A Greedy Divide-and-Conquer Approach to Optimizing Large Manufacturing Systems using Reinforcement Learning

Gang Wang

Dept. of Computer Science and Engg.
Michigan State University
East Lansing, MI 48823
wanggan1@cse.msu.edu

Sridhar Mahadevan

Dept. of Computer Science and Engg.
Michigan State University
East Lansing, MI 48823
mahadeva@cse.msu.edu

Abstract

Manufacturing is a challenging real-world domain for studying hierarchical MDP-based optimization algorithms. We have recently obtained very promising results using a hierarchical reinforcement learning based optimization algorithm for a 12-machine transfer line. Transfer lines model factory processes in automobile and many other product assembly plants. Unlike domains such as elevator scheduling, where the individual elevator "agents" are effectively independent, individual machines in a transfer line are directly affected by each others behavior. This interaction creates a highly non-stationary environment that defeats "flat" learning algorithms that train each machine while they are part of the line. Our hierarchical optimization algorithm comprises of an average-reward Q-learning algorithm for semi-Markov decision processes called SMART for learning low-cost policies for operating each individual machine, and a greedy algorithm for combining the base level policies to obtain an overall policy for running the transfer line. Unlike other approaches to hierarchical optimization, in our system policies at the lowest level are modeled using semi-Markov decision processes (SMDPs). In our results to date, we can show that the hierarchical approach is not only much faster than the "flat" algorithm, but also appears to outperform well-known heuristics for running transfer lines used in many factories today.

1 Introduction

There is much recent interest in hierarchical reinforcement learning (RL) as a way of scaling standard "flat" algorithms like Q-learning to converge to a good policy more rapidly. Among the various approaches proposed are macro-based models called "options"[17], hierarchical finite-state machine models called "HAMs" [11], and "weakly-coupled" MDP models [10]. In this paper, we present some promising evidence from a realistic manufacturing problem, where we show that the "flat" version of a semi-Markov average reward Q-learning algorithm called SMART [9] does fairly poorly, but a modified hierarchical version of the same algorithm does extremely well.

The notion of hierarchy in our task is very similar to that in the work on "weakly coupled" MDPs, and that in the "options" models. The principal difference is that in our task, we use a semi-Markov decision process (SMDP) model throughout, both at the primitive level, and at the abstract level. In the "options" model, for example, abstract actions are built out of discrete-time unit actions. The work on "weakly-coupled" MDPs has largely assumed constant time actions thus far. In our domain, abstract policies are themselves constructed out of non-constant-time actions, requiring a semi-Markov model at the lowest level. We believe that these other approaches can be applied to our task, however, and in particular, might provide a more powerful learning mechanism for combining abstract policies (which might be superior to our current greedy method).

The manufacturing problem we study is a discrete-part systems consisting of a (parallel or serial) combination of distinct machines, whose overall goal is to satisfy target demand while minimizing work-in-process inventory [6]. The machines age as they are used, and are subject to breakdown and consequently need repair. Periodic maintenance can avoid unnecessary and costly repair. Transfer lines are a well-known model of factories (in particular, assembly plants) where product buffers are placed in between machines to reduce the effect of a breakdown of one machine on the overall production process. Since it is intractable to analytically compute an optimal control policy for transfer lines with more than 2 machines [6], some heuristic control method must be employed. Many heuristics for controlling a transfer line have been studied and used in real manufacturing sites, the most significant being KANBAN [1] and constant work-in-progress(CONWIP) [15]. A key disadvantage of these heuristics is that they only address the problem of minimizing work-in-process inventory. However, in reality, many other costs need to be considered, including the cost of production of parts, and the repair and preventive maintenance costs.

An *analytical* approach to optimizing manufacturing systems consists of modeling each machine using a standard stochastic process, such as a semi-Markov decision process (SMDP) [12]. The optimal procedure can be determined using a dynamic programming algorithm, such as value or policy iteration. Unfortunately, this approach suffers from two central limitations: a "curse of dimensionality", since the number of possible states is often very large (e.g. 10^{15} states for the 12-machine transfer line studied here); and the requirement that a transition model of the one-step dynamics of the manufacturing plant be available.

