Proceedings of the 37th IEEE
Conference on Decision & Control
Tampa, Florida USA • December 1998

Stochastic Production Scheduling to meet Demand Forecasts

Jeff G. Schneider[1] Justin A. Boyan Andrew W. Moore[1]
The 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 requirements of maintaining product inventories in the face of unpredictable demand and stochastic factory output make the problem difficult. Existing approaches commonly fall into one of two groups: either demand forecasts are discarded and linearizing assumptions are made so methods based on optimal control can be applied, or AI search methods are used to tackle the large search spaces and the ability to handle stochasticity optimally is sacrificed.

In this paper, we describe a Markov Decision Process (MDP) formulation of production scheduling which captures stochasticity, while retaining the ability to construct a schedule to meet demand forecasts. The solution to this MDP is a value function, specific to the current demand forecasts, which can be used to generate optimal scheduling decisions online. We then describe an industrial application and a reinforcement learning method for generating an approximate value function in this domain. Our results demonstrate that in both deterministic and noisy scenarios, value function approximation is an effective technique.

1 Introduction

Production scheduling is a critical problem throughout the manufacturing industry. Often a choice is made between the ability to handle stochasticity, which is done well in an optimal control framework (e.g. [12]), and the ability to search large combinatorial spaces arising from demand forecasts or nonlinear production alternatives, which is done well in an AI search framework (e.g. [17]). In this paper we propose a Markov Decision Process (MDP) formulation of the problem that includes the strengths of each approach. Our paper is organized as follows:

- Section 2 describes the abstract task of production scheduling and the features that make the task difficult for various existing 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. We then discuss a reinforcement learning algorithm, Memory-based RTDP, which is applicable for solving large-scale MDPs by value function approximation.
- Section 4 presents experimental results with Memory-based RTDP on a slightly simplified version of the real-world manufacturing task. The results compare favorably to greedy and simulated annealing algorithms in both noisy and 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 "make to stock" model. We assume 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 most goods 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

[1] Also at Schenley Park Research, Inc.

0-7803-4394-8/98 $10.00 © 1998 IEEE

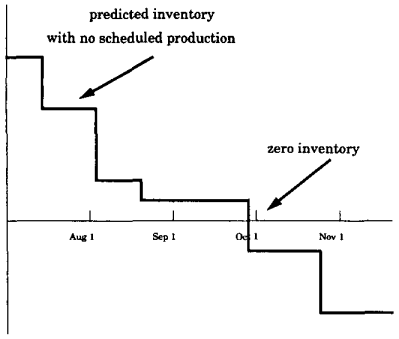

Figure 1: A demand curve for one product (see text for explanation)

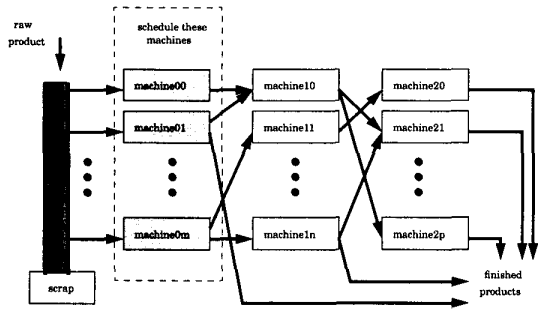

Figure 2: Factory layout (see text for explanation)

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, an estimated 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 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, which are all real dollar costs. 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. The final cost (or profit) of a schedule is the sum of all these real dollar costs, heuristic penalties, and revenue.

Given this problem description, the task of production scheduling is to maximize expected profit by selecting factory configurations (including choosing machine speeds) over a period of time. In cases where the production rates and demand curves are assumed to be deterministic, the problem reduces to finding the optimal *open-loop* schedule: that is, finding 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 outcome 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. 2. 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 and the chosen machine speeds, the rate of production of each of the twelve products varies. Production costs 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 on the order of $50 million to $2 billion worth of product annually, so the opportunities for cost savings via improved scheduling are large.

2.3 Conventional Solution Methods

General production scheduling is difficult to solve with standard models. Most approaches can be grouped in one of two categories centered around either optimal control or AI based search.

