

A reinforcement learning approach to stochastic business games

KIRAN KUMAR RAVULAPATI¹, JAIDEEP RAO² and TAPAS K. DAS³

¹*Delta Technology, Atlanta, GA 30354, USA*

E-mail: kirankumar@delta.com

²*Pilgrim Software, Tampa, FL 33618, USA*

E-mail: raoj@pilgrimsoftware.com

³*Department of Industrial and Management Systems Engineering, University of South Florida, Tampa, FL 33620, USA*

E-mail: das@eng.usf.edu

Received February 2001 and accepted November 2003

The Internet revolution has resulted in increased competition among providers of goods and services to lure customers by tearing down the barriers of time and distance. For example, a home buyer shopping for a mortgage loan through the Internet is now a potential customer for a large number of lending institutions throughout the world. The lenders (players, in generic game theory nomenclature) seeking to capture this customer are involved in a nonzero-sum stochastic game. Stochastic games are among the least studied and understood of the management science problems, and no computationally tractable solution technique is available for multi-player nonzero-sum stochastic games. We now develop a computer-simulation-based machine learning algorithm that can be used to solve nonzero-sum stochastic game problems that are modeled as competitive Markov decision processes. The methodology based on this algorithm is implemented on a supply chain inventory planning problem with a limited state space. The equilibrium reward obtained from the stochastic game problem is compared with a logical upper bound obtained from the corresponding Markov decision problem in which a single decision maker (player) is substituted for all the competing players in the game. Several numerical versions of the problem are studied to assess the performance of the methodology. The results obtained from our methodology for the inventory planning problems are within 0.8% of the upper bound.

1. Introduction

The explosion in E-commerce has created an unprecedented level of competition in today's markets. The struggle to capture customers, which even a few years ago was mostly confined to the neighborhood bricks and mortar type retailers/producers, has suddenly opened up to the rest of the world. Decision-making in such a scenario can often be characterized and studied using a nonzero-sum stochastic game model; a nonzero-sum game is where the gain of one player is not necessarily the loss of another player. Such multi-player game situations have become extremely commonplace; however, not much is currently known in terms of how to analyze these games and how to examine the possible rewards. Thus, businesses are left largely to their gut feelings when making critical business decisions.

This paper presents the development of a new Reinforcement Learning (RL) algorithm and its complete implementation methodology for finding (near-) equilibrium points of nonzero-sum stochastic games having state spaces that are relatively much larger than the problems that have been

studied in the literature. The methodology is tested on an inventory planning problem from the supply chain arena, which is of significant current interest. The supply chain considered could either be in the E-business domain or in the traditional domain. The choice of the problem was partially motivated by a recent paper by Anupindi *et al.* (2001) that presents a thorough discussion of the supply chain's general competitive framework. We are not aware of any other method that can be effectively implemented on stochastic games of comparable complexity both from the standpoint of modeling and also the size of the state space. The methodology takes advantage of a growing area of machine learning, which is founded on the well-known principles of stochastic dynamic programming and stochastic approximation.

RL allows the decision-making agents (or players) to learn from the rewards obtained from the actions taken over time and thus reach (near-) optimal strategies. RL-based methods work with simulated models of the system and can thereby deal with problems having complex reward and stochastic structures. RL can also integrate within it

various function approximation methods (regression, neural networks, etc.) which makes it possible to address problems with larger state spaces. Convergent and optimal algorithms for problems having a single decision maker, that are modeled as Markov decision problems, have been proposed in the RL literature (Watkins, 1989; Abounadi *et al.*, 1998; Sutton and Barto, 1998; Gosavi, 2000). The algorithm presented in this paper is an extension of the single agent RL framework to stochastic games under an average reward criterion. Since the RL algorithm is based on the value iteration algorithm of dynamic programming, our focus is on the class of stochastic games that has the structure of a Competitive Markov Decision Process (CMDP). The results of this study motivate further research to develop a much needed analytical/computational tool base which would allow the corporate world to examine competitive business policies.

Stochastic games are evolutionary in nature, where the players take actions with an objective of maximizing their benefits while anticipating others' actions. As a game evolves, the players learn more about the game outcomes and the tendencies of the other players. In most practical cases, a game eventually reaches an *equilibrium point*. Of the many characterizations of an equilibrium point, the most common one is called the *Nash equilibrium* (Nash, 1951). While volumes of literature exist for deterministic games, the theory of nonzero-sum stochastic games is in its infancy (Filar and Vrieze, 1997). Single or multiple equilibria exist for discounted reward nonzero-sum games with a finite state and action spaces with bounded rewards. For average reward nonzero-sum games, the existence of an equilibrium point is guaranteed only for irreducible games with finite state and action spaces and bounded rewards. However, identifying an equilibrium strategy for average reward nonzero-sum games is very challenging, especially for problems that are encountered in practice. Recently, RL has been used in attempts to solve simple test cases of both deterministic and stochastic games (Littman, 1994; Erev and Roth, 1998; Hu and Wellman, 1998). Encouraged by these applications and our experiences with the use of RL techniques for single player average reward stochastic decision-making problems (Das *et al.*, 1999; Paternina and Das, 2000; Gosavi *et al.*, 2002), we developed a multi-agent RL algorithm for games with average reward as the decision criterion. The average reward criterion, although it provides little analytical tractability compared to the discounted criterion, has significant practical relevance for performance measures such as: (i) demand fill rate, stock level and profit for inventory planning problems; (ii) number and average size of the loans (policies) served in banking (insurance); and (iii) the percentage of market held in the retail industry. Multi-agent RL applications to-date on stochastic games include the discounted reward Q -learning algorithm of Hu and Wellman (1998), and the minimax Q -learning algorithm for a two-player zero-sum Markov game of Littman (1994). We believe that our paper presents the first attempt to develop a

computational algorithm for solving large-scale nonzero-sum stochastic games with an average reward criterion.

2. The stochastic game model

A stochastic game is an extension of a single player stochastic decision-making problem and a brief description of the latter is in order. In a single player problem, the system is characterized by a numerical attribute generally referred to as the *state* of the system. The system is usually dynamic in the sense that the state changes from time to time in a stochastic manner. State changes in the system are influenced by the *actions* selected by the decision maker and are guided by the state transition probabilities. Each action, together with the state change between the decision epochs results in some rewards (or costs). The objective of studying such a problem is to find the actions in the different decision-making states that maximize some function of the net reward. A class of such single player problems is called Markov or Semi-Markov Decision Problems (MDPs/SMDPs). The modeling framework for this class of problems is referred to as *stochastic dynamic programming* (Bellman, 1957).

A natural extension of the single player decision problem is the case where there is more than one decision maker, and each decision maker has a different competitive objective. Such problems are referred to as stochastic games, a special class of which having Markovian state transition probability structure is called CMDPs. A description of a stochastic game, as given in Filar and Vrieze (1997, pp. 156–157) follows. Let the state of a system at any time belong to a finite set $E = \{1, 2, \dots, |E|\}$. At different points in time, players (decision makers) have the ability to influence the course of the system by choosing actions from a finite set of available actions. The time epochs in which actions are taken are also called decision epochs. System states are assumed to be fully observable to the players. However, an action chosen by a player is not observable by other players at the time of decision-making. Let $m^i(l)$ denote the number of actions available to player i in state $l \in E$. Then $A^i(l) = \{1, 2, \dots, m^i(l)\}$ denotes the action set for player i in state $l \in E$. To simplify the exposition, hereafter, we assume two players. Let, in state l , $a^1 \in A^1(l)$ and $a^2 \in A^2(l)$ denote the independent actions chosen by the players. Note that a^1 and a^2 are functions of l , but are written without l in order to simplify notation. Also, let $r^1(l, a^1, a^2, q)$ and $r^2(l, a^1, a^2, q)$ denote the immediate payoffs (rewards) earned by the players 1 and 2 when actions a^1 and a^2 are chosen in state l resulting in a transition to state q . Then the system state at the next decision epoch is determined by a stochastic transition probability matrix $\mathbf{P}(l)$ given as:

$$\mathbf{P}(l) : A^1(l) \times A^2(l) \rightarrow \left\{ \mathbf{x}; \mathbf{x} \in \Re^{|E|}, x_j \geq 0, \sum_{j=1}^{|E|} x_j = 1 \right\},$$

where $A^1(l) \times A^2(l)$ denotes the matrix of action combinations over which the transition probability vectors (\mathbf{x}) are defined. A vector of the $\mathbf{P}(l)$ matrix, when $a^1 \in A^1(l)$ and $a^2 \in A^2(l)$ are the actions chosen by players 1 and 2 respectively, can be given as

$$\begin{aligned} \mathbf{P}(l, a^1, a^2) \\ = (P(1|l, a^1, a^2), P(2|l, a^1, a^2), \dots, P(|E||l, a^1, a^2)), \end{aligned}$$

where $P(q|l, a^1, a^2)$ indicates the probability of system transition from state l to state q as a result of the chosen actions a^1 and a^2 , and $\sum_{q=1}^{|E|} P(q|l, a^1, a^2) = 1$.

