Proceedings of the 2003 Winter Simulation Conference
S. Chick, P. J. Sánchez, D. Ferrin, and D. J. Morrice, eds.

A REINFORCEMENT LEARNING APPROACH TO PRODUCTION PLANNING IN THE FABRICATION/FULFILLMENT MANUFACTURING PROCESS

Heng Cao
Haifeng Xi

IBM T. J. Watson Research Center
Yorktown Heights, NY 10598, U.S.A.

Stephen F. Smith

The Robotics Institute, School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213, U.S.A.

ABSTRACT

We have used Reinforcement Learning together with Monte Carlo simulation to solve a multi-period production planning problem in a two-stage hybrid manufacturing process (a combination of build-to-plan with build-to-order) with a capacity constraint. Our model minimizes inventory and penalty costs while considering real-world complexities such as different component types sharing the same manufacturing capacity, multi-end-products sharing common components, multi-echelon bill-of-material (BOM), random lead times, etc. To efficiently search in the huge solution space, we designed a two-phase learning scheme where "good" capacity usage ratios are first found for different decision epochs, based on which a detailed production schedule is further improved through learning to minimize costs. We will illustrate our approach through an example and conclude the paper with a discussion of future research directions.

1 INTRODUCTION

Today's manufacturers, facing the intensifying competition and steady pressure for higher levels of customer service, are compelled to continuously improve their supply chain management. A very appealing approach is a blend of the "push" and "pull" production control philosophy that combines build-to-plan with make-to-order operations, which we refer to as *fabrication/fulfillment* process in this paper. The fabrication stage is a build-to-plan process, where components are procured, tested and assembled. The component inventory is then kept in stock ready for the final assembly into the end-products. The fulfillment stage is a make-to-order process, which means that no finished goods inventory is kept for end-products and the final assembly starts after the customer order is received. This fabrication/fulfillment process works especially well when the final assembly time and resource requirements (in terms of both labor and machinery) are negligible compared with the more substantial production/replenishment lead-times and resource requirements, and when there are a lot of common components shared by various final products. The fabrication/fulfillment process is an ideal business process model that enables both mass customization and the speed and efficiency provided by mass production.

The work presented in this paper tries to find a near-optimal production plan for the fabrication-stage components. The objective of such a plan is to minimize the unsatisfactory end-product fulfillment penalty cost and the component inventory holding cost. The constraint to this problem is limited production capacity with an additional complexity that similar components can share the same type of manufacturing resources. This study is motivated by an inventory planning problem at IBM, where a skewed demand pattern, i.e., the weekly order numbers gradually increase and usually reach their peaks at the end of the quarter, can be observed for most end-products, while components production needs to be relatively smoothed across different periods due to the capacity constraint. We also take into consideration such real-world situations as multi-echelon BOM and end-products sharing common components, which are very typical of PC and Server manufacturing. Problems with such scale of complexity are extremely hard to solve mathematically, if not completely intractable.

Because of its versatility and potential at generating good solutions to extremely complicated problems, simulation-based optimization has rapidly gained popularity in recent years (Ólafsson and Kim 2002, Finke et al. 2002). Monte Carlo simulation is employed as a natural way to capture sophisticated details of the underlying system; the simulation results are fed into the optimization algorithm to direct its search for an optimal or near-optimal solution. The most successful developments in this area are the applications of heuristic search algorithms, such as Tabu search (Finke et al. 2002), simulated annealing, genetic algorithms (Joines et al. 2002) and nested partitions method of Shi and Ólafsson (2000). The basic idea is to use the search methods to find control parameters that improve the expected value of the objective.

In the supply chain problem described above, the decision variables are build quantities for each component in

each planning period (i.e., the detailed build plan), and the objective is to minimize the expected value of costs. For a moderate-size setting with a few hundred components and a planning horizon of 10 or so periods, the number of control parameters can easily reach thousands, which creates an overwhelming computational burden on those iterative search algorithms. Besides, to obtain a reasonable estimation of expected cost value, multiple simulation runs using different random seeds are required, yet we know that even a single run of the real-world supply chain simulation can get very slow (Cao et al. 2002). Such computing intensity renders those popular heuristic based search algorithms impracticable for our problem.

