Adaptive Partitioning of State Spaces using Decision Graphs for Real-Time Modeling and Planning

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 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 spaces. In order to learn and plan in such domains effectively, experience must be generalized over time and space. This system achieves generalization by incrementally adapting a discretization of the state space using a decision graph 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.

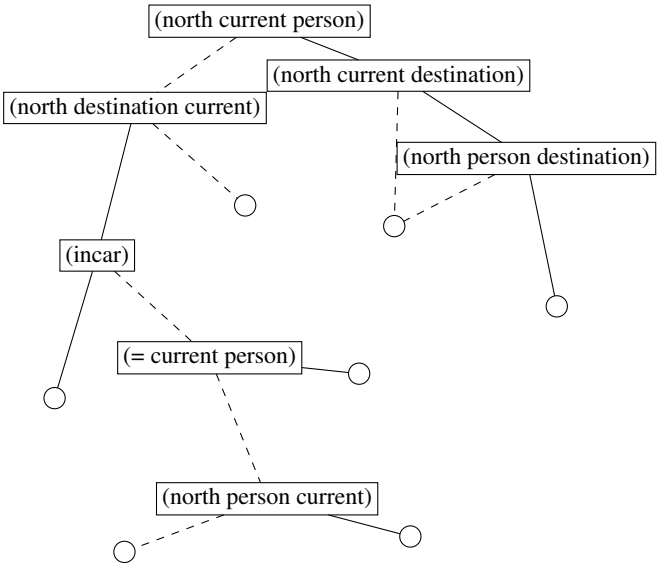

Figure 1: A decision graph constructed by AMPS during early exploration of the Taxi World domain. Decision nodes are represented by rectangles and leaf nodes, corresponding to regions of the state space, are represented by circles. When the test at a decision node evaluates to true, the solid edge is followed. Otherwise, the dashed edge is followed.

1 Introduction

This paper presents a new system that integrates incremental model learning and reactive plan construction for real-time control of an agent situated in a stochastic world. This system is a specialized instance of our more general "Adaptive Modeling and Planning System" (AMPS). Under consideration in this paper are problems with large or infinite state spaces and durative actions that 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 its experience in the world to competently maximize its expected discounted reward.

In order to make problems with large or infinite state spaces tractable, a *decision graph* partitions the state space into regions and treats all states in the same region collectively. A decision graph is a generalization of a decision tree where nodes may have multiple parents. Associated with every internal node is a condition, which is a binary test that maps states to truth values. The leaf nodes represent regions of the state space. A state is mapped to a region through a series of tests starting at the root and continuing through the graph according to the results of these tests until arriving at a leaf. Figure 1 is an example of a decision graph that was incrementally constructed by AMPS.

AMPS begins with a decision graph consisting of a single node representing the entire state space. As the agent accumulates experience, AMPS incrementally refines its partition of the state space by splitting and merging regions, thereby growing the decision graph, aiming to arrive at a model that is consistent with its experience and conducive to building a successful reactive plan.

The primary contribution of this paper is a method for incrementally adapting the partition of the state space by splitting and merging regions. One of the methods for splitting regions in AMPS is inspired in part by the work of Uther and Veloso [1998] where regions of the state space are split when there are observed differences in expected discounted reward

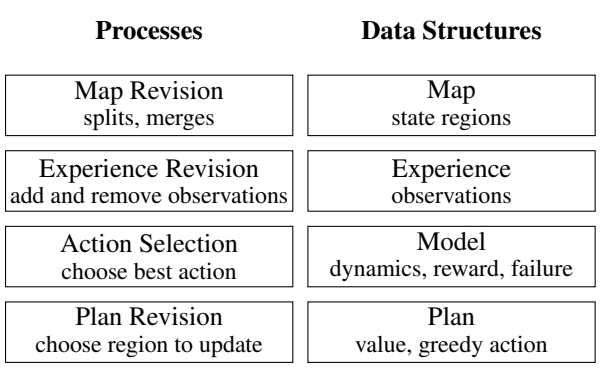

Figure 2: A high-level view of AMPS showing the four processes and the four data structures.

