	6	
2	1.0	6	4
3	1.0	9	8

In addition to the per-configuration costs listed in the table, there is an additional cost of 8 for each unit not produced by the end of two days. The following table shows the expected cost of each of the possible schedules. (Note that in this example, the expected cost of a schedule $[ab]$ is the same when the sequence is reversed, so redundant schedules are omitted from the table.)

Config Sequence	Config Cost	Expected Missed Production Cost	Total Expected Cost
1 1	2	0.25*48 + 0.5*24	26
1 2	5	0.5*24	17
1 3	9	0	9
2 2	8	0	**8**
2 3	12	0	12
3 3	16	0	16

Based on these costs, a replanning-based scheduler will choose sequence [2 2]. It will execute configuration 2 on the first day, and then have an opportunity to replan for day 2 based on the results of day 1. Since the production from configuration 2 on day 1 is deterministic (6 units), the scheduler will again choose configuration 2 on day 2, thereby completing the 2-day production run with a total cost of 8.

The replanning-based scheduler makes a suboptimal decision on day 1 because it doesn't "know" that it will be given the chance to replan after the first day's production is observed. By contrast, with the ability to exploit this knowledge, the MDP solution makes the correct scheduling decision of action 1 on day 1. The following table evaluates the choices for day 1 by showing all the possible outcomes followed by the optimal day 2 choice for each outcome.

Day 1 Config	with prob	units produced	Day 2 Config	with prob	units produced	Expected Cost
1	0.5	3	**3**	1.0	9	0.5*9 + 0.5*5
	0.5	6	**2**	1.0	6	= **7**
2	1.0	6	2	1.0	6	= 8
3	1.0	9	1	0.5	3	0.5*9 + 0.5*9
				0.5	6	= 9

By considering all the possible outcomes and the optimal decisions that will be made for each one, the MDP solution chooses configuration 1 on the first day and achieves an expected cost of 7 as compared to 8 obtained by replanning. This type of tradeoff exists in real factories as well. There is often a choice of how fast to run the production line that trades off higher production rates against higher unit costs.

3.3 Value Function Approximation

In practical scheduling problems, tabulating $V^*(x)$ for every possible state of the factory is completely intractable. Instead, we use reinforcement learning methods to compactly represent V^* with a function approximator, such as global or local polynomial regression. The two methods we tested are Memory-Based RTDP and ROUT.

3.3.1 Memory-Based RTDP

Memory-Based RTDP is a reinforcement learning approach that is closely related to RTDP (Real-Time Dynamic Programming) [2] and to Tesauro's application of TD(0) to the game of backgammon [10, 11]. It is also similar to the instance based approach to representing value functions is done in [9]. Trajectories through the MDP model are generated repeatedly, using the current approximation of the value function to guide Boltzmann-style exploration. At each step of each trajectory, a one-step backup operation (Eq. 1) is performed and the function approximator is updated.

In Memory-Based RTDP, the value function is represented by a nonparametric memory-based function approximator [5, 7, 1]. Memory-based learning simply accumulates training data points, rather than running a training algorithm on them. Then whenever a query is made, the approximator's output is computed by an average or regression over nearby points in memory.

Achieving good performance with Memory-Based RTDP requires an appropriate choice of the Boltzman exploration temperature and the local regression kernel width. These values were tuned empirically to obtain the results presented in Section 4. Although the training points generated by Memory-Based RTDP's early trajectories are sure to be inaccurate samples of V^*, we did not find it necessary to include an explicit "forgetting" mechanism in the learning.

3.3.2 ROUT

ROUT is an active learning algorithm for value function approximation that is specifically designed for the subclass of acyclic MDPs [4]. (The scheduling MDP is certainly acyclic, since the time counter t is part of its state representation.) Using simulations of the process, ROUT repeatedly identifies a state x at which (1) the function approximator is currently in error, and (2) an accurate sample of V^* can be obtained from a 1-step backup. Unlike Memory-Based RTDP and most other reinforcement learning methods, ROUT explicitly tries to prevent the function approximator from seeing *any* inaccurate samples of V^*.