In this paper we propose a Reinforcement Learning (RL) based approach that addresses the above-mentioned search space explosion from two perspectives. Firstly, we show that the production planning problem in question can be modeled as a Markov Decision Process (MDP), which reduces the search space because of the implicit dependency of control parameters between any two successive decision steps. Reinforcement learning, *Q*-learning in particular, is used to search for a good decision (a.k.a. action in RL term) in each MDP step, and the decision is sent to a Monte Carlo simulator (Cao et al. 2002) which evaluates the outcome, in terms of the concerned costs, of this certain action in the corresponding decision epoch. Secondly, to further reduce the action space and state space, we have introduced a two-phase RL scheme where desirable capacity usage ratios are first found for different decision epochs, based on which a detailed production schedule is finally developed.

In the following section, we define the problem in terms of its input parameters, decision variables, constraints and cost objective. Section 3 takes a close look at our solution framework by elaborating on the MDP modeling, *Q*-learning algorithm and the two-phase learning scheme. Section 4 discusses the supply chain model dynamics and the Monte Carlo simulator, followed by an illustration of the framework being applied to a simple example in Section 5. In the last section, we conclude our methodology and point out future research directions.

2 PROBLEM DEFINITION

As illustrated in Figure 1, we consider a two-stage manufacturing process: in the fabrication stage, a set of components are built to plan through several steps, which require certain amount of lead time of access to the necessary manufacturing resources (assembly lines, test machines and labor). Components of the same type (for example, CPU, memory, etc), share the same type of manufacturing resources. Upon receipt of customer orders in the fulfillment stage, end-products are assembled from components inventories according to their BOM configurations.

To help define the problem, we will first introduce some notations, and then break the detailed definitions into

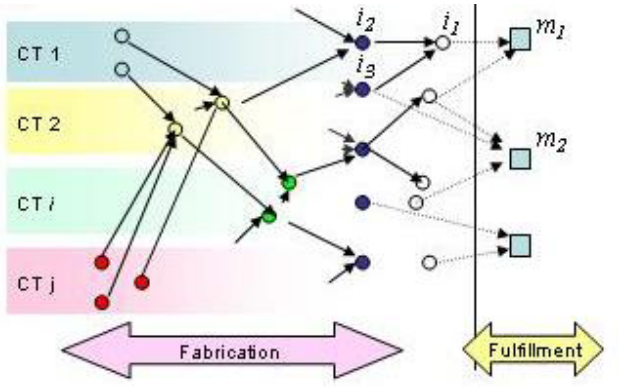


Figure 1: Fabrication/Fulfillment Manufacturing Process

four parts: input parameters, decision variables, capacity constraint, and objective. Please note that the problem is always solved for a planning cycle starting sometime in the future; a planning cycle contains a given number of periods, each of which in turn consists of a certain number of work days.

Notation

- Let $p = \{1,..., p_e\}$ index time periods in the planning cycle. We assume the available end-product demand plan starts from period 1.
- Each end-product is indexed by $m, m \in M$, where M denotes the set of all end-products.
- Each component is indexed by $i, i \in I$, where I denotes the set of all components. Components are grouped into mutually exclusive commodity types. Assume there are n commodity types, then I can be partitioned into n disjoint subsets, i.e., $I = \cup_{j=1}^{n} I_j$ and $I_{j_1} \cap I_{j_2} = \varnothing$ when $j_1 \neq j_2$, where I_j denotes the set of components belonging to the jth commodity type.