If $r^1(l, a^1, a^2, q) + r^2(l, a^1, a^2, q) = 0$, the game is called a zero-sum game, otherwise it is called a nonzero-sum game. Strategies for players are rule vectors that tell them what action to choose in any given system state. Two types of strategies that are of interest here are *pure strategy* and *mixed strategy*. Let the strategy of player 1 be given as $\mathbf{s}^1 = (s^1(1), s^1(2), \dots, s^1(|E|))$, where $s^1(l) = (f^1(l, 1), f^1(l, 2), \dots, f^1(l, m^1(l)))$. The term $f^1(l, n)$ denotes the probability of choosing action n in state l by player 1, and $\sum_{n=1}^{m^1(l)} f^1(l, n) = 1$. A strategy $s^1(l)$, as defined for player 1 in state l , is called *pure* if $f^1(l, a^1) \in \{0, 1\}$ for all $a^1 \in A^1(l)$ and $l \in E$, else the strategy is called *mixed*.

Let the mixed strategy spaces for players 1 and 2 be denoted by S^1 and S^2 . A strategy $(\mathbf{s}^1, \mathbf{s}^2) \in S^1 \times S^2$ is a Nash equilibrium point of a stochastic game Γ if:

$$\begin{aligned} v^1(\mathbf{s}^1, \mathbf{s}^{2*}) &\leq v^1(\mathbf{s}^{1*}, \mathbf{s}^{2*}) \quad \forall \mathbf{s}^1 \in S^1, \\ v^2(\mathbf{s}^{1*}, \mathbf{s}^2) &\leq v^2(\mathbf{s}^{1*}, \mathbf{s}^{2*}) \quad \forall \mathbf{s}^2 \in S^2, \end{aligned}$$

where $v^1(\cdot, \cdot)$ denote the overall payoff from a strategy for player 1 under a preselected criteria (e.g., average reward, discounted reward etc.) and a maximization scenario. Thus, Nash equilibrium dictates that unilateral deviations from $(\mathbf{s}^{1*}, \mathbf{s}^{2*}) \in S^1 \times S^2$ either by player 1 or player 2 are not worthwhile. In the next section, we give an overview of the multi-agent RL framework.

3. A RL approach

Stochastic games are naturally plagued by the same drawbacks as MDP/SMDP, such as: (i) the difficulty in developing the transition probability matrices (called the *curse of modeling*); and (ii) the difficulty in solving for the “values” of each element of the state space in every iteration (called the *curse of dimensionality*). The RL approach addresses the modeling issue by using a simulation-based model that can accommodate almost any complexity of a system. The issue of dimensionality is addressed to an extent by the use of an asynchronous value updating approach. The theory of RL is founded on two important scientific principles: Bellman’s equation, and the theory of stochastic approximation (Robbins-Monro type algorithm (Robbins and

Monro, 1951). We now briefly explain how a multi-agent RL system works with the help of Fig. 1 that shows a schematic of a two-agent system.

Any learning model contains four basic elements: (i) the system environment (simulation model); (ii) the learning agents (players); (iii) a set of actions for each agent (action spaces A); and (iv) the system response (immediate reward). At a decision-making state (say, l), the learning agents select an action vector ($\mathbf{a} = (a^1, a^2) \in A$). These actions and the system random variables collectively lead the system to the next decision-making state (say, q). As a consequence of this action choice (\mathbf{a}) and the resulting transition from state l to state q , the agents get their reward responses ($r^1(l, \mathbf{a}, q)$ and $r^2(l, \mathbf{a}, q)$) from the system environment. Using these responses, the agents update their knowledge base (Q -values, also called *reinforcement values*) for the most recent state and action combination encountered (l, \mathbf{a}). This completes a learning step. At this time the agents select their next actions based on the Q -values for the current state (q) and the available action combinations. The policy of selecting an action based on the Q -values is often violated by adopting a random choice. This is known as *exploration*, since this allows the decision makers to explore other possibilities. This learning process repeats and the agent performances continue to improve. A sample greedy updating scheme for the Q -value for player i after the visit to the state-action combination (l, \mathbf{a}) at the $(n + 1)$ th step (denoted as $Q_{n+1}^i(l, \mathbf{a})$) can be given as:

$$\begin{aligned} Q_{n+1}^i(l, \mathbf{a}) &= (1 - \alpha_n)Q_n^i(l, \mathbf{a}) \\ &\quad + \alpha_n[r^i(l, \mathbf{a}, q) - \rho_n^i + \max_{b \in A} Q_n^i(q, b)], \end{aligned}$$

where α_n is the learning rate at step n (which is decayed after every step), and ρ_n^i is a scalar for which current average reward of player i can be used. The current average reward can be obtained by dividing the running sum of rewards for agent i by the number of decision steps encountered. In some RL algorithms (as in ours), the current average reward values are also learned using a learning mechanism to avoid large fluctuations. This is referred to in the literature as two time-scale learning. After continuing learning for a large number of steps, if the Q -values for all state-action combinations converge (which depends on the nature of the system and the RL algorithm used), the learning process is said to be complete. The converged Q -values are then used to construct separate $A \times A$ value matrices $\{Q^i(l, a^1, a^2) : a^1, a^2 \in A\}$ for each player and each state. The matrix game formed by these matrices is solved to find the Nash equilibrium policy for each state of the system.

In what follows, we present the average reward RL algorithm for solving competitive Markov decision problems that was developed by this research effort. But, before formal presentation of the algorithm, we give a step-by-step description of the algorithm. The key elements of this

