Learning Control Policies in System Dynamics Models

Hazhir Rahmandad, Saeideh Fallah-Fini

Virginia Polytechnic Institute and State University, Industrial and Systems Engineering Department

Room 430, Northern Virginia Center, 7054 Haycock Road, Falls Church, VA 22043

Tel: +1 (703) 538-8434

hazhir@vt.edu, fallah@vt.edu

Abstract

Advances in artificial intelligence and optimal control provide increasingly better algorithms for controlling dynamical systems. These algorithms can be applied for policy design in system dynamics models. In this paper we introduce some basic solution concepts and apply the Q-learning algorithm to a simple dynamic model from system dynamics literature to demonstrate potential value of such cross-fertilization. We also extend a state aggregation and partitioning algorithm that may increase the efficiency of basic reinforcement learning models in application to continuous time and space problems. Simulation analysis demonstrates the value of this approach and offers guidelines for future research.

Key words: Reinforcement learning, State space aggregation, System dynamics, Machine learning, Optimal control

1. Introduction and literature review

Over the last fifty years system dynamics has grown to be a successful field in modeling the dynamics of complex social, business, and economics problems (Sterman 2007). The basic premise of system dynamics is that real world problems of interest to managers and policy makers are replete with non-linear feedback structures with counter-intuitive behavior for un-assisted mind (Forrester 1971). Therefore effective policy making in these contexts requires a two stage process of building a dynamic model of problem at hand and analyzing the model to gain insights into what are the most promising interventions that can solve the problem.

The first stage, which includes defining the problem and the model building process, has emerged as the core function of the field. Available textbooks in system dynamics cover different stages of modeling process in detail (e.g. Forrester 1961; Sterman 2000). Different methodologies have been developed for enhancing the effectiveness of modeling process through iterative model development (Randers 1980), engaging groups of stake-holders in modeling process (Vennix 1999; Andersen, Vennix et al. 2007), eliciting qualitative knowledge from stake-holders (Ford and Sterman 1998), and developing model blocks (Hines 2003), among others.

However, the second stage, finding effective policies using a current model, remains ripe with opportunities for development of tools and methods. In essence, the traditional and most widespread model analysis process relies on modeler's insight and experience to go through manual alterations of a model's different parameters and policy levers to both understand the origins of observed dynamics and to evaluate the value of different strategies. Algorithmic developments in feedback loop analysis promise more reliable methods for understanding model behavior (Mojtahedzadeh, Andersen et al. 2004; Ford and Flynn 2005; Kampmann and Oliva 2006). For the second step, when a more efficient method for finding effective policies is sought, non-linear optimization is the only tool available.

Non-linear optimization engines built in typical simulation software (e.g. Vensim ™, Powersim ™, Anylogic ™) allow a modeler to maximize a payoff function of model variables by changing a pre-defined set of parameters in the model. The overall structure of these optimization problems can be summarized as follows:

$$Maximize\left(\int_{t=0}^{t=T} r(x(t))\right)$$

$$dx/dt = f(x; b, p_{optim})$$

Subject to: $x(0) = x_0$

$$p_{optim} \in P$$

Here the function r(.) represents the continuous reward (cost) function the integral of which over the simulation time is to be maximized (or minimized). The vector *x(t)* represents the states (or stock variables) in the system. The first condition in the "subject to" line is saying that the dynamics of change in the *x* vector follow a system of differential equation specified by the function *f(.)*, and parameter vectors *b* and p_{optim}. In other words, there is a system dynamics model (*f*) that generates the values of state vector, *x*, at every point in time. Members of b vector parameter are not subject to change in the optimization process. They represent the intrinsic characteristics of the system

modeled, and thus not policy parameters of interest. In contrast, p_{optim} represents the set of policy parameters that can be changed in the model to produce the desired outcome. They are parameter, in the sense that their value does not change throughout a single simulation, yet they are control levers at decision-maker's hand to shape the evolution of the dynamics in the model. The policy parameters are subject to some additional constraints in their values (e.g. investment can not be negative, etc), captured here in the feasible set P for these parameters. For example, after modeling a firm's sales and marketing dynamics (specifying function f and parameters b), an analyst may use an optimization procedure to find the best annual price and marketing expenditure (two parameters constituting vector p_{optim}) to maximize discounted profits ($r(x(t))$ over a five year time horizon (T). Given the general, and often non-linear, form of functions f and r, the optimization process is not guaranteed to find the global optimum for most problems. Yet, with a good selection of optimization algorithm and settings, satisfactory results can be obtained for many practical questions.

However, the application of this method is limited by two general constraints. First, policy parameters are assumed to be constant throughout the simulation. In the above example the manager is not allowed to change the price at different points in time. This problem can be partly alleviated by inclusion of multiple parameters, each representing the policy lever of interest over a specific time horizon. For example we can have five different parameters to represent prices in years one to five respectively. This solution, however, is not perfect. The increase in the number of parameters increases the size of parameter space very quickly and makes it increasingly harder and more time consuming for the optimization algorithm to find the optimum solution.

A second problem is potentially harder to solve. So far we have assumed that the system is starting from a given initial condition (i.e. x_0) and the dynamics unfold deterministically thereafter. The optimal policy parameters are derived based on these assumptions. If the system started from a different initial condition, the best policy parameter value may no more be the same. More generally, the optimum parameters are contingent on the state of the system. If the system is not deterministic, different contingencies can arise that require different reactions (policy parameters) and therefore the optimal policy parameter setting can not be determined independent of the state of the system. In our example, the optimal price may not only change from year to year, but also may depend on competitor's price. It is not enough for the optimization to tell what the price should be for the third year, rather, it should make the recommendation contingent on the competitor's price (and other relevant information, e.g. firm's inventory level at the end of year two, which may depend on a partially stochastic sales figure in that year). Making the optimization contingent on the state of the system is much harder, if not practically impossible, in the basic non-linear optimization paradigm. One needs to run a new optimization for every possible state the system may be found in. Yet the problem is important and relevant to policy makers. The evolution of the system may not follow the deterministic path originally predicted by the optimization process. Moreover, policy makers prefer to preserve their control by having a set of heuristics for how to act given different contingencies, rather than being bound to a pre-defined set of actions to take. As analysts, we need better tools to provide more effective policy recommendations that use all the critical information available about the state of the system under analysis.

This challenge is not unique to the field of system dynamics and researchers in operations research, optimal control, and artificial intelligence have long been exploring ways to find optimal control policies for dynamical systems. The main goal of this paper is to address the practical challenge of finding efficient policies in dynamic models of social systems in light of developments in complementary fields and to provide basic concepts and emerging algorithms that can benefit the system dynamics community. We therefore first outline the main characteristics of a successful algorithm for application in practical problems faced by system dynamics modelers, and then briefly review the available approaches in neighboring literature in light of these characteristics. In section 2 we outline a learning algorithm from artificial intelligence literature and outline a modified algorithm that moves closer to satisfying the requirements for system dynamics applications. Section 3 provides analysis and illustrative results. We discuss the implications, contributions, and future research opportunities in section 4.

