Input Generalization in Delayed Reinforcement Learning: An Algorithm And Performance Comparisons

David Chapman and Leslie Pack Kaelbling
Teleos Research
576 Middlefield Road
Palo Alto, CA 94301 U. S. A.

Abstract

Delayed reinforcement learning is an attractive framework for the unsupervised learning of action policies for autonomous agents. Some existing delayed reinforcement learning techniques have shown promise in simple domains. However, a number of hurdles must be passed before they are applicable to realistic problems. This paper describes one such difficulty, the *input generalization problem* (whereby the system must generalize to produce similar actions in similar situations) and an implemented solution, the *G algorithm.* This algorithm is based on recursive splitting of the state space based on statistical measures of differences in reinforcements received. Connectionist backpropagation has previously been used for input generalization in reinforcement learning. We compare the two techniques analytically and empirically. The G algorithm's sound statistical basis makes it easy to predict when it should and should not work, whereas the behavior of backpropagation is unpredictable. We found that a previous successful use of backpropagation can be explained by the linearity of the application domain. We found that in another domain, G reliably found the optimal policy, whereas none of a set of runs of backpropagation with many combinations of parameters did.

1 Background

Delayed reinforcement learning is a framework for learning, without supervision, to act in an environment [Sutton, 1988]. In this framework, an agent is given on each tick, in addition to "perceptual" inputs, a numerical *reinforcement,* which is a measure of the *immediate* value of the state corresponding to the inputs. The goal of the agent is to choose actions to maximize the sum of reinforcement over time.[1]

Delayed reinforcement learning is attractive due to its similarity to the problem faced by a person or other creature placed in unfamiliar surroundings and expected to act intelligently. Such problems are of increasing theoretical and practical interest due to recent progress in the construction of autonomous agents such as mobile robots. Though such systems have achieved new levels of performance, they generally depend on elaborate hand-coded policies in computing how to act. When the environment for which they were designed is changed slightly, they may fail gracelessly; they are unable to adapt to new environments; and the process of hand-coding policies they require is slow and error-prone. Agents whose action policies are developed autonomously from a reinforcement signal might transcend all these problems [Chapman, 1991, Appendix B].

The bulk of experience with delayed reinforcement learning methods has been in simple domains that do not stretch their capabilities. The work described in this paper began by applying existing techniques to a more difficult task domain. This domain raised technical problems that our work here addresses.

1.1 Temporal difference learning

The best-understood approach to delayed reinforcement learning is *temporal difference (TD) learning,* codified by Sutton and his colleagues [Barto *et* al., 1989a; Sutton, 1988]. Two forms of TD learning have been studied in detail, the *adaptive heuristic critic* of Sutton [1988] and *Q-learning,* due to Watkins [1989]. Several authors have compared these methods empirically and found Q-learning superior [Kaelbling, 1991; Lin, 1990; Sutton, 1990], so we adopted Q-learning as our starting point.

Q-learning is based on estimating the values of *Q(i, a),* which is the *expected future discounted reinforcement* for taking action *a* in input state i and continuing with the optimal policy. The discounted reinforcement is the sum of all future reinforcement weighted by how close they are; specifically

$$Q(i(t), a(t)) = \sum_{j=0}^{\infty} \gamma^j \, r(t+j),$$

where t is the present time, $0 < \gamma < 1$ is a *discount factor* close to one, and *r{t)* is the reinforcement received

*This work was supported by the Defense Advance Research Projects Agency under NASA contract NAS2-13229.

