**Proceedings of the 2007 IEEE Symposium on Approximate Dynamic Programming and Reinforcement Learning (ADPRL 2007)**

# Knowledge Transfer Using Local Features

Martin Stolle
Christopher G. Atkeson
Robotics Institute,
Carnegie Mellon University
5000 Forbes Ave, Pittsburgh, PA 15213
http://www.cs.cmu.edu/∼mstoll
{mstolle, cga}@andrew.cmu.edu

***Abstract*— We present a method for reducing the effort required to compute policies for tasks based on solutions to previously solved tasks. The key idea is to use a learned intermediate policy based on local features to create an initial policy for the new task. In order to further improve this initial policy, we developed a form of generalized policy iteration. We achieve a substantial reduction in computation needed to find policies when previous experience is available.**

## I. Introduction

Finding policies, a function mapping states to actions, is computationally expensive, especially in continuous domains. The alternative of computing a single path, although computationally much faster, does not suffice in real world domains where sensing is noisy and perturbations from the intended paths are expected. When solving a new task in the same domain, planning algorithms typically start from scratch. This paper presents an algorithm which decreases the computation needed to find policies for new tasks based on solutions to previous tasks in the same domain. This is accomplished by initializing a policy for the new task based on policies for previous tasks.

As policies are often expressed using state representations that do not generalize across tasks, policies cannot be copied directly. Instead, we propose the use of local features as an intermediate representation which generalizes across tasks. By way of this local state representation, policies can be translated across tasks and used to seed planning algorithms with a good initial policy.

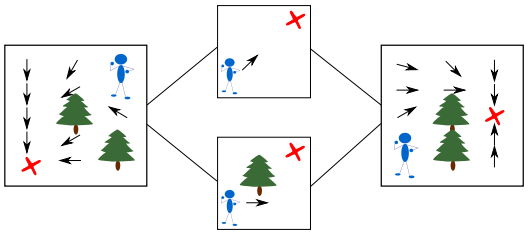

Fig. 1. Example navigation domain, left: original terrain, middle: feature-based policy, right: new terrain

For example, in a navigation domain, a policy is usually defined in terms of $(x, y)$ coordinates. If the terrain or goal changes, the same $(x, y)$ position will often require a different action. For instance, on the left terrain in figure 1, the policy of the upper left corner is to go down, whereas in the right terrain the policy of the same position is to go right. However, one can represent the policy in terms of local features that take into account the position of the agent with respect to the goal and obstacles. A new policy is initialized by looking up what the local features are for each state and setting the action of that state to the action that is associated with the local features. By reverting back to the global $(x, y)$-type state representation, the policy can be refined for the new task without being limited by the local state representation.

## II. Related Work

The transfer of knowledge across tasks is an important and recurring aspect of artificial intelligence. Previous work can be classified according to the type of description of the agent's environment as well as the variety of environments the knowledge can be transferred across. For symbolic planners and problem solvers, the high level relational description of the environment allows for transfer of plans or macro operators across very different tasks, as long as it is still within the same domain. Work on this goes back to STRIPS [1], SOAR [2], Maclearn [3] and analogical reasoning with PRODIGY [4]. More recent relevant work in discrete planning can be found in [5], [6].

In controls, work has been done on modeling actions using local state representations [7], [8]. Other work has been done to optimize low-level controllers, such as walking gaits, which can then be used in different tasks [9], [10], [11], [12]. Some work has been done in automatically creating macro-actions in reinforcement learning [13], [14], [15], [16], however those macro actions could only transfer knowledge between tasks where only the goal was moved. If the environment was changed, the learned macro actions would no longer apply as they are expressed in global coordinates, a problem we are explicitly addressing using the local state representation. Another method for reusing macro actions in different states using homomorphisms can be found in [17].

[18] explores learning from observation using local features, and learning from practice using global features on the marble maze task. Our work focuses on the use of local features in model-based planning.

