Proceedings of the 36th
Conference on Decision & Control
San Diego, California USA • December 1997

A Neuro–Dynamic Programming Approach to Retailer Inventory Management

Benjamin Van Roy[†‡] (bvr@mit.edu)
Dimitri P. Bertsekas[‡] (dimitrib@mit.edu)
Yuchun Lee[†] (ylee@unica-usa.com)
John N. Tsitsiklis[‡] (jnt@mit.edu)

[†]Unica Technologies, Inc.
Lincoln North
Lincoln, MA 01773

[‡]Laboratory for Information and Decision Systems
Massachusetts Institute of Technology
Cambridge, MA 02139

Abstract

We discuss an application of neuro–dynamic programming techniques to the optimization of retailer inventory systems. We describe a specific case study involving a model with thirty-three state variables. The enormity of this state space renders classical algorithms of dynamic programming inapplicable. We compare the performance of solutions generated by neuro–dynamic programming algorithms to that delivered by optimized s–type ("order–up–to") policies. We are able to generate control strategies substantially superior, reducing inventory costs by approximately ten percent.

1 Introduction

Many important problems in engineering and operations research involve sequential decision-making under uncertainty, or *stochastic control*. Dynamic programming (Bertsekas, 1995) provides a general framework for studying such problems, as well as a suite of algorithms for computing optimal decision policies. Unfortunately, the overwhelming computational requirements of these algorithms render them inapplicable to most realistic problems. As a result, complex stochastic control problems that arise in the real world are usually addressed using drastically simplified analyses and/or heuristics.

An exciting new alternative that is more closely tied to dynamic programming is being developed in the emerging field of neuro-dynamic programming (Bertsekas and Tsitsiklis, 1996). This approach makes use of ideas from artificial intelligence involving simulation–based algorithms and functional approximation techniques such as neural networks. The outcome is a methodology for approximating dynamic programming solutions without demanding the associated computational requirements.

Over the past few years, neuro-dynamic programming methods have generated several notable success stories. Examples include a program that plays Backgammon at the world champion level (Tesauro, 1992), an elevator dispatcher that is more efficient than several heuristics employed in practice (Crites and Barto, 1996), and an approach to job shop scheduling (Zhang and Dietterich, 1996). Additional case studies reported by Bertsekas and Tsitsiklis (1996) further demonstrate significant promise for neuro-dynamic programming. However, neuro-dynamic programming is a young field, and the algorithms that have been most successful in applications are not fully understood at a theoretical level. Furthermore, there is a large conglomeration of algorithms proposed by researchers in the field, and each one is complicated and parameterized by many values that must be selected by a user. It is unclear which algorithms and parameter settings will work on a particular problem, and when a method does work, it is still unclear which ingredients were actually necessary for success. Because of this, application of neuro-dynamic programming often requires trial and error, in a long process of parameter tweaking and experimentation.

In this paper, we provide a brief overview of work directed towards developing a neuro-dynamic program-

0-7803-3970-8/97 $10.00 © 1997 IEEE

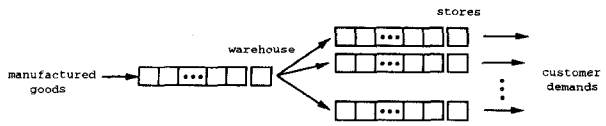

Figure 1: Schematic diagram of a retailer inventory system. the squares represent buffers at which inventory can be located at any given time. Buffers are associated with the warehouse, stores, and transportation delays.

ming approach for optimizing performance of retailer inventory systems (Nahmias and Smith, 1993). This is the problem of ordering and positioning retailer inventory at warehouses and stores in order to meet customer demands while simultaneously minimizing storage and transportation costs. This problem can also be viewed as a simple example from the broad class of multi-echelon inventory control problems that has received significant attention in the field of supply-chain management (Lee and Billington, 1993). Our presentation will brief and coarse, and we refer the interested reader to our full–length report (Van Roy, et al., 1997) for a more detailed account.

2 A Model of Retailer Inventory Systems

In this section we sketch the general structure of the model that is fully described in (Van Roy, et al., 1997). The characteristics of our model are largely motivated by the studies of (Nahmias and Smith, 1993) and (Nahmias and Smith, 1994). The general structure is illustrated in Figure 1 and involves several stages:

1. Transportation of products from manufacturers
2. Packaging and storage of products at a central warehouse
3. Delivery of products from the warehouse to stores
4. Fulfillment of customer demands using either store or warehouse inventory

Stochastic demands materialize at each store during each time period. Each unit of demand can be viewed as a customer request for the product. If inventory is available at the store, it is used to meet ongoing demands. In the event of a shortage, the customer will, with a certain probability, be willing to wait for a special delivery from the warehouse. If the customer is in fact willing to wait, the demand is filled by inventory from the warehouse (if it is available).