[1] Or, more accurately, to maximize a future-discounted sum of reinforcement, as we will explain.

at time *t*. Thus the *Q* values say how good an action is not only in terms of immediate reinforcement, but also in terms of whether it gets the agent closer to future reinforcement; so in effect they allow the learner to dynamically create subgoals based on reinforcement that may be far in the future. Given *Q* values, we can also compute the value of a state as the value of the best action in that state:

$$U(i) = \max_a Q(i, a).$$

The learning algorithm works by creating a two-dimensional table of *Q* values indexed by actions and inputs and then adjusting the values in the table based on actions taken and reinforcement received. This process is based on the observation that *Q* of the current (input,action) pair can be computed based on the immediate reinforcement received and the value of the next input:

$$Q(i(t), a(t)) = r(t) + \gamma U(i(t+1)).$$

For further details, see [Kaelbling, 1991; Watkins, 1989].

1.2 The domain

The domain used in this research was the videogame Amazon, previously used in the system Sonja [Chapman, 1991]. More accurately, we studied an apparently simple subproblem from this domain, which turned out to be unexpectedly difficult.

In Amazon, the player controls an "amazon" icon which is attacked by ghosts. On each tick, the player can move the amazon one pixel in any of the eight "king's move" directions. Since the amazon is fifty pixels high, the player has very fine-grained control over motion. (The otherwise similar videogame domains used by other researchers have coarse motion control, in which the player icon moves its entire width in a single tick. We'll see that this difference matters.) Ghosts similarly move a single pixel on each tick. The player can cause the amazon to shoot a projectile in the amazon's direction of motion. Projectiles move in a straight line, four pixels per tick; if they collide with a ghost, the ghost dies. Thus, the amazon must be aligned with a ghost in one of the king's move directions to kill it. The reinforcement given is 10 when a ghost dies, otherwise —.1 if a shot is taken, otherwise 0.

The subproblem from Amazon we studied generates a ghost at a random distance and orientation from the amazon, waits for the amazon to kill it, and then repeats. We experimented with various forms of perception, some described in [Chapman and Kaelbling, 1990], all delivering a relatively small number of input bits.

This domain is difficult for several reasons. First, some states are rare, and it is hard to gain enough experience with them to find the optimal policy. Second, ghosts are programmed to avoid alignment with the amazon. When the amazon and the ghost are unaligned, there are limited opportunities for the system to learn. In particular, the system is not (locally) reinforced for actions that lead to killing the ghost, because it is impossible to kill the ghost while unaligned. Under a random strategy, the amazon and ghost will typically stay unaligned for stretches of several hundred ticks punctuated by brief periods of alignment terminated by ghost death. Thus most experience is of very limited value, and the interesting reinforcements (ghost deaths) occur infrequently. Third, there is a great variance in the value of states that the limited perception does not make available. How nearly aligned the ghost is and how close it is to the amazon determine how long it will be before it is possible to kill it. If the system gets a series of "easy" ghosts in a row it can readily come to wrong conclusions about a state value. This means that the learning rate must be sufficiently low to even such differences out; but that makes learning slow.

A fourth difficulty in this domain lead directly to the G algorithm. We hoped to use Sonja's visual system as an input representation for Q-learning, and that the learner could come to control this active visual system in which the computations performed are chosen top-down. Progress along these lines has been reported by Whitehead and Ballard [1990]. However, this visual system provides more than a hundred bits of input, corresponding to more than 2^{100} distinct inputs. This is a problem for two reasons, space and time. One simply cannot allocate an array one of whose dimensions is 2^{100}. Even if you could, it would divide the state space up into tiny pieces, each of which would occur extremely rarely, so the system could never accumulate enough experience to gauge the value of most states. Somehow a learning algorithm must guess about the value of states based on experience with similar states. But how can it know which states are similar when it has no experience with them?

2 The G algorithm

The G algorithm addresses this problem of input, generalization in a Q-learning framework.[2] Although generalization over inputs is a large field of study in machine learning, delayed reinforcement learning puts special constraints on the problem that make most general techniques inapplicable.

We can motivate the G algorithm in two ways. The first is to see it as collapsing the exponential sized *Q* table engendered by large numbers of input bits. In most domains, large chunks of the table should have identical entries, because many of the bits will be irrelevant to acting under certain circumstances. If we can figure out which bits are irrelevant, we can summarize a large region of the state space with only one *Q* value, thereby saving both space (to store the values) and time (since experience with any state in the region can be used to update the single *Q* value).