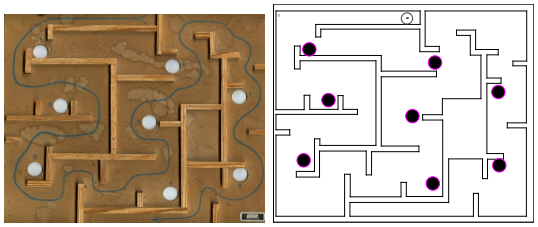

Fig. 2. Original (left) and simulation (right)

## III. Case Study: Marble Maze

The domain used for gauging the effectiveness of our knowledge transfer approach is the Marble Maze domain (figure 2). It consists of a plane with walls and holes. A ball (marble) is placed on a specified starting position and has to be guided to a specified goal zone by tilting the plane. Falling into holes is to be avoided and the walls both restrict the marble and can help it in avoiding the holes. The simulation used in this project uses a four-dimensional state representation $(x, y, dx, dy)$ where $x$ and $y$ specify the 2d position on the plane and $dx$, $dy$ specify the 2d velocity. Actions are also two dimensional $(fx, fy)$ and are force vectors to be applied to the marble. This is not identical but similar to tilting the board. Other simplifications are made in the simulator: the marble is only a point and has no physical extent, the physics are simulated as a sliding block (simplifies friction and inertia) and the simulator adds no artificial noise. Hence, all actions are deterministic. A more realistic but also higher dimensional marble maze simulator was used by Bentivegna [19].

The reward structure used for reinforcement learning in this domain is very simple. Reaching the goal results in a large positive reward. Falling into a hole terminates the trial and results in a large negative reward. Additionally, each action incurs a small negative reward. The agent tries to maximize the reward received, resulting in policies that roughly minimize the time to reach the goal while avoiding holes.

Solving the maze from scratch was done using value iteration. In value iteration, dynamic programming sweeps across all states and performs the following update to the value function estimate $V$ for each state $s$:

$$V^{t+1}(s) = \max_a \{r(s,a) + V^t(s(a))\} \quad (1)$$

where $a$ ranges over all possible actions, $r(s,a)$ is the reward received for executing $a$ in state $s$ and $s(a)$ is the next state reached after $a$ is executed in state $s$.

The simulator served as the model for value iteration. The state space was uniformly discretized and multi-linear interpolation was used for the value function [20].

The positional resolution of the state space was 3mm and the velocity resolution was 12.5mm/s. The mazes were of size 289mm by 184mm and speeds between -50mm/s to +50mm/s in both dimensions were allowed, resulting in a state space of about 380,000 states. Variable resolution methods such as [21] could be used to limit high-resolution representation to parts of the space where it is strictly necessary. The maximum force on the marble in each dimension was limited to 0.0014751N and discretized into -.001475N, 0 and +.001475N in each dimension, resulting in 9 possible actions for each state. With a simulated mass of the marble of .0084kg, maximal acceleration was about 176mm/s$^2$ in each dimension. Time was discretized to 1/60th of a second.

### A. Local State Representation

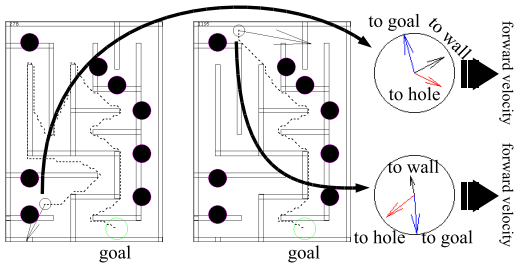


Fig. 3. Local state representation

The local state representation, chosen from the many possible local representations, depicts the world as seen from the point of view of the marble, looking in the direction it is rolling. Vectors pointing towards the closest hole, the closest wall as well as along a path towards the goal (dashed line in figure 3) are computed. These vectors are normalized to be at most length 1 by applying the logistic function to them. The path towards the goal is computed using A* on a discretized grid of the configuration space (**position only**). A* is very fast but does not take into account velocities and does not tell us what actions to use. Two examples of this local state representation can be seen in figure 3. In the circle representing the relative state of the marble, the forward velocity is towards the right. In the first example, the marble is rolling towards a hole, so the hole vector is pointing ahead, slightly to the right of the marble, while the wall is further to the left. The direction to the goal is to the left and slightly aft. This results in a state vector of (.037; -.25, -.97; .72, -.38; .66, .34), where .037 is the scalar speed of the marble (not shown in figure), followed by the relative direction to the goal, relative direction to the closest wall and relative direction to the closest hole. The second example has the closest hole behind the marble, the closest wall to the left and the direction to the goal to the right of the direction of the marble, resulting in a state vector of (.064; .062, .998; -.087, -.47; -.70, .58). As all vectors are relative to the forward velocity, the velocity becomes a scalar speed only. Actions can likewise be *relativized* by projecting them onto the same forward velocity vector.

