Modeling and Planning in Large State and Action Spaces

Mykel J. Kochenderfer and **Gillian Hayes**
Institute of Perception, Action and Behaviour
School of Informatics, University of Edinburgh
Edinburgh, United Kingdom EH9 3JZ
m.kochenderfer@ed.ac.uk, gmh@inf.ed.ac.uk

Abstract

This paper introduces a general framework that integrates incremental model learning and reactive plan construction for real-time control of an agent situated in a stochastic world. This system is designed to achieve competent performance in continuous-time problems involving large or infinite state and action spaces. In order to learn and plan in such domains effectively, experience must be generalized over time, state, and action. This system achieves generalization by incrementally adapting a discretization of the state and action spaces as the agent gains experience and refines its plan. In contrast with other approaches, the framework presented in this paper makes no assumption about the representation of the state space, making it broadly applicable across relational and continuous-valued domains. The mechanism of this framework is illustrated on an artificial problem.

1 Introduction

This paper presents the "Adaptive Modeling and Planning System" (AMPS), a general framework for integrating incremental model learning and reactive plan construction for real-time control of an agent situated in a stochastic world. Under consideration are problems with large or infinite state and action spaces where individual actions operate in continuous time. The agent has no prior knowledge of the dynamics of the world or its task. The objective of the agent is to use the experience it gains through interacting with the world to maximize its expected discounted reward.

In order to make problems with large or infinite state spaces tractable, AMPS partitions the state space into regions and treats all states in the same region collectively. Since the action space is also large or infinite, AMPS also partitions the action space into regions and treats all actions from the same region identically. AMPS incrementally refines its partition of the state and action spaces by splitting and merging regions of the state and action spaces to arrive at a model that is consistent with its experience and conducive to building a successful reactive plan.

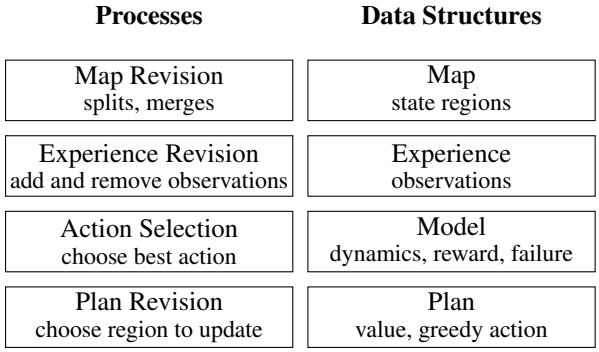

Figure 1: A high-level view of AMPS showing the four processes and the four data structures.

The primary contribution of this paper is a method for incrementally adapting the partitions of the state and action spaces by splitting and merging regions. One of the methods for splitting regions in AMPS is inspired by the work of Uther and Veloso [1998] where regions of the state space are split when there are observed differences in utility for different states within the same region. Another method for splitting regions in AMPS is inspired by the PARTI-GAME algorithm [Moore and Atkeson, 1995]. In PARTI-GAME and AMPS, the agent detects when it has become "stuck," meaning that the repeated application of the same action is not likely to achieve success in transitioning out of the current region. Both PARTI-GAME and AMPS use this information to refine their discretization. In AMPS this is known as failure revision. In addition to splitting regions, AMPS merges regions when possible, which is not typically done in other discretization algorithms.

AMPS is composed of four processes and four data structures as illustrated in Figure 1. The Map Revision process splits and merges regions of the state and action spaces. The Experience Revision process maintains past experience. The Action Selection process makes control decisions. The Plan Revision process incrementally improves the plan. These processes read from and modify the Map, Experience, Model, and Plan data structures.

The next section defines the problem under investigation, followed by a description of the data structures and processes

involved in AMPS. We then present our experiments and results. We conclude with comparisons to related work and a discussion of further research.

2 Problem Statement

The class of problems considered in this research involves an agent situated in a world. The agent continuously (or at high frequency) observes and acts in this world. At any point in time, the agent is in exactly one state belonging to a potentially infinite set $\mathcal{S}$ and the agent may execute exactly one action from a potentially infinite set $\mathcal{A}$. Actions may be *durative* meaning that they may be interrupted at any time (e.g. "rotate left at 1 rad/s") or *ballistic* meaning that they return control to the agent after some amount of time (e.g. "rotate left 1 rad").