Details of how ROUT identifies such states automatically are given in [4]. One by one, these useful states are accumulated into a training set of accurate samples of $V^*(x)$. The training set grows backwards from the terminal states. As soon as the start state x_0 is itself added to the training set, ROUT declares victory, outputs its learned training set and learned approximation of V^*, and terminates.

If the function approximator cannot represent V^* accurately, then ROUT may become stuck, repeatedly adding points near the terminal states and never progressing backwards. However, if the function approximator can represent V^* to within the specified tolerance, then unlike Memory-Based RTDP, ROUT can be guaranteed to eventually find it. For ROUT to find V^* efficiently, the function approximator must extrapolate well from a small training set.

4 Experimental Results

We have experimented with two instances of the real-world production scheduling task described in Section 2.2. The first instance is heavily simplified so that the exact optimal closed-loop scheduling policy can be calculated tractably. The second instance is a more realistic model, for which only heuristic solutions are available.

4.1 Simplified Scheduling Instance

In the simplified instance, the task is to schedule 8 weeks of production; however, configurations may be changed only at 2-week intervals, and only 17 configuration choices are available. Of these 17, nine have deterministic production rates; the other eight each have two stochastic outcomes, producing only 1/3 of their usual amount with probability 0.5. With a total of $9\times 1+8\times 2 = 25$ outcomes possible from every state, there are $25^4 = 390,625$ possible trajectories through the space. The optimal policy can be computed by tabulating $V^*(x)$ at every possible intermediate state x of the factory, of which there are $1 + 25 + 25^2 + 25^3 = 16,276$. The optimal policy results in an expected cumulative reward of $-$\$22.8M. By contrast, a random schedule attains a reward of $-923M$ (!). A greedy policy, which at each step selects a configuration to maximize only the one-step reward from the current state, attains $-$\$97.9M.

We applied ROUT to this instance, trying three different function approximators: 1-nearest neighbor, locally-weighted linear regression, and global quadratic regression. ROUT's exploration and tolerance parameters were tuned manually. Table 4.1 summarizes the results.

When nearest-neighbor was used as the function approximator, ROUT did not obtain sufficient generalization from its training set and failed to terminate within a limit of several hours. However, with both local linear and global quadratic regression models, ROUT did run to completion and

Algorithm	Mean Profit	95% C.I.	optimal runs
Optimal	-22.8		1
Random	-923.2	$\pm$58.7	0
Greedy	-97.9	$\pm$15.1	0
ROUT + global quadratic	-54.5	$\pm$28.1	8/12
ROUT + local linear	-48.3	$\pm$23.4	4/8

Table 1: Results for 4-timestep, 17-configuration stochastic scheduling problem.

produced an approximate value function which significantly outperformed the greedy policy. Moreover, over half of the ROUT runs did indeed terminate with the *optimal* closed-loop scheduling policy. In these cases, ROUT's final self-built training set for value function approximation consisted of only about 110 training points—a substantial reduction over the $16,276$ required for full tabulation of V^*. ROUT's total running time (≈ 1 hour on a 200 MHz Pentium Pro) was roughly half of that required to enumerate V^* manually.

From these preliminary results, we conclude that ROUT does indeed have the potential to approximate V^* extremely well, given a suitable function approximator for the domain. However, since it runs quite slowly on even this simple problem, we believe ROUT will not scale up to practical scheduling instances without further modification.

4.2 Practical Scheduling Instance

In this section we present experimental results on a larger scheduling problem. In doing so, we lose the ability to determine the optimal policy for comparison. However, it gives a better demonstration of how the competing methods perform on industrial scale scheduling problems. The task is to schedule eight weeks of production at one week intervals. There are eight products, eight machines, and a total of 421 legal configurations to consider including the "closed" configuration. Our experiments consider both deterministic and noisy variants of the problem. Noisy outcomes are simulated by adding 10% Gaussian noise to the production of each product by each configuration on each timestep.

