UNIVERSITY OF OKLAHOMA

GRADUATE COLLEGE

LEARNING ACTION-STATE REPRESENTATION FORESTS FOR IMPLICITLY RELATIONAL WORLDS

A DISSERTATION

SUBMITTED TO THE GRADUATE FACULTY

in partial fulfillment of the requirements for the

Degree of

DOCTOR OF PHILOSOPHY

By

THOMAS J. PALMER
Norman, Oklahoma
2015

LEARNING ACTION-STATE REPRESENTATION FORESTS FOR IMPLICITLY RELATIONAL WORLDS

A DISSERTATION APPROVED FOR THE SCHOOL OF COMPUTER SCIENCE

BY

Dr. Andrew H. Fagg, Chair

Dr. Dean F. Hougen

Dr. S. Lakshmivarahan

Dr. Amy McGovern

Dr. David P. Miller

Dr. Rick P. Thomas

To Grandpa Huber, and also to Kristine and our clan.

Acknowledgments

First, I'd like to thank the people directly involved in this work. This includes Andy Fagg, of course, for his mentoring and great support over the years. I'd also like to thank my other committee members for their contributions: Dean Hougen, S. Lakshmivarahan, Amy McGovern, Dave Miller, and Rick Thomas. Also, thanks go to Matt Bodenhamer for the SMRF project that I build on here. Thanks also to Dan Fennelly, Dougal Sutherland, and Sam Bleckley, other students who have worked on SMRF. Other students in the Symbiotic Computing Laboratory have also helped out a lot in my time here, including Di Wang, Josh Southerland, and Kim Houck, including in reviewing this document.

A lot of other folks at OU have also been good support in various roles, including professors Deborah Trytten, Sridhar Radhakrishnan, and John Antonio. Fellow student coworkers in TA and instructor roles have also been great. Among various folks in the CS office, I'd like to thank Barbara Bledsoe, Chyrl Yerdon, Emily Pierce, Jaime Hoots, Lisa Moore, and Virginie Perez Woods. Y'all are great! I'd also like to thank the students I've taught at OU, and my friends in CSGSA have also been awesome. I'm still leaving out way too many people, but thanks anyway, y'all.

Also, during a substantial portion of my time at OU, I've been supported by a University of Oklahoma Foundation Fellowship. I appreciate that support immensely.

I should also acknowledge all my friends from Los Alamos and White Rock that make it seem like getting a PhD is a normal thing. And I'd like to thank folks at Eyefinity for sending me off with well wishes, and thanks also to Mike Goodrich at BYU and other faculty and fellow students who supported me there and whom I

still count among my friends.

Of course, I also need to thank my amazing wife, Kristine, who let me come back to school and who never stopped believing in me. Our kids have put up with a lot because of this, too, so I'd like to thank each of them: Jacob, David, Sam, Anna, Zeke, and Lydia. Thanks also to Kristine's and my parents and many other family members for their support. I've been enough trouble, but they haven't stopped putting up with me.

And, finally, 2 Nephi 4:16–35.

Table of Contents

List of Tables

List of Figures

List of Algorithms

Abstract

Real world tasks, in homes or other unstructured environments, require interacting with objects (including people) and understanding the variety of physical relationships between them. For example, choosing where to place a fork at a table requires knowing the correct position and orientation relative to the plate. Further, the quantity of objects and the roles they play might change from one occasion to the next; the variables are not fixed and predefined. For an intelligent agent to navigate this complex space, it needs to be able to identify and focus on just those variables that are relevant. Also, if a robot or other artificial agent can learn such physical relations from its own experience in a task, it can save manual engineering effort and automatically adapt to new situations.

Relational learning, while often focused on discrete domains, applies to situations with arbitrary numbers of objects by using existential and/or universal quantifiers from first-order logic. The field of reinforcement learning (RL) addresses learning task execution from scalar rewards based on agent state and action. Relational reinforcement learning (RRL) combines these two fields.

In this dissertation, I present an RRL technique emphasizing relations that are merely implicit in multidimensional, continuous object attributes, such as position, color, and size. This technique requires analyzing permutations of possible object comparisons while simultaneously working in the multidimensional spaces defined by their attributes. Existing similar RRL methods query only one dimension at a time, which limits effectiveness when multiple dimensions are correlated.

Specifically, I present a representation policy iteration (RPI) method using the spatiotemporal multidimensional relational framework (SMRF) for learning rela-

tional decision trees from object attributes. This *SMRF-RPI* algorithm interleaves the learning of relational representations and of policies for agent action. Further, SMRF-RPI includes support for continuous actions. As a component of the SMRF framework, I also present a novel multiple instance learning (MIL) algorithm, which is able to learn parametric, existential decision volumes within a feature space in a robust manner.

Finally, I demonstrate SMRF-RPI on a variety of developmentally motivated blocks world tasks, as well as effective transfer and sample efficient learning in a standard keepaway soccer benchmark task. Both domains involve complicated, simulated world dynamics in continuous space. These experiments demonstrate SMRF-RPI as a promising method for applying RRL techniques in multidimensional, continuous domains.

Chapter 1

Introduction

The real world is a complicated place. For example, a robot working in an everyday kitchen needs to avoid obstacles, retrieve items, open containers, and combine and mix ingredients, among other activities. In these tasks, each action involves only a subset of the objects in the environment. When pouring flour into a bowl, the relative positions of the bowl and measuring cup are important. The pouring edge should be approximately centered above, but not *too* far above, the bowl. Other nearby objects might also interfere, but the exact locations of items on the spice shelf are probably irrelevant, and different objects might be present at different times. Some attributes of key objects also are likely irrelevant, such as the color of the bowl. For the objects and attributes that do matter, rather complicated world dynamics are at play, including gravity and the dispersion of flour in the air. The robot's own developmental experience (see Fagg, 1993) can help it to form the concepts needed to successfully perform tasks such as these. For example, from its experience dropping of objects on each other, the robot can learn to interpret the world, *for this specific kind of action*, in terms of just the variables most likely to be relevant.

For an intelligent agent to learn to accomplish such tasks on its own, a number of questions are salient. What high-level relations or *predicates* (such as *over* or *between*) might exist in the raw physical data? Which are the key objects or participants in a particular situation or scene? How should the agent behave to accomplish its task, and how do its actions affect the world? Learning discrete

predicates in a continuous space (such as for position or color) requires finding a meaningful decision volume in the space. Discovering which objects take on which roles requires iterating through permutations of possible assignments. Although it is computationally untenable to optimize across all possible solutions, an agent learning on its own still needs to answer these questions in an effective and efficient fashion.