1.1 Characteristics of useful control policy solutions

The available algorithms for finding optimal control policy for dynamic problems span a large set of problem characteristics, formulation constraints, and computational methods. For successful application to dynamic models of social systems an algorithm should partially or fully satisfy the following criteria.

Problem structure independence

Dynamic models of social systems are often non-linear and include soft variables, and therefore rarely follow a simple and analytically solvable model structure. For general application to this category of models, an algorithm should be applicable to different non-linear dynamical systems without major changes. In fact, a major reason for successful application of current non-linear optimization algorithms in this domain (e.g. the optimization engines developed inside current simulation software) is the structure independence of these methods. In general they view the simulation model as a black-box which provides a payoff for any given set of parameter sets. Similarly, we expect successful algorithms for finding control policies to be applicable to different problems regardless of specific model structure (e.g. no constrains calling for linearity of system or quadratic payoff function).

Scalability

Most practical problems of significance in modeling social systems include at least a few state variables. The algorithm should therefore be able to handle systems of moderate complexity (e.g. a dozen state variables) without running into computational barriers. This may sound like a relatively low standard compared to scalability requirements for well structured optimization algorithms such as linear programming, which are applied to problems with hundreds of thousands of variables. Yet the complexity of optimal control problem is such that even handling moderate size system dynamics models can not be taken for granted.

Separation of mathematics from application

The majority of potential users of any control policy development algorithm for system dynamics models are pragmatic in their approach. They want an algorithm that

can give satisfactory result for their application. These users are often reluctant to spend a significant time on learning complex mathematical algorithms to find a control policy as part of a larger project. Therefore any algorithm developed for these users should separate, and automate, the mathematical operations and require the minimum user involvement in design and operation of the algorithm. In return these users are often not concerned with proofs of convergence, global optimality, or efficiency of an algorithm.

Generation of insights for human decision makers

Whereas many applications in engineering and artificial intelligence only search for an optimal policy, social scientists and consultants not only search for an effective policy, but also require a way of interpreting and translating that policy into insights for human decision makers. A successful algorithm should therefore include some way of intuitively communicating the results of algorithm to human decision makers. Application of model analysis techniques (e.g. loop dominance and pathway participation methods) to inform the structures responsible for value of optimal control policy may prove helpful for this purpsoe.

1.2. Approaches for optimal control of dynamic systems

The general problem of finding optimal control policy for a dynamic system can be summarized as follows. We have a dynamic continuous time system defined by an n-dimensional state vector *x*, and a control vector *u* which belongs to a policy space *U*:

$$\dot{x}(t) = f(x(t),u(t)) \qquad 0 \le t \le T\ ,\ x(0) : given\ ,\ x(t) \in \Re^n\ ,\ u(t) \in U$$

A control policy is defined by the set $\{u(t) \mid t \in [0,T]\}$. Following this policy leads to a state trajectory that the system follows: $\{x^u(t) \mid t \in [0,T]\}$. The goal of an optimal control algorithm is to find a control trajectory that maximizes a profit function for the system defined based on the value of final conditions of the system at the end of T(*V(x(T))*), as well as, the accumulation of incremental rewards received throughout the journey (*r(x(t),u(t))*)[1]:

$$V(x(T)) + \int_0^T r(x(t),u(t)).dt$$

Different solution methods for this general problem can be put into two general categories. First, continuous time and space solutions that build on the Hamilton-Jacobi-Bellman equation that specifies the necessary conditions for an optimal control trajectory in terms of a reward(/cost)-to-go function. The details of these approaches go beyond the scope of this paper (For detailed treatment and applications see (Athans and Falb 1966; Bertsekas 2007)) however, a few important characteristics of these methods are relevant to the discussion at hand. These algorithms seek to analytically solve the control policy, and prove the optimality of the resulting solution. As a result they require extensive mathematical treatment and analytical solutions are often limited to very simple problems with specific structures. Numerical methods can be found for solving larger problems, and their efficiency is often a function of problem structure.

[1] For simplicity we have defined the problem for deterministic systems, but the main solution methods are expanded to stochastic systems as well.

A second category of algorithms turns the continuous time and state problem into an approximate discrete time and state model. It then uses one of the dynamic programming or reinforcement learning algorithms to solve the optimal control policy problem (Bertsekas and Tsitsiklis 1996). Learning methods often solve the problem iteratively, through a process of updating the value of different state (and action) combinations in terms of expected rewards of visiting that state (and action) and following the optimal policy thereafter. Dynamic programming methods try to find the same value function analytically. In fact numerical methods for solving dynamic programs tend to be very similar to reinforcement learning algorithms and solution methods to dynamic programs can be seen as a specific extreme in the spectrum of reinforcement learning algorithms (Sutton and Barto 1998). Among these algorithms the ones that use reinforcement learning methods have the highest flexibility in terms of model structure they can be applied to. In fact many of these methods are model-free (that is, they can treat the underlying model largely as a black-box) and therefore allow the application of algorithm to be (largely) separate from the underlying dynamic model (Kaelbling, Littman et al. 1996).

In comparison of the two general categories of algorithms, we find that the continuous time and state category of methods do poorly on satisfying the "Problem structure independence" and "Separation of mathematics from application" criteria. Reinforcement learning algorithms in the second category seem to be the most promising for the application to models of dynamic social systems because they satisfy these criteria. In fact successful application of reinforcement learning algorithms for robotic and process control applications provides empirical support for this conclusion (Santamaria, Sutton et al. 1997; Lee and Lee 2004). We therefore focus on introducing reinforcement learning and its application for solving the control policy problem in system dynamics models. Moreover, we introduce an extension to one of the well-known reinforcement learning algorithms in order to improve the intuitive interpretation of the results for human decision-makers.

2. Problem Definition and Algorithms

In this section we present a discrete time, discrete action and discrete state space reinforcement learning model. A reinforcement learning algorithm enables the learning of what is the best action given different states a system may land in. It does so through trying different actions from different states and following the rewards received to re-evaluate the different states (and actions) (Sutton and Barto 1998). Consider a dynamic system where at each point in time the learner observes information about the state of the system (the values of different stock variables), takes action, and receives payoff. At every step the learner receives a set of information cues about the state of the system and based on those cues decides what action(s) he will take during the next period[2], so that his long-term performance is maximized. The long-term performance integrates the immediate payoffs received throughout the completion of the task, or may be limited to a final performance measure observed at the end of the task. Learning entails improving the

[2] The size of period can be made arbitrarily small to converge to true continuous time formulation. However in practice the costs of information gathering and changing action make larger periods empirically more realistic.

mapping from the possible cues to possible actions so that performance is enhanced over the long-run.

Formally, denoting the state of the system at time t as $x_t \in S$ where *S* is the set of feasible states, the learner decides to choose action $u_t \in U(x_t)$, where $U(x_t)$ is the set of possible actions at state x_t. This action leads the system to a new state, x_{t+1}, and the learner receives an immediate reward $r(x_t, u_t, x_{t+1})$. The learning task for the learner can thus be summarized in finding a mapping from the state space *S* to the allowable control space U ($\pi : S \to U$) with respect to a particular measure.