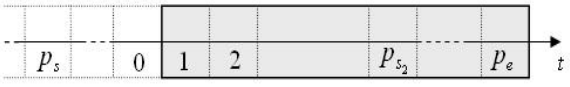
Input Parameters

1. *BOM*: Each end-product and component is characterized by its distinct BOM.
2. *Lead Time*: There are two types of lead time in the system, one is the assembly lead time of end-products, which is negligible compared to the component lead times. We assume the component lead time is a random variable with a known distribution, and its mean is defined in terms of work days.
3. *Resource Capacity for Component Type*: There is a resource capacity limit for each commodity type j, in terms of the maximum number of components $i \in I_j$ that can be processed in each planning period, denoted as $C_j, j = 1,..., n$
4. *Cost*: We consider two types of cost in the system. Inventory holding cost $hc_i, i \in I$ is the unit cost of the on-hand inventory of component i for each period. Penalty cost $pc_m, m \in M$ is the unit cost of not having sufficient stock on hand to satisfy a

demand. The penalty cost is much larger than the inventory holding cost.

5. *Demand Plan*: End-product demand is the output from an enterprise demand planning process and is a biased forecast derived from the unbiased demand forecast, revenue targets, and overall manufacturing capacity in the planning cycle. It is denoted as: $d_m(t),\ t = 1,\ldots,\ p_e,\ m \in M$.

Decision Variables. Decision variables are build quantities of each component *i* for each period in the planning cycle, denoted as $x_i(t), i \in I, t = p_s,\ldots,\ p_e$. Due to manufacturing lead times, in order to satisfy end-product demand in a certain period, components need to be built earlier, therefore we have $p_s \le 1$, where p_s is the earliest manufacturing start time for those leaf components on the BOM tree. There will be a $p_{s_2}\ (1 \le p_{s_2} \le p_e)$ that corresponds to the earliest manufacturing start time for the next planning cycle as well. If we use $x_{i1}(t)$ and $x_{i2}(t)$ to represent the number of components produced to satisfy end-product demand for the current and next planning cycle respectively, we have:



$$x_i(t) = x_{i1}(t) + x_{i2}(t)$$

$$\begin{cases} x_{i1}(t) \ge 0, & p_s \le t \le p_e \\ x_{i2}(t) = 0, & p_s \le t < p_{s_2} \\ x_{i2}(t) \ge 0, & p_{s_2} \le t \le p_e \end{cases}$$

If we assume that the unknown demand plan for the next cycle follows a similar pattern as the current cycle, then we can have the following approximation:

$$x_{i1}(t) \approx x_{i2}(p_e - t), \quad p_s \le t \le 0$$

We can then focus on solving the problem, i.e., finding $x_{i1}(t)$ and $x_{i2}(t)$, for $1 \le t \le p_e$, and use the above approximation to determine $x_{i1}(t)$ for $p_s \le t \le 0$.

Capacity Constraint.

$$\sum_{i \in I_j} x_i(t) \le C_j(t), \quad j \in \{1,\ldots, n\}, \quad 1 \le t \le p_e$$

Objective. The objective is to minimize the total cost including inventory holding cost in the fabrication stage and late shipments penalty cost in the fulfillment stage:

Cost =

$$\sum_{t=1}^{P_e} \left(\sum_{i \in I} hc_i \cdot E[Inv_i(t)] + \sum_{m \in M} pc_m \cdot E[Backlog_m(t)] \right)$$

where $Inv_i(t)$ is the average component inventory quantity in each period and $Backlog_m(t)$ is the average backlogged end-product quantity in each period.

Due to the highly nonlinear nature of the cost function plus multi-echelon product structure with stochastic lead times, coming up with a closed form analytical optimization solution is extremely hard, if not impossible.

3 SOLUTION FRAMEWORK: MDP AND REINFORCEMENT LEARNING

An MDP is characterized by four components: a state space ***S*** that specifies all possible configurations of the system; the action space ***A*** that lists all available actions for the learning agent to perform; the transition function that specifies the possibly stochastic outcomes of taking each action in any state; and a reward function that defines the possible reward of taking each of the actions. To model a problem as an MDP, the ***S*** and ***A*** should be carefully defined so that the transition function only depends on the current state and action, and not on earlier states or actions.