A variety of research areas come to bear in answering these questions. Reinforcement learning (RL) addresses learning to perform multistep tasks, where the agent's decision-making *policy* is defined by the outcome of each agent action, including a scalar reward signal, which might be positive or negative. Representation learning addresses the construction of features whereby to interpret the environment. Relational learning addresses environments in which the relationships between multiple objects is important for making predictions or taking actions, and where the number of objects might change in different situations. Relational concepts can be used as representations for task execution. For example, when considering passing the ball in a soccer game, I might care if there exists an opponent between me and a teammate. This concern is relevant no matter how many players are on the field. In first-order logic, this is called *existential quantification*. Relational learning might also address first-order *universal* quantification, which asks whether some property holds for *all* objects. For example, are all the appliances in the kitchen in working order? Again, this question is meaningful no matter how many appliances are present.

Relational reinforcement learning (RRL) combines relational learning and reinforcement learning (Blockeel and De Raedt, 1998; van Otterlo, 2005, 2012). RRL is ultimately concerned with how an agent should behave in relational worlds. However, traditional relational learning methods do not emphasize continuous, multidimensional environments. Often, they rely on hand-crafted predicates (e.g., Pasula

et al., 2007) or query continuous variables in only one dimension at a time (e.g., Blockeel and De Raedt, 1998). These limitations can prove detrimental when multiple dimensions are correlated. For example, the color yellow in the RGB color space requires covariance between the red and green color channels. Approximating a covariant volume with single dimensional queries may require numerous conjunctions. Methods that seek to learn relations that directly model multidimensional data sometimes do so without the ability to consider either existential or universal quantification during the relation learning process (e.g., Kulick et al., 2013). These methods also commonly consider only binary relations, such as those based on relative positions of pairs of objects.

In this dissertation, I present a method for RRL that addresses multidimensional continuous variables in a fashion that is unique among relational learning systems. In this approach, I both utilize and contribute a key component to the spatiotemporal multidimensional relational framework (SMRF) of Bodenhamer (2014), which learns relations to answer existential questions. The SMRF learning algorithm builds a decision tree from binary-labeled samples, where each sample is a set of objects. Each object has associated real-valued, multidimensional attributes, such as position or color. Such attributes increase the complexity of relational learning, which must already address permutations of objects in the scene when considering their possible roles. Concerning permutations, Bodenhamer (2014) demonstrates SMRF's effectiveness at pruning the search space when high numbers of irrelevant objects are present. Often, only a small fraction of object permutations needs to be considered. Bodenhamer also demonstrates that SMRF's ability to learn decision volumes in multidimensional spaces can be more effective than working in individual dimensions, especially when covariance exists in the data.

The SMRF learning algorithm is a binary classifier that learns high-level relations (such as *over* or *between*) beginning from continuous object attributes. To do

this, the SMRF framework uses *mapping functions* of the attributes of two or more objects to provide relative measures. As an example, a mapping function of relative position could provide a set of vectors corresponding to the relative positions of all pairs of objects in a scene. A relational existential question, such as whether any object exists above another, then becomes a question of which vectors are important among a set where most are irrelevant. The field of multiple instance learning (MIL) addresses this existential learning problem (Dietterich et al., 1997). In this dissertation, I contribute a novel multiple instance learning (MIL) algorithm called *covariant aggregation*, which includes direct support for covariance between dimensions. This algorithm is used for learning existential decision volumes within SMRF.

As a prelude and potential complement to relational reinforcement learning (RRL), I also present a learning method, using SMRF, to predict binary success or failure of actions. I approach this action outcome prediction in light of Gibsonian *affordances* (Gibson, 1977); if an action is likely to be successful, then one might consider it as an affordance provided to the agent. In predicting outcomes, trees learned by SMRF provide representational features for other learning algorithms, such as support vector machine (SVM) classification or approximate reinforcement learning (e.g., Lagoudakis and Parr, 2003). Among these features, in a matter reminiscent of relational model trees (see, e.g., Vens et al., 2007), I also present a novel method for extracting continuous, radial features from SMRF trees.

For reinforcement learning, I apply a representation policy iteration (RPI) mechanism (Mahadevan, 2005a), wherein representation learning is interleaved with policy learning. Across RPI iterations, a forest of SMRF trees is built for each kind of action (e.g., placing one object on another vs. rotating an object). In my work, I also present a method for continuous actions, rather than considering only actions parameterized on the discrete objects in a scene. My method for continuous

actions involves sampling possible actions as virtual objects. To summarize, key contributions of this dissertation include the following:

- a novel multiple instance learning (MIL) algorithm, emphasizing support for covariant decision volumes and instance labeling, and which provides the method for learning decision volumes in SMRF;
- an affordance-oriented method for relational learning to predict action success or failure in multidimensional, continuous domains;
- an algorithm for relational reinforcement learning (RRL) in multidimensional, continuous domains, using a representation policy iteration (RPI) framework;
- a mechanism for automatic extraction of continuous, radial features from SMRF trees;
- demonstration of ternary mapping functions in continuous domains that consider the relative positions of three objects; and
- demonstration of a mechanism for continuous actions in RRL.

In experimental evaluation, I minimally adapt parameters for the core learning system between different tasks. I test relational learning primarily in 2D simulated soccer and block stacking domains. In a keepaway soccer benchmark task (Stone et al., 2006), I show sample-efficient learning, compared to common, existing techniques. I also show effective transfer between tasks with different numbers of objects/participants.

Going forward in this dissertation, Chapter 2 discusses related prior work, including the reinforcement learning methods on which I base my work, and also discusses other relational reinforcement learning methods. Chapter 3 presents my covariant aggregation MIL algorithm. Chapter 4 discusses the SMRF relational tree learning

algorithm of Bodenhamer, which I use for relational learning in agent tasks, and to which I have contributed the MIL method for learning decision volumes. Chapter 5 presents my SMRF-forest-based method for binary action outcome prediction, including the use of ternary mapping functions. Chapter 6 presents my *SMRF-RPI* method for RRL. This chapter also presents my method for extraction of continuous, radial features and also my method for support of continuous actions. Chapter 7 presents experimental results for the SMRF-RPI method. Chapter 8 discusses conclusions and future work.

Chapter 2

Related Work

This work is primarily concerned with the learning of relational representations for reinforcement learning. Therefore, I first address the topic of reinforcement learning, with a focus on least squares policy iteration (Lagoudakis and Parr, 2003), a method for evaluating actions using linear approximation, given a feature basis for representing states and actions. My primary contribution in this dissertation is a technique for iteratively learning relational features using multidimensional object attributes to form such a basis for approximation. I therefore follow the discussion of approximate reinforcement learning with a review of representation learning and relational learning, especially as applied to relational reinforcement learning.