Reinforcement learning uses computational methods to find the optimal control policy π^*. This policy maximizes the total expected rewards for the learner. Action value function estimates the total expected reward for a learner starting from x, taking action u, and following the policy π afterwards and is used to find the optimal policy: $Q^\pi(x,u), \forall x \in S$ and $\forall u \in U$. For an infinite horizon problem, where the terminal time T approaches infinity, the action-value function (or long-term performance) is given by $Q^\pi(x,u) = \underset{T\to\infty}{Lim} E\{\sum_{t=0}^{T-1} \gamma^t r(x_t, u_t, x_{t+1})\}$ where $0 < \gamma \le 1$ is a discount rate for reward and E is for expected value function[3]. The optimal action-value function is defined as $Q^*(x,u) = \max_\pi Q^\pi(x,u), \quad \forall x \in S, \ \forall u \in U$. In addition, the value of each state defined as the largest of all action-values for that particular state is given by $V^\pi(x) = \max_{u \in U(x)} Q^\pi(x,u), \quad \forall x \in S.$

Typically the algorithms in reinforcement learning domain approximate the action-value or value function ($Q^\pi(x,u)$ or $V^\pi(x)$) for a policy, improve the policy, and continue with this iterative process until they find the optimal strategy that according to the bellman optimality condition satisfies:

$Q^*(x,u) = \underset{\pi}{Max} Q^\pi(x,u) = \underset{x_{t+1}}{E}\{r(x,u,x_{t+1}) + \gamma . \underset{u}{Max}(Q^*(x_{t+1},u))\}$, where $x_t = x$, $u_t = u$, and $Q^*(x,u)$ is the optimal action-value function.

An important innovation in reinforcement learning has been the development of an off-policy[4] algorithm known as Q-learning (Watkins 1989; Watkins and Dayan 1992). Its simplest form, one step Q-learning is defined by: $Q(x_t,u_t) \leftarrow Q(x_t,u_t) + \alpha[r_{t+1} + \gamma \underset{u}{Max} Q(x_{t+1},u) - Q(x_t,u_t)]$ where α is a constant step size parameter. In this case, the learned action-value function, Q, directly approximate Q^*, the optimal action-value function, independent of the policy being followed. This dramatically simplifies the analysis of the algorithm. The policy still has an effect in that it determines which state-action pairs are visited and updated.

---

[3] The expected value is important because in general the system and the policies are not deterministic, so many different evolutionary pathways can emerge from a single policy. Expected discounted reward over all these pathways informs the calculation of Q function.

[4] Off-policy methods do not require the following of a single policy for multiple time steps before improving on that policy.

The basic idea behind Q-learning is that the value of any state-action pair ($Q(x,u)$) depends on both the immediate reward received from that pair (r_{t+1}), and the value of the best state-action pair accessible from the resulting state ($\gamma \underset{u}{Max} Q(x_{t+1},u)$). This formulation effectively reduces the complex problem of estimating the value of different state-action pairs into a readily available piece (immediate reward) and an iteratively estimated piece (the value of the resulting state).

In order to obtain a more general algorithm that may learn more efficiently, any TD method such as Q-learning can be combined with eligibility traces. The basic idea is to allow not only the next visited state to impact the value of the current state-action, but also to involve value of states visited in multiple steps to have an impact on the current state's value. The eligibility trace $\zeta(x,u)$ of the state-action pair *(x,u)* is a temporary record of visiting state *x* and taking action *u*. The eligibility trace version of Q-learning is called Q(λ) which updates the action-value function $Q(x_t,u_t)$ by:

$$Q(x_t,u_t) \leftarrow Q(x_t,u_t) + \alpha\,\zeta(x_t,u_t)[r_{t+1} + \gamma \underset{u}{Max} Q(x_{t+1},u) - Q(x_t,u_t)]$$

where $\zeta(x_t,u_t) = \begin{cases} \gamma\lambda\zeta(x_{t-1},u_{t-1}) & \text{if } x_t \neq x_{t-1} \text{ and } u_t \neq u_{t-1} \\ 1 & \text{if } x_t = x_{t-1} \text{ and } u_t = u_{t-1} \end{cases}$.

Although Q(λ) is a powerful algorithm for solving learning tasks, it applies to problems with discrete state and action space. Since most problems of interest in our setting involve continuous state variables, a discretization process is essential before Q(λ) can be applied. In essence, the discretization process partitions the continuous state space into a finite number of subsets each of which represents a discrete space of the dynamic system. Convergence to the optimal solution is of prime importance which is guaranteed under appropriate conditions; However, the general form of the problem suffers from the curse of dimensionality and thus can be computationally intractable (Bellman 1957). For example consider a system dynamics model with two stock variables: inventory and workforce. If we discretize each variable into ten different regions (e.g. inventory between 0-0.1Inv_{Max}, 0.1 Inv_{Max} -0.2 Inv_{Max}, etc; Inv_{Max} is the maximum inventory we expect to be observed in this system) then 100 different states are defined for the discretized system. The different points within each of these states are assumed to have same Q and V functions. Additional stock variables can be similarly discretized and included, but the obvious cost is the exponential growth in the number of states for which the optimum value function should be estimated. A simple discretization method applied with Q(λ) algorithm to a simple system dynamics model is one of the two algorithms we will use in this paper.

Aggregation of the states of the problem is one of the methods for simplifying the state representation and reducing the dimensionality of the problem. The basic premise is that state aggregation offers a simplified set of states with minimum sacrifice in terms of relevant information. Simplified states can be used to design simple information cues for human learners that allow individuals to learn without being overwhelmed by complexity of the problem. In continue we review the previous work on state abstraction and discuss the discretization process we use in this paper.

State Space Discretization

The early work on state abstraction focused on network-type dynamic models where a final goal state could be identified in a discrete state space (Mendelssohn 1982; Bean, Birge et al. 1987). These methods created macro-nodes (states) from aggregation of multiple nodes, solved the aggregate problem, and then decomposed the solution for the original problem. Bertsekas and Castanon (Bertsekas and Castanon 1989) improved this method with adaptive definition of macro-nodes. Convergence results are obtained for some classes of these algorithms (Singh, Jaakkola et al. 1995). Moore and Atkeson (Moore and Atkeson 1995) provided an algorithm for dynamically splitting a continuous state space into finer grids. However, this algorithm is limited to tasks with a final goal state. Tsitsiklis and Van Roy (Tsitsiklis and VanRoy 1996) offer a general framework for compact representations including state aggregation and provide performance bounds for two general algorithms. Lee and Lau (Lee and Lau 2004) used vector quantization to partition the state space into a set of K disjoint regions. Each region is represented by a code word (i.e. the center point in that region of continuous space). The value for each region is constant and the regions are adaptively created and trimmed with new information. Their algorithm therefore simplifies an exponentially complex state space into a number of code words (examples) in which the agent's learning is embedded. This method may be specially relevant for human learners, as research on cognitive architectures suggest that an example-based process explains human learning very well (Gonzalez, Lerch et al. 2003; Anderson, Bothell et al. 2004) and codewords are synonymous with examples. Further research suggests substantial gains can be achieved through combining state-abstraction with temporally extended actions that consist of several simple actions (Provost, Kuipers et al. 2006).