At the end of each day, the warehouse orders additional units of inventory from the manufacturers, and the stores place orders to the warehouse. The warehouse manager fills store orders as much as possible given current levels of inventory. As materials travel from manufacturers to the warehouse and from the warehouse to the stores, they are delayed by transportation times. Coupled with the uncertainty of future demands, these delays create the need for storage of inventory at stores.

The differing impact of inventory at the warehouse on costs and service performance makes it desirable to also maintain stock there. For example, inventory stored at the warehouse provides a greater degree of flexibility than that maintained at a single store. In particular, inventory stored at the warehouse can be used to fill special orders made by customers at any store (for individual customers who are willing to wait), and can also be sent to any store in the advent of a shortage of goods. On the other hand, a surplus of inventory at one store cannot be used to compensate for a shortage at another. Furthermore, storage costs at stores are often higher than at the warehouse.

3 Heuristic Policy

A heuristic policy for controlling the retailer inventory system was implemented and used as a baseline for comparison against neuro-dynamic programming approaches. The type of heuristic used is known as an s-type, or "order-up-to," policy and is accepted as a reasonable approach to problem formulations that have independent identically distributed demands. Examples of research where such policies are the focus of study are discussed in (Nahmias and Smith, 1993) and include (Nahmias and Smith, 1994).

The s–type policy we implemented is parameterized by two values: a warehouse order-up-to level and a store order-up-to level. Essentially, at each time step the inventory manager tries to order inventory such that all inventory at and expected to arrive at the warehouse is equal to the warehouse order-up-to level and all the inventory at or expected to arrive at any particular store is equal to the store order-up-to level.

4 Dynamic Programming Formulation

Let S be the state space of the retailer inventory system (each element corresponds to a particular combination of inventory levels in stores, the warehouse, as well as in transport). We associate two states $x_t, y_t \in S$ to any nonnegative integer time t. We refer to x_t as the "pre-decision state" and y_t as the "post-decision state." These variables identify the system state just before and just after orders are placed. The orders are represented as a decision u_t selected from a finite set U at each time step. The state evolves according to two difference equations: $x_{t+1} = f_1(y_t, w_t)$ and

$y_t = f_2(x_t, u_t)$, where f_1 and f_2 are some functions describing the system dynamics and w_t is a random noise term taken from a fixed distribution, independent from all information available up to time t. There is a cost $g(y_t, w_t)$ associated with the system affected by a noise term w_t while the post-decision state is y_t. This function takes into account storage as well as transportation costs.

A *policy* is a mapping $\mu : S \mapsto U$ that determines a decision as a function of pre-decision state, i.e., $u_t = \mu(x_t)$. The goal in stochastic control is to select an optimal policy (i.e., one that minimizes long-term costs). We express the long-term cost to be minimized as the expectation of a discounted infinite sum of future costs, as a function of an initial post-decision state, i.e.,

$$J^\mu(y) = E\left[\sum_{t=0}^{\infty} \alpha^t g(y_t, w_t) | y_0 = y, \mu\right].$$

Here, $\alpha \in (0,1)$ is a discount factor and $J^\mu(y)$ denotes the expected long-term cost given that the system starts in post-decision state y and is controlled by a policy μ. An optimal policy μ^* is one that minimizes J^μ simultaneously for all initial post-decision states, and the function J^{μ^*}, known as the value function, is denoted by J^*. In our case studies, we chose to use a discount factor of $\alpha = 0.99$. The reason is that policies will be evaluated in terms of average costs and setting the discount factor close to one makes the discounted problem resemble the average cost problem.

5 Neuro-Dynamic Programming

The main idea in neuro-dynamic programming is to approximate the mapping $J^* : S \mapsto \Re$ using an approximation architecture. An approximation architecture can be thought of as a function $\tilde{J} : S \times \Re^k \mapsto \Re$. NDP algorithms try to find a parameter vector $r \in \Re^k$ such that the function $\tilde{J}(\cdot, r)$ closely approximates J^*.