2.1 Reinforcement Learning

2.1.1 Markov Decision Processes and Q Learning

Multistep tasks occur regularly in daily life. Such tasks sometimes involve reaching a particular goal, such as in driving to work. Others involve maximizing some quantity, such as the amount of work performed during the day. An agent might also receive negative rewards (corresponding, perhaps, to costs or punishment) while working through a task. The reward framework adapts easily to goal-oriented tasks simply by, for example, providing positive reward for reaching a goal. Multistep tasks involving reward are often formulated as a Markov Decision Processes (MDPs), in which the current world state and agent action fully determine the probability

of next state and reward. That is, the MDP model assumes that the future is independent of the past, given the present. This assumption is called the *Markov property*. Formally, an MDP is a tuple of $(S, A, \mathcal{P}, R)$ where:

- S is the set of states,
- A is the set of actions available to the agent,
- $\mathcal{P}(s', s, a) = \Pr(s_{t+1} = s' | s_t = s, a_t = a)$ is the state transition function where $s' \in S, s \in S, a \in A$, and t is the current time step, and
- $R(s', s, a) = E[r_t | s_{t+1} = s', s_t = s, a_t = a]$ is the expected reward function, where r_t is the reward at the state transition (Bellman, 1957a; Sutton and Barto, 1998).

For MDPs, commonly, the goal is for the agent to learn policy:

$$\pi(s, a) = \Pr(a_t = a | s_t = s)$$

such that expected long-term reward is maximized. Deterministic policies might be represented simply as functions from S to A. The cumulative reward, or *return*, to be maximized could perhaps be the total reward for finite tasks or the discounted future reward for infinite horizon tasks. The latter option,

$$\mathcal{R}et_t = \sum_{i=0}^{\infty} \gamma^i r_{t+i},$$

is perhaps most common, where $\gamma \in [0, 1)$ is the reward discount factor.

More generally, actions can take arbitrary amounts of time. An MDP with temporally extended actions is called a Semi-Markov Decision Process (SMDP, Sutton et al., 1999), and such extended actions are called *options*. In discrete-time SMDPs, options consist of a sequence of primitive actions where each primitive action takes

one time step. The reward for option o taken during agent experience at time t is merely the discounted sum of rewards for the individual actions in the sequence:

$$r_{o,t} = \sum_{i=0}^{D-1} \gamma^i r_{t+i},$$

where D is the duration of the option, and subsequent primitive actions or options are discounted as for D time steps in the future. For the sake of RL formalisms discussed in this section, I ignore the issue of temporally extended actions, as the discount modifications are straightforward.

For learning policies, different techniques exist. Forward planning (such as in Fikes and Nilsson, 1971) is a common technique to solve such tasks in *ad hoc* situations. Another common technique is *reinforcement learning* (RL, see Sutton and Barto, 1998), which seeks to learn proper behavior for all situations based on past experience. RL is commonly based on *dynamic programming* (DP) theory (Bellman, 1957b). The learning process includes estimation of the values of states and state-action pairs. The value of a state for a given policy π gives the expected return for starting in state s and following π afterward:

$$V_\pi(s) = E[\mathcal{R}et_t | s_t = s, \pi].$$

Making use of a state value function requires knowledge of $\mathcal{P}$. That is, the agent needs to know which action leads to which next state in order to choose the best next state. On the other hand, by knowing the value of state-action pairs, it is possible to choose an action without explicit knowledge of future states. The Q function (Watkins and Dayan, 1992),

$$Q_\pi(s, a) = E[\mathcal{R}et_t | s_t = s, a_t = a, \pi],$$

is the expected return for choosing action a in state s and following policy π afterward. Given the MDP definitions for state transition and reward functions:

$$Q_\pi(s,a) = \sum_{s'} \mathcal{P}(s',s,a) \left(R(s',s,a) + \gamma \sum_{a'} \pi(s',a') Q_\pi(s',a') \right). \tag{2.1}$$

Techniques such as the popular Q-learning (Watkins and Dayan, 1992) are able to learn an optimal policy π^* for an MDP without explicitly modeling $\mathcal{P}$. Indicating the Q function for π^* as Q^* and noting that π^* always chooses the action that maximizes Q^*, Equation 2.1 becomes:

$$Q^*(s,a) = \sum_{s'} \mathcal{P}(s',s,a) \left(R(s',s,a) + \gamma \max_{a'} Q^*(s',a') \right).$$

Q^* can be learned iteratively for finite states and actions during interaction with the world by the update rule:

$$Q_t(s,a) = (1-\alpha_t) Q_{t-1}(s,a) + \alpha_t \left(r_t + \gamma \max_{a'} Q_{t-1}(s',a') \right),$$

where α_t is the learning rate at time t. Given certain conditions on α_t and exploration, Q_t converges in the limit to Q^*. The optimal policy then is:

$$\pi^*(s) = \operatorname*{argmax}_a Q^*(s,a). \tag{2.2}$$

2.1.2 Least Squares Policy Iteration

For discrete states and actions, given conditions on α_t and repeated sampling of all actions in all states, the Q learning algorithm converges in the limit to Q^* and π^* (Watkins and Dayan, 1992). However, for large or continuous state-action spaces, it might be infeasible to calculate an exact Q function. Standard function approximation techniques can be applied in such cases; for an overview, see Sutton and Barto

(1998, Chapter 8).

One common method is to learn a linear combination of nonlinear features that capture sufficient information about the world:

$$\widehat{Q}_\pi(s,a) = \phi(s,a)^\intercal w_\pi,$$

where $\phi(s,a)$ is a $k \times 1$ vector of arbitrary features, and w_π, also a k-vector, is of feature weights. The dimensions of ϕ are presumed to be linearly independent.

One high-profile method for linearly weighted Q-learning is Least-Squares Temporal-Difference Q-learning (LSTDQ) of Lagoudakis and Parr (2003). For reinforcement learning in this dissertation, I use LSTDQ as well as the Least Squares Policy Iteration (LSPI) method (Lagoudakis and Parr, 2003) for iteratively bootstrapping feature weights from a fixed batch of experience. In the remainder of this section, I present an overview of these methods.

Assuming rewards don't depend directly on next state, Equation 2.1 can be expressed as follows:

$$Q_\pi(s,a) = \mathcal{R}(s,a) + \gamma \sum_{s' \in S} \mathcal{P}(s',s,a) \sum_{a' \in A} \pi(s',a') Q_\pi(s',a').$$

For finite states and actions, this can be expressed in matrix form:

$$Q_\pi = \mathcal{R} + \gamma \mathbf{P} \mathbf{\Pi}_\pi Q_\pi,$$

where Q_π and $\mathcal{R}$ are column vectors of size $|S||A| \times 1$, $\mathbf{P}$ is a matrix of size $|S||A| \times |S|$ where:

$$\mathbf{P}((s,a),s') = \mathcal{P}(s',s,a),$$

and $\mathbf{\Pi}_\pi$ is a matrix of size $|S| \times |S||A|$ where:

$$\mathbf{\Pi}_\pi(s,(s,a))) = \pi(s,a).$$

Then $\mathbf{P\Pi}_\pi$ is a matrix of size $|S||A| \times |S||A|$, where the rows indicate the current state and action, and columns indicate the next state and action. That is, if the agent is in a given state and takes a given action, what is the probability that it will be in a given next state and take the given next action? Rows of $\mathbf{P}$, $\mathbf{\Pi}_\pi$, and $\mathbf{P\Pi}_\pi$ each sum to 1.

