Reinforcement Algorithms Using Functional Approximation for Generalization and their Application to Cart Centering and Fractal Compression

Clifford Claussen, Srinivas Gutta and Harry Wechsler
Department of Computer Science
George Mason University
4400 University Dr.
Fairfax, VA 22030
U.S.A

Abstract

We address the conflict between identification and control or alternatively, the conflict between exploration and exploitation, within the framework of reinforcement learning. Q-learning has recently become a popular off-policy reinforcement learning method. The conflict between exploration and exploitation slows down Q-learning algorithms; their performance does not scale up and degrades rapidly as the number of states and actions increases. One reason for this slowness is that exploration lacks the ability to extrapolate and interpolate from learning and to a large extent has to "reinvent the wheel". Moreover, not all reinforcement problems one encounters are finite state and action systems. Our approach to solving continuous state and action problems is to approximate the continuous state and action spaces with finite sets of states and actions and then to apply a finite state and action learning method. This approach provides the means for solving continuous state and action problems but does not yet address the performance problem associated with scaling up states and actions. We address the scaling problem using functional approximation methods. Towards that end, this paper introduces two new reinforcement algorithms, QLVQ and Quad-Q-learning, respectively, and shows their successful application for cart centering and fractal compression.

1 Introduction

Reinforcement learning is "the on-line learning of a mapping from situations to actions so as to maximize a scalar reward or reinforcement signal. The learner is not told which action to take, but instead must discover which actions yield the highest reward by trying them. In most interesting and challenging cases, actions may affect not only the immediate reward, but also the next situation, and through that all subsequent rewards. These two characteristics - trial-and-error search and delayed reward - are the two most important distinguishing features of reinforcement learning" [Sutton, 1992].

In what is called on-policy learning [Sutton and Barto, 1998], the agent learns to predict the long-term reward from a fixed policy. With on-policy learning, the learning agent does not change its policy function during learning. The agent only learns to predict the long term expected reward for each state of the environment given that it maintains the same policy and it learns the value function V^{π}. With off-policy learning, on the other hand, the learning agent changes its policy during learning so as to improve long term expected reward. The objective of off-policy learning is to find an optimal policy function, i.e., to find the policy that always leads to the best possible long term expected reward. Off-policy learning is accomplished through a process of trial and error. The learning agent has to probe the environment in order to determine the directional information needed to modify its behavior. This probing action slows the operation of the learning agent because the changes in behavior required to explore the environment are generally in conflict with how behavior should be changed to exploit the gradient information determined by exploration. This problem is known as the conflict between identification and control or alternatively, the conflict between exploration and exploitation.

2 Q-Learning

Q-learning derives the optimal policy function incrementally as it interacts with the environment. It assumes no a-priori knowledge of rewards or transition probabilities. The idea of Q-learning is to learn a Q-function that maps the current state *s* and action a to a utility value Q(s, *a)* that predicts the total future discounted reward that will be received from current action *a* and from all subsequent actions, assuming the optimal policy, π^* is followed in subsequent actions. The optimal policy π^* is determined

from the Q function as $\pi^*(s) = \text{argmax}_a Q(s, a)$. The objective of Q-iearaing is to learn the Q function directly without ever explicitly learning transition probabilities and expected values of rewards. Suppose that we know the Q function and that currently the environment is characterized by the state i. An agent then chooses an action a so as to maximize $Q(i, a)$. Choosing action a results in an environmental state transition from state i to state j. The agent then chooses the next action b so as to maximize $Q(j, b)$. Given that the optimal policy is followed after action a is taken, $Q(i, a)$, which is the estimate of utility of an action a taken from state i, is just the immediate reward of taking action a from state i plus the maximum utility possible from the next state j, discounted by the discount factor λ. Therefore Q satisfies

$$Q(i,a) = R(i,a) + \lambda \sum_j P_{i,j}(a) \max_b Q(j,b)$$

where $R(i, a)$ is the immediate reward received when action a is taken from state i and $P_{i,j}(a)$ is the probability of transitioning to state j, given that the current state is i. O-learning employs a one step back-up using the estimated Q function value at the state of the next time step [Watkins and Dayan, 1992].