### B. Knowledge Transfer

The next step is to transfer knowledge from one maze to the next. For the intermediate policy, expressed using the local state representation, we used a nearest neighbor classifier with a kd-tree as the underlying data structure for efficient querying. After a policy has been found for a maze, we iterate over the states and add the local state representation with

their local actions to the classifier. It is possible to use this intermediate policy directly on a new maze. For any state in the new maze, the local representation is computed and the intermediate policy is queried for an action. However, in practice this does not allow the marble to complete the maze because it gets stuck. Furthermore, performance would be expected to be suboptimal as the local representation alone does not necessarily determine the optimal action and previous policies might not have encountered certain states that now appear on the new task.

Instead, an initial policy based on global coordinates is created using the classifier by iterating over the states of the new maze and querying the classifier for the appropriate action based on the local features of that state. This policy is then refined.

### C. Improving the Initial Policy

Originally, we wanted to use policy evaluation to create a value function from the initial policy which could then be further optimized using value iteration. In policy evaluation, the following update is performed for every state to update the value function estimate:

$$V_\pi^{t+1}(s) = r(s,a) + V_\pi^t(s(a)) \tag{2}$$

where $a = \pi(s)$, the action chosen in state $s$ by the policy $\pi$.

Compared to value iteration (equation 1), policy evaluation requires fewer computations per state because only one action is evaluated as opposed to every possible action. We hoped that the initial value function could be computed using little computation and that the subsequent value iterations would terminate after a few iterations.

However, some regions of the state space had a poor initial policy so that values were not properly propagated through these regions. In goal directed tasks such as the marble maze, the propagation of a high value frontier starting from the goal is essential to finding a good policy as the agent will use high valued states in its policy. If high values cannot propagate back through these bad regions, the values behind these bad regions will be incorrect and value iteration will not be sped up. Similarly, if a policy improvement step was used to correct the policy in these states, the policy of states behind these bad regions would be updated based on an incorrect value function.

We overcame these two problems by creating a form of generalized policy iteration [22]. The objective in creating this dynamic programming algorithm was to efficiently use the initial policy to create a value function while selectively improving the policy where the value function estimates are valid. Our algorithm performs sweeps over the state space to update the value of states based on a fixed policy. In a small number of randomly selected states, the policy is updated by checking all actions (a full value iteration update using equation 1). As this is done in only a small number of states (on the order of a few percent), the additional computation required is small.

In order to avoid changing the policy for states using invalid values, the randomly selected states are filtered. Only those states are updated where the updated action results in a transition to a state which has been updated with a value coming from the goal. This way we ensure that the change in policy is warranted and a result of information leading to the goal. This can easily be implemented by a flag for each state that is propagated back with the values. Note that as a result, we do not compute the value of states that cannot reach the goal.

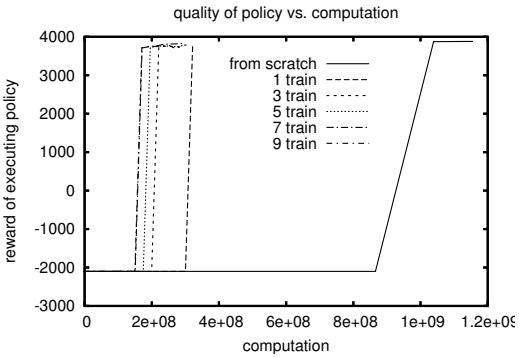


Fig. 4. Results for one test maze

## IV. Experimental Results

In order to gauge the efficiency of the algorithm, a series of experiments was run. First, pools of 30 training mazes and 10 test mazes where created using a random maze generator (mazes available from [23]). We trained the intermediate classifier with an increasing number of training mazes to gauge the improvement achieved as the initial policy becomes more informed. The base case for the computation required to solve the test mazes was the computation required when using value iteration.

**Computational effort was measured by counting the number of times that a value backup was computed** before a policy was found that successfully solved the maze. The procedure for measuring the computational effort was to first perform 200 dynamic programming sweeps and then performing a trial in the maze based on the resulting policy. Following that, we alternated between computing 50 more sweeps and trying out the policy until a total of 1000 dynamic programming sweeps were performed.

When performing a trial, the policy was to pick the best action with respect to the expected reward based on the current estimate of the value function. Figure 4 shows the quality of the policy obtained in relation to the number of value backups. The right most curve represents value iteration from scratch and the other curves represent starting with an initial policy based on an increasing number of training mazes. The first data points show a reward of -2100 because policy execution was limited to 2100 time steps, at which point the trial was aborted if the goal was not yet reached.