2.3.1 Optimal control approaches: Optimal control based methods (see [12] for example) are

appealing because they naturally deal with the problem of stochasticity and sometimes give performance guarantees or even achieve optimality. Unfortunately, these guarantees often come with assumptions that are difficult to meet in real world problems.

For example, a common model of demand is that the demand rate is generated by a Markov process. This assumption is excellent for handling stochasticity in demands, but ignores the fact that many manufacturing companies produce demand forecasts extending far into the future for use by their schedulers. These forecasts may not be accurate, but they allow for much more profitable production policies than are possible if the forecasts are ignored and only the current demand rate is known.

Another serious problem is the existence of complex constraints on what machines can produce what products and production rates that depend nonlinearly on control decisions. Incorporation of these characteristics often leads to an unmanagably large combinatorial set of controls to consider as well as nonlinear system equations that are difficult to solve. These difficulties are probably the reason that the scheduling software produced by successful companies such as i2, Manugistics and Peoplesoft is based on the methods described in the next section.

2.3.2 Artificial intelligence methods: AI methods (see [17] for example) are popular because they can naturally represent complex constraints and are targeted specifically to searching large spaces of discrete decisions. Unfortunately, stochasticity often must be ignored and handled with frequent "replanning." Guarantees of optimality are usually lost as well, and some methods drop the goal of optimization altogether and instead merely look for feasible solutions.

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 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, applying 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.

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. Possible modifications to improve this situation include:

- **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 method by itself is not a realistic choice 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 the previous option. 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 result of an action with a stochastic outcome 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 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 [11], 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

Here we describe a principled approach to generating closed-loop production scheduling policies with reinforcement learning methods. It combines the capabilities of both optimal control and AI search based methods. The approach is based on representing the problem as an MDP and representing the solution as an approximate value function. In contrast to many optimal control based methods, it produces a time-dependent policy specifically built to match current demand forecasts, rather than a time-invariant policy that ignores all demand information other than the current rate. Our experiments also demonstrate the ability to search hundreds of alternative factory configurations.

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 $\pi^* : X \rightarrow A$ 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 *optimal value function* $V^*(x)$, which is the unique solution to the *Bellman equations* [3]:

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

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 modeled 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 one 0 time step. This model handles random variations in production rates straightforwardly; it also handles changeover times by simply decreasing production in proportion to the (possibly stochastic) down time. 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, raw material 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 and outcomes, but only on a relatively 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 of 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 shown in [11].

3.2 Value Function Approximation

In practical scheduling problems, tabulating $V^*(x)$ for every possible state of the factory is intractable. Instead, we use reinforcement learning methods to represent V^* compactly with a function approximator, such as global or local polynomial regression. We tested a method called 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 [13, 14]. It is also similar to the instance-based approach to representing value functions used in [10]. Trajectories through the MDP model are generated repeatedly, using the current approximation of the value function to guide standard Boltzmann-style exploration [2]. 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 [4, 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 a weighted average or weighted polynomial regression over nearby points in memory.

Achieving good performance with Memory-based RTDP requires an appropriate choice of the Boltzmann 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 undoubtedly inaccurate samples of V^*, we did not find it necessary to include an explicit "forgetting" mechanism in the learning; the bad points are quickly overwhelmed by later, more accurate samples. A second algorithm, ROUT, which explicitly produces only accurate training samples of V^* is described in

Noise level	Algorithm	Profit	95% C.I.
No Noise	Random	-466.35	±59.45
(≈ 10 min)	PlanIt	13.81	±0.08
	Simulated Annealing	5.66	±3.68
	Greedy + Exploration	-1.93	±3.21
	Memory-based RTDP	7.70	±1.57
10% Noise	Greedy (c.l.)	-17.69	±1.94
(≈ 45 min)	Sim. Annealing (o.l.)	6.48	±1.21
	Sim. Annealing (c.l.)	9.03	±1.04
	Mem-based RTDP	**10.16**	**±0.84**
20% Noise	Greedy (c.l.)	-25.92	±1.12
(≈ 45 min)	Sim. Annealing (o.l.)	2.55	±1.91
	Sim. Annealing (c.l.)	2.40	±3.95
	Mem-based RTDP	**7.02**	**±0.67**

Table 1: Results for 8-timestep, 421-configuration scheduling problem. The numbers shown represent profits in millions of dollars. On the noisy problems, Memory-based RTDP is statistically better than the other algorithms at the 95% significance level.