As mentioned earlier, we can approximate Q_π by introducing abstract state features, now in matrix form:

$$\widehat{Q}_\pi = \mathbf{\Phi} w_\pi,$$

where $\mathbf{\Phi}$ is a $|S||A| \times k$ matrix of transposed feature vectors for every state and action. This approximation gives:

$$\begin{aligned} \widehat{Q}_\pi &\approx \mathcal{R} + \gamma \mathbf{P\Pi}_\pi \widehat{Q}_\pi \text{ and} \\ \mathbf{\Phi} w_\pi &\approx \mathcal{R} + \gamma \mathbf{P\Pi}_\pi \mathbf{\Phi} w_\pi. \end{aligned}$$

Note that choosing $\mathbf{\Phi}$ as the identity matrix is equivalent to using original states and actions as the features for a finite state-action space.

Lagoudakis and Parr emphasize a least squares fixed-point solution to this system, which finds weights that minimize the distance between $\widehat{Q}_\pi$ and $\mathcal{R} + \gamma \mathbf{P\Pi}_\pi \widehat{Q}_\pi$ using projection onto the space spanned by $\mathbf{\Phi}$:

$$\mathbf{\Phi} w_\pi = \mathbf{\Phi} \left(\mathbf{\Phi}^\intercal \mathbf{\Phi} \right)^{-1} \mathbf{\Phi}^\intercal \left(\mathcal{R} + \gamma \mathbf{P\Pi}_\pi \mathbf{\Phi} w_\pi \right).$$

Algorithm 2.1 Least Squares Temporal Difference Q-learning (LSTDQ)

procedure LSTDQ$(D, k, \phi, \gamma, \pi)$ ▷ D is a bag of data samples from any agent behavior.
  $\widetilde{\mathbf{A}} \leftarrow \mathbf{0}_{k \times k}$
  $\widetilde{b} \leftarrow 0_{k \times 1}$
  for $(s, a, r, s') \in D$ **do**
    $\widetilde{\mathbf{A}} \leftarrow \widetilde{\mathbf{A}} + \phi(s, a)\Big(\phi(s, a) - \gamma\phi\big(s', \pi(s')\big)\Big)^\intercal$
    $\widetilde{b} \leftarrow \widetilde{b} + \phi(s, a)r$
  end for
  $\widetilde{w}_\pi \leftarrow \widetilde{\mathbf{A}}^{-1}\widetilde{b}$
  return $\widetilde{w}_\pi$
end procedure

Algorithm 2.2 Least Squares Policy Iteration (LSPI)

procedure LSPI$(D, k, \phi, \gamma, \epsilon, w_0)$
  $w' \leftarrow w_0$
  repeat
    $w \leftarrow w'$
    $w' \leftarrow$ LSTDQ$(D, k, \phi, \gamma, \pi_w)$
  until $\|w - w'\| < \epsilon$
  return w
end procedure

This yields the solution:

$$w_\pi = \left(\mathbf{\Phi}^\intercal \left(\mathbf{\Phi} - \gamma \mathbf{P}\mathbf{\Pi}_\pi \mathbf{\Phi}\right)\right)^{-1} \mathbf{\Phi}^\intercal \mathcal{R}.$$

To express that some states and actions have greater importance or are visited more frequently than others, a diagonal weighting matrix Δ_μ may be included (which is equivalent to nonorthogonal projection):

$$w_\pi = \left(\mathbf{\Phi}^\intercal \Delta_\mu \left(\mathbf{\Phi} - \gamma \mathbf{P}\mathbf{\Pi}_\pi \mathbf{\Phi}\right)\right)^{-1} \mathbf{\Phi}^\intercal \Delta_\mu \mathcal{R}.$$

The above solution requires an enumeration of all states and actions, whether or not they are encountered. It also requires knowledge of $\mathbf{P}$. Alternatively, the following components can be approximated iteratively over examples from actual

agent experience:

$$\begin{aligned} \mathbf{A} &= \left(\mathbf{\Phi}^\intercal \Delta_\mu \left(\mathbf{\Phi} - \gamma \mathbf{P} \mathbf{\Pi}_\pi \mathbf{\Phi}\right)\right), \text{ and} \\ b &= \mathbf{\Phi}^\intercal \Delta_\mu \mathcal{R}, \end{aligned}$$

where $w_\pi = \mathbf{A}^{-1} b$. Algorithm 2.1 shows the LSTDQ algorithm, which approximates these components (as $\widetilde{\mathbf{A}}^{-1}$ and $\widetilde{b}$) and calculates w_π. Finally, Algorithm 2.2 shows the Least Squares Policy Iteration (LSPI) algorithm for bootstrapping policy learning from a fixed set of examples D. Each iteration of LSTDQ provides a new policy, which in turn affects the next calculation of $\widetilde{\mathbf{A}}$. Here, w_0 contains initial weights, which might be zero, small random values, or based on prior learning, if any, and ϵ is a cutoff level to test for approximate convergence.

2.2 Representation Learning

In reinforcement learning, linear approximations of Q_π depend intimately on feature space $\phi(s, a)$. Features could be manually specified in a domain-dependent fashion, or perhaps in a somewhat generic form such as polynomials, radial basis functions, overlapping tilings (Sutton and Barto, 1998), or cosine waves (Konidaris et al., 2011). Alternatively, features can be automatically derived from knowledge of the task space or learned from agent experience.

Feature selection and construction are both common concerns in the general field of machine learning (Guyon and Elisseeff, 2003), not just for representations for RL. Feature *selection* assumes an existing set of features, some of which might be better for the task at hand than others. Many selection techniques are classified as either *filter* methods, operating before the primary learning mechanism is used, or else *wrapper* methods, selecting features by testing the performance of the primary learner given different feature subsets. Feature *construction* assumes that the best representation might be (possibly nonlinear) functions of one or more of

the originally available features. Newly constructed features might be aggregated to the existing set, increasing dimensionality, or they might be used to replace existing features, possibly decreasing dimensionality (such as for principle component analysis). Also, the recent explosion in deep learning methods (Bengio, 2009), building on older work such as multilayer artificial neural networks (Hornik et al., 1989; Schmidhuber, 2015), is largely concerned with the topic of feature construction.