Clearly, an initial policy based on the intermediate policy reduces the computation required to find a good policy. However, final convergence to the optimal policy is slow because only a

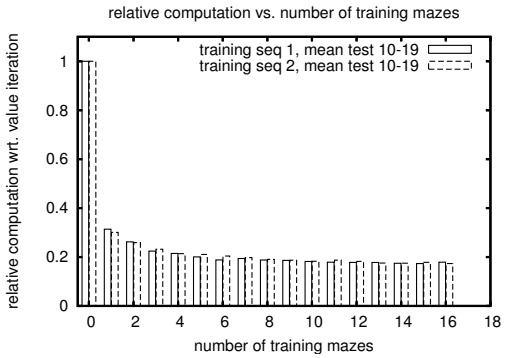


Fig. 5. Relative computation, averaged over 10 test mazes, using two different sequences of training mazes

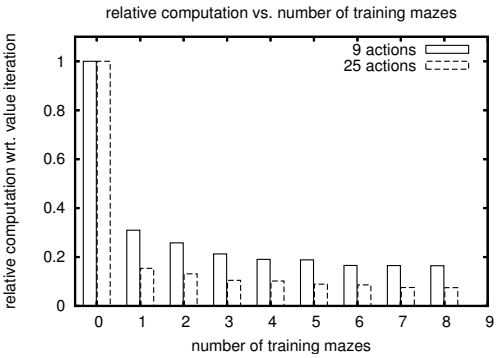


Fig. 7. Relative computation required for one test maze and different number of actions

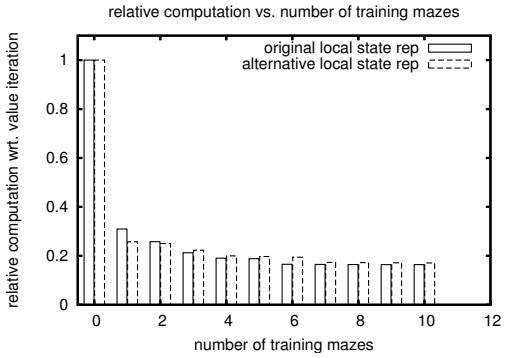


Fig. 6. Relative computation required for one test maze for two different local state representation

small number of states are considered for policy updates. This results in a slightly lower solution quality in our experiments.

In order to ensure that the savings are not specific to this test maze, we computed the relative computation required to find a policy that successfully performs the maze for ten different test mazes and plotted the mean in figure 5 (solid). Additionally, in order to exclude the peculiarities of the training mazes as a factor in the results, we reran the experiments with other training mazes. The results can be seen in figure 5 (dashed). Clearly, the individual training mazes and their ordering do not influence the results very much.

## V. Discussion

**State Representation:** The local features that we are proposing as a solution to this problem are intuitively defined as features of the state space that are in the immediate vicinity of the agent. However, often the agent is removed from the actual environment and might even be controlling multiple entities or there may be long-range interactions in the problem. A more accurate characterization of the features we are seeking are that they influence the results of the actions in a consistent manner across multiple tasks and allow, to a varying degree, predictions about the relative value of actions. These new features have to include enough information to predict the outcome of the same action across different environments and should ideally not include unnecessary information that does not affect the outcome of actions. They are similar in spirit to predictive state representation [24]. These conditions will preclude features such as position on a map, as this alone will not predict the outcome of actions – obstacles and goals are much more important.

In order to gauge the effect of different local state representations, we created an alternative state representation. In this alternative representation, the world is represented as seen from the marble, but aligned with the direction to the goal instead of the direction of the movement. Furthermore, the view is split up into 4 quadrants: covering the 90 degrees towards the path to the goal, 90 degrees to the left, to the right and to the rear. For each quadrant, the distance to the closest hole and closest wall are computed. Holes that are behind walls are not considered. The velocity of the marble is projected onto the path towards the goal. The resulting state representation is less precise with respect to direction to the walls or holes than the original local representation but takes into account up to four holes and walls, one for each quadrant. As can be seen in figure 6, the results are similar for both state representations. The new state representation performs slightly better with fewer training mazes but loses its advantage with more training mazes.