3 Learning and Functional Approximation

The basic Q-learning algorithm is very slow to converge. One reason for this slowness is that the Q function must be learned for each state and action pair independently; having a good idea of the value of Q(i, *a)* provides no information about the value of Q(j, *b)* for nearby state *j* and action *b.* Algorithms may converge to local optima or may not converge at all. With on-policy learning, the convergence of the value function, V^π is all that is of concern because the policy remains fixed. However, with off-policy learning, the objective is to find an optimal policy by incrementally changing it to improve system performance. Hence with off-policy learning, the exploration aspect is important and convergence to the optimal policy is of primary concern, although convergence of the value function is also important.

For finite state and action systems, both on-policy and off-policy algorithms exist that can be proven to converge [Sutton and Barto, 1998]. The problem that arises with these algorithms is that although they converge, they do not scale up well; that is their performance degrades rapidly as the number of states and actions increases. Moreover, not all reinforcement problems one encounters are finite state and action systems. Our approach to solving continuous state and action problems is to approximate ("discretize") the continuous state and action spaces with finite sets of states and actions and then to apply a finite state and action learning method. This approach provides the means for solving continuous state and action problems but does not address the performance problem associated with scaling up states and actions. We address the scaling problem using functional approximation methods. Sect. 4 and 6 describe two new reinforcement algorithms, QLVQ and Quad-Q-learning, respectively, that address the discretization and scaling up problems.

4 QLVQ

Q-learning performance deteriorates as the dimension of the state space increases. For the case when the policy function is a piecewise constant function with relatively smooth transition regions, we describe in this section QLVQ, a novel reinforcement algorithm, that tessellates the state space into regions with piecewise smooth boundaries using Labeled Vector Quantization (LVQ) [Kohonen, 1990]. The motivation behind integrating LVQ and Q-learning comes from the fact that LVQ (i) discretize the phase space for control problems and significantly reduces the state space requirements, and (ii) improves the overall accuracy as it estimates ('interpolates') between neighboring cells.

The new Q - learning and Learning Vector Quantization (QLVQ) algorithm is shown in Fig. 1. Steps 1 through 5 are similar to those employed by Q-learning. First one positions *K* (random) action prototypes z_{ij} for each of the *C* possible actions $A = \{a_1, \ldots, a_C\}$ on the state *X* space. The action a_i consists of prototypes $\{z_{i1}, \ldots, z_{iK}\}$, whose labels correspond to their positioning on the state space *X,* A total of KC prototypes are used by the LVQ component of QLVQ. If prototype z_{ij} is nearest to state *x* then the action selected is $\pi(x) = z_{ij}$, while ties are arbitrarily broken. The estimate for the utility $Q(x, a)$ is then updated as it was the case with the conventional Q-learning by combining the immediate reward r with a discounted utility estimate from the next state *y.* Let dQ be the change in the value of Q after Step 5. The prototypes z_{ij} are then repositioned (at iteration *t*) (see Step 6).

Initialize the action-value function Q and the learning rates α and β.

1. While stopping condition is false
2. Randomly generate state x.
3. Select an action *a* to execute: $\pi(x) = z_{ij}$.
4. Execute action *a*, and let *y* be the next state and r be the reward received.
5. Update $Q(x,a) : Q(x,a) \leftarrow (1-\alpha)Q(x,a) + \alpha[r + \lambda U(y)]$
6. Let dQ be the change measured after Q update (step 5)

$$z_{ij}(t+1) = z_{ij}(t) + \text{sign}(dQ)\,\beta(t)\,[x(t) - z_{ij}(t)]$$
$$\text{and} \quad \beta(t) \dashrightarrow 0$$

7. Update the policy function π: $\pi(x) \leftarrow a$ such that $Q(x,a) = \max_{b \in A} Q(x,b)$

Figure 1. QLVQ Algorithm

5 Cart Centering Using QLVQ

Control problems represent a good test bed to assess reinforcement algorithms due to their need to determine decision control boundaries expressed as policy (stimulus -action) functions (mappings). The cart-centering problem, discussed in this section, was chosen because its analytical solution is readily available and comparative performance is thus feasible. Cart centering is modeled using a phase space representation and the control aspect is handled using QLVQ reinforcement learning.