2.2.1 Feature Construction for RL

Early work in automatic feature or basis function construction specifically for reinforcement learning includes the Proto-Value Functions (PVFs) of Mahadevan (2007), which generates basis functions from the state adjacency matrix. Another common technique is to build functions based on the Bellman error (BE) of the current Q function (Parr et al., 2008; Wu and Givan, 2010). For any particular example of agent experience

$$BE = r + \gamma Q_\pi(s', a') - Q_\pi(s, a). \tag{2.3}$$

As earlier discussed, the Bellman error is 0 in expectation for the correct Q_π. Features constructed in this fashion are sometimes called Bellman Error Basis Features (BEBFs). Mahadevan and Liu (2010) shows that BEBFs converge slowly for high γ and suggests the use of Bellman Average Reward Bases (BARBs) as an alternative. Features can also be constructed based on actual agent performance (Girgin and Preux, 2008b). The techniques above are immediately useful to LSPI, although it is worth noting that feature learning techniques also exist that are more directly tied to other RL methods (e.g., Girgin and Preux, 2008a; Kersting and Driessens, 2008). Finally, feature selection (in addition to construction) is also a concern in RL (for example, Kolter and Ng, 2009).

The technique of interleaving feature learning and policy learning is called Rep-

Algorithm 2.3 Generic Representation Policy Iteration (RPI) using LSTDQ

```
procedure RPI(γ, ε, w_0)
    repeat
        Gather agent experience D.
        Learn representation φ, whose dimensionality is k.
        w' ← w_0
        repeat
            w ← w'
            w' ← LSTDQ(D, k, φ, γ, π_w)
            Optionally adapt the basis.
        until ||w − w'|| < ε
    until converged, if desired
    return w
end procedure
```

resentation Policy Iteration (RPI) by Mahadevan (2005b). Although, in his work, he primarily focuses on particular forms of basis function construction, this framework is applicable to a full variety of techniques, as discussed above. I give a modified RPI overview in Algorithm 2.3. When using LSTDQ, the inner loop of RPI is very similar to LSPI, except with the added optional step for basis adaptation. Mahadevan also leaves the outer loop optional, and other small variations also exist in different presentations of the framework. Within my present work, I use the term RPI generally to mean *any* interleaving of representation and policy learning.

2.3 Relational Reinforcement Learning

2.3.1 Relational Learning

In many real-world contexts, multiple objects exist with relationships between each other. For example, a plate could be on a table, or one person might be standing between two others. Such contexts are often described via first-order logic. For example:

$$\exists_X \, plate(X) \wedge on(X, table)$$

expresses that there exists at least one plate such that is on the table, which, here, is presumed to be a constant. First-order logic, which includes quantifiers $\exists$ and $\forall$, differs from *propositional* logic, which operates on fixed constants.

Logical quantifiers loop over possible objects in a scene. Importantly, the number of objects in a scene might not be constant from one situation to the next. This is often touted as a benefit of relational learning over propositional learning, where *relational* also is often taken to mean *first-order* (Blockeel and De Raedt, 1998; Džeroski et al., 2001). This same perspective consideres traditional machine learning algorithms, operating on vectors of fixed length, to be propositional.

TILDE

While much of the field of Inductive Logic Programming (ILP; see Muggleton, 1991; De Raedt and Kersting, 2008) includes some amount of first-order logical learning, the most relevant alternative to my present work is the TILDE (Top-down Induction of Logical Decision Trees) algorithm of Blockeel and De Raedt (1998). TILDE recursively builds a decision tree, choosing query nodes that maximize some metric (such as information gain) with respect to the training examples and background knowledge. Each training example is a graph with objects and propositions giving information about them, including relations between objects.

The types of queries available are manually given in the form of *refinement specifications*. These specifications indicate how many variables may be introduced, which may be reused from existing variables, which predicates can be used, and so on. Further, manual lookahead specifications allow multiple queries to be introduced at the same time (Blockeel and De Raedt, 1997), which otherwise is prohibited due to exponential growth in the search space. For example, when playing soccer, it might be relevant that an opponent is nearby. However, that *some* player is nearby might not be very meaningful on its own, and that *some* player is an opponent

might also be meaningless for making a decision, without also emphasizing where that player is. However, asking *all* pairs (or more) of questions in a row without having some meaningful distinctions made along the way is a deep search problem. Therefore, a lookahead specification might be given to allow asking about team membership followed by specific kinds of questions about location.

Free variables are implicitly existentially quantified. A query *succeeds* if it has at least one true binding of constants to the variables. A recursive query down a tree follows only one path, so if a query fails, it means that there exists no binding. In this fashion, TILDE also supports universal quantifiers, although negative forms of predicates need to be explicitly specified to allow positive universals. For example, to say that all parts of a machine are working, a specification needs to be given to allow asking that a part is *not* working, and a query can then test that there exists no broken part. That is, the query would fail, and the example would proceed down the negative branch, thus saying that all parts are in working order.

Core TILDE works with discrete data rather than with continuous values. Therefore, continuous values must be discretized into bins before learning a tree or querying an existing one (Blockeel and De Raedt, 1997). TILDE usually determines thresholds in a supervised fashion based on information entropy in the training set. For multidimensional attributes (e.g., position in 3D space), each dimension is discretized as a separate variable, and a single query can also ask about only one dimension. Multiple queries down a single branch a required to carve a volume in multidimensional space.

TILDE has also been used for regression, in a batch-learning algorithm called TILDE-RT (Blockeel et al., 1998) or an incremental version called TG (Driessens et al., 2001). For TILDE regression, each scene has a single real-valued label. Each leaf has a value which is the prediction for scenes arriving there. In learning, the goal is typically to minimize mean squared error. Queries are selected that minimize

this, where leaf values are selected as the mean of values of the scenes arriving at them. Finally, ReMauve (Vens et al., 2007) is a variation that allows for linear regression in the leaves by selecting aggregate or single values from the constants bound to variables in a tree.

Other Relational Learning Approaches

As mentioned above, a variety of other relational learning systems exist. Most notable for my present work is the Spatiotemporal Multidimensional Relational Framework (SMRF) of Bodenhamer (2014), which I use for relational learning for the methods presented in this dissertation. SMRF is a decision tree framework that operates directly on multidimensional continuous attributes, as opposed to the discretization and single-dimensional approach of TILDE. SMRF has also been shown to handle large numbers of distractor objects better. I cover this framework in detail in Chapter 4.

Additionally, a large body of work also exists in the field of Multiple Instance Learning (MIL), which most commonly addresses simple existential questions in multidimensional continuous spaces. As demonstrated by Bodenhamer (2014), such algorithms can be applied to pairwise relational learning problems, again by creating instances as pairs of existing objects, with attributes as relative measures. I address MIL in more detail in Chapter 3.

2.3.2 Combining Relational Learning with Reinforcement Learning

As relational learning inherently addresses worlds with variable numbers of objects, much work exists in the use of relational representations for reinforcement learning (van Otterlo, 2005, 2012). Of course, relational planning mechanisms are one of the traditional foci in artificial intelligence (e.g., Fikes and Nilsson, 1971).