**Computational Saving:** There are several factors that influence the computational saving one achieves by using an informed initial policy. The computational reduction results from the fact that our generalized policy evaluation only computes the value of a single action at each state, whereas value iteration tries out all actions for every state. As a result, if the action space is discretized at high resolution, resulting in many possible actions at each state, the computational savings will be high. If on the other hand there are only two possible actions at each state, the computational saving will be much less. The computation can be reduced at most by a factor equal to the number of actions. However, since in a small number of states in the generalized policy evaluation we also try all possible actions, the actual savings at every sweep will be less. In order to show the effects of changing the number of actions,

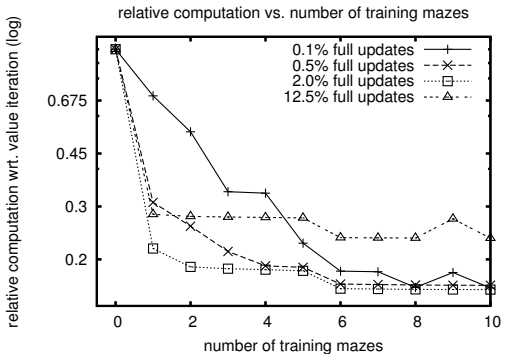


Fig. 8. Relative computation required for one test maze and different percentages of full updates

we reran the experiments for one maze with actions discretized into 25 different actions instead of 9. As seen in figure 7, the relative computational saving becomes significantly larger, as was expected.

We also ran experiments to determine the effect of performing policy updates on a varying number of states. If many states are updated at every sweep, fewer sweeps might be necessary, however each sweep will be more expensive. Conversely, updating fewer states can result in more sweeps, as it takes longer to propagate values across bad regions which are now less likely to be updated. The results are presented in figure 8. When reducing the percentage of states updated to 0.1%, the computational saving is reduced as it now takes many more sweeps to find a policy that solves the maze, unless the initial policy is very good (based on several mazes). The savings become more pronounced as more states are updated fully and are the greatest when 2.0% of the states are updated, performing better than our test condition of 0.5%. However, increasing the number of states updated further results in reduced savings as now the computational effort at every sweep becomes higher. Comparing the extreme cases shows that when updating few states, the initial policy has to be very good (many training mazes added), as correcting mistakes in the initial policy takes longer. On the other hand, if many states are updated, the quality of the initial policy is less important – many states are updated using the full update anyways.

**Intermediate Policy Representation:** Another issue that arose during testing of the knowledge transfer was the representation of the intermediate policy representation. We chose a nearest neighbor approach, as this allows broad generalization early on, without limiting the resolution of the intermediate policy once many training mazes were added to the intermediate policy. However, after adding many mazes, the data structure grew very large (around 350,000 data points per maze, around 5 million for 15 mazes). While the kd-trees performed well, the memory requirements became a problem. Looking at the performance graph, adding more than 5 mazes does not seem to make sense with the current state representation. However, if a richer state representation was chosen, it might be desirable to add more mazes and then pruning of the kd-tree becomes essential.

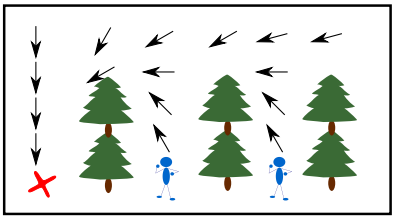

Fig. 9. Aliasing problem: same local features, same policy but different values

The nearest neighbor algorithm itself is modifiable through the use of different distance functions. By running the distances to the closest hole and wall through a logistic function, we have changed the relative weight of different distances already. However, instead one could imagine rescaling distance linearly to range from 0 and 1, where 1 is the longest distance to either hole or wall observed.

**Dynamic Programming on Local State Space:** As we are using the local state space to express an intermediate policy, it might be interesting to perform dynamic programming in this state space directly. Due to the possible aliasing of different states to the same local state, the problem becomes a partially observable Markov decision process (POMDP). This is aggravated if one keeps the value function across multiple tasks, as now even more states are potentially aliased to the same local state. A policy is less sensitive to this aliasing, as the actions might still be similar while the values could be vastly different. An example can be seen in figure 9. Both positions with the agent have the same features and the same policy, but the value would be different under most common reward functions which favor short paths to the goal (either with discounting or small constant negative rewards at each time step).

## VI. Conclusion and Future Work

We presented a method for transferring knowledge across multiple tasks in the same domain. Using knowledge of previous solutions, the agent learns to solve new tasks with less computation than would be required without prior knowledge. Key to this knowledge transfer was the creation of a local state representation that allows for the representation of knowledge that is independent of the individual task.

We would like to explore other uses for this intermediate state representation and would like to verify the claim that doing dynamic programming using this intermediate state representation directly has limited use. Also, we plan to apply this method to a stochastic simulator of the maze and a physical, actuated maze. Furthermore, we would like to use this local state representation for the creation of macro actions that can be carried across multiple tasks.