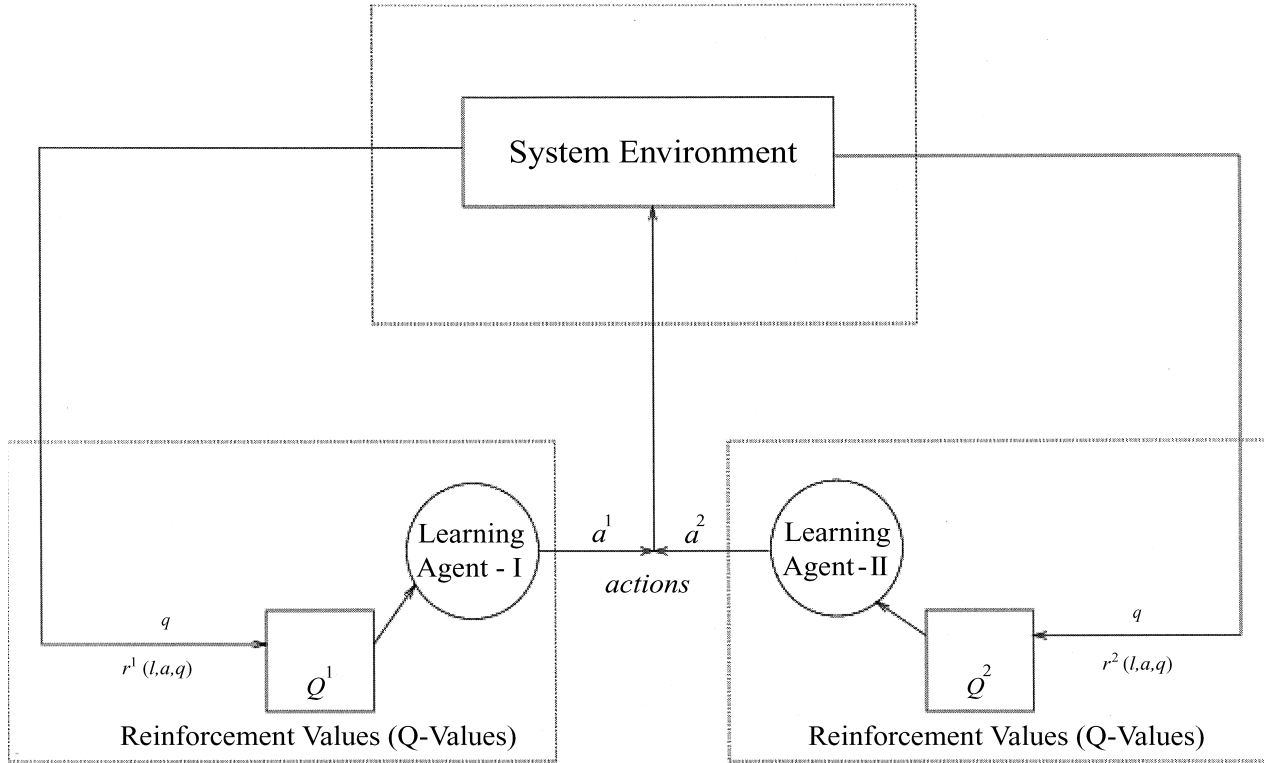


Fig. 1. A multi-agent RL model.

algorithm are a system simulation program within which the decision-making component is embedded. The simulation program simulates the actions and the resulting system transitions, and also generates the system response (rewards). The decision component of the algorithm then uses the system responses to learn Q -values, which is then used to construct an equivalent matrix game for a stochastic game. An existing method is used to obtain the Nash equilibrium policy from the matrix game and to evaluate the corresponding rewards of the players. The step numbers used in the following explanation correspond exactly to the step numbers of the algorithm presented in Section 3.1.

1. As a first step, the following variables are initialized: the Q -value matrices for each player and each state, the current average reward values for the players, the input parameters for two different learning rates and an exploration rate, and the quantities $R^i(l, k)$ defined as the sum of the $Q^i(\cdot)$ -values for all combinations of action choices of the remaining players when player i chooses action k in state l .
2. Assume that at the iteration count $n < \text{MaxSteps}$, the system state is l . MaxSteps is the termination criterion.
 - (a) For each player i , assign the probability value of one minus the current exploration rate for choosing action k with the highest $R^i(l, k)$ value. All of the remaining actions (that are possible choices for an exploratory action) are assigned equal portions of the

exploration probability. Uniform(0,1) random variables are generated for selecting action choices for the players according to the above probabilities.

- (b) The system simulation is initiated with the chosen actions and continued till the next decision-making state (say, q) is reached. Immediate rewards of the players resulting from the competitive action choices are generated.
- (c) At this time, for each player, the Q -value for the most recent state-action combination is updated using a learning scheme. Also updated is the player's average reward value (ρ_n^i) which is used in Q -value updating. This type of a dual updating scheme is called a *two time-scale updating* procedure.
- (d) The learning parameters and the exploration parameter are now updated (decayed). We have used a well-known decay scheme proposed by Darken *et al.* (1992).
- (e) Update the current system state as q and the iteration count as $n + 1$.
- (f) If the current iteration count is less than the MaxSteps , the algorithm continues at Step 2(a). Else, it moves to Step 3. The value of "MaxSteps" is selected (based on preliminary experimentation) such that after many iterations, the Q -values do not change appreciably anymore and, perhaps, converge. These Q -values are used to construct an equivalent matrix game.

3. Compute the Nash equilibrium for each state from the matrix game using an existing software GAMBIT[©] available at www.hss.caltech.edu/gambit. The software provides various approaches to finding one or more equilibrium points, such as solving the linear complementarity problem, a simplicial subdivision method, and Lyapunov function method, among others.
4. Start a fresh simulation of the system with the Nash equilibrium policy and estimate the rewards of the players. Note that the values of ρ_n^i at the completion of the learning phase give somewhat lower estimates of rewards, since they are averaged over all “good” and “bad” actions taken by the agents during learning.

In what follows, we give a formal description of the RL algorithm. Thereafter, we examine the efficiency of the RL algorithm through its application to a noncooperative inventory planning problem.

3.1. A RL algorithm for average reward competitive MDPs

1. Let N be the number of players and the iteration count $n = 0$. Initialize reinforcement values $Q^i(l, a^1, a^2, \dots, a^N) = 0$ for all players $i \in \mathcal{N} = \{1, 2, \dots, N\}$, states $l \in E$, and actions $a^i \in A^i(l)$, which denotes the action space for player i at state l . The above Q -values form N -dimensional matrices ($\mathbf{Q}^i(l)$, for each player and state combination) of size $|A^1(l)| \times |A^2(l)| \times \dots \times |A^N(l)|$, where $|A^i(l)|$ denotes the cardinality of $A^i(l)$. Set the average reward for the players $\rho_0^i = 0$ for all $i \in \mathcal{N}$. Define $R_n^i(l, k) = \sum_{\{a^1, \dots, a^N\} \setminus a^i} Q_n^i(l, a^1, \dots, a^i = k, \dots, a^N)$, which is the sum of all entries of a reinforcement value matrix $\mathbf{Q}_n^i(l)$ corresponding to action k of player i and all possible combinations of other players' actions at the n th iteration. Initialize the values of $R_0^i(l, k)$ to zero. Initialize the input parameters ($\alpha_0, \beta_0, \gamma_0, \alpha_\tau, \beta_\tau, \gamma_\tau$) for choosing the learning rates (α and β) and exploration rate (γ) according to the Darken-Chen-Moody (DCM) search-then-converge scheme in Step 2(d) (Darken *et al.*, 1992).
2. While $n < \text{MaxSteps}$, do
 - Let the system state at iteration n is $l \in E$.
 - (a) For all $i \in \mathcal{N}$ and $k \in A^i(l)$, compute the probability of choosing action k by player i , $P\{a^i = k\}$, denoted by $p^i(l, k)$, as follows.

$$p^i(l, k) = \begin{cases} (1 - \gamma_n) & \text{if } R_n^i(l, k) \\ & = \max_{k \in A^i(l)} R_n^i(l, k), \\ \gamma_n \times 1/(|A^i(l)| - 1), & \text{otherwise.} \end{cases}$$

As per above, the action $a^i = k$, for which $R_n^i(l, k)$ is maximum is chosen with probability $(1 - \gamma_n)$, and the rest of the actions are chosen with an equal fraction of the exploration probability γ_n at the current step n of the algorithm.

Then, for each player $i \in \mathcal{N}$, generate a random variable y^i from a uniformly distributed random vari-

able $U(0, 1)$ to select an action as follows. Find, using $p^i(l, k)$, the cumulative probability distribution function $F^i(l, k)$, over the range of actions $k \in A^i(l)$. Choose action $a^i = k$, for which the condition $F^i(l, k - 1) < y^i \leq F^i(l, k)$ is satisfied.