Another way of looking at the algorithm is as a technique for incrementally constructing the sort of action selection networks that have recently been used in various situated machine agents [Beer, 1990; Brooks, 1989; Chapman, 1991; Connell, 1990; Kaelbling and Rosenschein, 1990]. These networks are digital circuits that compute what to do based on perceptual inputs. The circuits are kept shallow in order to compute quickly when

[2] For further discussion of the algorithm, see [Chapman and Kaelbling, 1990].

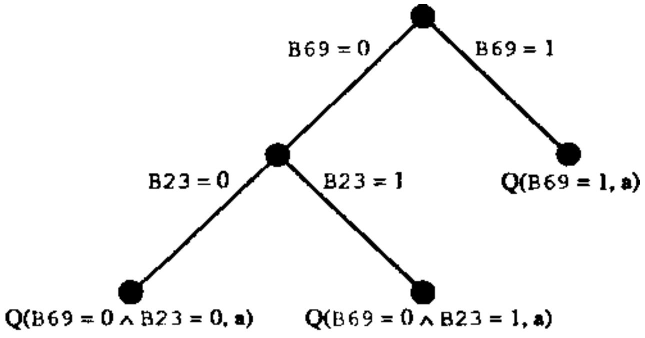

Figure 1: A G tree. Internal nodes correspond to input bit splits; the algorithm collects statistics about the subspaces represented by the leaf nodes.

implemented on slow, massively parallel hardware. Typically they maintain little or no state. Because the networks are shallow, they have the property that although every input bit is used in computing how to act, in most situations most bits are ignored. For example, Sonja searches for the amazon whenever the visual system reports losing track of it; all other inputs are irrelevant in this case.

The G algorithm incrementally builds a tree-structured *Q* table. It begins by supposing that all input bits are irrelevant, thereby collapsing the entire table into a single block. It collects *Q* values within this block. G also collects statistical evidence for the relevance of individual bits. When it discovers that a bit is relevant, it splits the state space into the two subspaces corresponding to the relevant bit being on and off. Then it collects statistics (action and relevance) within each of those blocks. These blocks can in turn be split, giving rise to a tree-structured *Q* table (figure 1). The system acts on the basis of the *Q* statistics in the leaf node corresponding to an input. Thus the tree acts as a boolean input classification network, essentially similar to the sorts of action networks described above.

The incremental, one-bit-at-a-time construction of the G tree puts a constraint on the sorts of environments that G can learn in: the relevance of bits must be apparent in isolation. The algorithm will fail if groups of bits are collectively relevant but individually irrelevant. If we consider the perceptual system of an agent to be part of the "environment" of its learning system, as we must, then this constraint can be placed on that system rather than the world. In other words, we hypothesize that *a well-designed perceptual system orthogonahzes inputs such that they are individually relevant.*

The G splitting technique is related to existing algorithms, such as ID3 [Quinlan, 1986] and CART [Breiman *et* a/., 1984], for inducing decision trees. The crucial difference is that the decision-tree algorithms are presented with input/output pairs rather than reinforcement data; for this reason, the statistical tests used to make splits must be different. Also, our work has emphasized making incremental decisions with a fixed amount of computation per tick rather than learning the shallowest or smallest tree.[3]

The G algorithm applies to input generalization. A similar problem arises on the output side: if the number of actions is very large, the learner can not hope to try each in every state. For example, Sonja's visual system has several dozen control bits that, the action policy must set on every tick. Kaelbling [1991] has described an approach to this problem. We believe, however, that the G algorithm should be directly applicable to the output generalization problem. That is, the system could keep track of the effect of individual output bits on reinforcement received in particular input blocks, and construct a tree of output bit relevance analogous to the input bit relevance tree. We have not implemented this, however.