The experience of the agent may be recorded as a sequence

$$s_1, a_1, t_1, f_1, r_1, s_2, a_2, t_2, f_2, r_2, \ldots$$

where s_k is the state at the kth sample, a_k is the action taken after the kth sample, t_k is the time spent between sample k and $k+1$, f_k indicates whether a failure was encountered after the kth sample, and r_k is the reward received after the kth sample. The objective of the agent after sample k is to maximize its expected discounted reward,

$$E\left[\sum_{k=0}^{\infty} e^{-\beta\sigma_{k+i}} r_{k+i}\right]$$

where $\sigma_k = \sum_{i=1}^{k} t_k$ and $\beta \in (0, \infty)$ is the continuous compound discount rate. Discounting the reward pressures the agent to aggressively pursue reward.

AMPS is expected to perform well on problems that are modeled sufficiently well by a semi-Markov decision process [Puterman, 1994]. It is assumed that $\mathcal{S}$ may be partitioned into a finite set of regions N and for each region $n \in N$ there exists a partition of $\mathcal{A}$ into regions $U(n)$ such that the transitions, reward, and failures are determined by fixed probability distributions:

- $P(\cdot|n,u)$ is the probability distribution over the regions transitioned to after starting in n and continuously applying actions in u.
- $P_t(\cdot|n,u,n')$ is the c.d.f. for the time required to transition from n to n' by continuously applying actions in u.
- $P_r(\cdot|n,u,n')$ is the c.d.f. for the lump sum reward received after transitioning from n to n' by continuously applying actions in u.[1]

If N, U, and these probability distributions are known, then the optimal value function, V^*, and optimal reactive plan, π^*, satisfy:

$$V^*(n) = \max_u Q(n,u,n') \quad (1)$$

[1]Other SMDP models considered in the literature [Bradtke and Duff, 1995] involve reward that is accumulated at some constant rate as opposed to lump sum rewards following transitions. In this paper we will only consider lump sum rewards, although AMPS can easily be extended to handle reward rate models.

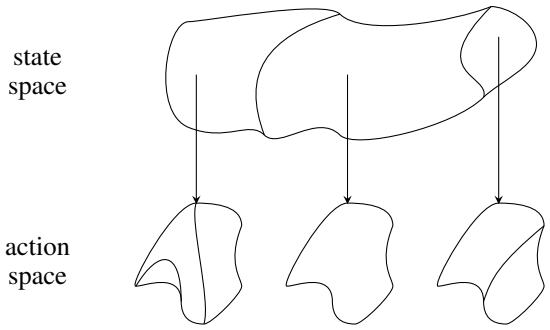

Figure 2: Associated with each region of the state space is a different partition of the action space.

$$\pi^*(n) = \arg\max_a Q(n,u,n') \quad (2)$$

where

$$\gamma(n,u,n') \triangleq \int_0^\infty e^{-\beta t} dP_t(t|n,u,n')$$

$$R(n,u,n') \triangleq P(n'|n,u)\gamma(n,u,n') \int_{-\infty}^{\infty} r dP_r(r|n,u,n')$$

$$Q(n,u,n') \triangleq R(n,u,n') + \sum_{n'} P(n'|n,u)\gamma(n,u,n')V^*(n)$$

The value function $V^*(n)$ is the expected discounted reward given that the agent starts in region n and follows an optimal policy. The expected discounted reward given that the agent starts with a transition from n to n' by actions in u and then follows an optimal policy is given by $Q(n,u,n')/P(n'|n,u)$, a value that will be used in the map revision process described in Section 4.1.

The agent has no prior knowledge of N, U, P, P_t, or P_r. AMPS incrementally adapts its partitioning of the state and action spaces and revises its estimates of P, γ, and R accordingly. As these estimates are revised, AMPS uses prioritized value iteration to improve its reactive plan.

3 Data Structures

The four data structures in AMPS store information about state and action regions, observations, world models, and plans. These data structures are manipulated by the processes described in the next section, and changes in one data structure can result in changes in another data structure. Of the four data structures, the Map is the most significant and so most of the following discussion will focus on it.