In this dissertation, my own focus is on model-free reinforcement learning, typi-

cally centered on variations around Q-learning. As seen earlier, learning the Q function allows policy learning without explicit modeling of the state following agent action. From a relational perspective, one common Q-learning strategy is to employ relational regression methods to approximate the Q function as experience increases. One of the seminal works in this field is the relational reinforcement learning work of Džeroski et al. (2001). This work uses TILDE-RT, where each training example consists of (s_i, a_i, q_i), and each $q_i = r_i + \gamma \max_{a'} \hat{Q}_j(s_{i+1}, a')$, in standard fashion. The Q function is subscripted here by experience iteration j. Specifically, the agent gathers a batch of experience based on current function Q_j, and this generates learning examples as described. With each new batch of examples, TILDE-RT learns a new tree. After the first iteration, each new batch is aggregated to prior examples. Each old example has its associated q_i updated to match the latest Q_j. Using the entire set of examples, a new tree is learned which replaces the old tree. This provides the Q-RRL algorithm. Džeroski et al. (2001) also present the P-RRL algorithm which learns an additional tree predicting the policy directly and generalizes better to environments where the Q function itself might be inconsistent.

Later work in this vein employs other relational regression algorithms in addition to incremental variations on the algorithms using TG (Driessens et al., 2001). When incremental, each new MDP sample (s, a, s', r) can be immediately used for updating the Q function. Other relational regression algorithms employed including relational instance-based regression (RIB) of Driessens and Ramon (2003) and the TRENDI method of Driessens and Džeroski (2005), which learns trees via TG, but uses RIB in the leaves rather than a constant mean value. The kernel-based relational regression (KBR) method of Gärtner et al. (2003) uses graph kernels and Gaussian processes for Q-function regression. Kersting and Driessens (2008) differ in learning relational trees (using TG) to follow a probabilistic policy gradient; Natarajan et al. (2011) use this same method for imitation learning. Wu and Givan

(2010) use a beam search through relational expressions finding those that correlate with Bellman error within a representation policy iteration framework. Jetchev et al. (2013) use a relational nearest neighbor approach to learn grounded symbols for MDP planning techniques. Zaragoza and Morales (2009) employ an RRL system for robot navigation that discretizes state and action space but also constructs continuous actions by interpolation. Many other techniques for RRL have also been employed (van Otterlo, 2012).

Of note, across these techniques, while various algorithms other than trees have been introduced for RRL, tree learning, and especially TILDE in various forms, is one of the common methods that continues to be employed, suggesting its versatility and effectiveness. However, as noted, SMRF has been shown to be more effective than TILDE in multidimensional, continuous domains for binary classification (Bodenhamer, 2014), but prior to my current work, it has not been used for reinforcement learning.

2.3.3 Other Continuous, Relation-Oriented Task Learning

Of note beyond model-free RRL, other planning and RL techniques exist for continuous, multidimensional tasks of a relational nature. Some are relational in the first-order sense, and some merely consider physical relations without support for quantifiers. Some emphasize learning of abstract predicates and symbols, and others use human-designed, high-level representations.

One notable work in relational MDPs for continuous tasks is that of Pasula et al. (2007), which employs human-designed predicate representations of a 3D physics-based, simulated block-stacking domain. Their algorithm learns rules predicting the outcomes of actions, thus yielding a relational model useful for MDP planning. Lang and Toussaint (2010) explore more advanced planning algorithms built atop models learned by the method of Pasula et al. (2007). Kulick et al. (2013) continue this

work, learning specific relations as noted earlier, working from simulated and real robot action. They work from specific pairs of objects, though, rather than being concerned with quantification when learning relations. Of note, they also include an action with a continuous parameter (not associated with an actually present object), but this action is used only for active learning and not for relational task planning.

Mugan and Kuipers (2009), Modayil and Kuipers (2008), and Konidaris et al. (2014) also learn symbolic grounding of various sorts from multidimensional, continuous data, but only in propositional form. Mugan and Kuipers (2009) is also an example of the field of Qualitative Reasoning (QR), which focuses on the use of high-level representation for perhaps otherwise continuous values. QR can include first-order relational work as well (for example Zhang and Renz, 2014). Xu and Laird (2011) learn symbolic, relational rules and also continuous, non-relational action models, fusing information from the two in making predictions about action outcomes.

As yet another and much different approach, Verbancsics and Stanley (2010) use the evolutionary HyperNEAT learning algorithm with a grid-based representation, where objects in the scene are marked onto the grid. This inherently supports an arbitrary number of objects in the grid, although the technique has a different nature than the set-based approach of standard relational representations.

Chapter 3

Multiple Instance Learning via Covariant Aggregation

A primary concern in relational learning is identifying the objects that play particular roles. For example, in soccer, does there exist an opponent between me and the teammate to which I want to pass the ball? Such an opponent has the role of a potential interceptor, and the opponent has this role *because* of the physical relationship of being between me and my teammate. There are other players on the field who do not have this role. Mathematically, calculating the positions of all opponents relative to me and my teammate yields a set of vectors, and only those vectors within a particular region represent potential interceptors. In learning how to play, I need to determine what is in common between the cases where I make a successful pass as opposed to when I lose control of the ball.

This becomes an existential binary classification problem, commonly called a multiple instance learning (MIL) problem. For the larger relational setting, many relationships might need considered. Is there someone between the passer and the receiver? Is an opponent near the ball? Are multiple opponents nearby? Each such question becomes a MIL classification problem, and the specific nature of the layout of vectors for each question might vary. We might also want to ask more than one question about the same players or objects. Also, because of the multidimensional nature of physical attributes, it is possible that correlation exists between multiple dimensions. These are among the concerns at hand in solving the MIL problem for the relational tasks of interest in this dissertation. Therefore, in this chapter, I contribute a novel MIL algorithm for learning simple, covariant decision volumes

which performs robustly and quickly in a variety of example MIL data sets, without parameter tuning to each case.

3.1 Introduction

Machine learning, including classification, usually addresses inputs of individual, fixed-length feature vectors. In contrast, multiple instance learning (MIL) addresses unordered sets or bags of instances, where each instance is commonly a feature vector (Dietterich et al., 1997). Binary MIL classification problems commonly consider a bag *positive* if it has at least one instance that is a positive example of some concept; in other words, at least one instance is in a positive region of the vector space. In this formulation, MIL is an existential classification problem:

$$positive(B) = \exists_{x \in B}\, positive(x), \tag{3.1}$$

where B is a bag, and each x is an instance. MIL therefore goes beyond ordinary vector-based learning, in that the learning algorithm must identify relevant instances while also learning to classify them. It also therefore provides a foundation for existential relational learning, as will be further discussed in Chapter 4.

Common examples of MIL include that of molecule classification, where each molecule has a set of potential foldings, and any one of those foldings might signify the key behavior of the molecule. Image classification is another example, where the task is to identify whether or not an image contains a particular object, given a bag of visual features, where it does not matter where an object is in the image or if other objects are present. A third example, relevant to relational task learning within the scope of this dissertation, in playing a game of soccer, passing a ball is likely to fail if there exists an opponent between the passer and the potential receiver; positions of other opponents might be less relevant.