3 Statistics

3.1 Discrete reinforcement: D statistics

We found the standard Q technique insufficiently sensitive in the Amazon domain. The problem is that Q simply sums all the reinforcement it gets, without distinguishing between different reinforcement values. For example, if the system is acting at random, as it does initially, it will typically have to shoot off many projectiles before killing a ghost. As the value of killing a ghost is only 10 and the cost of shooting is —.1, the 10 can get lost when summed with enough —.Is. To solve this problem, we extended Q to make more distinctions. Specifically, we effectively added a third dimension to *Q(i,* a), keeping track of D(z,a,r), the discounted future probability of receiving reinforcement *r* after performing action *a* given input *i:*

$$D(i(t), a(t), r) = \sum_{k=0}^{\infty} \gamma^{t+k} p(r(t+k) = r).$$

This extension separates out the various possible rein-for cement values and so gives better statistical information. The *Q* values can be recovered with the identity

$$Q(i, a) = \sum_{r \in R} r\, D(i, a, r)$$

where *R* is the space of reinforcements. This extension to Q-learning is possible only when the reinforcement given is discrete and takes on only a relatively small number of values (though it might be possible to use buckets to apply it in cases of continuously varying reinforcement).

It's not surprising that D-learning works better than Q-learning; it is superior for the same reason that Q-learning is better than the adaptive heuristic critic: it keeps track of more distinctions. The logical next step in this progression would be to keep track of input-action-input triples, as (for instance) Drescher [1991] has done. This raises questions of combinatorial feasibility, however.

[3] Utgoff's [1988] ID5 algorithm works incrementally, but a single instance can require a large amount of work if it causes a node to be "pulled up" in the tree.

3.2 The bit relevance test

C uses a standard statistical test, the *Student's t test* [Snedecor, 1989], to determine when a bit is relevant. The *t* test tells you, given two sets of data, how probable it is that distinct distributions gave rise to them. That is, how likely is it that these two sets of data arose from the same underlying process? This is just what we need in order to determine whether an input bit is relevant: is the learner/environment interaction the same when the bit is on versus off, or is it different?

Two sorts of relevance statistics are kept: a bit may be relevant because it affects the value of a state or because it affects the way the system should act in that state. Two sorts of statistics are used to determine value, corresponding to the mean *immediate* value of the state and its mean *discounted future* value. Both sorts are required; immediate value is used to "bootstrap" the process by recognizing the states that themselves give large reinforcements (e. g. those in which a projectile is flying toward the ghost) and discounted value is used to find states that lie on a path toward externally reinforced states (such as those in which the ghost and the arnazon are aligned). For each bit in each state block, G keeps track of the immediate and discounted values of the state block subdivided by the bit being on and off, and compares these values with the *t* test.

A bit may also be relevant because it affects how the agent should act; for example the input bits indicating the direction to the ghost do not affect the values of states, but they do determine which direction the player should head in. To discover such relevance, G keeps track, for each action in each state block, of the discounted value of taking the action in that state block when the bit is on versus when it is off, and compares these values with the *t* test.

- When a bit is shown to be relevant in a block, that block is split on the bit. When a block is split, all discounted statistics (both action value and relevance) must be zeroed. The reason is that a state block whose mean value is low may have a subblock whose value is high. Before the split is made, this high-valued subblock is effectively invisible, and the estimated values of all states that can transition to that subblock will be too low. Throwing away all experience accumulated thus far on each split seems too drastic. We are exploring ways of avoiding this, and expect that they will substantially increase the learning rate.

3.3 Enforcing normality

Unfortunately, the *t* test depends on the assumption that items sampled are distributed normally. Most statistical techniques make such assumptions.[4] The normality assumption is violated by Amazon, because the interesting reinforcement value (for killing a ghost) occurs so rarely. We found, as a result, that the test as specified in the last section often made incorrect judgements of relevance.