Phase space for a dynamical system refers to the entire set of allowable states. For a non-autonomous system, such as cart centering, an external control variable, in addition to the initial state and a specified control function, determine the future trajectory of the system. For discrete control problems such as cart centering ("bang-bang") control, the optimum policy or decision function is piecewise constant on disjoint regions of the phase space. These regions are separated by hypersurfaces called "switching boundaries". Determination of optimum control then reduces to the computation of these boundaries. The table look up approach of Q-learning, however, does not exploit the property of the policy function being piecewise constant. The LVQ algorithm exploits this property in the determination of the optimal switching boundaries and this motivates the hybrid QLVQ reinforcement learning scheme.

The cart-centering problem involves a cart that can move either to the left or to the right on a frictionless one-dimensional track [Bryson and Ho, 1975; Koza, 1992]. The problem is to center the cart in minimal time, by applying a force of fixed magnitude. The following state information is available: the current position of the cart along the track (x) and the current velocity of the cart (v). At each time step (t), the controller must decide in which direction the force should be applied to the cart so as to bring the cart to a target state of rest (velocity: 0.0; position: 0.0). The cart-centering problem can be described using the following set of equations:

$$a(t) = \frac{F(t)}{m}, v(t+1) = v(t) + \tau a(t), x(t+1) = x(t) + \tau v(t)$$

x: position of the cart, v: velocity of the cart, t: time, a: acceleration, r. size of the time step (normally set to 0.02 sec), m: mass of the cart (here 2.0 kg) and F: magnitude of force (here IN).

The exact (analytical) time-optimal solution specifies that for any given current position x(t) and current velocity v(t), the applied fixed force F would accelerate the cart in the positive direction **(u = +1),** if the inequality shown below holds true:

$$-x(t) > \frac{v(t)^2 \, Sign\, v(t)}{2|F|/m}$$

The applied fixed force F accelerates the cart in the negative direction **(u = -1)** when the above inequality does not hold. Fig. 2 depicts the computed time-optimal solution map for the cart-centering problem. The boundary between the shaded and unshaded portions of the graph is the switching boundary for the problem.

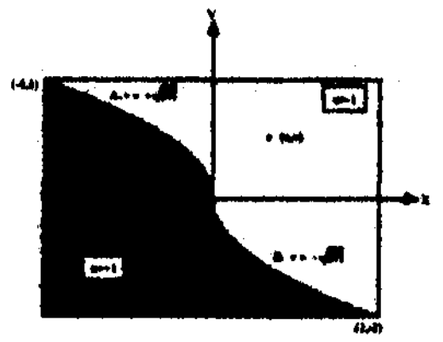

Figure 2. Time-Optimal Solution Map for the Cart-Centering Problem.

Standard Q-learning for cart-centering is run on two (x, v) grid sizes, (101, 501) and (202, 1002), respectively, to assess how it scales up. Q-learning computes the overall change in the policy Q function from one cycle to the next and should stop as soon as the change becomes less than a given threshold θ. Using such stopping criteria the Q-learning algorithm failed to otop for both grid sizes. When the phase space dimension is (101, 501), the Q learning algorithm is terminated after (arbitrary) 567 cycles. Note that even though a state is randomly selected, the chance of a state (x, v) not being considered during several cycles is very small as Q-learning is executed for hundreds of cycles, each cycle consisting of x * v iterations. Fig. 2a below shows the Q (phase state) - map for Q- learning algorithm after 176 cycles, the number of cycles required by QLVQ (see Fig. 2b) to stop, while Fig. 2c shows the Q - map after Q-learning has been aborted. Similar behavior for Q-learning is observed when the grid size is doubled to (202, 1002). Q-learning is aborted after 1,016 cycles, while QLVQ requires only 328 cycles before it stops.