Early MIL algorithms include axis-parallel rectangles (APR) of Dietterich et al. (1997) and diverse density (DD) of Maron and Lozano-Pérez (1997). APR greedily selects discriminating features, choosing minimum and maximum bounds for each feature value to ensure that at least one instance from each positive bag is included. DD defines a probabilistic metric emphasizing instances with many positive and few negative bags nearby. Using the DD metric, gradient ascent selects the best central feature vector and the scaling for each feature. Many other MIL algorithms and multiple-instance problem formulations (beyond just existential queries) have since been developed (see Foulds and Frank, 2010). These include such concerns as bag distance metrics and MIL-oriented kernels and constraints for support vector machines. While many of these techniques are often quite effective, learned models are not always intuitively clear to humans. Further, some techniques, including DD and APR, label individual instances as positive or negative, while others focus exclusively on labeling each bag as a whole.

In our work, we seek to determine instance labels, as well as to provide a simple description of the relevant volume in the feature space, such as those provided by APR and DD. However, neither APR nor DD acknowledge covariance that can occur across features. Covariance can arise, for example, in color models (such as yellow in RGB space, which contains equal parts red and green) or in spatial configurations (where interesting cases have some object along a particular vector). In MIL, simply realigning data to principal components is not ideal, as the representative instances might align very differently than the aggregate set of instances from positive bags. While Zhao et al. (2013) address learning of covariant distance metrics within MIL, they focus neither on simple decision volumes nor instance classification. Most similar to our own work is the recent work of Kandemir and Hamprecht (2014), who learn a Gaussian mixture model for MIL instance prediction using a Bayesian formulation. In this chapter, we present a learning technique that directly addresses

the learning of simple, covariant decision volumes for instance label prediction, and we show that our method is robust across a variety of learning problems without parameter tuning.

In the remainder of this chapter, we describe our MIL algorithm, which learns a covariant decision volume via iterative aggregation of volume-describing instances from positive bags. We then evaluate our method, on both synthetic and real-world data sets, against other well-known algorithms (including DD) that either describe ellipsoidal regions or else use radial feature kernels.

3.2 Learning Algorithm

We follow the common existential, instance-classifying MIL formulation established in Equation 3.1. In this form, the classifier's job is to classify individual instances. If any instance in a bag is found to be positive, then the bag itself is labeled as positive.

The differentiating aspect of our approach is that of covariant decision volumes. This contrasts with the axis-aligned volumes of APR and DD. Specifically, we describe a decision volume by its mean μ, covariance V, and radius r. Parameters μ and V provide the Mahalanobis distance from the volume center:

$$D_M(x|\mu, V) = \sqrt{(x-\mu)^T V^{-1}(x-\mu)}. \tag{3.2}$$

Any instance x within radius r is considered positive:

$$positive(x|\mu, V, r) \iff D_M(x|\mu, V) \leq r. \tag{3.3}$$

Throughout our discussion, we treat instances as feature vectors within a Euclidean topology. As such, we also refer to instances as *points*.

We now describe an algorithm for learning covariant decision volumes from training data with labeled bags. Algorithm 3.1 gives informal pseudocode for the learning process, and Figure 3.1 shows several steps of the process using a synthetic data set in which positive bags are more likely to have at least one instance along the diagonal. In summary, our algorithm seeks a set of *key points*, K, from which to estimate parameters μ and V. This process begins with a single key point from one sampled positive bag. At each iteration, the algorithm chooses a new key point from an unrepresented positive bag to add to this set, where the candidate key point from a bag is its point nearest to the current decision volume center (by Equation 3.2), also called the bag's *witness point*. Given the new potential set of key points, μ and V are reestimated, and a new radius, r, is chosen to maximize the training accuracy (similar to the method of Auer and Ortner, 2004). Key point aggregation continues while training set accuracy across recent iterations suggests possible improvement.

In the remainder of this section, we elaborate on each step of the algorithm.

3.2.1 Covariance Estimation

Our algorithm determines a covariant, ellipsoidal decision volume starting from a single key point. Additional key points are aggregated iteratively. We calculate μ as the mean of the key points. However, when there are few key points relative to the dimensionality, covariance V is poorly defined. To condition the covariance, we begin with an inverse Wishart prior based on all points from all positive bags. The inverse Wishart distribution is conjugate prior to the multivariate covariance matrix (see Gelman et al., 2013). That is, given additional observations defining a covariance matrix, the posterior distribution is still inverse Wishart.

The inverse Wishart distribution has two parameters: a scale matrix, Ψ, and degrees of freedom, ν. The matrix Ψ represents a prior observation of covariance

Algorithm 3.1 Covariant Aggregation

```
Begin set K with one initial key point.
Let B_R be all positive bags not containing initial point.
Estimate μ and V from K.
Let W be witness points of min D_M(x|μ, V) for all bags.
Choose radius r to maximize training set accuracy.
Remember (μ, V, r) as initial best.
while progress continues do
    Let B_S be M sampled bags from B_R.
    for all B in B_S do
        Let K' be K ∪ the witness point from W for B.
        Determine μ', V', W', and r' using K'.
        Evaluate training accuracy for K'.
    end for
    Update K, μ, V, W, and r for best B sampled.
    Update B_R to exclude best B.
    if new training accuracy ≥ previous best then
        Remember new best (μ, V, r).
    end if
end while
Return best (μ, V, r).
```

scaled by ν presumed observations. The posterior Ψ_{post} is given by:

$$\Psi_{post} = \text{ncov}(X) + \Psi$$

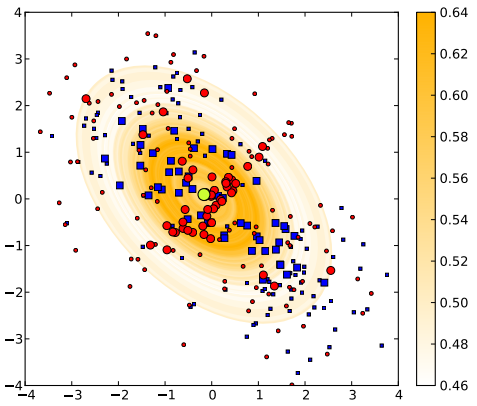
where

$$\text{ncov}(X) = (X - E[X])(X - E[X])^T \tag{3.4}$$

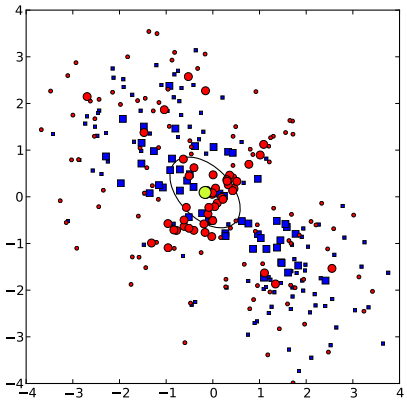
is a scale matrix based on observations as column vectors in the matrix X.