- (b) Simulate the system with the chosen actions for all the players until the next decision epoch. Let the system state at the next decision epoch be q and $r^i(l, \mathbf{a}, q)$ be the reward for player i corresponding to the action combination \mathbf{a} .
- (c) Update the Q -value for the most recent state-action combination $Q_{n+1}^i(l, \mathbf{a})$ and the average reward ρ_{n+1}^i for all the players $i \in \mathcal{N}$ as follows. Let $\hat{\mathbf{a}} = (\hat{a}^1, \hat{a}^2, \dots, \hat{a}^N)$ denote a possible action vector in state q , where $\hat{a}^i \in A^i(q)$. Define $Q_{\text{exp}}^i(q)$ as the expected Q -value for player i in state q calculated over all possible action choices of the players. Thus, we have that:

$$Q_{\text{exp}}^i(q) = \sum_{\hat{\mathbf{a}} \in \{A^1(q) \times \dots \times A^N(q)\}} (p^1(q, \hat{a}^1) \times \dots \times p^N(q, \hat{a}^N)) Q_n^i(q, \hat{\mathbf{a}}).$$

With the above, the two time-scale updating equations are given as

$$Q_{n+1}^i(l, \mathbf{a}) = (1 - \alpha_n) Q_n^i(l, \mathbf{a}) + \alpha_n (r^i(l, \mathbf{a}, q) - \rho_n^i + Q_{\text{exp}}^i(q)), \quad (1)$$

and

$$\rho_{n+1}^i = (1 - \beta_n) \rho_n^i + \beta_n \left[\frac{n \rho_n^i + r^i(l, \mathbf{a}, q)}{(n + 1)} \right]. \quad (2)$$

- (d) Find the updated values of the learning rate parameters α_{n+1} and β_{n+1} , and the exploration rate parameter γ_{n+1} using the DCM scheme as given (in generic notation ϕ) below:

$$\phi_{n+1} = \frac{\phi_0}{1 + u}, \quad \text{where } u = \frac{n^2}{\phi_\tau + n},$$

where ϕ_0 is the starting value of the rate parameter, and ϕ_τ is a large constant chosen suitably to control decay of the rate parameter ϕ .

- (e) Set $l \leftarrow q$ and $n \leftarrow n + 1$.
 - (f) If $n < \text{MaxSteps}$, go to Step 2(a), else go to Step 3.
3. Compute the Nash equilibrium in each state $l \in E$ using the final form of the Q -matrices $\{\mathbf{Q}^i(l)\}$ for all $i \in \mathcal{N}$, and $l \in E$. (This step can be accomplished for problems with limited state spaces using GAMBIT[©]).
 4. Run a fresh simulation of the system with the fixed Nash equilibrium policy as obtained from Step 3, without the learning component, to estimate the average system reward.

4. A multiple retailer/warehouse inventory planning system: a test bed for the competitive RL algorithm

To develop an insight into the abilities of a RL-based approach to handle nonzero-sum stochastic game problems under the average reward criterion, we adopted an inventory planning problem for a test bed. The problem has the desired characteristics, such as finite state and action spaces, and a bounded reward. The inventory system consists of N retail outlets belonging to independent agents dealing with a single product, who are interested in maximizing their own performance, rather than the entire system. The retail agents may pool inventories at W central warehouses. The pooling of inventory in central warehouses is a common practice among businesses to deal with the demand uncertainty and to take advantage of the transportation and storage costs. Each of the N retail locations experiences demand independently from the other outlets. The warehouses do not experience any demand. Inventory decisions for the warehouses are made by the retailers. For simplicity, it is assumed that each retailer controls one outlet. The retailers share the residual inventory (inventory left after realizing the daily demand) at local outlets as well as the stocks at the central warehouses to meet the residual demands (unsatisfied demands) at the outlets at the end of a business day. Note that the warehouse inventories are used only for meeting residual demands at the end of a day. In such a system, there are two stages of decision-making that are separated in time. The first stage involves the inventory decision that is made before realizing the demand. This stage includes the decision of stock levels at local as well as central locations. The retailers choose retail and warehouse stock levels unilaterally without any knowledge of the decisions taken by other retailers (pure competition). However, it is assumed that at the end of the business day the inventory decisions made earlier are revealed. The second stage involves the transshipment decision that is made after the demand realization. All the agents must come to an agreement (complete cooperation) regarding the shipping of residual inventory at the retailers and at the warehouses to meet the residual demands, and also regarding the allocation of the excess profit made from inventory sharing. However, the decision about the allocation mechanism of excess profit among the agents should be made before realizing the demands. The Profit Allocation Rule (PAR) should be designed such that it does not give incentive for individual retailers or groups of retailers to break away from the grand coalition; such allocations are called *core allocations* (Owen, 1975).

A thorough description and a very general framework for study of this problem can be found in Anupindi *et al.* (2001). We extend this general framework by developing a CMDP-ILP (Integer Linear Programming) model for the noncooperative scenario, and a MDP-ILP model for the centralized (noncompetitive) scenario of the problem. (The ILP model used for determining inventory sharing is identical

to what is presented in Anupindi *et al.* (2001).) We solve the CMDP-ILP model using our machine learning algorithm. We also optimally solve the MDP-ILP model using the well-known value iteration algorithm of stochastic dynamic programming. In the remainder of this section, we present our model formulations. We include a brief description of the ILP model for completeness and for the benefit of the readers.

4.1. Formulating the inventory planning problem as a CMDP and ILP

Let the system state of the problem with N retailers and W warehouses be denoted by a 2-tuple $X = (r, w)$ where r and w indicate the retailer and the warehouse inventory levels respectively. The elements of the system state (r, w) are $r = (r_1, r_2, \dots, r_N)$ and $w = (w_1, w_2, \dots, w_W)$, where r_i is the inventory level at retailer i and w_j is the inventory at warehouse j . Let K be the maximum inventory level that can be stored by a retailer at his/her retail location and in each of the warehouse locations. Thus, the quantity ordered by a decision maker at any decision epoch is between zero and K for the retail and each of the warehouse locations. Thus, the size of the state space is bounded by $|E| = (K + 1)^N \times (NK + 1)^W$. Let $\mathbf{d} = (d_1, d_2, \dots, d_N)$ be the demand vector where d_i is the demand at the i th retailer. Demands are assumed to be independent across all retailers. Let $\mathcal{N} = \{1, 2, \dots, N\}$ and $\mathcal{W} = \{1, 2, \dots, W\}$ be the sets of retailers and warehouses respectively. Thus, the action vector of any retailer i can be given as $\mathbf{a}^i = (a_i, a_{N+1}^i, \dots, a_{N+W}^i)$, where, for $i \in \mathcal{N}$, a_i represents the “order up to” level of inventory of retailer i , and for $j \in \mathcal{W}$, a_{N+j}^i denotes the “order up to” level of inventory of retailer i at warehouse location j . The system state is observed at the end of every business cycle, and based on the system state an action is taken by each retailer (decision maker). Let X^l denote the system state at the end of the l th business day, where $l \in \mathcal{L}$ and \mathcal{L} denotes the set of natural numbers. Define a random process $\mathbf{X} = \{X^l : l \in \mathcal{L}\}$ as the system process, where $X^l \in \{(r_i, w_j) : \forall i \in \mathcal{N} \text{ and } j \in \mathcal{W}\}$, $r_i \in \{0, \dots, K\}$, and $w_j \in \{0, \dots, NK\}$. For any given inventory policy and Poisson demand arrivals, the random process \mathbf{X} can be shown to be a Markov chain. The actions available to retailer i at any system state $m \in E$ are given by the set as follows.

$$A^i(m) = \{a^i(m) : a_i(m) \in \{r_i(m), r_i(m) + 1, \dots, K\},$$

$$\{a_{N+j}^i(m) \in \{w_j^i(m), w_j^i(m) + 1, \dots, K\} : \forall i \in \mathcal{N}, j \in \mathcal{W}\},$$

where $r_i(m)$ and $w_j^i(m)$ denote the inventory levels at the i th retailer, and the i th retailer's inventory at the j th warehouse respectively for system state m . With the “order up to” policy, if the action $a_i(m)$ is K , the number of new items ordered is $K - r_i(m)$. The Markov chain \mathbf{X} and the action space $\mathcal{A} = \{A^i(m) : i \in \mathcal{N}, m \in E\}$ together define the complete MDP. However, the noncooperative nature

of the decision-making processes with a separate objective function for each retailer makes this MDP a CMDP. The system state transition in the Markov chain depends on the demand arrival processes at the retailers and also on the inventory sharing between the retailers and the warehouses toward the end of a business cycle. The ILP model for the inventory sharing aspect of the problem, as given in Anupindi *et al.* (2001), is given next.