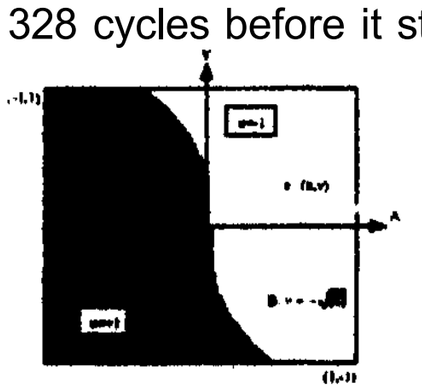

Figure 2a. Q - Map for Cart-Centering Using a Phase Space (101,501) Grid After 176 Cycles

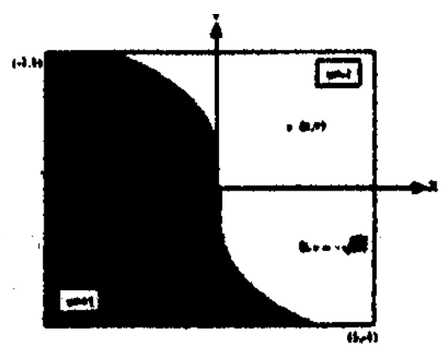

Figure 2b. QLVQ - Map for Cart-Centering Using a Phase Space (101, 501) Grid After 176 cycles

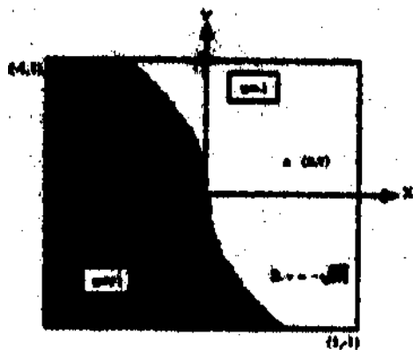

Figure 2c. Q - Map for Cart-Centering Using a Phase Space (101, 501) Grid After 567 cycles

We also assessed whether there exists a solution leading to the state of "rest", i.e., the state (x, v) = (0 ± ε, 0 ± ε), starting from every point in the phase space and within a specified time limit, e, arbitrarily set to 3, is used to compensate for the grid being discrete rather than continuous. The time limit for each point in the phase space is a slight increase (two) on the number of steps required by the analytical solution to reach the rest state.

If the grid size is of dimension (101, 501), then the number of points in the phase space trying to reach the state of rest neighborhood is 101 * 501 - 1 ('rest state') = 50,600 while for a grid size dimension of (202, 1002) the number of points is 202,4031 Experimental data shows that for a grid size dimension of (101, 501), 6, 578 or 13% of the points in the phase space, using Q - learning, do not reach the state of rest neighborhood within the time limit, while for QLVQ, only 3.9% of the points in the phase space do not reach the rest state neighborhood. For the case when the grid size is increased to (202, 1002), 24,086 or 11.9% of the points in the phase space, using the Q - learning solution, fail to reach the state of rest neighborhood within the time limit, while for the QLVQ solution 3,643 or only 1.8% of the points in the phase space, fail this test.

6 Quad-Q-Learning

Quad-Q-learning is an outgrowth of Q-learning. Quad-Q-learning, requires a different notion of state than that of Q-iearning. In Quad-Q-learning there are two types of actions: those that create new states, called *split* type actions, and those that do not, called *no split* type actions. When a split type action is taken, four new environments arise, each with their own state. When a no split type action is taken, the corresponding environment's state does not change and no further actions are taken with respect to that environment. To avoid confusion, we will use the term block attribute, or just attribute, with respect to Quad-Q-learning, instead of state. Quad-Q-learning is best understood in the context of a Quad tree partitioned image, where an attribute vector such as size and variance represents each image block in the Quad tree partition.