Generally, the learning and state aggregation algorithms are used to simplify the state space, map out the value of different regions, and find good policies to manage the simulated development process. State aggregation algorithms differ in their implementation and suitability for the task at hand. In this paper we modify the algorithm developed by Lee and Lau (Lee and Lau 2004) to a simple managerial task. The aim of this algorithm is to find an efficient discretization of the continuous state space. The partitioning of the state space is done based on a set of codewords, c_i, which are representative points (of different partitions) of the state space. Each point in the state space is assigned to one of these partitions depending on which codeword the point is closest to (e.g. in Euclidian distance). A perfect partitioning would put all the neighboring state-actions with similar values in the same partition. However, the value of different states is unknown (and in fact the main target of learning), thus the state aggregation and learning algorithms iteratively interact with each other and improve on their partitioning and value estimates. The aggregation algorithm provides the function, $\boldsymbol{g}$, for the interpretation and evaluation of different states, while the learning module provides the action-value function that guides adjustments in the state aggregation.

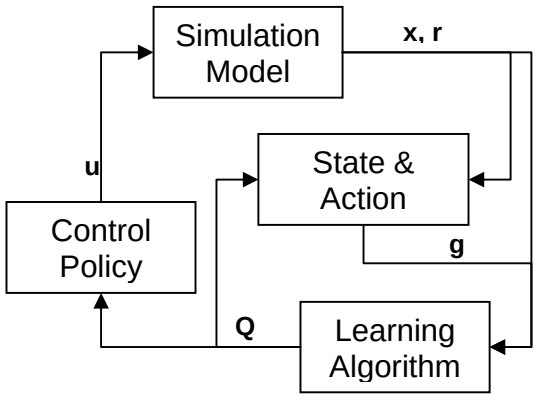

Figure 1: Learning and state aggregation process

Initially the value function is constant for all the states in the space and the algorithm starts with a single codeword, viewing all the state-action space to have the same value. New experiments (state *x* and reward *r* information) signal the need for appending the code word list with a new member when the following two criteria hold:

1. The difference between action-value function of the pair (x,u) and value function related to the closest codeword exceeds a threshold χ. This criterion limits the variation of action-value function within a particular cell.
2. The Euclidian distance between the new code word and its nearest neighbor code word is greater than a threshold value Δ. This criterion maintains a predefined minimum resolution for practical implementation reasons.

By appending new codewords, new regions with different values are defined in the state space. The learning algorithm can now learn about the value of these regions. Through this process of appending, the partitioning of the state-action space is continuously improved along with the precision of action values learned through the learning module. The number of resulting partitions depends on the thresholds used for appending operation and can be adjusted to obtain the desirable balance between quality of the resulting strategy and the complexity of the partitions for providing insight to human learners. Figure 1 represents an overview of the learning and state abstraction processes described above. The simulation model receives the decisions (u) of a control policy and applies them to the system which leads to the change in the state of the dynamic system[5]. The state (x) and the rewards (r, e.g. revenue-costs) are then passed to the state aggregation and learning modules.

If the optimal value function is known a priori, each region defined by the corresponding codeword should include all the states with approximately the same value, and optimal action. However, TD learning is a computational algorithm that iteratively improves its estimate of the value function. Therefore, the codewords have to be defined and adjusted along with the TD operations to reflect the most recent estimate of the value function. Exhibit 1 discusses the discretization process and the TD learning algorithm that are carried out simultaneously based on the real-time feedback signals gathered as the system interacts with its environment.

In the section 3 a number of simulated experiments are performed to demonstrate the properties and efficacy of the described algorithms.

[5] The change in the state of dynamic system is achieved by simulating the system dynamics model from the current time to the next period.

Define reward threshold (χ) and minimum resolution (Δ)

Initialize the first codeword c_1 (e.g. c_1 = initial state)

Initialize $Q(c_1, u) = \omega$, $\forall u$ (ω is any arbitrary value)

Initialize eligibility trace $\zeta(c_1, u) = 0$, $\forall u$

Initialize x_t , u_t arbitrarily

determine codeword c_t using the nearest neighbor rule: $\| x_t - c_i \| \leq \| x_t - c_j \|, \forall\, i \neq j$

execute action u_t

Repeat

observe next state x_{t+1} and immediate reward $g(x_t, u_t, x_{t+1})$

determine activated codeword c_{t+1} using the nearest neighbor rule:

$$\| x_{t+1} - c_i \| \leq \| x_{t+1} - c_j \|, \forall\, i \neq j$$

calculate the action value function corresponding to the pair (x_t, u_t):

$$\hat{Q}(x_t, u_t) := g(x_t, u_t, x_{t+1}) + \gamma \underset{i}{Max} \{Q(c_{t+1}, u_i)\}$$

delta$Q$ = $\hat{Q}(x_t, u_t)$ - $Q(c_t, u_t)$

if ($Q(c_t, u_t) \neq 0$ and delta$Q > \chi$ and $\| c_t - x_t \| > \Delta$)

append the new codeword $c_{new} := x_t$

initialize and append $Q(c_{new}, u) := \omega$ $\forall u \neq u_t$ and $Q(c_{new}, u_t) := Q(c_t, u_t)$

$\zeta(c_{new}, u) := 0$, $\forall u$

$c_t := c_{new}$

update c_{t+1} after appending a new codeword: $\| x_{t+1} - c_i \| \leq \| x_{t+1} - c_j \|$, $\forall\, i \neq j$

end

calculate the TD error: $\tau := g(x_t, u_t, x_{t+1}) + \gamma \underset{i}{Max} \{Q(c_{t+1}, u_i)\} - Q(c_t, u_t)$

update the eligibility trace of current codeword-action pair: $\zeta(c_t, u_t) := \zeta(c_t, u_t) + 1$

refine the estimate of action value function using the TD learning rule:

$$Q(c, u) := Q(c, u) + \alpha\tau\zeta(c, u), \forall c,u$$

decay eligibility trace: $\zeta(c, u) := \lambda\gamma\zeta(c, u), \forall c,u$

determine u_{t+1} for c_{t+1} using the ε-greedy policy derived from $Q(c_{t+1}, u)$

$c_t := c_{t+1}$, $u_t := u_{t+1}$

execute action u_t

end of trial

Exhibit 1: The complete aggregation process and Q(λ) learning algorithm

3. Simulation Experiments

The dynamic system we have considered for implementing control algorithms in this section is a simple organizational task accomplishment model composed of one main stock. It is a core structure in models of organizational performance, where task accomplishment rate depends on the stock of "Tasks Under Development" (Rudolph and Repenning 2002). This stock represents the load of tasks that are under development at any point in time and impacts the level of pressure on organizational members. The core feature of this model is the inverse U-shaped function of how Tasks Under Development impact Task Completion rate. At lower levels of backlog of tasks, incremental pressure increases the productivity of the organizational members and thus the Task Completion. However, beyond some optimum level of pressure, further increase will simply *reduce* the overall performance due to stress, burnout, and increased error rate and thus rework. The inverse U-shaped relationship is documented across different domains of activity (Rudolph and Repenning 2002; Rahmandad and Repenning 2006; Taylor and Ford 2006). The table function relating Tasks Under Development and Task Completion is shown in Figure 2. The maximum Task Completion (27 tasks per unit of time) is achieved when Tasks Under Development are at one hundred level.