3.1 Map

The *Map* data structure maps states and actions to regions and adapts this mapping by splitting and merging regions. The map partitions the state space into a finite set of state regions, N. Associated with every state region is a finite partition of the action space, as illustrated in Figure 2. Let U be the set of all action regions defined across all state regions. The computational mechanism implementing the Map should be such that mapping from $\mathcal{S} \rightarrow N$ and $N \times \mathcal{A} \rightarrow U$ is efficient.

The representation used by the Map must allow the Map Revision process to split and merge regions. The Map implements the following functions to support revision:

1. SPLIT$(n, S_1, \ldots, S_m)$. This function splits the state region n into some number of new regions such that each set of states $S_1, \ldots, S_m \subset \mathcal{S}$ that were all mapped to n become mapped as well as possible to separate state regions.
2. SPLIT$(u, A_1, \ldots, A_m)$. This function splits the action region u into some number of new regions such that each set of actions $A_1, \ldots, A_m \subset \mathcal{A}$ that were all mapped to u become mapped as well as possible to separate action regions.
3. MERGE$(n_1, \ldots, n_m)$. This function merges the state regions $n_1, \ldots, n_m$.
4. MERGE$(u_1, \ldots, u_m)$. This function merges the action regions $u_1, \ldots, u_m$ belonging to the same state region.

The Map can be implemented, in principle, using any representation that supports these operations. For example, if a distance metric may be defined between any two states and any two actions, then a mapping based on nearest-neighbor [Cover and Hart, 1967] may be used. There have been computationally efficient algorithms and data structures, such as vantage point trees [Yianilos, 1993], suggested for computing nearest neighbors without making any additional assumptions about the representation of the space (e.g. that the space is Euclidean).

Another representation that can be used is a decision graph. Decision graphs are extremely well suited because they allow regions to be split and merged very quickly, and this is the representation used in the experiments here.

Merging regions is straightforward in a decision graph, but splitting regions requires a mechanism called a *separator* that produces the (typically binary) tests in the decision graph. The separator produces a test based on sample states belonging to multiple categories, much like a supervised learning classifier [Duda *et al.*, 2000]. The test returned by the separator should be simple. The condition should not be, for example, an entire decision tree, a neural network, or a lengthy sentence in first order logic. Instead, simple tests such as $x_2 < 2.3$ or $\exists x \, \text{ON}(\text{A}, x)$ should be used. Simple tests are to be combined using the decision graph to create more complex boundaries between regions.

The separator function must be engineered according to how the state and action spaces are represented. It is interesting to note that the separator function is the only component in the system that interacts with the state space representation directly, enabling the system to be applied across domains with only the separator requiring changing.[2]

[2]Other partitioning algorithms in the literature make strong assumptions about how states and actions are represented. For example, Chapman and Kaelbling [1991] and Munos and Patinel [1994] assume that states are represented as fixed-length binary strings, McCallum [1995] assumes an attribute-value representation, Dean *et al.* [1998] assume a factored state and action representation, and Mahadevan and Connell [1992] assume that states are represented as real-valued vectors.

In the experimental domain described later, the state and action space requires a representation in the form of a vector of real values. The separator used in the experiments was designed to run in time linear with respect to the number of samples. The function returns the condition that best separates the points along some axis-parallel hyperplane according to information gain [Shannon, 1948].

It should be noted that the mechanism for partitioning the state space does not have to be the same mechanism for partitioning the action space. For example, the state space may be partitioned using nearest-neighbor, but the action space associated with each state region may be partitioned using a decision graph.

3.2 Experience

The *Experience* data structure keeps a record of the states and transitions experienced by the agent and associates them with the state and action regions managed by the Map. When state and action regions are split and merged, the association of past observations to regions is updated appropriately.

3.3 Model

The *Model* data structure models state transition dynamics, failure, and reward. Let $P_f(n, u)$ be the probability of encountering failure when executing an action in u from region n. This may be estimated directly from experience as can $P(n'|n, u)$. Instead of estimating the c.d.f.s P_t and P_r directly, AMPS estimates γ and R instead. The integrals in γ and R are approximated as follows [see Kalos and Whitlock 1986, pp. 89–116]:

- $\int_0^\infty e^{-\beta t} dP_t(t|n, u, n')$ is approximated by averaging $e^{-\beta \tau_i}$ where τ_i is the time required to make the ith transition from n to n' by actions in u.
- $\int_{-\infty}^\infty r dP_r(r|n, u, n')$ is approximated by averaging the reward received while transitioning from n to n' by actions in u.