RL is a simulation-based technique for solving complex MDP (Kaelbling et al. 1996, Van Roy et al. 1997). Based on the Bellman equation, RL judiciously makes use of the stochastic approximation thereby eliminating the difficulty of estimating transition and reward function. *Q*-learning is an easy-to-implement RL algorithm that solves the Bellman equation iteratively through estimating the values of state-action pairs. The value $Q(s,a)$ is defined to be the expected sum of future payoffs obtained by taking action *a* from state *s* and following an optimal policy thereafter. Once these values have been learned, the optimal action from any state is the one with the highest *Q* value. Please refer to Kaelbling et al. (1996) for a more detailed discussion of *Q*-learning.

For the problem we formulated in Section 2, it may seem tempting at the first glance to form a vector using all $Inv_i(t)$ and $Backlog_m(t)$ to define the state at time *t*, and the vector of each component's build quantity as the action set. However, this simplistic formation creates extremely large state and action spaces, and poses tremendous difficulty for the learning algorithm to converge.

To handle the "curse of dimensionality", we will carefully examine the structure of the problem. It is easy to understand that, without the capacity constraint and the randomness in the lead times, the optimal cost value can be achieved by matching the build quantity to the demand generated from standard MRP demand explosion technique (Nahmias 1997). Using this set of derived build quantities, denoted as $x_i'(t)$, as initial values, we can apply RL to learn the policies to optimally adjust those values, so the capacity constraint can also be satisfied. Using $x_i'(t)$ as a "hint" for the initial build plan value, we have significantly reduced the "searchable" action space.

To further improve the learning efficiency, we transform this constraint optimization problem into a two-phase MDP learning problem. In the first phase, we try to find the range of resource usage ratios for each period that will most probably lead to a near-optimal solution. In the second phase, the resource usage ratio is fixed for each period, RL is used to develop the detailed build plan.

We first introduce some notation: $x_i''(t)$ denotes the "target" build plan and is initialized as $x_i'(t)$; the *resource demand profile* for each of the resource type is defined as

$$d_j(t) = \sum_{i \in I_j} x_i''(t), \quad j \in \{1, ..., n\}, t \in \{1, ..., p_e\}$$

For the first phase learning, we define the state $\boldsymbol{S}$ at time t as an n-dimensional vector S^t such that $S_j^t = \left[C_j - d_j(t)\right]/C_j$. The action $\boldsymbol{A}$ at time t is also an n-dimensional vector A^t, such that $A_j^t = \lambda_j$ where λ_j is the usage ratio of the spare capacity $S_j^t \cdot C_j$ when $S_j^t > 0$. When $S_j^t \le 0$, λ_j has 0 as the only possible value. Both state element S_j^t and action element λ_j need to be discretized when constructing the Q table entry $Q(S^t, A^t)$. In Section 5, we will discuss the performance difference in using different discretization schemes.

Once A^t is decided, it needs to be translated into $x_i(t)$, so the incurred cost for this action can be evaluated by the simulator. For $\lambda_j > 0$, there are $\Delta = \lambda_j \cdot S_j^t \cdot C_j$ number of type j components that can be built one period ahead of its original target build plan. For $\lambda_j = 0$ and $S_j^t < 0$, there are $\Delta = -S_j^t \cdot C_j$ quantity of j type components that have to be postponed to the next period. When it is not clear which component's build plan should be advanced (or postponed), we randomly select i form I_j, and advance (or postpone) one quantity from its original target build plan, until Δ becomes 0.

The following is the Q-learning algorithm for phase-one learning:

1. For each S^t and A^t in different planning periods, initialize the Q table entries $Q(S^t, A^t)$ to zero.
2. Initialize the target build plan $x_i''(t)$ as $x_i'(t)$.
3. For t from 1 to p_e, do:
   3.1 Observe the current state vector S^t, select an action A^t with a Boltzmann exploration policy, so that the action with a larger reward (corresponding to lower cost) will receive a bigger chance to be explored.
   3.2 For each resource type j, derive $x_i(t)$ from $xi''(t)$ based on the selected $A_j^t = \lambda_j$:
   if $S_j^t > 0$, randomly select $\delta_i(t)$ from $x_i''(t+1)$ in the way explained before, such that $\sum_i \delta_i(t) = \lambda_j \cdot C_j$, and then
   let $x_i(t) \leftarrow x_i''(t) + \delta_i(t)$
   and $x_i''(t+1) \leftarrow x_i''(t+1) - \delta_i(t)$.
   if $S_j^t < 0$, A_j^t equals to 0, randomly select $x_i(t)$ from $x_i''(t)$ in the way explained before, such that $\sum_{i \in I_j} x_i(t) = C_j$, and then let