5.1 On-Line Temporal-Difference Learning

Variants of the temporal-difference algorithm (Sutton, 1988; Tsitsiklis and Van Roy, 1996) have been applied successfully to several large scale applications of NDP. The algorithm updates the parameter vector of an approximation architecture during each step of a single endless simulation. In particular, we start with an arbitrary parameter vector r_0 and generate a sequence r_t using the following procedure:

1. Given the initial pre–decision state x_0 of the simulator, generate a control u_0 by letting

$$u_0 = \arg\min_{u \in U} \tilde{J}(f_2(x_0, u), r_0);$$

2. Run the simulator using control u_0 to obtain the first post-decision state

$$y_0 = f_2(x_0, u_0);$$

3. More generally, at time t, run the simulator using control u_t to obtain the next pre-decision state

$$x_{t+1} = f_1(y_t, w_t),$$

and the cost $g(y_t, w_t)$;

4. Generate a control u_{t+1} by letting

$$u_{t+1} = \arg\min_{u \in U} \tilde{J}(f_2(x_{t+1}, u), r_t);$$

5. Run the simulator using control u_{t+1} to obtain the post-decision state

$$y_{t+1} = f_2(x_{t+1}, u_{t+1});$$

6. Update the parameter vector via

$$r_{t+1} = r_t + \gamma_t \Big(g(y_t, w_t) + \tilde{J}(y_{t+1}, r_t) - \tilde{J}(y_t, r_t) \Big) \nabla_r \tilde{J}(y_t, r_t),$$

where γ_t is a small step size parameter;

7. Return to step (3).

5.2 Active Exploration

In the case studies we will describe, performance depends critically on the incorporation of "active exploration" into the temporal–difference method. Note that the algorithm described in the previous section always updates the parameter vector to tune the approximate values $\tilde{J}(x, r)$ at states x visited by the current policy, which in turn are determined by the parameter vector r. In some sense, the exploration here is *passive*, i.e., only states that naturally occur on the basis of the current approximation to the value function are visited. By active exploration, we refer to a mechanism that brings about some tendency to visit a larger range of states.

Except for the steps involving generation of control decisions, the temporal-difference algorithm that we used *with* active exploration follows the same routine as that *without* active exploration. In particular, the algorithm can be described by the steps enumerated in the previous section, except with the equations of Steps (1) and (4) replaced by

$$u_0 = n_0 + \arg\min_{u \in U} \tilde{J}(f_2(x_0, u), r_0),$$

and

$$u_{t+1} = n_t + \arg\min_{u \in U} \tilde{J}(f_2(x_{t+1}, u), r_t),$$

respectively, where each n_t is a noise term. Note that the only difference is the addition of a noise term. The structure of the noise term is described on a case-by-case basis in the next section.

6 Results With the NDP Approach

In this section, we present the results obtained from applying the NDP approaches we developed to the retailer inventory management problem. Through our development, much of which occurred in the process of experimentation, we arrived at an approach that was successful relative to the heuristic s-type policies.

6.1 Initial Experiments With a Simple Problem

The first set of experiments involved optimization of a very simple retailer inventory system. The purpose of these experiments was to debug the software packages developed in the initial stages of research and also to ensure that the NDP methodologies worked reasonably well on a simple problem, before moving to complex situations.

The system included only one store in addition to the warehouse. There was no delay for goods ordered by the warehouse, and there was a delay of only one time unit between the warehouse and the store. There were therefore only three state variables involved (each corresponding to the quantity of goods within a buffer). The list of parameter settings for this problem are provided in the table below.

number of stores	1
delay to stores	1
delay to warehouse	0
production capacity	10
warehouse capacity	50
store capacity	50
probability of customer waiting	1
cost of special delivery	10
warehouse storage cost	1
store storage cost	2
mean demand	6.2
demand stdev	6.2
shortage cost	50

As a baseline for comparison, we developed an s-type policy, optimizing the order-up-to levels associated with the warehouse and store. The optimal order-up-to levels turned out to be 10 for the warehouse and 16 for the store. The corresponding average cost was 51.7.

We used an approximation architecture consisted of a multilayer perceptron with ten hidden nodes in a single hidden layer. Three features were taken as input to the network, each a normalized version of one of the state variables.

Next, we tried adding a small degree of exploration to the on-line temporal difference method. In particular, each time a decision was generated using the approximation architecture $\tilde{J}$, a noise term was added to the decision. Recall that there are two decision variables: the warehouse order and the store order. The noise term was generated by adding a unit normal random variable to each decision variable, rounding off to the closest integer in each case, and then making sure the decision variables stayed within their limits. That is, if the noise term made a variable negative, the variable was set to zero, and if the noise term made a variable too large (e.g., having a warehouse order greater than the production capacity), then the variable was set to its maximum allowable value.