These approximations are used by the Plan data structure, which is described next.

3.4 Plan

The *Plan* data structure maintains the value function and the greedy policy. The estimated value function, $V(n)$, is an estimate of the expected discounted reward starting in region n and proceeding with an estimate of optimal behavior. The estimated greedy policy, $\pi(n)$, associates with each state region an estimate of the greedy action that maximizes the expected discounted reward.

As can be seen in the equations given in Section 2, the value and the greedy action associated with each state region depend upon the value of other state regions. The Plan data structure supports a function called UPDATE(n) that updates the value and greedy action of state region n using Equation 1 assuming that the values of all other regions are correct. The Plan Revision process is responsible for calling UPDATE on state regions in response to changes in the Model and Plan.

4 Processes

The four processes in AMPS are responsible for splitting and merging regions of the state and action space, recording experience, selecting actions, and constructing plans. All processes are designed to be incremental and perform (relatively) simple operations on the data structures at a frequency dependent on available computational resources.

4.1 Map Revision

The Map Revision process is responsible for dynamically splitting and merging state and action regions in response to changes in the Model and Plan. If the agent has no prior knowledge of how the state and action space should be partitioned when it commences its interaction with the world, the entire state space is contained within a single region. Over time, this region is incrementally refined and simplified as experience is acquired. The objective is to find a partition of the state and action space that has the following properties:

1. All transitions resulting from a greedy action have approximately the same estimated value.
2. The expected failure of greedy actions is minimal.
3. The partition is as simple as possible.

Different types of revision can be done to bring the Map closer to the criteria listed above. Each type of revision is given a priority at each region that indicates (heuristically) the likelihood that performing that particular type of revision will improve the Map. When the Map Revision process runs, it will perform the highest priority revision. The Map Revision process ignores revisions below a certain threshold so as to not over-fit potentially noisy experience.

The three types of revision used in this system correspond to the three criteria above, and they are are as follows:

Value Revision

This type of revision attempts to separate state-action observations that have resulted in transitions with differing estimated value. This separation can be done by splitting action regions within a state region or by splitting the state region. Deciding whether to split by state or by action can be done using information gain.[3] The priority of performing this type of revision is proportional to a measure of variation of estimated value. The experimental implementation separates samples involved in greedy transitions based on whether $Q(n, \pi(n), n')/P(n'|n, u)$ is above or below the mean. The priority is proportional to the variance of $Q(n, \pi(n), n')/P(n'|n, u)$.

Failure Revision

This type of revision separates states that have led to success from those that have led to failure. The priority of this type of revision for a state region is related to the estimated probability of failure in the Model.

[3] If the Map is implemented using a nearest-neighbor classifier, it does not make sense to make splits based on information gain since nearest neighbor can always separate two different sets (assuming that the same exact sample does not appear in both sets). Instead, an approach based on cross-validation or bootstrapping would be appropriate [Weiss, 1991].

Simplification Revision

This type of revision merges state and action regions when their distinction seems to be irrelevant to the task. Non-greedy action regions are merged with low priority. State regions are merged in two situations. The first situation occurs when $P(n'|n, \pi(n)) \approx 1$ and $\pi(n) \approx \pi(n')$, in which case n and n' are merged. Since action regions are partitioned differently in different state regions, it is necessary to define the equivalence relation $u \approx u'$, which means that all the actions experienced in u can be mapped to action region u' in the state region of u' and vice versa. The second situation when two state regions are merged is when $P(n|n', \pi(n')) \approx 1$ and $P(n|n'', \pi(n'')) \approx 1$ and $\pi(n') \approx \pi(n'')$. In this case, n' and n'' can be merged. These conditions for merging are related to the SQUISH algorithm described by Nilsson [2000] for deterministic teleo-reactive trees.