$$x_i''(t+1) \leftarrow x_i''(t+1) + x_i''(t) - x_i(t)$$

   else if $S_j^t = 0$, let $x_i(t) \leftarrow x_i''(t)$.
   3.3 Receive the immediate reward as the negative value of the cost c incurred from time t and arrival at a next state $\boldsymbol{S}^{t+1}$.
   3.4 Update

$$Q(S^t, A^t) \leftarrow \alpha \cdot [-c + \max_{A^{t+1}} Q(S^{t+1}, A^{t+1})] + (1-\alpha) \cdot Q(S^t, A^t),$$

   where α is the learning rate.
4. Go to step 2, or exit when the learning converges.

Since the penalty cost is much larger than the inventory holding cost, through the first phase learning, the spare capacities can be used wisely for building safety stock ahead, so that "bad" states can be avoided when $S_j^t < 0$.

In the second phase, we concentrate the learning on finding the near-optimal combination of components to be built ahead of the target build plan when there is extra resource capacity, or to be postponed when $C_j - d_j(t) < 0$. The usage ratio λ_j learned from the first phase is used. In the second phase, the state $\boldsymbol{S}$ at time t is defined as the target build plan, and $\boldsymbol{A}$ at time t is the combination of components to be built ahead from or need to be postponed to next period. The second-phase learning is pretty straightforward, we skip the detailed algorithm here.

Figure 2 presents the overall solution flow. The two-phase learning is achieved through the RL learner interactively working with the simulator. At each learning decision epoch t, the learner sends the build plan for that period to the simulator. After simulating one planning period, the simulator sends back the inventory snapshot, which is used by the learner to calculate the immediate reward, update the Q table, and pick up a valid action for the decision epoch t+1.

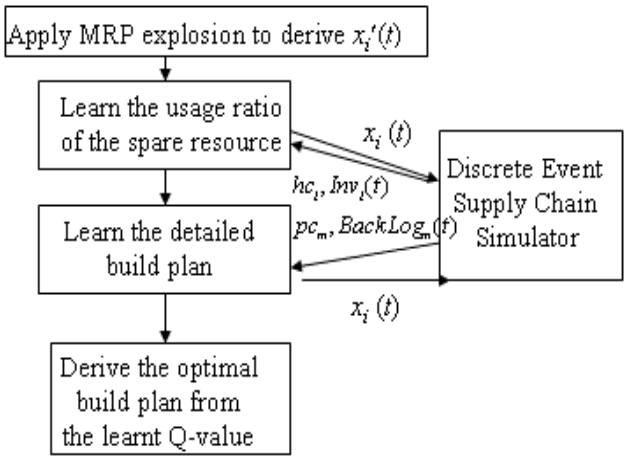


Figure 2: Overall Solution Framework

4 SIMULATE THE SYSTEM DYNAMICS

We built the simulator for the fabrication/fulfillment process by using our Java based supply chain simulation library (Cao et al. 2002). In addition to supporting generic discrete event simulation functionalities like time advancement and event management, the library provides some basic supply chain simulation components as well, such as backlogged orders, BOM, inventory pipeline etc.

The underlying supply chain structure of our problem is relatively simple and involves only two entities, i.e., one abstract *Customer* places orders for various end-products to a *Manufacturer*. According to their intended roles in the supply chain, each entity has its dynamics specified via a set of properties including knowledge about itself and the other entities, internal state, and event handlers/dispatchers.