Table 4.2 shows experimental results. The computation times reported are on a 200 MHz Pentium Pro. The first section contains results for the case where the factory output is deterministic and known. The purpose of the first two lines is to delimit the range of results we should expect from good algorithms. The "Random" algorithm builds a schedule by choosing 8 configurations at random and loses an enormous amount of money. Much of the cost is due to heuristic penalties for failing to satisfy customer demand.

The "PlanIt" algorithm, developed by Schenley Park Research, is the proprietary algorithm currently used to schedule the real factory's production. It has several advantages over the other algorithms in this table. First, it is finely tuned to schedule this factory using a combination of simulated annealing, linear programming, constraint propagation, and several heuristics. Second, it is not restricted to choosing configurations for pre-discretized time steps, but can choose an arbitrary number of configurations and switch between them at arbitrary times. Our experience with this scheduler leads us to believe that the average profit of 13.81 million is very near optimal (and unachievable when restricted to 8 fixed-size time steps) for this instance, so this can be considered an unattainable upper bound for the other results. It usually incurs no heuristic penalties in its schedules, so that figure is a profit in real dollars.

Simulated annealing, greedy exploration, and Memory-Based RTDP are run as described in the previous sections. The simulated annealing runs made use of the successful "modified Lam" adaptive annealing schedule [8]. The poor result from greedy shows that generating trajectories based solely on the one-step cost of configurations is not an effective way to search even when compared to a random search method such as simulated annealing. The search efficiency gained by computing a value function is shown by the favorable Memory-Based RTDP results. All of the algorithms can do better with more computation time, but they were cut off at 10 minutes since PlanIt gets its results in that much time. For example, Memory-Based RTDP improves to 11.39 ± 0.20 if allowed 20 minutes of computation.

The second section of the table shows results with 10% noise added. The PlanIt algorithm cannot be run in this case since it does not handle stochastic outcomes; however, we still expect its result in the deterministic case to be a reasonable upper bound for the other algorithms.

Open-loop simulated annealing means that all the computation is spent at the beginning and the resulting best schedule is executed without observing actual production statistics along the way. This algorithm suffers because it cannot update the schedule to account for variances in actual production. By contrast, closed-loop simulated annealing replans the rest of the schedule

Algorithm	Mean Profit	95% C.I.
Deterministic $\approx$ 10 min computation		
Random	-466.35	$\pm$59.45
PlanIt	13.81	$\pm$0.08
Simulated Annealing	5.66	$\pm$3.68
Greedy + Exploration	-1.93	$\pm$3.21
Memory-Based RTDP	7.70	$\pm$1.57
10% Noise $\approx$ 45 min computation		
Simulated Annealing (o.l.)	-3.82	$\pm$6.10
Simulated Annealing (c.l.)	2.97	$\pm$4.96
Greedy (c.l.)	-17.69	$\pm$1.94
Memory-Based RTDP	**7.38**	**$\pm$0.86**

Table 2: Results for 8 time step, 421 configuration scheduling problem. The numbers shown represent profits in millions of dollars. On the noisy problem, Memory-Based RTDP is statistically better than the other algorithms at the 95% significance level.

after each week of actual production is observed. In order to keep the total computation the same, the computation allotted for each week's decision was divided by the number of weeks (8). The results show that replanning does improve over open loop execution. We note that all the simulated annealing schedulers have high variance, which can be a disadvantage of using that algorithm.

Memory-Based RTDP uses its computation at the beginning to compute a value function. The value function determines a closed-loop policy valid for any state reached during actual production. As discussed earlier, it not only executes closed-loop, but also makes its decisions "knowing" that it will be executing closed-loop. The results show both a favorable expected profit as well as smaller variance across runs.