Normal statistics are frequently used to examine nonnormal data, and this is often successful due to the *central limit theorem* which states that the sum of a set of values from an arbitrary distribution will approach normality as the number of samples increases. We were able to eliminate most incorrect relevance judgements by delaying splitting until enough samples had been collected for their distribution to approach normality. This fix depends on a numerical threshold whose values may vary according to the domain. A better-motivated alternative would be to use statistical tests of normality (such as skew and kurtosis [Snedecor, 1989]) to decide whether enough samples have been collected to trust the data.

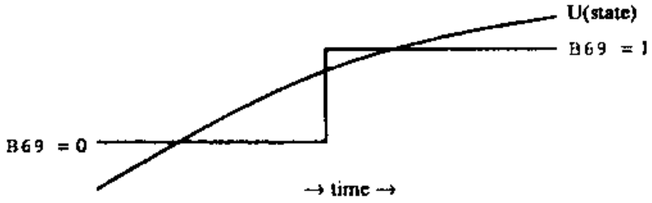

Figure 2: Noise bits that change slowly relative to estimated state values look relevant.

3.4 Low frequency noise

The techniques described in the previous sections were mostly sufficient in practice to ensure that the system did not split on irrelevant bits. An additional problem arose in some cases, however: while bits that changed rapidly presented no problems, irrelevant slowly-changing bits continued to pass the / test. Figure 2 illustrates the reason. If an input bit changes slowly relative to changes in estimated state values, the statistics collected to determine the discounted value of a subspace are skewed. In the figure, the estimated value of the state starts low and converges to a higher value. Initially the bit B69 is low, and later goes high. As a result, it will appear that B69 being on makes this subspace more valuable and the system will split.

The solution to this problem is to separate learning into action value and bit relevance phases. Estimated *Q* values are held constant while bit relevance statistics are collected. The system switches phases when values seem to have settled down, based on information about the derivatives of the statistical measures.

4 Performance comparisons

Relevance-splitting in G has performed well in our problem domain. We have run it for well over a million ticks with ten bits of noise given in addition to the standard inputs; it never split on any of these noise bits. On the other hand, on several runs on each of several variations of the problem G has always split on all the bits that *arc* relevant. Having done so, it has always learned the optimal policy for the domain. The total learning time for the simplest version of the problem runs around 35,000 ticks.

The system learns many times slower than Q on problems with few inputs bits, because it has to find the right

[4] An alternative is to use nonparametric statistics, which are unwieldy and seemed inappropriate to this domain (for reasons too complex to go into here).

splits before learning a policy. However, we have successfully run G on problems with thirty input bits, for which Q could not allocate memory to store its table, much less accumulate the billions of ticks of experience neccessary to fill it out.

Several other methods have been applied to input generalization for reinforcement learning. Watkins [1989] used the CMAC algorithm. Mahadevan and Connell [1990], working simultaneously with and independent of us, developed a statistical clustering method that is neatly dual to G. Rather than starting from a single merged state and splitting it, they start with fully differentiated inputs and merge them when they are found to behave similarly. We hope to compare G with CMAC and with this clustering technique in future work.

Anderson [1987] and Lin [1990] have successfully combined TD methods with connectionist backpropagation, which can generalize given a large number of input bits. Others [Chapman, 1991; Kaelbling, 1991; Shepanski and Macy, 1987] have attempted the combination and reported negative results; the combination of TD and backpropagation sometimes learns very slowly and converges to poor action choices. It is hard to resolve the discrepancy analytically because backpropagation is ill-characterized; it is impossible to know how the algorithm will perform when presented with a complex problem because it often converges to bad local minima.

To better understand backpropagation's success, we examined more closely Lin's domain, a video game in which a player collects food and avoids obstacles and enemies. It occured to us that potential field navigation, in which the direction of motion depends on a vector sum of attractive and repulsive forces, might be an adequate strategy for this domain, and furthermore that such a strategy would arise from a *Q(i,a)* function that is linear in Lin's retinotopic input representation. If this is the case, the good performance of backpropagation would not be surprising; by the perceptron convergence theorem, there should be no local minima to fall into.