[11]. It performed well on modest sized problems, but was difficult to scale up to the problems used in this paper's experiments.

4 Experimental Results

In this section we present experimental results on a large 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 versions of the problem. To build the deterministic version of the problem, we ran long (stochastic) simulations for each of the 421 actions and cached the mean observed production rate for each. For the noisy versions, we could have used noisy outcomes directly from the stochastic simulation, but instead we simply added Gaussian noise to the cached, deterministic production rates. This enabled our experiments to run significantly faster, and also allowed us to easily generate empirical results with varying amounts of noise.

Table 1 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 it 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 $13.81M is very near optimal for this instance, so it can be considered an unattainable upper bound for the other results. In particular, PlanIt achieves its results by using an average of about 13 configurations in its schedules while the other algorithms are restricted to 8 fixed-sized time steps. It usually incurs no heuristic penalties in its schedules, so that figure is a profit in real dollars.

The greedy algorithm considers only the one-step cost of each action. The simulated annealing runs made use of the successful "modified Lam" adaptive annealing schedule [9]. The Memory-based RTDP algorithm is run as described in the previous sections. It uses kernel regression with a kernel width of 2^{-5} of the range of each state variable, and uses KD-trees for efficiency [8]. Boltzmann exploration without cooling was used for the deterministic case (for both Greedy and Memory-based RTDP), but proved unnecessary in the stochastic case because the noise alone caused sufficient exploration.

The poor result from Greedy in the deterministic case 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 randomized search method such as simulated annealing. The search efficiency gained by computing a value function is shown by the favorable Memory-based RTDP results. They are obtained from only 200 trajectories through the state space, meaning the value function at each time step is represented with 200 training points. 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.

The second and third sections of the table show results with 10% and 20% noise added. The PlanIt algorithm cannot be run in these cases 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 com-

putation 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 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). As expected, replanning improves 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. Each run used 400 trajectories for these results. The value function determines a closed-loop policy valid for any state reached during actual production. 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 since replanning based algorithms are provably suboptimal 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 production scheduling including multiple products, multiple machines, demand forecasts, and stochasticity. The scheduling of machine maintenance is discussed in [5], and transfer line production scheduling is discussed in [6]. In their task, each product or sub-product is produced on a single machine and each machine makes a local decision on whether to produce one of its products or go down for maintenance. Demand is a random process that is not forecasted. A reinforcement learning approach to the Space Shuttle scheduling problem is described by [16]. 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 assumed deterministic.

Our empirical 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 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 [15]. 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. *Artificial Intelligence*, 1995.

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

[4] 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.

[5] 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.

[6] S. Mahadevan and G. Theocharous. Optimizing production manufacturing using reinforcement learning. In *Eleventh International FLAIRS Conference*, 1998.

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

[8] A. Moore, J. Schneider, and K. Deng. Efficient locally weighted polynomial regression predictions. In *International Conference on Machine Learning*, 1997.

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

[10] J. Peng. Efficient Dynamic Programming based Learning for Control. PhD. Thesis, Northeastern University, December 1993.

[11] J. Schneider, J. Boyan, and A. Moore. Value function based production scheduling. In *International Conference on Machine Learning*, 1998.

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

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

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

[15] G. Tesauro and G. R. Galperin. On-line policy improvement using Monte-Carlo search. In M. C. Mozer, M. I. Jordan, and T. Petsche, editors, *Advances in Neural Information Processing Systems*, volume 9. MIT Press, 1997.

[16] W. Zhang and T. G. Dietterich. A reinforcement learning approach to job-shop scheduling. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1114–1120, 1995.

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