Define a set $\mathcal{V} = \mathcal{N} \cup \mathcal{W} = \{1, 2, \dots, N, N+1, \dots, N+W\}$. For all $k \in \mathcal{V}$, let c_k and v_k denote the unit cost and the unit salvage price respectively, and, for all $i \in \mathcal{N}$, let p_i denote the revenue per item at retailer i . Also, let $s_i = \min\{a_i, d_i\}$ be the sales from the local inventory at retailer i , and $h_i = \max\{(a_i - d_i), 0\}$ be the residual inventory after satisfying local demand. Recall that residual inventories are shared among the retailers to meet residual demands. Let $q_{k,i}$ and $t_{k,i}$ denote the number of units shipped from location $k \in \mathcal{V}$ to retailer $i \in \mathcal{N}$ and the corresponding transportation cost, respectively. Let $\mathbf{q} = \{q_{k,i} : k \in \mathcal{V}, i \in \mathcal{N}\}$ denote the shipping pattern. For each combination of the action vector (\mathbf{a}), the demand arrival vector (\mathbf{d}), and the shipping pattern (\mathbf{q}), the excess profit available to the N retailers in this system can be expressed as

$$\omega_{\mathcal{N}}(\mathbf{a}, \mathbf{d}, \mathbf{q}) = \sum_{k \in \mathcal{V}, i \in \mathcal{N}} (p_i - v_k - t_{k,i})q_{k,i}.$$

The optimal shipping pattern that maximizes the profit is found by solving the following ILP:

$$\omega_{\mathcal{N}}(\mathbf{a}, \mathbf{d}) = \max_{\mathbf{q}} \omega_{\mathcal{N}}(\mathbf{a}, \mathbf{d}, \mathbf{q}), \quad (3)$$

subject to

$$\sum_{l \in \mathcal{N}} q_{i,l} \leq h_i, \quad \forall i \in \mathcal{N}, \quad (4)$$

$$\sum_{l \in \mathcal{N}} q_{j,l} \leq \sum_{i \in \mathcal{N}} (a_{N+j}^i), \quad \forall j \in \mathcal{W}, \quad (5)$$

$$\sum_{k \in \mathcal{V}} q_{k,i} \leq E_i, \quad \forall i \in \mathcal{N}, \quad (6)$$

$$q_{k,i} \geq 0 \text{ and integers, } \quad \forall k \in \mathcal{V}, \forall i \in \mathcal{N}, \quad (7)$$

where $E_i = \max\{(d_i - a_i), 0\}$ is the residual demand at retailer i which cannot be satisfied from the local inventory. The constraints (4) and (5) ensure that the total quantities shipped from retailer i and warehouse j respectively do not exceed their available inventories. The constraint (6) ensures that the total quantity shipped to retailer i does not exceed its residual demand. Non-negativity and integrality restrictions are given by Equation (7).

The total profit made by a retailer i following the dual price (DP) allocation rule (which ensures *core allocation* (Owen, 1975; Shapley and Shubik, 1975)) is given by Anupindi *et al.* (2001):

$$P_i^{\text{DP}}(\mathbf{a}, \mathbf{d}) = p_i s_i + (v_i - \mu_i) h_i - c_i a_i + \alpha_i^{\text{DP}}(\mathbf{a}, \mathbf{d}), \quad (8)$$

where

$$\alpha_i^{\text{DP}}(\mathbf{a}, \mathbf{d}) = u_i h_i + \sum_{j \in \mathcal{W}} \gamma_j (a_{N+j}^i) + \delta_i E_i. \quad (9)$$

In Equation (8), v_k and μ_k represent the salvage price and holding cost per item, respectively. The allocation of the total excess profit to player i is denoted by $\alpha_i^{\text{DP}}(\mathbf{a}, \mathbf{d})$. The elements u_i , γ_j and δ_i in Equation (9) represent the DPs of the constraints (4), (5) and (6) respectively. This completes the presentation of the CMDP-ILP model. The MDP model for the centralized (noncompetitive) version of the problem is presented next.

If the competing decision agents (retailers) of the problem are substituted by a single agent who looks at the whole system state space and takes decisions for all retail locations to maximize average system reward, then the uncooperative inventory planning problem reduces to what can be modeled as a MDP. This Single Agent Problem (SAP) when solved optimally, gives a solution that is an upper bound for the uncooperative Multi-Agent inventory planning Problem (MAP). The ways in which the SAP and MAP solutions can differ are in the total system reward, and in the distribution of individual rewards to the players. In the SAP, the decision maker has complete information and makes decisions in order to maximize the total system reward. In the MAP, the players act with incomplete information while trying to maximize their individual rewards, without regard for the total system reward. As a result, the MAP solution is likely to produce a smaller system reward compared to the SAP solution. However, the rewards earned by the players in a MAP depend very much on the efficiency of the learning algorithm, and the total reward will at best be equal to the SAP solution. Hence, the SAP solution provides a very logical upper bound for the results of the RL algorithm.

4.2. A MDP model for SAP and its solution algorithm

The SAP is modeled as a MDP in a manner similar to the CMDP modeling of the MAP. The notation needed to describe the MDP is same for both MAP and SAP with the only exception of the action space. The “order up to” action vector for the decision agent in SAP is given by $\mathbf{a} = (a_1, a_2, \dots, a_N, a_{N+1}, a_{N+2}, \dots, a_{N+W})$, where a_i , $i \in \mathcal{N}$, denotes the inventory level at retailer i at the beginning of the business cycle immediately following the action a , and a_j , $j \in \{N+1, \dots, N+W\}$, denotes the inventory level at the warehouse j . Note that, the superscript i from the action vector is removed, since there is only one decision agent. The optimal solution of a MDP requires the transition probability matrix and the cost/profit matrix associated with each action. In what follows, the details of computing the transition probability and profit matrices for SAP are given.

Let $q = (r_1, \dots, r_N, w_1, \dots, w_W) \in E$ (state space) be the stock levels at the retailers and warehouses at the end of the most recent business cycle. For each “order up to” action

vector $\mathbf{a} = (a_1, \dots, a_N, a_{N+1}, \dots, a_{N+W})$, the starting inventory level at all locations at the beginning of the current business cycle is given by $(a_1, \dots, a_N, a_{N+1}, \dots, a_{N+W})$. Let the random variable for demand, D_i , at a retailer $i \in \mathcal{N}$ have a Poisson distribution with parameter λ_i . The maximum demand a retailer can satisfy is $d_{\max} = (a_1 + \dots + a_N + a_{N+1} + \dots + a_{N+W})$ (assuming that all other retailers experience a zero demand). Let $\mathbf{d} = (d_1, \dots, d_N)$ be the demand experienced by the retailers. Since demands are independent across the retailers, the probability of realizing a demand scenario \mathbf{d} is given by $\prod_{i=1}^N P(D_i = d_i)$. After satisfying the demands with local inventories to the extent possible, the residual inventories are shared, and the sharing pattern is determined by the ILP as in MAP. Subtracting from the starting inventory levels the demands realized by a retailer and the inventory transferred from that location to other locations gives the end state $s \in E$ of the system. One or many of the demand scenarios might lead the system to the same end state s . Thus, the transition probability $P(q, \mathbf{a}, s)$ for action \mathbf{a} at starting state q is given by the sum of the probabilities of each demand scenario that leads to the end state s .