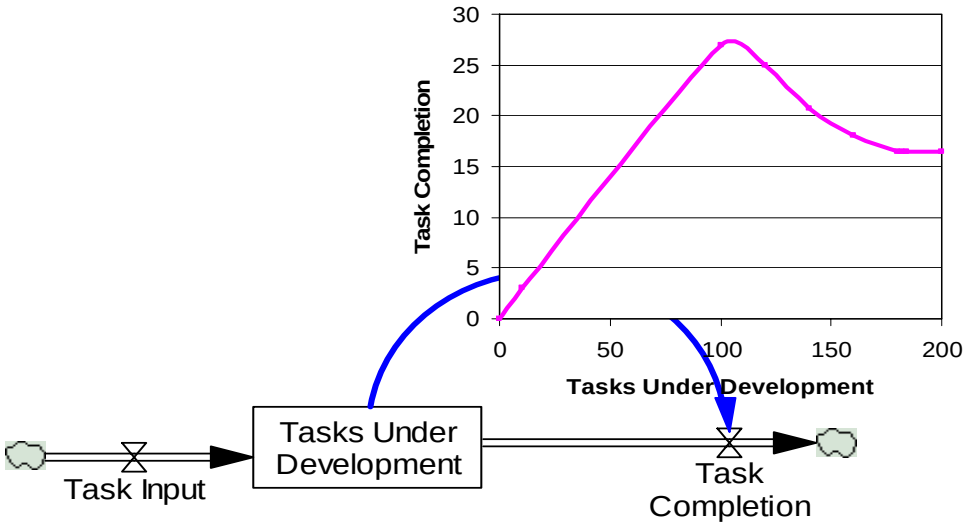

Figure 2- The overview of the dynamic model to be controlled in the experiments. A single stock of tasks under development determines the dynamics in this system. Task Input is the policy lever that is under manager's control. Task Completion is a non-linear function of Tasks Under Development (as shown in the figure). Maximum task completion is achieved when tasks under development=100.

The major control lever at management disposal is the "Task Input" rate, which represents the demand of the management from the organization. Management wants to maximize the output of the system by selecting the best input levels. However, the tipping dynamics in this model can make the control policy non-trivial. If the stock of tasks under development is pushed beyond its optimal level (here 100), the system can tip into a mode where reduced Task Completion leads to increased stock levels, and further reductions in the task completion. Therefore the Task Input should be contingent upon

the current state of the system: for lower levels tasks under development, Task Input could be higher than maximum output feasible, for levels higher than optimum stock level, however, it should be aggressively lowered to avoid tipping into problematic dynamics. Therefore this simple task provides a good test bed for examining algorithms that should find control policies contingent on state of the system.

In this model the number of tasks under development forms our one-dimensional continuous state space. Newly defined tasks which is the inflow rate to the stock of "Tasks Under Development" is our main decision variable or control policy that should be determined with respect to the stock of tasks under development such that keeps the productivity of the crew at its maximum possible level. Note that no priori knowledge regarding the dynamic of the system is available to the decision maker (learner); instead learner has to approximate the optimal control policy π (the number of tasks that should be assigned at any point of time depending on the state of the system) by learning through interaction with the system. We use this dynamic model as the task for online (adaptive) partitioning of the state space according to the action-value function profiles which are estimated and updated by reinforcement learning algorithm Q(λ).

The simulation model receives the decision u, i.e. the Task Input rate, and applies them to the simulation model above. The simulation is continued for one period with this fixed Task Input. As a result the state of the system (x), which is the number of tasks under development, changes. The accumulation of task completion rate over the whole period is considered as the system reward (r) for the action u taken at state x in the beginning of the period. The new system state (x) and reward (r) are then passed to the state aggregation and learning method. The control policy u in this algorithm is derived based on the current estimate of the state-action value function and an exploration strategy called ε-greedy policy. In brief, the ε-greedy policy selects actions stochastically based on the current estimates of action value function. All the possible actions are selected uniformly with probability ε; otherwise the action with the highest estimated value will be selected (with probability of 1- ε). The admissible action which is the number of tasks that can be assigned is also a continuous variable in general, but for simplicity in this simulation model, we assume that the number of newly defined tasks is selected from a predefined set of admissible actions U = {0, 8, 16, 24, 32, 40}. More granularity is achievable by having more discrete actions.

In all of the following simulation results values for parameters Δ and χ have been set to 20 and 10, respectively. Epsilon in ε-greedy algorithm is equal to 0.05, discount rate γ equals to 0.7 and λ is 0.6. In order to run the simulation we have used Anylogic 6.2 Advanced. The simulation result after 500 trials is shown in Figure 3. In this result simulation model has been run for 4000 time periods. Note that the initial state of the system which means the initial number of tasks under development is set randomly in our simulation (thus starting in average from 200 tasks). Moreover, we reset the state of the system randomly every 100 periods, so that the system visits the different points in the state space. Therefore tasks under development and payoff rate jump every 100 periods.

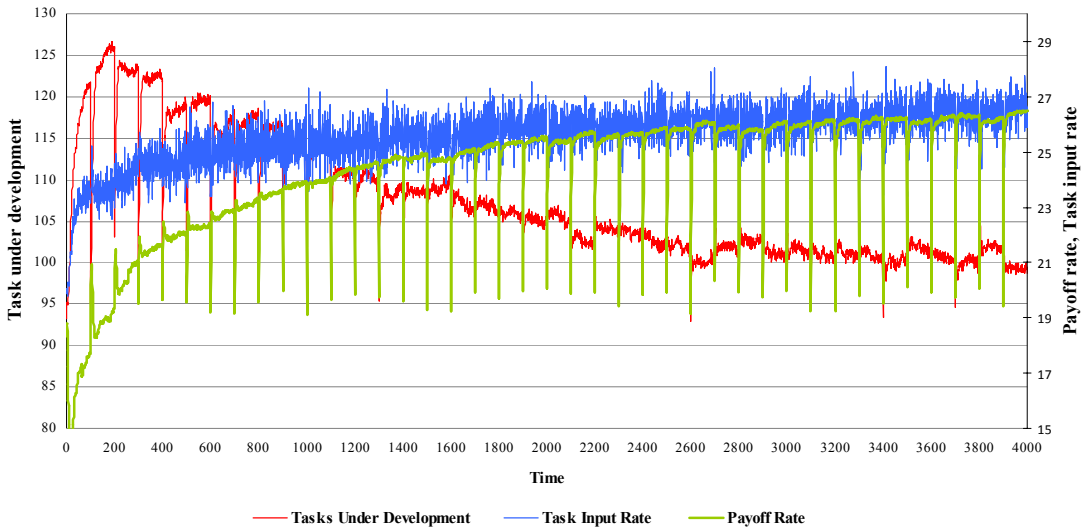

Figure 3: Task Under Development, Task Input Rate and Payoff Rate

As we can see in Figure 3, the adaptive algorithm in section 2 has been successful in learning through interaction with the system, since the number of tasks under development has gradually converged to its optimal value which is equal to 100 tasks and thus the PayoffRate (which is the one-period accumulated Task Completion) converges to 27. As the graph shows tasks under development, task input rate and payoff rate start their converging behavior roughly around period 2500. The optimal policy found by the algorithm, in majority of cases, consists of having high (32 or 40) task input level when the system is in a state with codewords smaller than 80, and having very little task input (0 or 8) when in codewords beyond 120. The optimum task input for the codewords around effective tasks under development (100) is often 24 or 32. The exact policy found in each simulation depends on the codewords that have emerged however.