There are two types of events being supported in supply chain entities, *internal* and *external*. An internal event signifies a simulation time advancement, and triggers internal state transitions. External events, such as *new order*, *order shipped* etc., are exchanged between entities as a result of their internal state transitions; these events are fired from an event dispatcher in the "source" entity to "sink" entities that have registered their event handlers with the source entity to indicate an interest in the events.

The Customer properties are defined as follows:

- *Knowledge*. Demand plan for each end-product in each simulated period.
- *State*. Unfilled orders, number of orders received on time, and number of orders received late.
- *Event Handlers and Dispatchers*. The internal event handler generates new order events based on the demand plan; these external events get queued up in the event dispatcher which will pass them on to the Manufacturer. The external event handler receives order shipped events from the Manufacturer, and updates Customer internal state accordingly.

The Manufacturer properties are defined as follows:

- *Knowledge*. BOM structure for each end-product, manufacturing lead times given in statistical distributions, inventory policy (build-to-plan or build-to-order), component and end-product stores each having an inventory pipeline queue.
- *State*. Unfilled orders, on-hand inventory and pipe line inventory for each node in the BOM tree.
- *Event Handlers and Dispatchers*. There are two external event handlers, one for new order events from the Customer, the other for *build plan update* events from the *Q*-learning algorithm. There is one internal event handler that simulates the two-step manufacturing process (illustrated below in pseudo code), and sends order shipped events to the Customer during the process.

Manufacturer::processInternalEvent

1. If today is the first work day in the planning period *p*, establish the build quantity of each component *i* for every work day in that period $b_i^{(p)}(t), t \in \{1,..., d_p\}$, where d_p is the number of work days in period *p*. This is done by apportioning the period-level build plan quantity evenly into work days in that period, such that:

$$\sum_{t=1}^{d_p} b_i^{(p)}(t) = x_i(p) \cdot$$

2. Simulate the fabrication process for work day *d* in period *p*:
   a. Shift inventory pipeline for each component *i* as follows:
      1. $pl_i(0) \leftarrow pl_i(0) + pl_i(1)$
      2. $pl_i(k) \leftarrow pl_i(k+1)$,
      3. *k* from 1 to the second to last element remove the last element in the pipeline where $pl_i(k)$ represents the quantity of component *i* that will become available on the *k*th work day starting from today.
   b. For each component *i*, a lead time *lt* is sampled from its stochastic distribution, and the corresponding pipeline element is updated as follows:

$$pl_i(lt) \leftarrow pl_i(lt) + b_i^{(p)}(d)$$

      For non-leaf components, remove the used sub components from their safety stock according to the BOM structure.
3. Simulate the fulfillment process as follows:
   a. Fill orders from the unfilled order list according to the order in which they were received, until the on-hand component inventory is depleted or there is no more unfilled order. For

orders received on the same day, fulfill the one with larger penalty cost first.

5 ILLUSTRATION

We have applied our approach to several small data sets for which the optimal solutions are known and have validated its effectiveness. In this section, we use one of the test data sets, which has 3 end-products and 4 components, for illustration purposes. The planning cycle has two periods each having 30 work days. The two-level BOM structure is shown in Figure 3, where usage ratios are listed adjacent to the links. Component P1 and P2 are of the same commodity type and share capacity $C1$ of 70 per period. Similarly, P3 and P4 share capacity $C2$ of 65 per period. All the end-product assembly lead times are neglected, and component lead times are given as normal distributions, with means and standard deviations shown in Table 1.

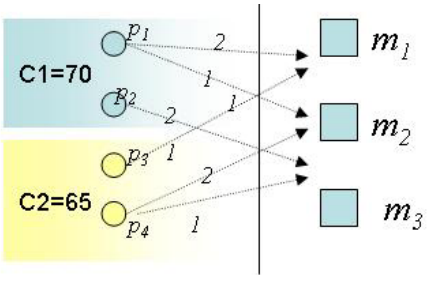


Figure 3: A Two-Level BOM Example

Table 1: Lead Times Distribution (Mean, Std. Dev.)