In this paper, we adopt the average-reward framework where agents maximize the expected reward per step [8]. In particular, we have developed an average-reward reinforcement learning method that is suited to optimize discrete-event simulation models. The algorithm called SMART (for Semi-markov Average Reward Technique) was used to learn a flexible maintenance policy for a large multi-product machine with buffers, and shown to be superior to heuristic maintenance procedures, such as coefficient of operational readiness (COR) and age replacement (AR) [9]. In this paper, we describe a hierarchical version of the SMART algorithm, and apply it to optimize an even larger 12-machine transfer line. We demonstrate that the hierarchical SMART algorithm produces a transfer line policy that is superior to both CONWIP and KANBAN, and is also superior to the previous "flat" version of SMART.

2 Stochastic Model of a Transfer Line

Before introducing transfer lines, let's briefly examine a stochastic model of a single machine. Each machine produces parts, which fill a buffer, but the machines are responsible for their own maintenance. So the single machine's action set for each decision epoch includes producing, maintenance and idling. The machine interacts with the one before it through its *supply rate*, and the machine interacts with the one after it by a *demand rate*. Each machine has a failure time modeled according to some distribution, and when it fails, it needs to be repaired. The repair time and maintenance time are stochastic variables according to some distributions respectively. Thus, the single machine process above can be described by a semi-Markov decision process(see figure 1). Each point in the state space is an vector of $< t, l >$, where t denotes the running time of the machine since last maintenance or repair; and l denotes the inventory size right after the machine. Since time is an continuous variable, we coarsely discretize this value into intervals to turn the state space into finite one.

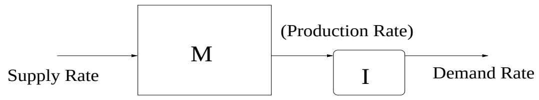

Figure 1: A single machine is modeled as a semi-Markov decision process, with non-constant time actions such as "produce" or "maintain". The SMDP process is parameterized by the supply and demand rate.

A transfer line [6] models an assembly line operation in many factories, where a sequence of machines is needed to make finished parts. Usually there will be a buffer between two machines, since the machines are unreliable, and a failure of one machine should not disrupt the working of the entire line (see figure 2). The state space of the whole transfer line is essentially the cross product of the individual machine state spaces, and the action space is the cross-product of the individual machine action spaces. Thus, the entire transfer line can be described by a usually huge semi-Markov decision process, parameterized by the input supply rate and the output demand rate.

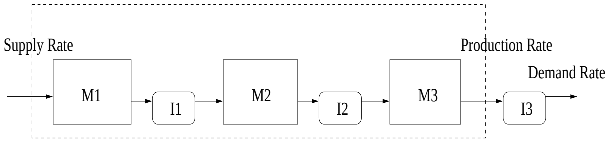

Figure 2: A transfer line with multiple machines can be viewed as a very large semi-Markov decision process, parameterized by supply and demand rate at its input and output.

However, if we look at the processes in a transfer line closely, we can see that the upstream machine's production rate is the downstream machine's supply rate; and the downstream machine's production rate is the upstream machine's demand rate. For example, in transfer line of figure 3, machine 1's and machine 3's production rate are machine 2's supply rate and demand rate respectively. Therefore, we can decompose the overall "flat" semi-Markov decision process into n smaller semi-Markov decision processes which are sequentially coupled with each other.

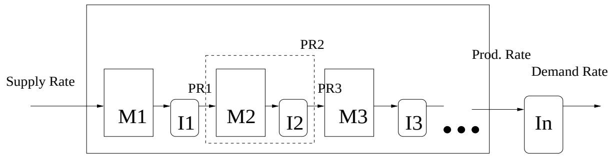

Figure 3: The overall semi-MDP model of a transfer line can be decomposed into a set of "weakly coupled" significantly smaller single machine semi-Markov decision processes.

Now we briefly explain the framework of semi-Markov decision processes (SMDPs). An SMDP is defined as a five tuple (S, A, P, R, F), where S is a 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 is a function giving probability of transition times for each state-action pair. $P_{xy}(a)$ 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. F is a function where $F(t \mid s, a)$ is the probability that the next decision epoch occurs within t time units, after action a is executed in state s. From F, and P, we can compute Q by

$$Q(t, y \mid x, a) = P(y \mid x, a)F(t \mid x, a)$$

where Q denotes the probability that the system will be in state y for the next decision epoch, at or before t time units after choosing action a in state s, at the last decision epoch. Q can be used to calculate the expected transition time between decision epochs.