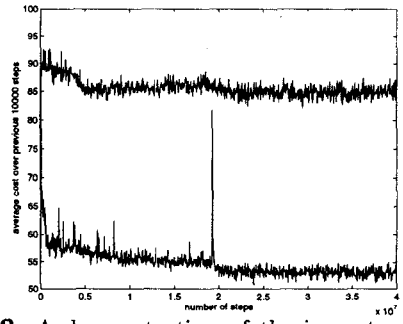

Figure 2: A demonstration of the importance of exploration. The two plots show the evolution of average cost using the on-line temporal-difference method with (lower plot) and without (higher plot) exploration.

With the extra exploration term, the on-line temporal-difference method essentially matched the performance of the heuristic (which is probably close to optimal, given the simplicity of the problem at hand). Figure 2 displays the evolution of average cost as the algorithm progresses in tuning the parameters of the multilayer perceptron, in experiments performed both with and without active exploration. In both cases, a step size $\gamma_t = 0.01$ was used for the first 2×10^7 time steps, and a step size $\gamma_t = 0.001$ was used for the next 2×10^7 time steps.

Note that in the graph of Figure 2 associated with the exploratory version of the algorithm, the average cost is computed during the execution of the algorithm, and is thus affected by the active exploration. In particular, a policy based on the final approximate value function without any exploratory term should perform better than the policy with active exploration (the exploration is there to improve the "learning and discovery" process that the algorithm goes through, rather than to improve performance of a policy at any given time). Indeed, a simulation employing a non-exploratory policy based on the final approximate value function generated an average cost of 52.6, which was slightly better than average costs sampled during execution of the exploratory on-line algorithm.

6.2 Case Study

With the success of the on-line temporal-difference method on a simple problem, a subsequent set of experiments was conducted on a more complex test bed. The parameters used for the retailer inventory management problem of this case study are given in the table below.

number of stores	10
delay to stores	2
delay to warehouse	2
production capacity	100
warehouse capacity	1000
store capacity	100
probability of customer waiting	0.8
cost of special delivery	0
warehouse storage cost	3
store storage cost	3
mean demand	8.6
demand stdev	9.8
shortage cost	60

Once again, an s–type heuristic policy was developed by optimizing over order-up-to levels. Since the properties of all stores were identical, we assumed that the order up-to-levels of all stores should be the same, and there were again only two variables to optimize: a warehouse order-up-to level and a store order-up-to level. The optimal order-up-to levels were 330 for the warehouse and 23 for each store. The corresponding average cost was 1302.

In the simple problem of the previous section, there were only two inventory sites for which orders had to be placed. In the more complex problem of this section, on the other hand, there are eleven inventory sites, and exhausting all possible combinations of orders that can be made for these eleven sites would take too long. In particular, the minimizations carried out in steps (1) and (4) of the on-line temporal-difference method would be essentially impossible to carry out. Because of this, we constrained the decision space to a more manageable subset.

First of all, we represented decisions in terms of two variables: a warehouse order and a store order-up-to level. Given particular values for the two variables, the individual store orders would be set to exactly what the s–type policy would set them to given the store order-up-to level. Note, however, that unlike the case of the heuristic s–type policy, the store order-up-to-level here is chosen at each time step, rather than taken to be a fixed constant.

The approximation architectures employed in this case study involved the use of 22 features, given by normalized versions of the following variables:

(1) total inventory at stores
(2) total inventory to arrive at stores in one time step
(3) total inventory to arrive at stores in two time steps
(4) inventory at warehouse
(5) inventory to arrive at warehouse in one time step
(6) inventory to arrive at warehouse in two time steps
(7) inventory to arrive at warehouse in three time steps
(8)-(14) the squares of (1)-(7)
(15) variance among stores of inventory levels
(16) variance among stores of inventory levels plus inventory to arrive in one time step
(17) variance among stores of inventory levels plus inventory to arrive within two time steps
(18) the product of (1) and (4)
(19) the product of (4) and the sum of (1) through (3)
(20) the sum of (4) through (7) times the sum of (1) through (3)
(21) the sum of (4) through (6) times the sum of (1) through (3)
(22) the product of (3), (4), and (7)

By variance among stores (as in features (15) through (17)), we mean the average among stores of the square of the difference between quantities associated with each store and the average of such quantities over the stores.

For active exploration, noise terms were added to the decisions generated using the approximate value function at each step of the temporal-difference algorithm. The way noise terms were added is completely analogous to the method employed in the previous section, except that this time the noise term added to the warehouse order involved a normal random variable with a mean of zero and a standard deviation of five. Furthermore, the noise terms added to the store orders were independent from one another.