The objective of Quad-Q-learning is to learn a policy function, denoted as π, that maps local attributes of a *range* block to a decision on whether or not to split the block and, if the decision is not to split the block, whether to use the affine or Bath transform to code it in terms of the pool of available *domain* blocks. This decision is made in a way that will lead to a decompressed image of the desired quality while maintaining good compression rates. The policy function is learned by generalizing from a few image blocks to make decisions about an entire image.

With respect to fractal compression, it is more convenient to work with costs instead of rewards. Costs can be thought of as negative reward, but it will be convenient to use a version of Quad-Q-learning that can handle costs directly. We are actually interested in two costs, one corresponding to compressed image quality, and one corresponding to image compression rate. Instead of learning π directly, we learn a Q function that maps * block attribute vector and related action to an expected cost. The block attribute consists of two components, block size and block variance. For each block size and each action the value of the Q functions is assumed to be a linear function of range variance. Block size is a discrete value because of the nature of Quad tree partitioning, while variance, on the other hand, is a continuous attribute. Hence the local block attribute of a block can be denoted as (i, x), where i is an index of block size, and *x* is the continuous valued block variance. There are four possible actions: (1) code block using affine transformation, (2) code block using Bath transformation, (3) split as an affine block, and (4) split as a Bath block. Hence the expected cost of an action a on a block with attribute (i, x) is denoted as $Q(i, x, a)$. The policy function *n* is related to *Q* and learned using $\pi(i, x) = \arg\min_a Q(i, x, a)$. We assume that the Q function can be expressed as

$$Q(i, x, a) = m_{i,a}\sqrt{x} + b_{i,a}$$

We learn the *Q* function by learning estimates of $m_{i,a}$ and $b_{i,a}$. At each time ste *n* uring learning one obtains estimates $m^n_{i,a}$ and $b^n_{i,a}$, which are used to estimate $Q^n(i, x, a)$. At each time step *n* of the learning procedure, a potential range image block with attribute vector (i, x) is considered. A no split decision corresponds to a decision to code the block using the affine transform or the Br th transform. A split type decision corresponds to a cecision to split the block and use only the affine or only the Bath transform to code its sub blocks.

For simplicity, let $z_k = \sqrt{x^k}$ and $\bar{z} = \frac{1}{n}\sum_{j=1}^{n} z_k$ where *k* is an index representing the *kth* block of size "i" considered for action 'V (code using the affine or Bath transform) and *n* is the number of blocks visited, Let $c_k(i, x^k, a)$ be an instance of the cost of no split action "a" for a block

with attribute (i, x^k) and let $\bar{c}^n(i,a) = \frac{1}{n}\sum_{k=1}^{n} c_k(i, x^k, a)$.

Then one way to estimate $m_{i.a}$ and b is as follows:

$$m_{i,a}^n = \frac{\sum_{k=1}^{n}(z_k - \bar{z}^n)c_k(i, x^k, a)}{\sum_{k=1}^{n}(z_k - \bar{z}^n)^2} \qquad (A)$$

$$b_{i,a}^n = \bar{c}^n(i,a) - m_{i,a}^n \bar{z}^n \qquad (B)$$

where the denominator of (A) is assumed to increase infinitely as *n* increases without bound. Equations (A) and (B) are just the equations for calculating regression coefficients where z_k are the independent variables and c_k are the dependent variables. In other words we are estimating the parameters of a linear function relating the block variance and the cost of coding. The split action estimation procedure is similar. In the no split case, we are simply trying to estimate the dependent variable cost, as a linear fimction of block variance. The split case is the same, except that the dependent variable is estimated from estimates of the children blocks. That is, we are bootstrapping our estimates.