Parameter epsilon (ε) plays an important role in our algorithm. Positive values for epsilon enable the algorithm to maintain a level of exploration that better estimates the value of new states and actions. It is not enough just to select the actions currently estimated to be best, because then no action-value function will be obtained for alternative actions and states, and it may never be learned that they are actually better, or change the current estimates of other state-action pairs. On the other hand large values for epsilon, select the actions more randomly to cover all possibilities. Unlikely for real organizations with high performance pressure, such purely explorative policies degrade the performance of the organization because they never exploit what is already learned. Figure 4 represents the result of running the simulation model under two possible values of epsilon, $\varepsilon = 0.05$ and $\varepsilon = 0.2$. The rest of the settings in the simulation model are the same and equal to the settings described above.

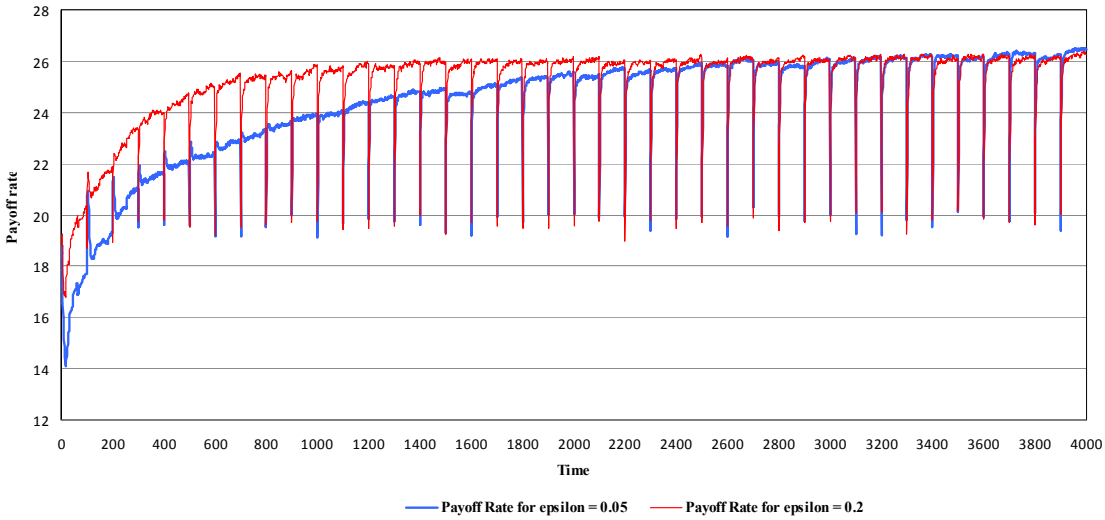

Figure 4: Comparison of payoff rates for different values of epsilon

As we can see in Figure 4, during the early periods at which not much experience and learning has been accumulated in the system, the payoff corresponding to epsilon = 0.2 is considerably greater than the payoff related to epsilon = 0.05. This is due to the fact that in the early periods not enough experience has been accumulated in the systems, so return on exploration is high; it enables the simulated firm to visit more alternative state-actions and thus estimate the value functions more efficiently. As time passes, the learning from experiences increases, so learner can rely more on the accumulated experience (exploitation) than exploration. Thus towards the end, lower epsilon actually pays off slightly better by keeping the organization on the optimum path for longer. A gradual reduction in epsilon, as done through softmax exploration functions, benefits from both good initial exploration and higher final exploitation.

In running the simulation model, we reset the system after a fixed number of time periods by setting the value of tasks under development equal to a random umber between zero and 200 (maximum value for task under development). This causes the simulation model to navigate through all possible values of the state space which leads to the generation of a set of codewords that can more uniformly partition the state space. Figure 5 shows the results of running the simulation with and without such reset. As we can see the payoff obtained from running the simulation with reset dominates in all periods over the reward obtained without resetting the system.

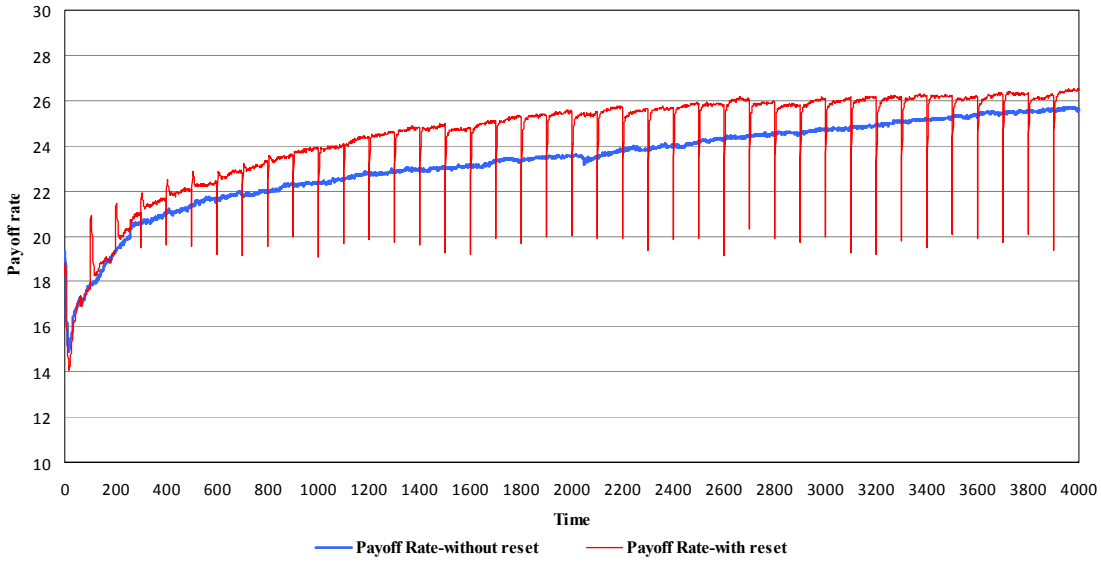

Figure 5: Comparison of payoff rate with and without resetting the system

To investigate the merits of adaptive codewords over fixed codewords, we run a comparative simulation with a fixed set of codewords. The same TD method and set of parameters were used to run the simulations with both algorithms. In the fixedcode algorithm we experiments with 5 and 10 codewords that have uniformly been selected in the interval 0 to 200. The distances between codewords are equal to each other and have been set equal to 40 and 20 as a result. Figure 6 shows the payoff resulteing from learning over fixed codeword in comparision with the payoff obtained from learning over adoptive (dynamic) codewords. The adaptive codeword simulation resulted in nearly 6 codewords. As we can see from Figure 6, during the entire learning process which comprises of 500 trials over 4000 time periods, the payoff resulted from adoptive codeword has almost always dominated the payoffs obtained from fixed codewords. Moreover, the fewer codewords are actualy helpful for quick initial learning, eventhough long-term performance is slightly degraded as a result of coarser coding.