In general, the reward function for SMDPs consists of a fixed reward $k(x, a)$, resulting from an action a taken in state x, and an additional reward that is 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.

Formally, the expected reward between two decision epochs, given that the system is 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. In transfer lines, for example, each unit of product sold provides a fixed lump-sum positive reward, whereas there is a continuous rate-cost for items that are stored in buffers. Failure, repair and maintenance results in a fixed cost.

The average reward ρ^π for a policy π can be defined by taking the limit of the ratio of the expected total reward up until the nth decision epoch to the expected total time until the nth epoch. For each transition, the expected transition time is defined as:

$$\tau(x, a) = E_s^a \tau = \int_0^\infty t \sum_{y \in S} Q(dt, y \mid x, a) \quad (2)$$

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

The Bellman optimality equation for *unichain* average reward SMDPs can be written as

$$V^*(x) = \max_a \left(r(x, a) - \rho\tau(x, a) + \sum_{y \in S} P_{xy}(a) V^*(y) \right) \quad (3)$$

where $V^*(x)$ is the average-adjusted sum of rewards received in state x under a gain-optimal (i.e. average-reward optimal) policy. In unichain SMDP's, the average reward is constant across states. Many real-world SMDP's, including the transfer line problem below and the elevator task [3] are unichain.

3 The SMART Algorithm

It is often too difficult to determine a gain-optimal policy for a given SMDP by analytically solving Equation 3, due to the large size of the state space and the lack of a one-step probability transition model. Instead, we use an approximate approach of solving the Bellman equation that uses the simulation model to generate sample trajectories through the state space. Each sample trajectory provides instances of states and rewards, and can be used to iteratively improve a given policy.

In this paper we develop a hierarchical method for solving SDMPs, based on a previously developed algorithm called SMART (for Semi-Markov Average Reward Technique) [9]. This algorithm, described in Figure 4 represents policies using an action value function $R(x,a)$, which is the average-adjusted sum of rewards of doing action a in state x once, and thereafter following the optimal policy.[1] The derivation of this algorithm from the Bellman equation for SMDPs (Equation 3) is fairly straightforward and resembles the derivation of algorithms such as Q-learning [16].

1. Initially, let decision epoch $n = 0$, and action values $R_n(x,a) = 0$. Let initial state of the simulation $= x$. Set total reward c_n and total time t_n to 0.
2. While $n <$ MAX_STEPS do
   (a) With high probability p_n, choose action a that maximizes $R_n(x,a)$, otherwise choose a random action.
   (b) Perform action a. Run the simulation model until the next decision epoch, when state $= z$, the transition time $= \tau$, and $r_{imm} =$ *cumulative* reward earned over epoch.
   (c) Update $R_n(x,a)$ using:
   $$R_n(x,a) \overset{\alpha_n}{\leftarrow} \left(r_{imm} - \rho_n \tau + \max_b R_n(z,b) \right)$$
   (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 $x \leftarrow z$, and $n \leftarrow n+1$.

Figure 4: The SMART algorithm solves average-reward semi-Markov decision processes.

The number of states in most optimization problems of interest is too large to use a table-based representation of the action value function. Consequently, a number of methods have been developed to approximate the action value function using parametric techniques, such as neural nets [3] and CMAC [16]. In our work, we used a combination of two approaches to reduce the size of the state space: hierarchical decomposition of the transfer line, and an interval-based discretization of continuous state variables (such as the age of a machine).

[1] The notation $u \overset{\alpha_n}{\leftarrow} z$ is an abbreviation for $u \leftarrow (1-\alpha)u + \alpha(z-u)$.

3.1 Hierarchical Optimization

As we can see from the discussion in the previous section, the number of states in a transfer line grows exponentially as the number of machines is increased. For example, in a 12-machine transfer line, the number of states can easily exceed 10^{15}, if we assume a maximum buffer size of 10, and a continuous age value which is discretized into 100 equal intervals. The exponential growth in states make it impossible to apply reinforcement learning to optimize the transfer line as a whole. Instead, we have to employ some method to decompose the transfer line into simpler optimization problems and solve it approximately.