We tested this hypothesis by using a linear associator in place of backpropagation. Figure 3 demonstrates that the linear learner does as well as backpropagation. This suggests that this domain is unexpectedly easy, and that the success of backpropagation should not necessarily be expected to transfer to other domains in which the *Q* function is nonlinear.

G would not work in this domain with the retinotopic input encoding because each of the 145 input bits is relevant in every situation. G would try to split on all of them and would soon generate too large a tree. However, as we have argued elsewhere [Chapman, 1991], the inputs to mammalian policy learning systems are almost certainly not retinotopic, and we should not try to optimize our learning systems for such inputs. We hypothesize that "intermediate" visual inputs should be easier to learn from than retinotopic ones.

The G algorithm is a more direct approach to the generalization problem than is backpropagation. It is mathematically well-characterized due to a sound statistical basis, and it is therefore easier to determine when and why it should or should not work. Given the difficulty in predicting the performance of Q plus backpropagation, a fair empirical comparison of the two methods would require tests on a spectrum of domains.

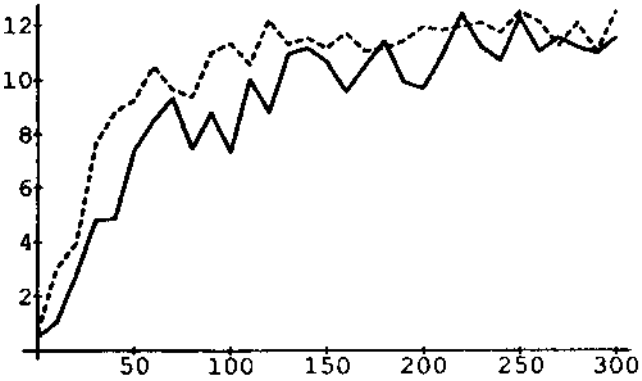

Figure 3: Typical learning curves for Q plus backpropagation (solid) and linear associator (dotted) in Lin's video game domain. The horizontal axis is games played and the vertical axis is the average number of pieces of food collected.

We tested Q plus backpropagation on the simplest Amazon problem with the same input representation used with G. Backpropagation has numerical parameters that must be tuned for a domain: the learning rate and the number of hidden units. We carried out several dozen runs with a wide variety of combinations of settings of these parameters. We checked every thousand ticks to see if the system had yet found the optimal policy; the average run length was about 70,000 ticks (twice the time required by (*), and some were as long as 210,000 ticks. Backpropagation never found the optimal policy. It is possible, though, that some other combination of parameter settings, or longer runs, would eventually find the solution. Also, our backpropagation engine does not implement momentum, so this parameter was effectively zero in all runs.

The input generalization problem is one of the most important in attempting to apply temporal difference learning to complex domains. Further analytic study and more detailed empirical testing, involving a spectrum of domains, is needed.

Acknowledgments

Thanks to Tom Dean, Long-Ji Lin, Stan Rosenschein, Stuart Russell, Steve Whitehead, and two anonymous reviewers for their various contributions to this work.

References

[Anderson, 1987] Charles W. Anderson, "Strategy Learning with Multilayer Connectionist Representations." In *Proceedings of the Fourth International Workshop on Machine Learning,* Ann Arbor, Michigan, 1987, pp. 103-114.

[Barto *et al.*, 1989a] A. G. Barto, R. S. Sutton, and C. J. C. H. Watkins, *Learning and Sequential De-*

cision Making. University of Massachusetts Department of Computer and Information Science Technical report 89-95, Amherst, Massachusetts, 1989.

[Beer, 1990] Randall D. Beer, *Intelligence as Adaptive Behavior: An Experiment in Computational Neuroethology.* Academic Press, San Diego, 1990.