7 Fractal Compression Using Quad-Q-Learning

The image compression process involves image presmoothing, lean domain pool construction, block classification, Quad-Q-learning, fractal compression, and parameter quantization, coding and storage. Quad-Q learns a Q function which is used to partition an image and to make decisions about which transform, affine or Bath, to use for regions in an image based upon local block attributes. The fractal compression phase exploits the Q function to encode an image.

Quad-Q-learning learns a Q function that captures the relationship between cost, local block attributes and a decision on how to encode a block. A decision determines whether or not a block will be split and what transform method, either affine or Bath, will be used to encode a range block. We will actually use two different costs. The first cost estimates the quality of the reconstructed image measured as the root mean square (RMS) distance between the original image and the decoded image. The second cost estimates the compression rate measured as the number of bits that it takes to encode a block. These two costs will be used for learning two different Q functions, one corresponding to image quality and the other corresponding to image compression. Image quality, is estimated using the RMS distance between the transformed domain block and the range block [Barnsley, 1993].

Quad-Q-learning proceeds in two stages. The first stage learns the Q function for no split type decisions, while die second stage learns the Q function for split type decisions. In general a block at level m of a quad-tree is of size $2^m \times 2^m$. In stage 2 no costs are calculated; rather the costs calculated in stage 1 and the Q function estimates from the previous level of stage 2 are passed up in the quad-tree, similar to boot strapping [Sutton and Barto, 1998]. That is the cost of splitting a block of size $2^m \times 2^m$ is an estimate of the sum of the minimum Q function estimates of each quadrant block.

Fractal compression uses the Q functions as follows. First one calculates the variance for the largest range block. One then uses the variance, block size, and the Q function to determine the estimated image quality cost (RMS) of directly coding the block using each of the affine and Bath transforms. If both transforms are acceptable, we use the affine transform since it requires fewer bits. If neither Bath nor affine transforms will achieve the desired quality by direct coding, then we use recursive block splitting. The Q function is used to determine the expected RMS from block splitting. If both transforms achieve the desired RMS through block splitting, as determined by the Q function, then the transform type yielding the minimum product of expected RMS and expected bit rate is chosen. Otherwise, the transform with the best expected RMS is used. Once the transform for the block of largest size is chosen, only that transform is used to code sub-blocks of that block. For smaller blocks, if the block is of smallest size and therefore cannot be split, the block is coded directly using the current transform type. If the block is not of smallest size, then a split or no split decision is made by first calculating the variance for the block and then using the Q function to determine the estimated image quality cost of coding the block directly. If the cost of direct coding is acceptable, the block is coded directly; otherwise, the block is split.

Our novel scheme for fractal compression is evaluated by comparing between JPEG [Nelson, 1992], Fisher's fractal compression [Fisher, 1995] as implemented in FRACOMP [Kassler, 1995], and fractal compression with Quad-Q-learning, denoted as Learn-Frac in the graphics below. Both rate-distortion curves and compression-time curves are calculated and depicted for each of the aforementioned methods for several images. Fig. 3 shows comparative results for the aforementioned compression methods on the LENA image.