## Acknowledgments

This material is based upon work supported in part by the National Science Foundation under NSF Grant ECS-0325383

and the DARPA Learning Locomotion Program.

We would like to thank Darrin Bentivegna for comments and Mike Stilman for thoughtful discussions.

## REFERENCES

[1] Fikes, Hart, and Nilsson, "Learning and executing generalized robot plans." *Artificial Intelligence*, vol. 3, pp. 251–288, 1972.

[2] J. Laird, P. Rosenbloom, and A. Newell, "Chunking in soar: The anatomy of a general learning mechanism," *Machine Learning*, vol. 1, pp. 11–46, 1986.

[3] G. A. Iba, "A heuristic approach to the discovery of macro-operators," *Machine Learning*, vol. 3, pp. 285–317, 1989.

[4] M. M. Veloso, "Learning by analogical reasoning in general problem solving," Ph.D. dissertation, Carnegie Mellon University, 1992.

[5] E. Winner and M. Veloso, "Automatically acquiring planning templates from example plans," in *Proceedings of AIPS'02 Workshop on Exploring Real-World Planning*, April 2002.

[6] A. Fern, S. W. Yoon, and R. Givan, "Learning domain-specific control knowledge from random walks." in *ICAPS*, 2004, pp. 191–199.

[7] S. Mahadevan, "Enhancing transfer in reinforcement learning by building stochastic models of robot actions," in *Proceedings of the Ninth International Conference on Machine Learning*, 1992.

[8] S. Chernova and M. Veloso, "Learning and using models of kicking motions for legged robots," in *Proceedings of International Conference on Robotics and Automation (ICRA'04)*, May 2004.

[9] N. Kohl and P. Stone, "Machine learning for fast quadrupedal locomotion," in *The Nineteenth National Conference on Artificial Intelligence*, July 2004, pp. 611–616.

[10] S. Chernova and M. Veloso, "An evolutionary approach to gait learning for four-legged robots," in *Proceedings of IROS'04*, September 2004.

[11] T. Röfer, "Evolutionary gait-optimization using a fitness function based on proprioception," in *Eighth International Workshop on Robocup 2004*, 2005.

[12] J. D. Weingarten, G. A. D. Lopes, M. Buehler, R. E. Groff, and D. E. Koditschek, "Automated gait adaptation for legged robots," in *International Conference in Robotics and Automation*. New Orleans, USA: IEEE, 2004.

[13] A. McGovern, "Autonomous discovery of temporal abstractions from interaction with an environment," Ph.D. dissertation, University of Massachusetts Amherst, 2002.

[14] M. Stolle and D. Precup, "Learning options in reinforcement learning," *Lecture Notes in Computer Science*, vol. 2371, pp. 212–223, 2002.

[15] Ö. Şimşek and A. G. Barto, "Using relative novelty to identify useful temporal abstractions in reinforcement learning," in *Proceedings of the Twenty-First International Conference on Machine Learning*, 2004.

[16] S. Mannor, I. Menache, A. Hoze, and U. Klein, "Dynamic abstraction in reinforcement learning via clustering," in *Proceedings of the Twenty-First International Conference on Machine Learning*, 2004.

[17] B. Ravindran and A. G. Barto, "SMDP homomorphisms: An algebraic approach to abstraction in semi markov decision processes," in *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence*, 2003.

[18] D. C. Bentivegna, C. G. Atkeson, and G. Cheng, "Learning similar tasks from observation and practice," in *Proceedings of the 2006 IEEE/RSJ International Conference on Intelligent Robots and Systems.*, Beijing, China, October 2006, pp. 2677–2683.

[19] D. C. Bentivegna, "Learning from observation using primitives," Ph.D. dissertation, Georgia Institute of Technology, 2004.

[20] S. Davies, "Multidimensional interpolation and triangulation for reinforcement learning," in *Advances in Neural Information Processing Systems*, vol. 9. Morgan Kaufmann Publishers, Inc., 1996.

[21] R. Munos and A. Moore, "Variable resolution discretization in optimal control," in *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*, 1999.

[22] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA., 1998.

[23] M. Stolle, "Images of mazes used," Internet/WWW, 2005, http://www.cs.cmu.edu/∼mstoll/files/adprl2007-mazes.tar.gz.

[24] M. L. Littman, R. S. Sutton, and S. Singh, "Predictive representations of state," in *Advances in Neural Information Processing Systems*, vol. 14. Morgan Kaufmann Publishers, Inc., 2002.