The profit made under each demand scenario is given by the following equation. Let $P_N^C(\mathbf{a}, \mathbf{d}, \mathbf{q})$ denote the total profit for a centralized system with N retailers having adopted action vector \mathbf{a} , experienced demand arrival vector \mathbf{d} , and with a shipping pattern \mathbf{q} . Then we have that:

$$P_N^C(\mathbf{a}, \mathbf{d}, \mathbf{q}) = \sum_{i=1}^N [p_i s_i + (v_i - \mu_i) h_i - c_i a_i] + \sum_{j \in \mathcal{W}} [(v_j - \mu_j) h_j - c_j a_j] + \sum_{k \in \mathcal{V}, i \in \mathcal{N}} (p_i - v_k - t_{k,i}) q_{k,i}.$$

The total profit is composed of three terms, which correspond to the profits made by: (i) selling items at the local outlets; (ii) salvaging all unsold items; and (iii) sharing inventory among the retailers. An average reward relative value iteration algorithm (Puterman, 1994) has been used to solve the MDP. Note that each business cycle is independent of the preceding cycle, and thus the system profit per business cycle is used as the measure of performance.

Relative Value Iteration (RVI) algorithm for SAP

Let $V^n(q)$ be the total expected value of evolving through n stages starting at state $q \in E$, and ψ is the space of bounded real valued functions on E .

Step 1. Select $V^0 \in \psi$, choose $k^* \in E$, specify $\epsilon > 0$, and set $n = 0$.

Step 2. For each $q \in E$, compute $V^{n+1}(q)$ by:

$$V^{n+1}(q) = \max_{\mathbf{a} \in A(q)} \left\{ r(q, \mathbf{a}) - V^n(k^*) + \sum_{s \in E} P(q, \mathbf{a}, s) V^n(s) \right\},$$

where $r(q, \mathbf{a})$ is the immediate reward obtained by taking action \mathbf{a} in state q , and $P(q, \mathbf{a}, s)$ is the one-step transition probability.

Step 3. If $sp(V^{n+1} - V^n) < \epsilon$, go to Step 4. Otherwise increment n by one and return to Step 2. sp denotes span, which for a vector \mathbf{v} is defined as $sp(\mathbf{v}) = \max_{q \in E} \mathbf{v}(q) - \min_{q \in E} \mathbf{v}(q)$.

Step 4. For each $q \in E$, choose ϵ -optimal action in steady state, $a_\epsilon(q)$, as:

$$\mathbf{a}_\epsilon(q) \in \operatorname{argmax}_{\mathbf{a} \in A(q)} \left\{ r(q, \mathbf{a}) - V^n(k^*) + \sum_{s \in E} P(q, \mathbf{a}, s) V^n(s) \right\},$$

and stop.

4.3. Solving the MAP

The RL-based solution methodology of the MAP has five different elements as depicted in Fig. 2. The arrows in the figure indicate the flow of information among the elements. Brief descriptions of the elements are given below.

- Simulation:** This block is activated after the inventory decisions are made by the retailers. The customer arrival processes during a business cycle in the inventory distribution system are simulated. The ARENA[®] simulation software package was used to develop the simulation model. At the end of a business cycle, based on the action taken (inventories ordered) at the beginning of the cycle and the realized demand during the cycle, the simulation block sends the information about the residual inventory and the residual demands to the transportation block.
- Transportation:** Upon receiving from the simulation block the information on residual demands and residual inventories, the transportation block decides the optimal sharing pattern for the residual inventories

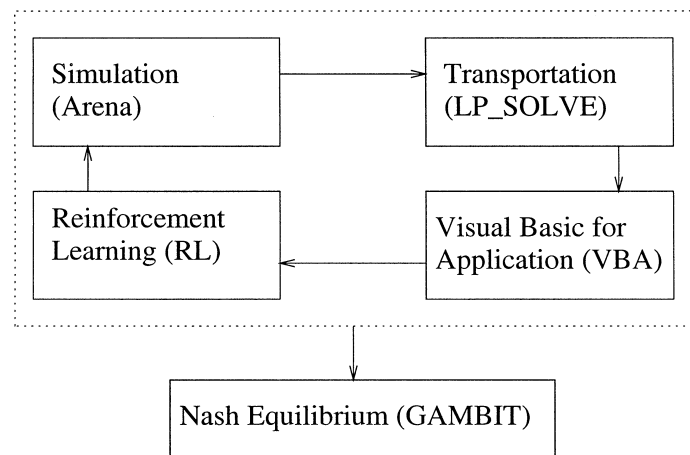


Fig. 2. Elements of the MAP solution methodology.

to meet the residual demands by solving the ILP. LP_SOLVE, a freely available software on the Internet (ftp://ftp.es.ele.tue.nl/pub/lp_solve/), was used to solve the ILP. The optimal shipping pattern, the excess profit thus made by sharing, the dual values of the constraints, and the end state of the cycle are then sent to the Visual Basic for Applications block (VBA), which is a backend software package module available with ARENA[®].

- (c) Visual Basic for Applications (VBA): The VBA block allocates to the retailers the excess profit made in a cycle by sharing of the inventory based on a DP allocation rule. This block sends the information regarding the starting and the end states of the system, and the total profit made by each retailer during the business cycle (known as reward response) to the reinforcement learning block.
- (d) Reinforcement Learning (RL): The RL block updates the reinforcement value matrix of each agent using Step 2(c) of the RL algorithm and the information received from the VBA block. This block then updates the values of the exploration and learning rates (Step 2(d)) using the DCM scheme. Then, if the stopping criterion is not reached, it decides on the next set of actions for the retailers (Step 2(a)) based on the ending state of the current cycle and the reinforcement values of the available actions. This set of actions is then sent to the simulation block (Step 2(b)). This cycle of simulation → transportation → VBA → RL → simulation continues until the condition in Step 2(f) is met. Then the matrices containing the reinforcement values (Q -values) are sent to the Nash Equilibrium (NE) block as input.
- (e) Nash Equilibrium (NE): This block solves a static game with the reinforcement value matrices as the payoff matrices. GAMBIT[®] was used to find the Nash equilibrium policy for each state. We used the simplicial subdivision algorithm approach (Van der Laan *et al.*, 1987) available through GAMBIT[®] to compute a Nash equilibrium. The Nash equilibrium policy thus found is sent back to the simulation module. This system operating under the above Nash policy is simulated again for a large number of replicates from which the “equilibrium rewards” for each player and the system are estimated.

5. Numerical results and analysis

This section presents numerical results for several variants of the test problem, which were solved using the RL-based methodology. In order to demonstrate the performance of the RL-based methodology, these results are compared with the corresponding upper bounds obtained by solving the SAPs using optimal value iteration algorithm. However, implementation of the value iteration algorithm involves: (i) complexity of modeling (resulting from the need to construct separate transition probability and reward matrices

for each player and each action of the large action spaces); and (ii) burden of dimensionality (resulting from the large size of the system state space). Hence, in our numerical problems, without any loss of generality, warehouses were not considered in order to reduce the size of the state space and thus ease the task of upper bound calculations.

5.1. Description of the numerical test problem

The numerical test problem consists of three retailers ($N = 3$). The parameters for the test problem, as described next, have been chosen somewhat arbitrarily. It was considered that the retailers 1, 2 and 3 experience Poisson distributed demand arrivals with mean rates of $\lambda_1 = 3$, $\lambda_2 = 3$ and $\lambda_3 = 2$. Each retailer can order inventory between zero and four (i.e., $K = 4$). All customers at a retailer are willing to be served from other locations when a stock out situation occurs. The unit cost (c_i), unit revenue (p_i) (unit cost + profit), and holding cost (μ_i) of an item are the same across all retailers. The salvage price of an unsold item (v_i) is considered to be the same as the unit cost. The unit cost is considered to be \$10. The profit, holding cost and the salvage price are taken as percentages of the unit cost. The transfer cost for moving an item between two retailers i and j (t_{ij}) is fixed and given in Table 1. The DP-based allocation of excess profit (as described in Equations (8) and (9)) was considered in the study of the test problem.