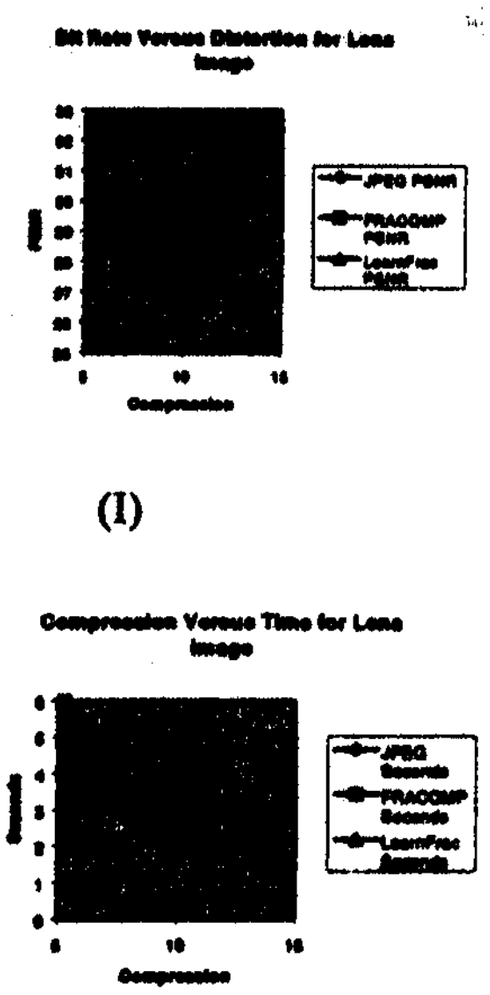

Figure 3. Comparison of Compression Methods

Bit rate (compression) is measured as the ratio of the file size of the original image measured in bytes to the file size of the compressed image measured also in bytes. Distortion (image quality) is measured as the peak signal to noise ratio (PSNR). The compression rate vs time to compression curves are important to ensure that improvements in the rate-distortion curves are not merely the result of increased compression time. In general, fractal compression with Quad-Q-learning requires approximately 0.5 seconds for learning time, independent of compression. The time advantage gained from learning at high PSNR more that offsets the cost of learning and hence fractal compression with Quad-Q-learning performs comparatively better than fractal compression without learning at low compression and high PSNR. For practical locations on the bit rate distortion curve, including the point of the crossover image, fractal compression with Quad-Q-learning generally performs better than fractal compression alone. The Quad-Q-learning method and the associated new fractal techniques taken together form a new fractal compression algorithm that frequently out performs other fractal compression algorithms with respect to the trade-off between bit rate, distortion, and compression time. At compression rates above 11:1, fractal compression with quad-Q-learning will produce better quality compressed images than JPEG and outperform it with respect to PSNR for high compression rates. JPEG will compress faster, but our proposed method still is well under 10 seconds for a 256 x 256 pixel image on a typical PC. In addition, the proposed method is faster than other fractal compression algorithms for the same compression rate and compressed image quality. Another advantage of our proposed method comes from the possibility of choosing the transform type. The Bath transform, because of its linear functional part, should be the choice for ranges with significant slope characteristics.

8 Conclusions

We addressed in this paper the conflict between identification and control or alternatively, the conflict between exploration and exploitation, within the framework of reinforcement learning. Our approach to solving continuous state and action problems has been to approximate the continuous state and action spaces with finite sets of states and actions and then to apply a finite state and action learning method. This approach provides the means for solving continuous state and action problems but does not yet address the performance problem associated with scaling up states and actions. We addressed the scaling problem through generalization using functional approximation methods. Towards that end, this paper introduced two new reinforcement algorithms, QLVQ and Quad-Q-learning, respectively, and showed their successful use of functional approximation for generalization purposes with application for cart centering and fractal image compression.

References

[Barnsley, 1993], *Fractals Everywhere,* Academic Press, San Diego, California.

[Bryson and Ho, 1975], *Applied Optimal Control,* Hemisphere.

[Fisher, 1995], *Fractal Image Compression, Theory and Application,* Springer—Verlag, Netherlands.

[Kassler, 1995], Fraktale Bildkompression unter Windows, Diplomarbeit im Studiengang Mathematik an der Mathematish-Naturwissenshaftlichen Fakultaet der Unversitaet Augsburg,

[Kohonen, 1990], The Self-Organizing Map, *Proceedings of the IEEE 78:1464*-1480.

[Koza, 1992], *Genetic Programming,* MIT Press, Cambridge, Massachusetts.

[Nelson, 1992], *The Data Compression Book,* M&T Books, San Mateo, California.

[Sutton, 1992], Special issue on Reinforcement Learning, In R. S. Sutton (Ed.), *Machine Learning,* 8(4): 1-395.

[Sutton and Barto 1998], *Reinforcement Learning, An Introduction,* MIT Press, Cambridge, Massachusetts.

[Watkins and Dayan 1992], Technical Note Q-Learning, *Machine Learning,* 8(4): 279-292.