In general, the advantage of adaptive state space partitioning over fixed partitioning is mainly due to the reduction in the number of distinguishable states, while effectively partitioning the state space. Fixed partitioning considers a discrete space with 10 (or 5) states, adoptive codeword simulation model produces just near 6 aggregate states, in average, given the current parameter settigns. Since the entire state space comprises a smaller number of distinguishable states, the chances of visiting all the states and taking every admissible action in each state are higher despite fixed ε-greedy exploration strategy. This improves the convergence of the control policy without sacrificing the quality of the learned policy.

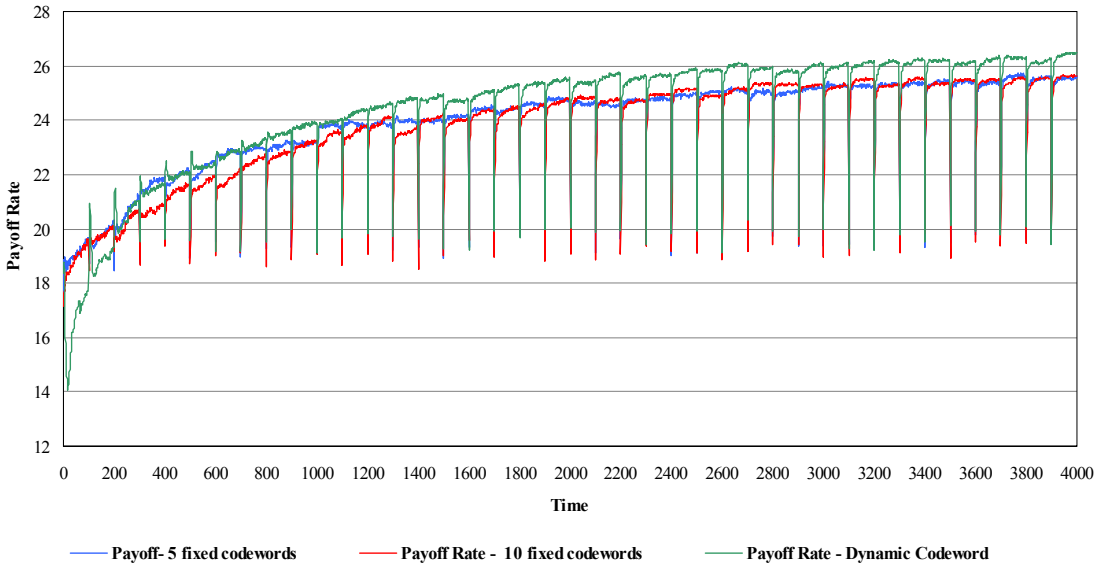

Figure 6: Comparision of payoffs for different number of fixed codewords

4. Discussion and future extensions

In this paper we discussed the value of more formal methods for finding efficient control policies in models of dynamic social phenomena. To address this problem we introduced solution concepts from artificial intelligence and operations research literature that can be applied to problems of interest for system dynamics community. We elaborated on Q-learning as one of these algorithms and established proof of concept for an extension to this algorithm that allows for emergence of more effective discretization of state-space. Simulation results were used to illustrate the basic concepts and their applicability to a very simple dynamic model of an organization.

Introduction of methods and algorithms that enrich modelers' toolboxes is an important contribution of this study. Formal methods for finding efficient policies for managing dynamic social systems have been limited in their capabilities. This study introduces an additional set of tools and algorithms for interested researchers and practitioners. The main value of these methods is in enabling the analyst to find optimal interventions at any state of the system and at any time, regardless of the history, or the stochasticity of system evolution. Building on the example in introduction, the control policy developed through a reinforcement learning algorithm can tell the managers what price they should pick depending on their inventory levels as well as the competitor's price at any time. This is a great improvement over the traditional optimization methods that only find a set of parameters fixed throughout the simulation to maximize the payoff function starting from a unique initial state.

Moreover, documented algorithms and models are made available for inspection and as building blocks for future work. We found that by providing both a modeling environment and a programming language inside the same software, Anylogic ™ simplifies the algorithm development and implementation significantly. In fact, with a little modification, the current reinforcement algorithm can be customized and applied to

any dynamic model that is built inside the Anylogic environment. We hope this arrangement encourages more modelers to try the methods introduced in this paper.

We also hope that this study opens more collaboration and cross-fertilization across system dynamics and operations research. The evolution of the field of system dynamics over last four decades has been largely separated from many potentially valuable developments in operations research, optimization, and control theory. Learning about these developments and their potential application to important real-world problems that system dynamics is most interested in, benefit both fields. Mathematical methods to find effective control policies for dynamic systems are among the most important areas of synergy. While traditional system dynamics studies can provide realistic and well-grounded models of dynamic problems, optimal control methods can shed light on better ways for managing and solving these problems, and potentially provide interesting insights not easily accessible through traditional tools in system dynamics.

Besides the obvious application to finding more efficient policies for managing dynamic social systems, we see at least two derivative applications of the work at hand. First, the optimal value function derived for different state-actions can be used for comparing different organizational conditions. For example, the field of performance measurement is concerned with comparing the value of organizations in different states (Kaplan and Norton 2005): is it better to have a lot of well-trained individuals but poor production equipment, or to have high quality production facilities but poor workforce? The value function of the two configurations for the same organization informs the relative value of these two cases, providing a concrete way of evaluating different organizational assets that pay off with different time lags.

Second, the derivation of code-words as we discussed, leads to the generation of relevant examples that can be used for training individuals to learn about successful policies. An individual can be put in charge of managing a simulated social system (e.g. an organization) in a management flight simulator (Sterman 2001) and offered start points that correspond to important code-words learnt through reinforcement learning. These code-words can organize the learners experience to maximize the value of different trials, for example starting from most advantages states and moving to ones that lead to those and so on, avoiding cognitive overload due to task complexity (van Merrienboer and Sweller 2005).

Future research can take on multiple shortcomings of this study. The code-word appending procedure we introduced in this paper is a first step towards a more flexible method of state aggregation in continuous space. A second element that should be added in the future research includes the merging of different code-words in order to trim the set of code-words towards those with most value added. The iteration of appending and merging code-words can help the algorithm converge to a more homogeneous set of code-words in terms of adding value for the learning task at hand. These methods should also be expanded for systems with arbitrary number of states, to be applicable for realistic problems.

The Q-learning algorithm we discussed in this paper is one of several approaches in the reinforcement learning field. It has been among the most widely used in the literature (Kaelbling, Littman et al. 1996). It follows a simple conceptual structure in which value function for the continuously improving policy is estimated at the same time as the algorithm explores the alternative policies. Other methods such as SARSA may separate

the estimation of value function for a control policy, and the improvement of control policy itself (Sutton and Barto 1998). Many successful methods for continuous state space are based on more complex function approximators that rely on an emerging function of current state (e.g. a linear function) to estimate the value function. This function is in turn updated based on new data as the experiments progress (Bradtke and Barto 1996; Lin 2003). Actor critic methods separate the algorithm into two parts, one (critic) maintains the estimated value function while the other (actor) chooses the next action in each state (Konda and Tsitsiklis 2003). The wide variety of available algorithms calls for additional research to find the most appropriate method for the dynamic models of social phenomena. Comparatives studies that measure the quality and speed of learning control policies using alternative algorithms in classic dynamic models (e.g. Forrester 1968; Cooper 1980; Sterman 1985) would be informative for this purpose.