There are two ways to approximately solve this problem. One is to individually and simultaneously learn policies for every machine on-line by independently running n copies of the reinforcement learning algorithms together. We refer to this below as the *flat-SMART* algorithm. However, for individual machine indexed i in a transfer line, its semi-Markov decision process is dependent on the machine's supply rate and demand rate, in other words, it depends on the production rate of machine $i-1$ and machine $i+1$. Therefore, if the production rates of the machine upstream and downstream are changing, the semi-Markov decision process of the machine i will also vary accordingly. This makes on-line learning of an individual machine's policy in a transfer line extremely difficult and slow.

Hierarchical methods of optimizing manufacturing systems have been studied before [4] [5] [13]. However, individual machines in a transfer line interact and straightforward decompositions methods can produce sub-optimal results. Our reinforcement learning based approach is quite distinct from these earlier hierarchical methods, and in essence is as follows. We already know that the overall semi-Markov decision process of the whole transfer line can be subdivided into a sequence of semi-Markov decision processes coupled one by one by the actual production rates. Therefore, instead of training each machine *in-situ* while it is part of the transfer line, we instead train each machine in isolation using SMART for a variety of different boundary conditions (in particular, supply rates of parts coming in, and target demand rates of parts consumed by machines down the line). Once each machine is trained individually, we then optimize the transfer line as a whole, where instead of selecting each action for operating a machine, we choose between different *policies*.

When we train each single machine in various types of supply/demand situation, we keep around every policy that it learned and reuse these policies when the machine is reinserted into the transfer line. Now we have two level of control: the lower one is the ordinary reinforcement learning level of learning control of individual actions, and the higher one is a heuristic one of deciding which policy to use to run each machine, according to the observed supply and demand rates from machines up and downstream. We call this technique as *hierarchical-SMART*.

4 Experimental Results

In this section we present detailed experimental results of the hierarchical SMART algorithm for optimization of transfer lines. We compare the performance of our algorithm to heuristic procedures such as KANBAN or CONWIP. We also compare the benefits of hierarchical decomposition with a "flat" model of optimization.

For the results described below, we used the following parameters. The unit (rate) cost for a single product on inventory is 2.0, the cost for failure is 40.0, and the maintenance cost is 5.0. For each product, a reward of 10 is received if it is "sold". Each machine has a failure time modeled as a normal distribution with mean of 10.0 and standard deviation of 2.0. Repair time, after each failure, is exponentially distributed at mean of 10.0. The time for maintenance is also exponentially distributed at mean of 2.0. The time to produce a product is lognormally distributed with mean of 1.0. We emphasize that our approach does not depend on the distributional or cost model, and we have conducted experiments with other cost and distributional models also.

4.1 "Flat" Optimization of a 3-machine Line

We begin by comparing SMART on a three machine transfer line problem with the KANBAN and CONWIP heuristic methods. Note that this is a non-hierarchical approach, where each machine is trained while it is part of the overall transfer line. Figure 5 compares the average reward and average inventory size for the policy learned by SMART with that produced by the CONWIP and KANBAN heuristics. Each graph shows the result over 50,000 time units. The supply/demand rate is infinite/0.6. The results are averaged over 10 separate runs. Clearly, the average reward of the policy learned by SMART is significantly better than that achieved by CONWIP or KANBAN. This improvement is partly due to the fact that the latter heuristics do not incorporate a maintenance policy to minimize repair costs. Also, the policy learned by SMART also achieves a significant reduction in inventory size over KANBAN and CONWIP.

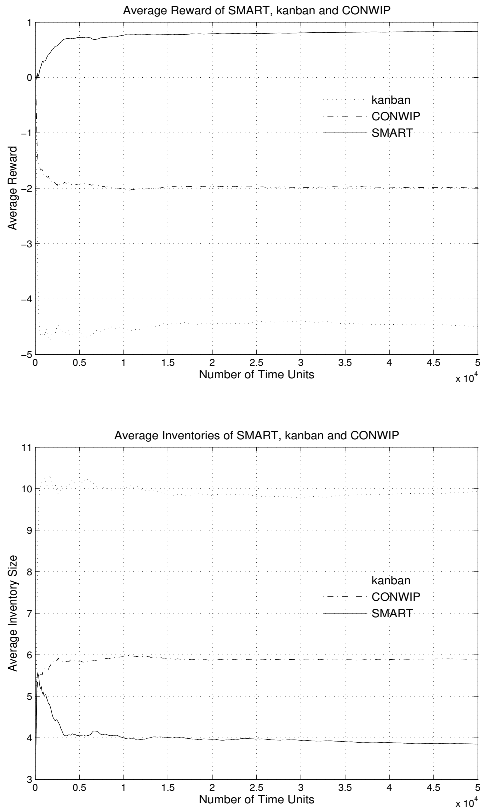

Figure 5: Comparing the average reward and inventory size learned by SMART for a three-machine transfer line with that produced by the KANBAN and CONWIP heuristics.