5 Discussion and Future Work

We expect Memory-Based RTDP to outperform simulated annealing on a stochastic problem based on the intuition from Sec. 3.2, and our experimental results show that it does. It is interesting to observe that Memory-Based

RTDP does well against simulated annealing even in the deterministic case where the stochastic modeling capability of MDPs is not needed. This provides further evidence that search based on value functions can improve efficiency. While simulated annealing is forced to try configurations at random, value function based methods can explicitly reason about which intermediate states are good and which actions will reach those states.

To our knowledge, this work represents the first application of reinforcement learning to the production scheduling task. The scheduling of machine maintenance is discussed in [6]. There, reinforcement learning and queueing theory techniques are used to make binary decisions about taking machines down for maintenance. A reinforcement learning approach to the Space Shuttle scheduling problem is described by [13]. In that framework, states are complete schedules and actions are modification operators applied to the schedules. Their feature representation introduces noise, but the underlying problem is deterministic.

Our work to date covers stochasticity only in production. Another large source of uncertainty in real problems is the inadequacy of demand forecasts. This can be handled heuristically within the MDP formulation described here by the addition of appropriate noise to the demands during simulations. However, it may also be possible to gain extra efficiencies by incorporating demands more explicitly into the MDP state space. Further empirical work is required to answer that question.

As the size of the scheduling problem increases, it becomes increasingly expensive to compute the value function accurately. However, even an inexact value function can be useful as the basis for a quasi-greedy search or "rollout" search performed online [12]. We intend to test such methods in future work on larger scheduling problems.

References

[1] C. Atkeson, S. Schaal, and A. Moore. Locally weighted learning. *AI Review*, 1995.

[2] A. G. Barto, S. J. Bradtke, and S. P. Singh. Real-time learning and control using asynchronous dynamic programming. *AI Journal*, 1995.

[3] R. Bellman. *Dynamic Programming*. Princeton University Press, 1957.

[4] J. A. Boyan and A. W. Moore. Learning evaluation functions for large acyclic domains. In L. Saitta, editor, *Machine Learning: Proceedings of the Thirteenth International Conference*. Morgan Kaufmann, 1996.

[5] W. Cleveland and S. Delvin. Locally weighted regression: An approach to regression analysis by local fitting. *Journal of the American Statistical Association*, pages 596–610, September 1988.

[6] S. Mahadevan, N. Marchalleck, T. Das, and A. Gosavi. Self-Improving Factory Simulation using Continuous-Time Average-Reward Reinforcement Learning. In *Proceedings of the 14th International Conference on Machine Learning (IMLC '97), Nashville, TN*. Morgan Kaufmann, July 1997.

[7] A. Moore, C. Atkeson, and S. Schaal. Locally weighted learning for control. *AI Review*, 1995.

[8] E. Ochotta. *Synthesis of High-Performance Analog Cells in ASTRX/OBLX*. PhD thesis, Carnegie Mellon University Department of Electrical and Computer Engineering, April 1994.

[9] Jing Peng. Efficient Dynamic Programming-based Learning for Control. PhD. Thesis, Northeastern University, December 1993.

[10] R. S. Sutton. Learning to predict by the methods of temporal differences. *Machine Learning*, 3, 1988.

[11] G. Tesauro. Practical issues in temporal difference learning. *Machine Learning*, 8(3/4), May 1992.

[12] G. Tesauro and G. R. Galperin. On-line policy improvement using Monte-Carlo search. In M. Mozer, M. Jordan, and T. Petsche, editors, *NIPS-9*, 1997.

[13] W. Zhang and T. G. Dietterich. A reinforcement learning approach to job-shop scheduling. In *Proceedings of IJCAI-95*, pages 1114–1120, 1995.

[14] M. Zweben and M. Fox. *Intelligent Scheduling*. Morgan Kaufmann, 1994.