[Breiman *et* a/., 1984] Leo Breiman, Jerome H. Friedman, Richard A. Olshen, and Charles J. Stone, *Classification and Regression Trees.* Wadsworth International Group, Belmont, California, 1984.

[Brooks, 1989] Rodney A. Brooks, "A Robot that Walks: Emergent Behavior from a Carefully Evolved Network." *Neural Computation* 1:2, Summer, 1989.

[Chapman, 1991] David Chapman, *Vision, Instruction, and Action.* MIT Press, 1991 (forthcoming).

[Chapman and Kaelbling, 1990] David Chapman and Leslie Pack Kaelbing, *Learning from Delayed Reinforcemeni in a Complex Domain.* Teleos Research TR-90-11, December, 1990.

[Connell, 1990] Jonathan Hudson Connell, *Minimalist Mobile Robotics: A Colony Architecture for an Artificial Creature.* Academic Press, San Diego, 1990.

[Drescher, 1991] Gary Drescher, *Made-up Minds: A Constructivist Approach to Artificial Intelligence.* MIT Press, 1991. Revised version of PhD Thesis, Department of Electrical Engineering and Computer Science, MIT, 1989.

[Kaelbling and Rosenschein, 1990] Leslie Pack Kaelbling and Stanley J. Rosenschein, "Action and Planning in Embedded Agents." *Robotics and Automation* 6 (1990).

[Kaelbling, 1991] Leslie Pack Kaelbling, *Learning in Embedded Systems.* MIT Press, 1991 (forthcoming). Revised version of Teleos Research TR-90-04, June 1990.

[Lin, 1990] Long-Ji Lin, "Self-improving Reactive Agents: Case Studies of Reinforcement Learning Frameworks." Carnegie Mellon University Technical Report CMU-CS-90-109, 1990. Also to appear in *Proceedings of the International Conference on Simulation of Adaptive Behavior: From Animals to Animats.*

[Mahadevan and Connell, 1990] Sridhar Mahadevan and Jonathan Connell, "Automatic Programming of Behavior-based Robots, using Reinforcement Learning." AAAI-91.

[Quinlan, 1986] J. Ross Quinlan, "Induction of Decision Trees." *Machine Learning* 1:1 (1986), pp. 81-106.

[Shepanski and Macy, 1987] J. F. Shepansky and S. A. Macy, "Teaching Artificial Neural Systems to Drive: Manual Training Techniques for Autonomous Systems." *Proceedings of the First Annual International Conference on Neural Networks,* San Diego, CA, June 21-24, 1987.

[Snedecor, 1989] G eorge W. Snedecor and William G. Cochran, *Statistical Methods.* Iowa State University Press, Ames, Iowa, eighth edition, 1989.

[Sutton, 1988] Richard S. Sutton, "Learning to Predict by the Method of Temporal Differences.[1]' *Machine Learning,* 3:1 (1988), pp. 9-44.

[Sutton, 1990] Richard S. Sutton, "Integrated Architectures for Learning, Planning, and Reacting Based on Approximating Dynamic Programming." *Proceedings of the Seventh International Conference on Machine Learning,* Austin, Texas, Morgan Kaufmann, 1990.

[Utgoff, 1988] Paul E. Utgoff, "ID5: An Incremental ID3." *Proceedings of the Fifth International Conference on Machine Learning,* Morgan kaufmaitn, Ann Arbor, Michigan, 1988, pp. 107 120.

[Watkins, 1989] Christopher John Cornish Hellaby Watkins, *Learning from Delayed Rewards.* PhD Thesis, King's College, Cambridge, 1989.

[Whitehead and Ballard, 1990] St even D. VVhitehead and Dana H. Ballard, "Active Perception and Reinforcement Learning." University of Rochester Computer Science Department Technical Report 331, 1990. To appear in *Machine Learning.*