4.2 Hierarchical Optimization of Large 12-Machine Line

We now turn to hierarchical optimization, where a "typical" machine is trained in isolation with different supply/demand rates. For each supply/demand rate combination, we ran SMART for 100,000 time units. The results are shown in Figure 6. Note that KANBAN performs poorly, when support rate is higher than demand rate, because it tends to fill inventory space. In such cases, SMART does a much better job at balancing the production rate along with supply and demand rate to get a higher average reward policy.

After training, we reinsert each machine into the line, and use a hierarchical approach to compute the

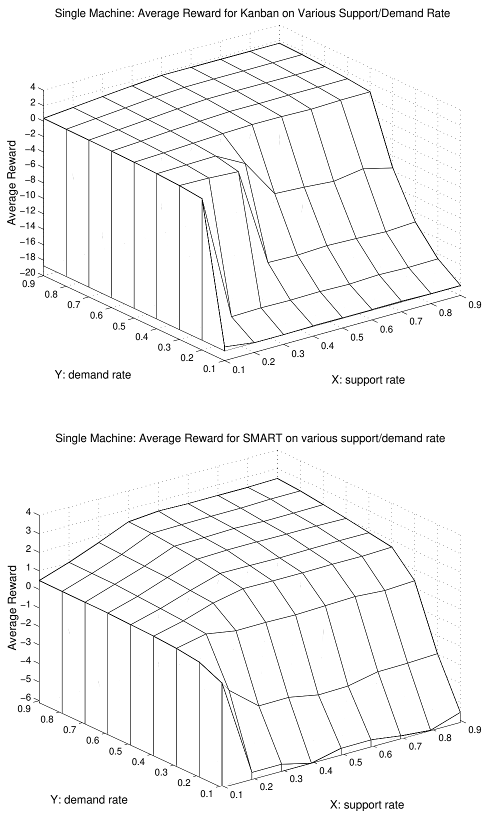

Figure 6: Average reward by kanban and SMART on various kinds of supply/demand rate situation.

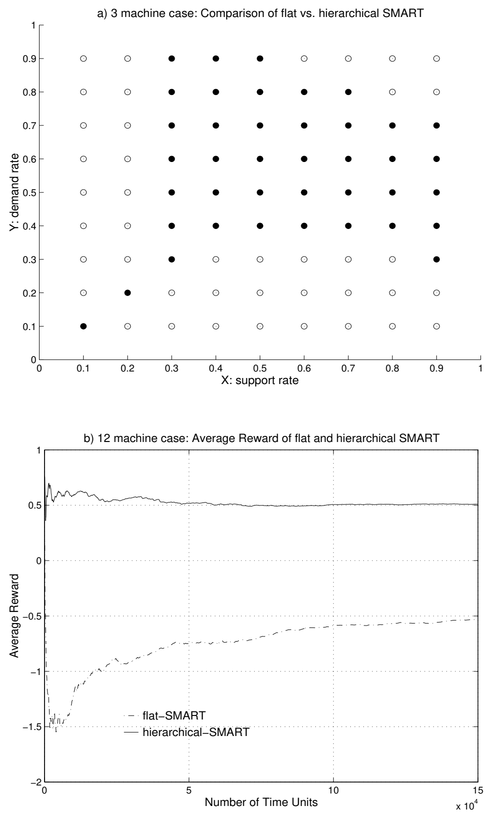

Figure 7: a) Comparison of average reward by hierarchical-SMART versus flat-SMART in 3 machine transfer line. The solid dot means that hierarchical-SMART outperformed the flat-SMART approach. b) Average reward plot in 12 machine transfer line for flat-SMART and hierarchical-SMART.

policy for the whole transfer line. Here, each machine is operated by selecting from one of the previously learned policies, according to its observed supply/demand rate. In this set of experiments, each machine keeps track of its supply and demand rate for the last 20 events and chooses the closest policy. Figure 7 (a) compares the average reward of the policy learned by the hierarchical-SMART algorithm versus the flat-SMART approach, averaging over 5 runs for each approach.