References:

Andersen, D. F., J. A. M. Vennix, et al. (2007). "Group model building: problem structuring, policy simulation and decision support." Journal of the Operational Research Society **58**(5): 691-694.

Anderson, J. R., D. Bothell, et al. (2004). "An integrated theory of the mind." Psychological Review **111**(4): 1036-1060.

Athans, M. and P. L. Falb (1966). Optimal control; an introduction to the theory and its applications. New York,, McGraw-Hill.

Bean, J. C., J. R. Birge, et al. (1987). "Aggregation in Dynamic-Programming." Operations Research **35**(2): 215-220.

Bellman, R. E. (1957). Dynamic programming. Princeton,, Princeton University Press.

Bertsekas, D. P. (2007). Dynamic programming and optimal control. Belmont, Mass., Athena Scientific.

Bertsekas, D. P. and D. A. Castanon (1989). "Adaptive Aggregation Methods for Infinite Horizon Dynamic-Programming." Ieee Transactions on Automatic Control **34**(6): 589-598.

Bertsekas, D. P. and J. N. Tsitsiklis (1996). Neuro-dynamic programming. Belmont, Mass., Athena Scientific.

Bradtke, S. J. and A. G. Barto (1996). "Linear least-squares algorithms for temporal difference learning." Machine Learning **22**(1-3): 33-57.

Cooper, K. G. (1980). "Naval Ship Production: A Claim Settled and a Framework Built." Interfaces **10**(6).

Ford, A. and H. Flynn (2005). "Statistical screening of system dynamics models." System Dynamics Review **21**(4): 273-303.

Ford, D. N. and J. D. Sterman (1998). "Expert knowledge elicitation to improve formal and mental models." System Dynamics Review **14**(4): 309-340.

Forrester, J. W. (1961). Industrial Dynamics. Cambridge, The M.I.T. Press.

Forrester, J. W. (1968). "Market Growth as Influenced by Capital Investment." Industrial Management Review **9**(2): 83-105.

Forrester, J. W. (1971). "Counterintuitive behavior of social systems." Technology Review **73**(3): 52-68.

Gonzalez, C., J. F. Lerch, et al. (2003). "Instance-based learning in dynamic decision making." Cognitive Science **27**(4): 591-635.
Hines, J. H. (2003). Creating Better Models Faster : System Dynamics Molecules and Their Use. Proceedings of the 21st International Conference of the System Dynamics Society, New York City, USA, The System Dynamics Society.
Kaelbling, L. P., M. L. Littman, et al. (1996). "Reinforcement learning: A survey." Journal of Artificial Intelligence Research **4**: 237-285.
Kampmann, C. E. and R. Oliva (2006). "Loop eigenvalue elasticity analysis: three case studies." System Dynamics Review **22**(2): 141-162.
Kaplan, R. S. and D. P. Norton (2005). "The balanced scorecard: Measures that drive performance." Harvard Business Review **83**(7): 172-+.
Konda, V. R. and J. N. Tsitsiklis (2003). "On actor-critic algorithms." Siam Journal on Control and Optimization **42**(4): 1143-1166.
Lee, I. S. K. and H. Y. K. Lau (2004). "Adaptive state space partitioning for reinforcement learning." Engineering Applications of Artificial Intelligence **17**(6): 577-588.
Lee, J. M. and J. H. Lee (2004). "Approximate dynamic programming strategies and their applicability for process control: A review and future directions." International Journal of Control Automation and Systems **2**(3): 263-278.
Lin, C. K. (2003). "A reinforcement learning adaptive fuzzy controller for robots." Fuzzy Sets and Systems **137**(3): 339-352.
Mendelssohn, R. (1982). "An Iterative Aggregation Procedure for Markov Decision-Processes." Operations Research **30**(1): 62-73.
Mojtahedzadeh, M., D. Andersen, et al. (2004). "Using Digest to implement the pathway participation method for detecting influential system structure." System Dynamics Review **20**(1): 1-20.
Moore, A. W. and C. G. Atkeson (1995). "The parti-game algorithm for variable resolution reinforcement learning in multidimensional state-spaces." Machine Learning **21**(3): 199-233.
Provost, J., B. J. Kuipers, et al. (2006). "Developing navigation behavior through self-organizing distinctive-state abstraction." Connection Science **18**(2): 159-172.
Rahmandad, H. and N. P. Repenning (2006). Dynamics of Multi-release New Product Development. Academy of Management Conference, Atlanta.
Randers, J. (1980). Elements of the system dynamics method. Cambridge, Mass., MIT Press.
Rudolph, J. W. and N. P. Repenning (2002). "Disaster Dynamics: Undrstanding the Role of Quantity in Organizational Collapse." Administrative Science Quarterly **47**: 1-30.
Santamaria, J. C., R. S. Sutton, et al. (1997). "Experiments with reinforcement learning in problems with continuous state and action spaces." Adaptive Behavior **6**(2): 163-217.
Singh, S. P., T. Jaakkola, et al. (1995). Reinforcement Learning with Soft State Aggregation. Advances in Neural Information Processing Systems. G. Tesauro and D. Touretzky, Morgan Kaufmann. **7**.
Sterman, J. (2000). Business Dynamics: systems thinking and modeling for a complex world. Irwin, McGraw-Hill.

Sterman, J. D. (1985). "A Behavioral-Model of the Economic Long-Wave." Journal of Economic Behavior & Organization **6**(1): 17-53.
Sterman, J. D. (2001). "System dynamics modeling: Tools for learning in a complex world." California Management Review **43**(4): 8-+.
Sterman, J. D. (2007). "Exploring the next great frontier: system dynamics at fifty." System Dynamics Review **23**(2-3): 89-93.
Sutton, R. S. and A. G. Barto (1998). Reinforcement Learning: An Introduction. Cambridge, The MIT Press.
Taylor, T. and D. N. Ford (2006). "Tipping point failure and robustness in single development projects." System Dynamics Review **22**(1): 51-71.
Tsitsiklis, J. N. and B. VanRoy (1996). "Feature-based methods for large scale dynamic programming." Machine Learning **22**(1-3): 59-94.
van Merrienboer, J. J. G. and J. Sweller (2005). "Cognitive load theory and complex learning: Recent developments and future directions." Educational Psychology Review **17**(2): 147-177.
Vennix, J. A. M. (1999). "Group model-building: tackling messy problems." System Dynamics Review **15**(4): 379-401.
Watkins, C. (1989). Learning from delayed rewards. Ph.D. Thesis. Cambridge, UK, King's College.
Watkins, C. J. C. H. and P. Dayan (1992). "Q-Learning." Machine Learning **8**(3-4): 279-292.