Period	P1	P2	P3	P4
1	(4, 1)	(5, 2)	(2, 0)	(1, 0)
2	(3, 1)	(4, 1)	(2,0)	(3, 1)

The optimal build plan for the above model can be derived through a simple Dynamic Programming based approach designed by Cao et al. (2003), and the corresponding minimal cost is 1670 given the holding costs and penalty costs listed in Table 2.

Table 2: Holding Costs and Penalty Costs

Component Holding Costs			
P1	**P2**	**P3**	**P4**
20.08	20.13	30.89	32.79
End-Products Late Fulfillment Penalty Costs			
M1	**M2**		**M3**
340	560		400

We use $\alpha = 0.7$ in Q-learning. The temperature parameter T in Boltzmann exploration policy is set to decrease over the number of simulation iterations (each iteration simulates one full planning cycle including two periods): T equals to 3 for the first 150 iterations, 2 for the next 150 iterations, and 1 afterwards. The stop criteria is that the objective value has no significant improvement for 25 consecutive iterations.

In the first learning phase, both state and action need to be discretized. Figure 4 shows two schemes adopted for state discretization, where x takes the continuous S_j^t values. We let the element in the state vector take integer values from interval [-5, 5]. For the nonlinear scheme, when $x \geq 0$, $y = 5\sqrt{x}$; when $x < 0$, $y = -5\sqrt{(-x)/5}$. For action discretization, we tried two schemes: in scheme A, λ_j takes a value from the set {0, 25%, 50%, 75%, 1}; in scheme B, λ_j takes a value from the set {0, 10%, 20%, 30%, 40%, 50%, 60%, 70%, 80%, 90%, 1}.

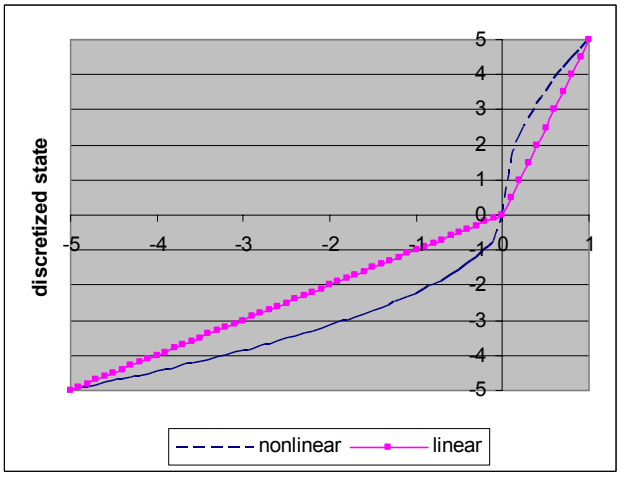


Figure 4: Phase 1 State Discretization Schemes

Table 3 compares the average results between different discretization schemes. For each different parametric setting, learning was carried out 10 times with different random seeds. It can be observed that the state discretization scheme affects the convergence speed, and the action scheme contributes to both speed and the quality of the solution. Both action discretization schemes found the best usage ratios in their respective action space. With a much finer grained scheme B, the learner managed to find a better result; however, more iterations were needed for the learning algorithm to converge. The best build plan derived from scheme B is presented in Table 4.

Further sensitivity analysis with lead time variance shows that when the standard deviations decrease, the total costs decrease as well.

6 CONCLUDING REMARKS

We have developed an RL based approach to solving a capacity constraint multi-period production planning problem in the fabrication/fulfillment manufacturing process. The near-optimal build plan for each planning period is learned by the RL learner through trial and error interaction with a

Table 3: Test Results

Discretization Scheme		Iterations in Each Phase		Period 1 Usage Ratio*		Total Costs
State	Action	1	2	C1	C2	
L**	A	245	330	75%	75%	2051
NL***	A	168	297	75%	75%	2049
L	B	879	267	70%	60%	1760
NL	B	787	246	70%	60%	1759
Optimal Solution				72%	53%	1670

* optimal usage ratios learned by first-phase learning
** linear state discretization scheme
*** nonlinear state discretization scheme

Table 4: The Best Build Plan*

Period	P1	P2	P3	P4
1	44	10	11	34
2	7	64	0	61

* total cost being 1759

Monte Carlo supply chain simulator. Through this simulation based approach, real-world situations such as multi-echelon BOM structure and manufacturing lead time randomness can be effectively addressed. To efficiently search in the very large state and action spaces, we designed a two-phase learning scheme, where the first phase learns the near-optimal usage ratios of the capacity, based on which a detailed build plan is derived in the second phase.