4.2 Experience Revision

The *Experience Revision* process adds state and transition observations to the Experience data structure. If the state-space is continuous and it is being sampled at a high frequency, it is impractical to add each sample. Instead, the agent should filter out samples that are not likely to be useful. The exact mechanism for determining significance is domain dependent, but one approach to filtering out states is to ignore all states until the agent has transitioned to a new state that is outside some threshold distance.

In addition to filtering out insignificant samples, the Experience Revision process is also responsible for removing old samples from the Experience data structure. The schedule for removing old samples depends upon memory and processor constraints. The removal of old samples also allows the agent to adapt to slowly changing environments.

4.3 Action Selection

The *Action Selection* process is responsible for continuously selecting a single action to execute. This process must be extremely efficient because it is typically executed as frequently as the agent samples the state of the world.

Usually, the agent should execute an action from the greedy region of the current state region as computed by the policy revision process. However, as with traditional reinforcement learning, there are issues with balancing exploration of the world and exploitation of the optimal policy. There have been many techniques proposed for balancing exploration with exploitation [Thrun, 1992], any of which may be used in this system.

As mentioned earlier, the agent is able to sense failures when they occur. What exactly counts as a failure depends upon the domain, but failures are generally situations where executing the same action repeatedly will not lead to success. In the Corner World domain described later, a failure might be trying to push through a wall. The Action Selection mechanism uses information about past failures to better direct the agent toward a goal.

4.4 Plan Revision

The *Plan Revision* process incrementally builds an optimal plan with the assumption that the current Model is correct.

This process calls the UPDATE(n) function supplied by the Plan data structure, which updates the value and greedy action region for n.

One way to arrive at an optimal policy is to repeatedly iterate through the state regions calling UPDATE until convergence [cf. Bellman 1957]. However, doing so is not likely to be feasible in real time since the Model changes in response to the Map Revision and Experience Update processes. Instead, it is better to use a method related to prioritized sweeping [Moore and Atkeson, 1993]. Prioritized sweeping was designed for solving Markov decision processes (MDPs) but it can be easily adapted for solving the class of problems under consideration here. The idea of the algorithm is to prioritize updates of regions based on observed changes in the value function from earlier updates.

5 Experiments and Results

In the Corner World domain the agent starts at a random location on the starting line and maneuvers along an L-shaped track to the finish line. The track is contained within a 10×10 meter square, and the width of the track is 1 m. Each experimental run consists of 20 episodes of this task. The continuous compound discount rate β was set to 0.01.

The state space is represented by the tuple (x, y) corresponding to the agent's location. The agent may translate in any direction at 5 m/s. The agent samples the environment and makes control decisions at 5 Hz. Also at 5 Hz, the position of the agent is perturbed by an amount selected from a normal distribution with 5 mm standard deviation.

Since AMPS has no prior knowledge about the dynamics of the world or its task, it considers all states to be equivalent. Hence, the agent must rely upon random exploration until it reaches a goal. Random exploration is not practical in the Corner World or most other interesting domains because the state space is so large, and therefore the agent needs some mechanism to bias it toward the relevant goal states [Whitehead, 1991]. In our experiments, we used a teacher to guide the agent to a goal for two training episodes. These two training episodes allowed AMPS to partition the state space and distribute the observed reward through the model. The teacher used in the experiments returned random actions 5% of the time.

For comparison purposes and to control for the information gained from the noisy teacher, we trained a decision tree based on the same teacher. The induction of the decision tree is done by using the separator function to partition the state space according to the actions taken by the teacher. This sort of supervised learning of control is known as *behavioral cloning* and has been successfully applied to a variety of domains including aircraft control [Sammut *et al.*, 1992]. The primary disadvantages of a behavioral clone is that it is entirely dependent on a good teacher and it is not able to adapt its behavior based on its own experience in contrast to AMPS.

In our experiments, AMPS was allowed to perform a merge or split of the state or action space at 5 Hz. Episodes terminated when the agent reached the finish line or after 100 s without success. The results of 100 repeated experiments are summarized in Table 1.

	Successes		Disc. Reward	
	mean	s.d.	mean	s.d.
AMPS	16.02	6.04	9.54	4.13
Clone	12.75	5.74	7.87	3.85
Teacher	20.00	0.00	13.00	0.18
Random	0.00	0.00	0.00	0.00

Table 1: Experimental results of various agents in the Corner World domain. Each experiment was repeated 100 times. Shown above are the means and standard deviations of the number of successes and discounted reward in each run of 20 episodes.