We conducted a further experiment (figure 7 (b)) on a bigger transfer line with 12 machines. The results are from a single run for both flat and hierarchical SMART. The supply/demand rate is 0.7/0.7. This experiment shows that on larger scale optimization problems, the hierarchical SMART method results in an even more dramatic improvement over the "flat" approach.

5 Conclusions and Future Work

In this paper, we presented a hierarchical algorithm for finding a low-cost control policy for running a large manufacturing system. The algorithm requires as input a simulation model and a cost function. We illustrated the method for the problem of optimizing a transfer line, and showed that it outperformed standard heuristics such as CONWIP and KANBAN. The hierarchical method uses a reinforcement learning algorithm called SMART to find low-cost control policies for operating individual machines. These policies are then combined to obtain a policy for the entire transfer line. On a 12-machine transfer line, the hierarchical method clearly outperformed the "flat" version of SMART.

Our approach currently suffers from some significant limitations. Our hierarchical algorithm for combining individual machine policies is non-adaptive, and the choice of initiating and terminating a policy is heuristic. A significant extension of our algorithm would be to implement some learning algorithm at the abstract level that can combine the individual machine policies to obtain a seamless policy for the entire line as a whole. There are difficulties in implementing a learning mechanism at the abstract level, however, since the the number of policies at the abstract layer grows quickly depending on the discretization of the demand and production rates.

Acknowledgments

This research is supported in part by an NSF CAREER Award Grant No. IRI-9501852, and by a Michigan State University (MSU) AURIG grant.

References

[1] Berkley, B. J. 1992. "A review of the kanban production control research literature" in *Production and Operations Management 1(4)*, 393-411.

[2] Bonvik, A. M. Couch, C. Gershwin, S. 1996. "A Comparison of Production-Line Control Mechanism". Accepted for *International Journal of Production Research.*

[3] Crites, R. Barto, A. 1996. "Improving elevator performance using reinforcement learning" in *Neural Information Processing System (NIPS).*

[4] Gershwin, S. Caramanis, M. Murray, P. 1988. "Simulation Experience with a Hierarchical Scheduling Policy for a Simple Manufacturing System", in *Proceedings of the 27th IEEE Conference on Decision and Control*, Austin, Texas.

[5] Gershwin, S. 1987. "A Hierarchical Framework for Manufacturing System Scheduling: A Two-Machine Example." in *Proceedings of the 26th IEEE Conference on Decision and Control*, Los Angeles, CA.

[6] Gershwin, S. 1994. *Manufacturing System Engineering.* Prentice Hall.

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

[8] Mahadevan, S. 1996. "Average Reward Reinforcement Learning: Foundations, Algorithms, and Empirical Results" in *Machine Learning.*, 22, 159-195.

[9] 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.*, D. Fisher (editor), Morgan Kaufmann. 202-210.

[10] Meuleau, N., Hauskrecht, M., Kim, K, Peshkin, L., Kaelbling, L., Dean, T., Boutilier, C., "Solving Very Large Weakly Coupled Markov Decision Processes", Proceedings of the Conference on Uncertainty in AI, 1998.

[11] Parr, R. and Russell, S. 1998. "Reinforcement Learning with hierarchies of machines" in *Advances in Neural Information Processing Systems (NIPS).*

[12] Puterman, M. L. 1994. *Markov Decision Processes: Discrete Stochastic Dynamic Programming.* Wiley Series in Probability and Mathematical Statistics. John Wiley & Sons, Inc.

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

[14] Singh, S. Bertsekas D. 1996. "Reinforcement Learning for Dynamic Channel Allocation in Cellular Telephone Systems" in *Neural Information Processing Systems (NIPS)*, 1996.

[15] Spearman, M. L., D. L. Woodruff, and W. J. Hopp. 1990. "CONWIP: a pull alternative to kanban" in *International Journal of Production Research 28(5)*, 879-894.

[16] Sutton, R., Barto, A. 1998. *Reinforcement Learning: An Introduction*, MIT Press, Cambridge, MA. A Bradford Book.

[17] Sutton, R., Precup, D., and Singh, S. 1998, "Intra-option Learning about Temporally Abstract Actions", Proceedings of the 15^{th} International Conference on Machine Learning (IMLC-98), pp. 556-564.