Preliminary numerical results have confirmed the validity of this approach, and we are testing the methodology with real-word model and data. There are several directions in which we can explore to enhance the performance of our algorithm. To improve learning efficiency, the simulation results from the first phase should be incorporated to initialize the second-phase *Q* table. Parallel learning execution can also be exploited to speed up the convergence.

ACKNOWLEDGMENTS

We are grateful to many colleagues for the help and discussions on the topics covered here. In particular, Dr. Feng Cheng deserves special mention for bringing up the original problem to the first author.

REFERENCES

Cao, H., G. Y. Lin, H. Xi and S. F. Smith. 2002. An Agent Based Enterprise Computing Framework for High Performance Supply Chain Simulation. In *Post-Conference proceedings of Int'l Conf. on Parallel and Distributed Processing Techniques and Applications (PDPTA'02)*, Las Vegas, Nevada, USA, June 24-27.

Cao, H., F. Cheng and S. Smith. 2003. *A Constraint-Based Method for Inventory-Service Optimization in a Fabrication/Fulfillment Manufacturing Process*. INFORMS Annual Meeting Atlanta, 2003.

Finke, A.D., D. J. Medeiros, and M. T. Traband. 2002. Shop Scheduling Using Tabu Search and Simulation. In *Proceedings of the 2002 Winter Simulation Conference*, ed. E. Yücesan, C.-H. Chen, J.L. Snowdon, and J.M. Charnes, 1013–1017. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers. 2002.

Joines, J., D. Gupta, M. A. Gokce, R. E. King and M. G. Kay. 2002. Supply Chain Multi-Objective Simulation Optimization. In *Proceedings of the 2002 Winter Simulation Conference*, ed. E. Yücesan, C.-H. Chen, J.L. Snowdon, and J.M. Charnes, 1306-1313. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers. 2002.

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

Nahmias, S. 1997. *Production and Operations Analysis*. McGraw-Hill Higher Education. 338–339.

Ólafsson, S. and J. Kim. 2002. Simulation Optimization. In *Proceedings of the 2002 Winter Simulation Conference*, ed. E. Yücesan, C.-H. Chen, J.L. Snowdon, and J.M. Charnes, 79-84. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers. 2002.

Shi, L. and S. Ólafsson. 2000, Nested Partitions Method for Global Optimization, *Operations Research*, (48) 3.

Van Roy, B., D. P. Bertsekas, Y. Lee, and J. N. Tsitsiklis. 1997. A Neuro-Dynamic Programming Approach to Retailer Inventory Management. In *Proceedings of the IEEE Conference on Decision and Control*.

AUTHOR BIOGRAPHIES

HENG CAO is a Staff Software Engineer at IBM T.J. Watson Research Center in Yorktown Heights, NY. She received her M.S. in Robotics from Carnegie Mellon University and is currently working toward her Ph.D. degree. In addition to modeling, analysis and simulation of supply chain systems, her interests include AI and data mining.

HAIFENG XI is an Advisory Software Engineer at the IBM T. J. Watson Research Center. He received the M.S. degree in Electrical and Computer Engineering from the University of Maryland. His current interests include Web application architecture, business integration, grid computing, and supply chain simulation.

STEPHEN F. SMITH is a Principal Research Scientist and the director of the Intelligent Coordination and Logistics Laboratory in the Robotics Institute, Carnegie Mellon University. His research interests include planning and scheduling, (re)configurable system architectures, machine learning and evolutionary computation, and intelligent systems in manufacturing, transportation and space applications.