5.2. Numerical results

This section first discusses the methodology that was used in determining the design parameters of the RL-based MAP solution methodology. This methodology requires three input parameters and six design parameters. The input parameters are: the cost per item, the profit per item, and the holding cost per item. The design parameters are the initial values of state-action learning parameters (α_0, α_τ), average reward learning parameters (β_0, β_τ), and the exploration parameters (γ_0, γ_τ). The parameters α_τ, β_τ and γ_τ affect the rate of decay of the corresponding learning and exploration rates α, β and γ , for which the initial values are α_0, β_0 and γ_0 respectively. In our study, all the learning and exploration parameters were decayed using the same parameter, and thus we had $\alpha_\tau = \beta_\tau = \gamma_\tau$. In order to obtain the best performance of the RL methodology, it is important to have good estimates for each of the design parameters. We obtained such estimates through a designed factorial experiment. Through this experiment, the sensitivities of

Table 1. Transfer cost matrix

—	A1	A2	A3
A1	—	1	1.5
A2	1	—	1.25
A3	1.5	1.25	—

Table 2. Levels of the input and design factors

Factor index	Factor description	High	Low
A	Profit (% of unit cost)	75	50
B	Holding cost (% of unit cost)	15	10
C	α (learning parameter for Q values)	0.3	0.05
D	β (learning parameter for average reward)	0.3	0.05
E	γ (exploration parameter)	0.3	0.05
F	θ_r (common decay rate for α , β and γ)	9×10^5	4.5×10^5

the RL algorithm toward the design variables were first assessed using the difference (in percent) between the MAP result and the corresponding SAP upper bounds as the response variable. We used Analysis Of Variance (ANOVA) and normal probability plots, as applicable, for this purpose. After identifying the significant variables/interactions, a regression model was developed which was then optimized to get the best design parameter values for the RL algorithm.

The factorial experiment was developed using two levels of values for each of the six input and design parameters. The input parameters, such as profit per item and holding cost per item, were expressed in percentages of the cost per item. Hence, the cost per item was not considered as a separate factor. The factors (indexed A through F), their descriptions, and their values at two levels are listed in Table 2.

A quarter fractional factorial 2^{6-2} experiment was designed with $I = ABCD = BCDF$ as the design generator. The SAP abstractions of the sixteen test problems were

solved optimally using the relative value iteration algorithm (presented in Section 4.2). The MAPs were then solved using the RL-based methodology (presented in Section 4.3). The average system profits obtained from the RVI and RL methods for all sixteen test problems are given in Table 3. The treatment combinations given in column 1 of the table is explained as follows: (1) denotes the experiment where all factors are maintained at low level, ae denotes the experiment where factors A and E are at high level while the rest of the factors are at their low levels, and so on. Note that the output of the RL algorithm is a policy. To estimate the average system profit for a policy, the system is simulated with the policy. To maintain an equal basis for comparison, the optimal policies obtained from the RVI algorithm were also simulated to assess their average profits, in spite of the fact that the RVI algorithm produces both the optimal policy and the optimal average profit from the policy. The 3σ limits given against both the RVI and the RL profits indicate the variability of the average system profit estimates obtained through simulation. The effects of the factors and the interactions were calculated using the percent difference between RVI and RL as the measure. A normal probability plot of the main effects and the interactions indicates that among the interaction effects, AB is the only outlying one and hence, perhaps, significant. An ANOVA was then performed in which all the main factors and one interaction effect (AB) were examined. Since this is a single replication experiment, the error estimate was obtained by pooling the sum of the squares of all interactions other than AB. The results of the ANOVA, shown in Table 4, indicate that the main factors, such as the profit per item (A), learning parameter (D) for average profit, and the decay parameter (F) are significant. The two-level interaction between profit and holding cost per item (AB) is also shown

Table 3. Treatment combinations and corresponding responses

Treatment combination	RVI(SAP)		RL(MAP)		RVI-RL	Percentage difference
	Average system profit	3σ limits	Average system profit	3σ limits		
(1)	35.1798	0.56	34.9582	0.56	0.2216	0.63
ae	55.2584	0.7925	53.7377	0.86	1.5207	2.75
bef	33.5699	0.58	32.9629	0.59	0.607	1.81
abf	53.124	0.84	52.8358	0.83	0.2882	0.54
cef	35.1798	0.56	35.178	0.55	0.0018	0.01
acf	55.2584	0.7925	54.7906	0.78	0.4678	0.85
bc	33.5699	0.58	33.0186	0.61	0.5513	1.64
abce	53.124	0.84	52.288	0.95	0.836	1.57
df	35.1798	0.56	35.0456	0.61	0.1342	0.38
adef	55.2584	0.7925	53.465	0.61	1.7934	3.25
bde	33.5699	0.58	31.8397	0.69	1.7302	5.15
abd	53.124	0.84	51.6047	0.9	1.5193	2.86
cde	35.1798	0.56	35.0132	0.54	0.1666	0.47
acd	55.2584	0.7925	53.635	0.93	1.6234	2.94
bcdf	33.5699	0.58	31.7427	0.43	1.8272	5.44
abcdef	53.124	0.84	53.0318	0.86	0.0922	0.17

Table 4. ANOVA results for selection of design parameters

Source of variation	Degrees of freedom (df)	Sum of squares (SS)	SS/df	F _o	F _c
A	1	0.526	0.526	6.318	5.317
B	1	0.144	0.144	1.729	5.317
C	1	0.315	0.315	3.783	5.317
D	1	1.205	1.205	14.47	5.317
E	1	0.001	0.001	0.012	5.317
F	1	0.546	0.546	6.558	5.317
AB	1	2.94	2.94	35.31	5.317
Error	8	0.666	0.08325		
Total	15	6.349			

to be significant. A regression model given in Equation (10) was obtained using all the main factors and the interaction AB.

$$Y = 0.836 + 0.1813X_A + 0.0951X_B - 0.1405X_C + 0.2745X_D + 0.007X_E - 0.1848X_F - 0.4288X_A X_B. \tag{10}$$

The above regression model resulted in a R^2 value of 0.726. This model was then used to obtain values for the design parameters (C, D, E and F) of the RL algorithm. A set of four problems were chosen by varying two input parameters, unit profit (A) and unit holding cost (B), as shown in Table 5. For each set of input parameters A and B, the regression model was optimized to get good estimates for the design parameters for the RL algorithm. The profits for the policies derived from the RL algorithm using optimized design parameters are given in Table 5. The corresponding RVI results are also given in the table. The results produced by the RL methodology are on average within 0.8% of the upper bounds given by the RVI results. Note that, in a uncooperative environment (as in MAP) it would almost never be possible to achieve the SAP results. Since in a SAP, the single agent works with complete information and, hence, can make optimal decisions that maximize the overall system profit. In a MAP, agents work with partial information and are focused on maximizing only the individual profits. As a result, the system profit obtained in a MAP may suffer.

6. Concluding remarks

Stochastic games, although increasingly prevalent in the current business environment, are among the least understood of the management science related problems. Game theoretic results for stochastic games are limited and are practically impossible to implement computationally, even for small sized problems. As a result, RL has long been proposed as a cure for this situation. However, to date, RL applications to game problems have been limited to test problems, such as two-player zero-sum games, robotic soccer and grid world games. Moreover, to our knowledge, the average reward criterion has never been tried for multi-agent problems.

An average reward RL-based solution methodology to study nonzero-sum stochastic games is developed in this paper. A supply chain inventory planning problem consisting of multiple retailers and warehouses dealing with a single product was considered as a test bed for the RL methodology. A set of sixteen numerical test problems having three retailers and no warehouses was solved using the methodology. Although the problem used is more realistic and has a larger state space compared to the problems that were attempted in the game theoretic literature, its size is still rather small compared to the problems encountered in real life. The results were compared with realistic upper bounds obtained from the optimal solutions of the corresponding single agent problems without competition. The single agent problems were solved using RVI. The small size of the test

Table 5. Comparison of system average rewards achieved using RL and RVI algorithms

Unit cost	Percentage of the unit cost		RVI(SAP)		RL(MAP)		Difference (RVI-RL)	Percentage difference
	Unit profit	Unit holding	Average profit	3σ limits	Average profit	3σ limits		
10	50	10	35.1798	0.56	34.9876	0.55	0.1922	0.546
10	50	15	33.5699	0.58	33.0228	0.60	0.547	1.6291
10	75	10	55.2584	0.79	54.79096	0.78	0.467	0.846
10	75	15	53.1240	0.84	53.024	0.87	0.099	0.187

problems was chosen to limit the computational and modeling complexities associated with RVI implementation. The RL results were found to be on average within 0.8% of the upper bounds. Although we noticed strong numerical evidence of convergence for our algorithm, the Robbins-Monro type stochastic approximation scheme used in our algorithm has not yet been analyzed analytically to guarantee convergence. We are currently investigating these issues and some of the results obtained by us so far can be seen in Li and Das (2003).