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 the discretization of the state space. 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 2. The Map Revision process splits and merges regions of the state space. 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 of potentially infinitely many states and the agent may execute exactly one of a small finite number of possible actions. 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, 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,

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

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] over a state space that has been partitioned into a finite set of regions N. Transitions, reward, and failures are determined by fixed probability distributions:

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

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

$$V^*(n) = \max_a Q(n, a, n') \quad (1)$$

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

where

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

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

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

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 action a and then follows an optimal policy is given by $Q(n, a, n')/P(n'|n, a)$, a value that will be used in the map revision process described in Section 4.1.

The agent has no prior knowledge of N, P, P_t, or P_r. AMPS incrementally adapts its partitioning of the state space 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.

[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.

3 Data Structures

The data structures in AMPS store information about state regions, observations, world models, and plans. This section will describe each data structure in turn.

3.1 Map

The *Map* data structure maps states to regions and allows these regions to be efficiently split and merged to adapt with the experience of the agent. The instance of AMPS considered in this paper performs these operations using a decision graph.[2]

Merging regions is straightforward in a decision graph, but splitting regions requires a mechanism called a *separator*. The separator mechanism creates binary tests based on sample states belonging to different categories, much like a supervised learning classifier [Duda and Hart, 1973]. The tests that are generated by the separator should be simple. A test 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.

Clearly, the separator mechanism must be engineered according to how the state space is represented. It is important to note that the separator is the only component in AMPS that interacts with the state space representation directly. This is intentional. AMPS is designed to be representation independent, unlike most other partitioning algorithms,[3] enabling it to be applied across both relational and continuous-valued domains with only the separator requiring changing.

We implemented a program that generates separators for any logical domain given the names of the constants, functions, and relations and their associated types. We integrated the JTP theorem prover [Fikes *et al.*, 2003] to prune unnecessary logical sentences from consideration based on a set of axioms about the domain. The separator selects the sentence that provides the greatest information gain [Shannon, 1948].

3.2 Experience

The *Experience* data structure keeps a record of the states and transitions experienced by the agent and associates them with the regions, i.e. the leaf nodes of the decision graph, managed by the Map. When 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 $P_f(n, a)$ be the probability of encountering failure when executing action a from region n. This may be estimated directly from experience as can $P(n'|n, a)$. 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):

- $\int_0^\infty e^{-\beta t} dP_t(t|n, a, n')$ 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 action a.
- $\int_{-\infty}^\infty r dP_r(r|n, a, n')$ is approximated by averaging the reward received while transitioning from n to n' by action a.

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 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 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 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, growing the decision graph. The objective is to find a partition of the state 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

[2] Alternatively, nearest-neighbor [Cover and Hart, 1967] may be used provided that a distance metric is defined over the state space.

[3] 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.

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 as follows:

Value Revision

This type of revision attempts to separate state observations that have resulted in transitions with differing estimated value. This separation is done with the separator mechanism for the decision graph. The priority of performing this type of revision is proportional to a measure of variation of estimated value. The experimental implementation separates states involved in greedy transitions based on whether $Q(n, \pi(n), n')/P(n'|n, \pi(n))$ is above or below the mean. The priority is proportional to the variance of $Q(n, \pi(n), n')/P(n'|n, \pi(n))$.

Failure Revision

This type of revision uses the separator mechanism to separate 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 when taking a greedy action, which is given by $P_f(n, \pi(n))$.

Simplification Revision

This type of revision merges state 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 there exists an approximately deterministic greedy transition[4] from n to n' and both nodes have the same greedy action, in which case n and n' are merged. The second situation occurs when there exists approximately deterministic greedy transitions from n and n' leading to the same node and $\pi(n) = \pi(n')$, in which case n and n' may 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] An approximately deterministic greedy transition from n to n' is one where $P(n'|n, \pi(n)) \approx 1$.