6 Related Work

There have been algorithms proposed that assume a discrete action space but partition a large state space using a decision graph such as the G-ALGORITHM [Chapman and Kaelbling, 1991] and U-TREE [McCallum, 1995]. Uther and Veloso extended the work by McCallum to continuous state spaces [1998] and temporally abstract actions [2003]. None of these algorithms consider continuous action spaces.

Other techniques have been proposed for solving problems with large or infinite state spaces that do not involve partitioning the state space into regions. For example, parameterized function approximators, such as neural networks, may be used with traditional reinforcement learning techniques to estimate the value function [Bertsekas and Tsitsiklis, 1996].

Doya [2000] developed a reinforcement learning framework for continuous state, action, and time that uses function approximation. However, it assumes that the dynamics of the system are deterministic and the state and action spaces are represented by real-valued vectors. Smith [2002] used self-organizing maps to handle continuous state and action spaces represented as vectors of real-values.

Other work has been done on problems with durative actions in relational domains [Benson and Nilsson, 1995; Ryan, 2004]. The way AMPS uses SMDPs to model durative actions connects to work done in hierarchical reinforcement learning [surveyed in Barto and Mahadevan 2003].

7 Conclusions and Further Work

This paper has introduced a real-time system that integrates the adaptive partitioning of the state and action space and the incremental planning over the estimated model to produce goal-directed behavior. The system was tested on the Corner World domain and was shown to quickly learn successful behavior in the presence of noise in the environment.

AMPS has been designed to be broadly applicable across domains and representations. The state and action space may be infinite, and actions may have a continuous effect on the world and may be of variable duration. The Map data structure has been defined generically so that states and action spaces may have any representation. The Map may be implemented using a decision graph, in which case a separator must be defined. Alternatively, the Map may be implemented using a nearest-neighbor classifier, in which case only a distance metric must be defined. In either case, the separator or

the distance metric can be built to exploit the structure inherent in the state and action representation.

AMPS is currently being tested in other more complex domains and compared against other methods such as traditional reinforcement learning with function approximation. Further work will investigate the use of different separators and distance metrics that take into account values of previous observations, making AMPS potentially applicable to domains where the current state is only partially observable.

Acknowledgments

The authors would like to thank Nils Nilsson and the anonymous reviewers for their helpful suggestions on an earlier draft of this paper.

References

[Barto and Mahadevan, 2003] Andrew G. Barto and Sridhar Mahadevan. Recent advances in hierarchical reinforcement learning. *Discrete Event Dynamic Systems*, 13(4):341–379, October 2003.

[Bellman, 1957] Richard Bellman. *Dynamic Programming*. Princeton University Press, 1957.

[Benson and Nilsson, 1995] Scott Benson and Nils J. Nilsson. Reacting, planning and learning in an autonomous agent. In K. Furukawa, D. Michie, and S. Muggleton, editors, *Machine Intelligence*, volume 14, pages 29–64. Oxford University Press, 1995.

[Bertsekas and Tsitsiklis, 1996] Dimitri P. Bertsekas and John N. Tsitsiklis. *Neuro-Dynamic Programming*. Athena Scientific, 1996.

[Bradtke and Duff, 1995] Steven J. Bradtke and Michael O. Duff. Reinforcement learning methods for continuous-time Markov decision problems. In Gerald Tesauro, David S. Touretzky, and Todd K. Leen, editors, *Advances in Neural Information Processing Systems*, volume 7, pages 393–400. MIT Press, 1995.

[Chapman and Kaelbling, 1991] David Chapman and Leslie Pack Kaelbling. Input generalization in delayed reinforcement learning: An algorithm and performance comparisons. In *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence*, pages 726–731. Morgan Kaufmann, 1991.

[Cover and Hart, 1967] T. M. Cover and P. E. Hart. Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, 13(1):21–27, January 1967.