We believe that the results obtained from this study will encourage more research in this area, since there are practically no computational tools available in the open literature that can effectively solve nonzero-sum stochastic games. The numerical problems that we have studied, although much larger compared to the problems that are found in game theoretic literature, have limited state spaces. Thus, we were able to use the look-up table approach for the reinforcement values during the RL algorithm implementation. For implementing the algorithm on problems with large state spaces, a suitable function approximation scheme must be devised. This would eliminate the need to store the Q -values explicitly, and can be stored as a function of a small number of parameters. This can be achieved using local regression or neural networks. Several schemes have been proposed in the literature (Bertsekas and Tsitsiklis, 1996; Van Roy, 1998) for function approximation. The selection of the right scheme very often depends on the problem structure and the nature of the value function which is rarely known beforehand. Also, since the reinforcement value function keeps changing its shape during the learning process, approximating it is a challenging task. The use of a backpropagation scheme has been widely advocated but it is well-known that since it depends on gradient-descent, getting trapped in local optima cannot be ruled out. A number of schemes have been suggested to overcome this drawback and they seem to have produced encouraging results. Nearest neighbor schemes are an alternative. Another is the use of regression with a piece-wise linear approximation of the value function. Using this approach with a neuron in every piece has been shown to produce good results on a large-scale implementation of RL on a problem of airline yield management (Gosavi *et al.*, 2002). A neuron produces a linear approximation (unlike backpropagation which can give a non-linear approximation) and its theory is based on the Widrow-Hoff rule (Ripley, 1996) which has a strong convergence analysis.

Acknowledgements

This work was supported in part by grant ITR:AP DMI-0113946 from The National Science Foundation. The authors appreciate the detailed review of an earlier version of this paper by the Department Editor Professor Izak Duenyas, an Associate Editor, and three anonymous

referees. The review comments of the Associate Editor and one of the referees have been especially useful in improving the paper.

References

- Abounadi, J., Bertsekas, D. and Borkar, V.S. (1998) Learning algorithms for Markov decision processes with average cost report. LIDS-P-2434, Laboratory for Information and Decision Systems, MIT, Cambridge, MA.
- Anupindi, R., Bassok, Y. and Zemel, E. (2001) A general framework for the study of decentralized distribution systems. *Journal of Manufacturing and Service Operations Management*, **3**(4).
- Bellman, R.E. (1957) *Dynamic Programming*, Princeton University Press, Princeton, NJ.
- Bertsekas, D. and Tsitsiklis, J. (1996) *Neurodynamic Programming*, Athena Scientific, Belmont, MA.
- Darken, C., Chang, J. and Moody, J. (1992) Learning rate schedules for faster stochastic gradient search, *Neural Networks for Signal Processing 2—Proceedings of the 1992 IEEE Workshop*, in White, D.A. and Sofge, D.A. (eds.), IEEE Press, Piscataway, NJ.
- Das, T.K., Gosavi, A., Mahadevan, S. and Marchallick, N. (1999) Solving semi-Markov decision problems using average reward reinforcement learning. *Management Science*, **45**(4), 560–574.
- Erev, I. and Roth, A.E. (1998) Predicting how people play games: reinforcement learning in experimental games with unique, mixed strategy equilibria. *The American Economic Review*, **88**(4), 848–881.
- Filar, J. and Vrieze, K. (1997) *Competitive Markov Decision Processes*, Springer-Verlag, New York, NY.
- Gosavi, A. (2004) Reinforcement learning for long-run average cost. *European Journal of Operations Research*, to appear.
- Gosavi, A., Bandla, N. and Das, T.K. (2002) A reinforcement learning approach to airline seat allocation for multiple fare classes with overbooking. *IIE Transactions*, **34**(9), 729–742.
- Hu, J. and Wellman, M.P. (1998) Multi-agent reinforcement learning: theoretical framework and an algorithm, *Proceedings of the 15th International Conference on Machine Learning*, pp. 242–250.
- Li, J. and Das, T.K. (2003) Learning Nash equilibrium for average reward irreducible stochastic games. Working paper, Department of Industrial and Management Systems Engineering, University of South Florida, Tampa, FL 33620.
- Littman, M.L. (1994) Markov games as a framework for multi-agent reinforcement learning, in *Proceedings of the 11th International Conference on Machine Learning*, pp. 157–163.
- Nash, J.F. (1951) Non-cooperative games. *Annals of Mathematics*, **54**, 286–295.
- Owen, G. (1975) On the core of linear production games. *Mathematical Programming*, **9**, 358–370.
- Paternina, C.D. and Das, T.K. (2000) Intelligent dynamic control policies for serial production lines. *IIE Transactions*, **33**(1), 65–77.
- Puterman, M.L. (1994) *Markov Decision Processes*, Wiley, New York, NY.
- Ripley, B.D. (1996) *Pattern Recognition and Neural Networks*, Cambridge University Press, Oxford, UK.
- Robbins, H. and Monro, S. (1951) A stochastic approximation method. *Annals of Mathematical and Statistics*, **22**, 400–407.
- Shapley, L. and Shubik, M. (1975) Competitive outcomes in the core of market games. Technical report R-1692-NSF, The Rand Corporation.
- Sutton, R.S. and Barto, A. (1998) *Reinforcement Learning*, MIT Press, Cambridge, MA.
- Van der Lann, G., Talman, A.J.J. and Van der Heyden, L. (1987) Simplicial variable dimension algorithms for solving the nonlinear complementary problem on a product of unit simplices using a general labeling. *Mathematics of Operations Research*, 377–397.

Van Roy, B. (1998) Learning and value function approximation in complex decision processes. Ph.D. thesis, Laboratory for Information and Decision Systems, MIT, Cambridge, MA.

Watkins, C.J.C.H. (1989) Learning from delayed rewards. Ph.D. thesis, Cambridge University, Cambridge, UK.

Biographies

Kiran Ravulapati is an Operations Research Specialist with Delta Technology, Atlanta, GA. His current areas of interest include pricing and revenue management in airlines, Markov decision processes, reinforcement learning and game theory. He received his MS in Industrial Engineering from the University of South Florida, Tampa and his BS in Mechanical Engineering from the Birla Institute of Technology and Science, Pilani, India. He also worked in the manufacturing area for 2 years with Tata Motors, India.

Jaideep Rao is an Applications Engineer with Pilgrim Software, Tampa, FL. He received a BS in Mechanical Engineering from the University of

Bangalore, India and a MS in Industrial Engineering from the University of South Florida, Tampa, FL. As a graduate student, under the directions of Dr. Tapas K. Das, he worked on numerous stochastic optimization projects. As an Applications Engineer, he provides solutions to various FDA regulated companies in the United States to meet stringent quality and regulatory requirements. He also worked as a Project Engineer with the Robert Bosch Corporation.

Tapas K. Das received his Ph.D. from Texas A&M University and now teaches at the University of South Florida where he serves as a Professor of Industrial and Management Systems Engineering. His teaching interests are in applied stochastic processes and quality engineering. His current research interests include simulation-based optimization of *co-operative* and *noncooperative* stochastic decision systems using machine learning techniques, design and control of service and supply-chain-type systems, and design of quality monitoring and control systems for multivariate processes using wavelet-based multiresolution analysis. He has published in *IIE Transactions* (in all areas), *IEEE Transactions (Reliability, Power Systems, and Semiconductor Manufacturing)*, *Management Science*, *Communications in Statistics*, *Naval Research Logistics*. His research has been funded primarily by the National Science Foundation and state agencies.