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 Taxi World domain described later, a failure might be trying to drop off a person who is not in the taxi. The Action Selection mechanism uses information about past failures to better direct the agent toward a goal.

In our current implementation of AMPS, the agent will take the greedy action $\pi(n)$ with probability $1 - P_f(n, \pi(n)) - \epsilon$, where ϵ is a small positive value allowing random exploration even when no failure has been observed. If $\pi(n)$ is not taken, then the last action a is followed with probability $1 - P_f(n, a) - \epsilon$. Otherwise, a random action is selected.

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 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 is continually changing. Instead, it is better to use a method related to prioritized sweeping [Moore and Atkeson, 1993], which was designed for solving Markov decision processes (MDPs) but can be adapted for solving SMDPs. The idea of the algorithm is to prioritize updates of regions based on observed changes in the value function from earlier updates. Pseudo-code is given in Figure 3.

5 Experiments and Results

This section demonstrates AMPS in the Taxi World domain. In this problem the agent must control a car in a grid world to pick up people and transport them to their desired destination. The agent may move up, down, left, and right and pick up and put down passengers. When the agent moves around in the world, it will move in a direction orthogonal to the direction it intended with 5% probability.

The state space is represented by a seven-dimensional vector containing the x and y coordinates of the car, person, and desired destination and a Boolean value indicating whether

PRIORITIZED-SWEEPING

while priority queue is not empty
  do remove region n from the front of the queue
    $v \leftarrow V(n)$
    UPDATE(n)
    $\Delta \leftarrow |V(n) - v|$
    for each pair $\langle n', a' \rangle$
      s.t. $n' \neq n$ and $P(n|n', a') > 0$
      do $p \leftarrow P(n|n', a')\gamma(n', a', n)\Delta$
        if $p > \epsilon$ and n' is either not in the queue or p is greater than its current priority
        then promote the priority of n' to p

Figure 3: The adapted version of prioritized sweeping used by the Plan Revision process.

the person is in or out of the car. The separator partitions the state space using the IN-CAR relation and various directional relations (e.g. NORTH and DIRECTLY-WEST) that take two objects as arguments, where the objects may be CURRENT, PERSON, and DESTINATION, representing the current position of the taxi, the person, and the destination respectively. Failures occur when the agent attempts to pick up or put down a person when not occupying the same square or when it attempts to move past the border of the world. Goal states are states where IN-CAR is true and the person is at its destination. The agent receives zero reward except at goal states where it receives unit reward. The continuous compound discount rate, β, was set to 0.01.

In our experiments we used a 40×40 grid, which allows for over 4 billion states. Since the agent 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 Taxi 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 decision tree is induced using the separator 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, we tested seven different agents: AMPS with all forms of map revision (A), AMPS without value revision (V), AMPS without failure revision (F), AMPS without simplification revision (S), behavioral clone (C), the noisy teacher that was used to train the other agents (T), and random (R). We ran 100 runs of 20 episodes (not including the two training episodes) each using different random seeds. For the AMPS agents, no more than one merge or split of the state space was allowed per time step. Each episode lasts 300 steps or when the agent successfully picks up and drops off its passenger at the desired destination. The results of 100 repeated experiments are summarized in Table 1.

	Successes		Disc. Reward		Regions	
	mean	s.d.	mean	s.d.	mean	s.d.
A	14.49	4.27	6.05	2.20	66.73	11.78
V	0.97	2.01	0.31	0.99	18.48	8.28
F	0.02	0.14	0.02	0.14	1.00	0.00
S	14.33	3.81	6.11	2.02	84.66	13.12
C	3.49	3.24	1.48	1.82	12.39	2.11
T	20.00	0.00	10.29	0.69	N/A	N/A
R	0.02	0.14	0.02	0.14	N/A	N/A

Table 1: Experimental results of various agents in the Taxi World domain. Shown are the means and standard deviations of the number of successes and discounted reward over 100 runs of 20 episodes each. Shown for each AMPS-based agent is the mean and standard deviation of the number of regions at the end of the run.