[Dean *et al.*, 1998] Thomas Dean, Robert Givan, and Kee-Eung Kim. Solving planning problems with large state and action spaces. In *Proceedings of the Fourth International Conference on Artificial Intelligence Planning Systems*, pages 102–110. AAAI Press, 1998.

[Doya, 2000] Kenji Doya. Reinforcement learning in continuous time and space. *Neural Computation*, 12(1):219–245, January 2000.

[Duda *et al.*, 2000] Richard O. Duda, Peter E. Hart, and David G. Stork. *Pattern Classification*. Wiley-Interscience, 2nd edition, 2000.

[Kalos and Whitlock, 1986] Malvin H. Kalos and Paula A. Whitlock. *Monte Carlo Methods*, volume 1. John Wiley and Sons, 1986.

[Mahadevan and Connell, 1992] Sridhar Mahadevan and Jonathan Connell. Automatic programming of behavior-based robots using reinforcement learning. *Artificial Intelligence*, 55(2–3):311–365, June 1992.

[McCallum, 1995] Andrew Kachites McCallum. *Reinforcement Learning with Selective Perception and Hidden State*. PhD thesis, Department of Computer Science, University of Rochester, 1995.

[Moore and Atkeson, 1993] Andrew W. Moore and Christopher G. Atkeson. Prioritized sweeping: Reinforcement learning with less data and less time. *Machine Learning*, 13(1):103–130, October 1993.

[Moore and Atkeson, 1995] Andrew W. Moore and Christopher G. Atkeson. The Parti-game algorithm for variable resolution reinforcement learning in multidimensional state-spaces. *Machine Learning*, 21(3):199–233, December 1995.

[Munos and Patinel, 1994] Rémi Munos and Jocelyn Patinel. Reinforcement learning with dynamic covering of state-action space: Partitioning Q-learning. In *Proceedings of the Third International Conference on Simulation of Adaptive Behavior*, pages 354–363. MIT Press, 1994.

[Nilsson, 2000] Nils J. Nilsson. Learning strategies for mid-level robot control: Some preliminary considerations and results. www.robotics.stanford.edu/users/nilsson/trweb, 2000. Computer Science Department, Stanford University.

[Puterman, 1994] Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley and Sons, 1994.

[Ryan, 2004] Malcolm R. K. Ryan. *Hierarchical Reinforcement Learning: A Hybrid Approach*. PhD thesis, University of New South Wales, 2004.

[Sammut *et al.*, 1992] Claude Sammut, Scott Hurst, Dana Kedzier, and Donald Michie. Learning to fly. In *Proceedings of the Ninth International Workshop on Machine Learning*, pages 385–393. Morgan Kaufmann, 1992.

[Shannon, 1948] Claude E. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27:379–423 and 623–656, July and October 1948.

[Smith, 2002] Andrew James Smith. Applications of the self-organising map to reinforcement learning. *Neural Networks*, 15(8–9):1107–1124, October–November 2002.

[Thrun, 1992] Sebastian B. Thrun. The role of exploration in learning control. In D. White and D. Sofge, editors, *Handbook for Intelligent Control: Neural, Fuzzy and Adaptive Approaches*, pages 527–559. Van Nostrand Reinhold, 1992.

[Uther and Veloso, 1998] William Uther and Manuela Veloso. Tree based discretization for continuous state space reinforcement learning. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence*, pages 769–775. AAAI Press, 1998.

[Uther and Veloso, 2003] William Uther and Manuela Veloso. TTree: Tree-based state generalization with temporally abstract actions. In *Adaptive Agents and Multi-Agent Systems: Adaptation and Multi-Agent Learning*, volume 2636 of *Lecture Notes in Computer Science*, pages 266–296. Springer, 2003.

[Weiss, 1991] Sholom M. Weiss. Small sample error rate estimation for k-NN classifiers. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(3):285–289, March 1991.

[Whitehead, 1991] Steven D. Whitehead. A complexity analysis of cooperative mechanisms in reinforcement learning. In *Proceedings of the Ninth National Conference on Artificial Intelligence*, pages 607–613. AAAI Press, 1991.

[Yianilos, 1993] Peter N. Yianilos. Data structures and algorithms for nearest neighbor search in general metric spaces. In *Proceedings of the Fourth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 311–321. Society for Industrial and Applied Mathematics, 1993.