We began by using an architecture that was linear in the features. This architecture delivered promising performance. Two variations on the initial architecture/algorithm were explored. One involved replacing the linear function approximator with a multilayer perceptron with a single hidden layer of ten nodes. The other variation used the original linear architecture, but with an increased degree of exploration. Here the random noise term added to the warehouse order involved a normal random variable with a standard deviation of ten, and that added to the store orders involved a normal random variable with a standard deviation of two. Figure 3 charts the evolution of average cost during the execution of the temporal-difference algorithm in all three of these cases. In the two cases involving linear architectures, the step size was maintained at $\gamma_t = 0.0001$, while with the multilayer perceptron-based architecture, the step size was $\gamma_t = 0.0001$ during the first 1.5 million steps and $\gamma_t = 0.00001$ thereafter. These step size schedules were chosen after some trial

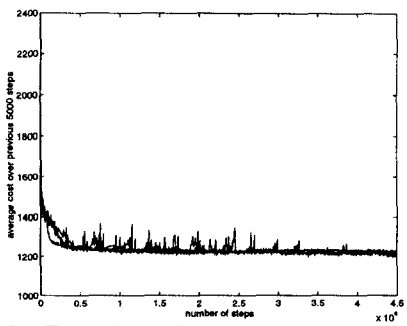

Figure 3: Evolution of average cost using the on-line temporal-difference method with a linear architecture and a small degree of exploration, a single hidden layer, 10 hidden node, multilayer perceptron and a small degree of exploration, and a linear architecture with a greater degree of exploration.

and error.

All three variants of on-line temporal-difference methods generated policies superior to the heuristic. In particular, the linear architectures generated policies with average costs of 1179 (less active exploration) and 1181 (more active exploration), while the multilayer perceptron architecture led to an average cost of 1209. Hence, the best policy cut costs by about ten percent relative to the heuristic.

7 Conclusions

Through this study, we have demonstrated that NDP can provide a viable approach to advancing the state-of-the-art in retailer inventory management. The method we have developed outperformed a well-accepted heuristic approach in two case studies.

Though the problems we solved in this research were truly complex from a technical standpoint, not much effort was directed at ensuring that the models reflected all the practical issues inherent in real-world retailer inventory systems. Further research is required to translate the methods we have developed into those that could be truly beneficial in a real-world application.

Acknowledgments

This material is based upon work supported by the National Science Foundation under award number 9561500. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the authors and do not necessarily reflect the views of the National Science Foundation.

References

Bertsekas, D. P. (1995) *Dynamic Programming and Optimal Control*, Athena Scientific, Bellmont, MA.

Bertsekas, D. P., and Tsitsiklis, J. N. (1996) *Neuro-Dynamic Programming*, Athena Scientific, Bellmont, MA.

Crites, R. H., and Barto, A. G (1996) "Improving Elevator Performance Using Reinforcement Learning," *Advances in Neural Information Processing Systems 8*, Touretzky, D. S., Mozer, M. C., and Hasselmo, M. E., eds., MIT Press, Cambridge, MA.

Lee, H. L., and Billington, C. (1993) "Material Management in Decentralized Supply Chains," *Operations Research*, vol. 41, no. 5, pp. 835-847.

Nahmias, S., and Smith, S. A. (1993) "Mathematical Models of Inventory Retailer Systems: A Review," *Perspectives on Operations Management, Essays in Honor of Elwood S. Buffa*, Sarin, R., editor, Kluwer Academic Publishers, Boston, MA, pp. 249-278.

Nahmias, S., and Smith, S. A. (1994) "Optimizing Inventory Levels in a Two Echelon Retailer System with Partial Lost Sales," *Management Science*, Vol. 40, pp. 582-596.

Sutton, R. S. (1988) "Learning to Predict by the Methods of Temporal Differences," *Machine Learning*, vol. 3.

Tesauro, G. J. (1992) "Practical Issues in Temporal–Difference Learning," *Machine Learning*, vol. 8.

Tsitsiklis, J. N., and Van Roy, B. (1996) "An Analysis of Temporal–Difference Learning with Function Approximation," to appear in *IEEE Transactions on Automatic Control.*

Van Roy, B., Bertsekas, D. P., Lee, Y., and Tsitsiklis, J. N. (1997) "A Neuro–Dynamic Programming Approach to Retailer Inventory Management," Unica Technologies, Lincoln, MA.

Zhang, W., and Dietterich, T. G. (1995) "A Reinforcement Learning Approach to Job Shop Scheduling," *Proceedings of the IJCAI.*