6 Discussion

As can be seen in the table of results, AMPS quickly learned how to navigate from a state selected randomly from the space of over 4 billion possible states to one of the 160 goal states. The success of AMPS is due to its ability to take advantage of the structure inherent in the task.

AMPS began with a simple model of the world where all states are equivalent. After two teaching episodes, AMPS proceeded on its own with the reactive plan that it developed. As AMPS encountered evidence indicating that its model and plan was incorrect, the Map Revision process revised the state space partition appropriately. The complexity of the agent's model increased rapidly as the agent initially explored the world but then this complexity leveled off. The number of regions commonly used in effective solutions to the Taxi World problem was typically in the range of 60–80. Figure 6 shows the connectivity between regions after five episodes.

The results demonstrate that AMPS without value revision performs extremely poorly and AMPS without failure revision performs like a random controller. As can be expected, AMPS without simplification revision is capable of quickly learning effective behavior. However, without simplification revision the model consists of about 27% more regions, and consequently requires more memory and processing power.

Behavioral clones did quite poorly, solving on average only 3.49 of the 20 random problems. This poor performance is due to the fact that behavioral clones do not adapt their policies in response to their experience. Because the two training instances provided such a limited exposure to the state space, the clones could not consistently generalize the training instances to new situations.

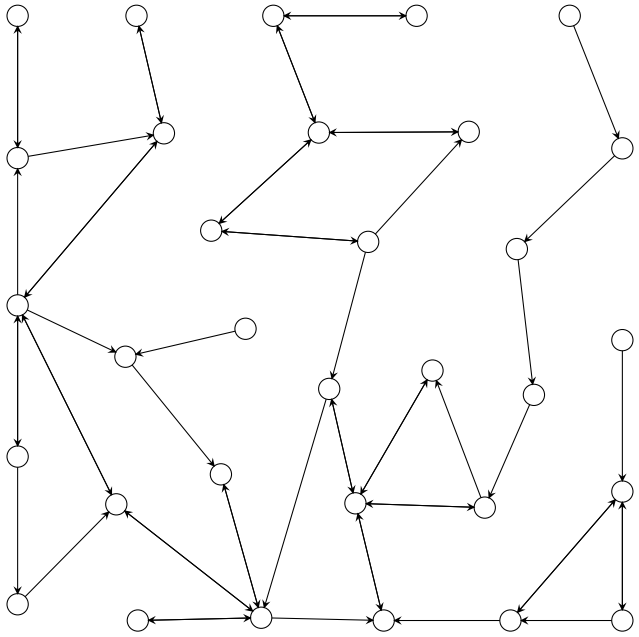

Figure 4: A graph illustrating the connectivity between regions after running AMPS for five episodes.

Although AMPS is unique in the way it decides how and when to split and merge regions of the state space, there have been other algorithms proposed, both model-free and model-based, that use decision trees to partition the state space [Chapman and Kaelbling, 1991; McCallum, 1995; Uther and Veloso, 2003; 1998]. 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]. Other work has been done on problems with durative actions [Benson and Nilsson, 1995]. The way AMPS uses SMDPs to model durative actions relates 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 adaptive partitioning of the state space and incremental planning over the estimated model to produce goal-directed behavior. The system was tested on the Taxi World domain and was shown to quickly learn successful behavior in the presence of noise in the environment and the teacher.

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 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, George Konidaris, Marc Toussaint, and the anonymous reviewers for their helpful suggestions on earlier drafts 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.

[Duda and Hart, 1973] Richard O. Duda and Peter E. Hart. *Pattern Classification and Scene Analysis*. John Wiley and Sons, 1973.

[Fikes *et al.*, 2003] Richard Fikes, Jessica Jenkins, and Gleb Frank. JTP: A system architecture and component library for hybrid reasoning. Technical Report KSL-03-01, Knowledge Systems Laboratory, Department of Computer Science, Stanford University, 2003.

[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.

[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.

[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.

[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.