Usually, the magnitude of the posterior covariance depends on the sum of ν and the number of new observations. However, in our case, we use our estimate of V only for the shape and orientation of the decision volume. Radius r is determined at a later step.

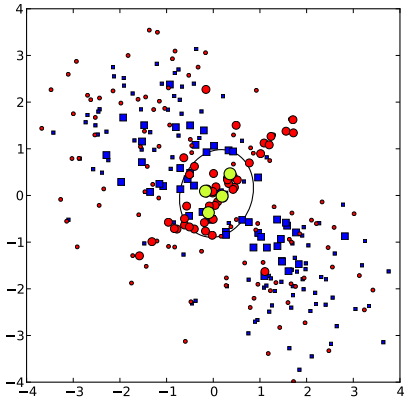
Also, because our learning problem is multiple-instance, there is no necessary relationship between all instances from positive bags and the final decision volume.



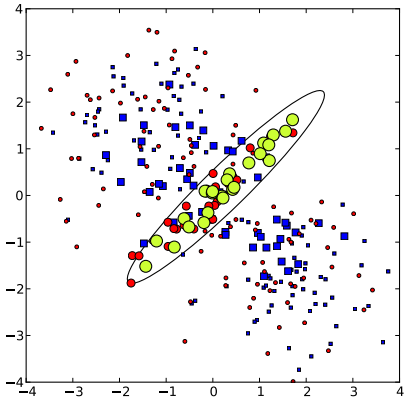
(a) Initial key point, with training set accuracy by radius.



(b) Radius with optimal accuracy.



(c) Decision volume after four key points.



(d) At 24 key points, the final decision volume.

Figure 3.1: Overview of the learning process. Circles indicate points from positive bags, and squares indicate negatives. Pale circles indicate key points. Larger markers indicate witness points: the nearest point from each bag, given the current mean and covariance.

Therefore, we treat the weight of the prior as independent of both the number of bags and the number of instances. Instead, we use a hand-selected parameter, w, to influence the overall weight of the prior. Our effective posterior covariance is therefore:

$$V_{post} = \text{ncov}(K) + w\,\text{ncov}(P)/p, \tag{3.5}$$

where K represents the key points and P represents all points (quantity p) from all positive bags. Figures 3.1(a) and 3.1(b) show an initial covariance based entirely

on the prior, while Figures 3.1(c) and 3.1(d) show how an increasing number of key points can increasingly change the volume shape.

However, the covariance could still be poorly conditioned, especially in situations of high dimensionality. If the condition number of the covariance matrix is above a particular threshold c, we add a constant diagonal to this covariance:

$$V = \begin{cases} V_{post} & \text{if } \left|\frac{\max \Lambda}{\min \Lambda}\right| \leq c \\ V_{post} + \frac{\max \Lambda - c \min \Lambda}{c-1} I & \text{otherwise,} \end{cases} \tag{3.6}$$

where Λ is the set of eigenvalues of V_{post}, and I is the identity matrix. This yields a covariance matrix with a maximum condition number of c.

3.2.2 Decision Volume Radius

To choose decision volume radius r, we follow the optimal ball algorithm of Auer and Ortner (2004). Specifically, we select the radius that maximizes classification accuracy across all bags in the training set. Because each bag is classified as to whether any of its points lie within the decision volume, only the nearest point to the volume center is of interest. This nearest point is called the *witness point* for its bag. We differ from Auer and Ortner primarily in our use of Mahalanobis distance. Formally, our set of witness points W is defined as:

$$W = \left\{ \underset{x \in B}{\operatorname{argmin}} \; D_M(x|\mu, V) \;\middle|\; B \in \mathcal{B} \right\}, \tag{3.7}$$

where $\mathcal{B}$ represents all training bags (both positive and negative).

To choose radius r, we first sort W by increasing distance, and then perform a linear scan for maximal accuracy, also as suggested by Auer and Ortner. Figure 3.1(a) illustrates training set accuracy as a function of radius, and Figure 3.1(b) shows the corresponding radius maximizing accuracy.

3.2.3 Key Point Aggregation

Before the outer loop of Algorithm 3.1, the algorithm samples a single positive bag. For each point in the bag, using only the covariance prior as V, a radius is chosen to maximize training accuracy. Iterative covariant aggregation, shown in Figure 3.1, begins from the point yielding highest accuracy.

At each aggregation step, the algorithm samples M positive bags currently unrepresented among the key points, where M is an algorithm parameter. Further, to keep the search localized, only bags whose witness points lie inside the current decision volume are sampled, where, as suggested in Figure 3.1, the set of candidate bags changes as the volume changes. The witness point (and no others) from a sampled bag is tentatively added to the key points, and a new decision volume is constructed. Among sampled bags, the tentative key point set with highest accuracy is selected, and the aggregation process repeats.

Early in the search process, with few key points, the decision volume's shape is very flexible. It becomes more established as the number of key points increases. Therefore, to terminate the aggregation process, we make a linear least squares estimate of training accuracy over the most recent A aggregation steps, where A is an algorithm parameter. We continue aggregation if the accuracy slope is positive and if, continuing the current slope through remaining positive bags, the accuracy would surpass the previous best (including across restarts, as discussed shortly). Otherwise, aggregation stops, retaining the model with the highest accuracy. If decision volumes at multiple iterations have equal accuracy, we select the one with the most key points, as this presumably better describes the volume. At minimum, a full window of A iterations is required before termination.

As is common for stochastic algorithms, we restart the learning process multiple times. After each aggregation process, we sample another positive bag for the next

restart. Algorithm parameter N determines the number of restarts. Of the N resulting models, the volume with the highest accuracy with respect to the training set (or, again, in case of a tie, the one with the most key points) is selected as the final volume.

3.3 Experimental Evaluation

We test our method on two types of data sets: (1) synthetic data in two or three dimensions, of which some exhibit covariant regions of interest (discussed below), and (2) standard third-party MIL data sets, which usually have high dimensionality and do not necessarily exhibit interesting covariance.

We compare against other algorithms with ellipsoidal decision boundaries or ellipsoidal kernels. Techniques with ellipsoidal boundaries include the non-boosted optimal ball algorithm of Auer and Ortner (2004), and the DD (Maron and Lozano-Pérez, 1997) and EM-DD (Zhang and Goldman, 2001) algorithms. None of these are capable of expressing covariance, although DD and EM-DD scale original axes. We use the Weka (Hall et al., 2009) implementations of these algorithms with default parameters.

We also compare against SVM techniques mi-SVM, MI-SVM (Andrews et al., 2002), and MILES (Chen et al., 2006). For mi-SVM and MI-SVM, we use (isotropic) RBF kernels, and the MILES feature set is also based on a radially decaying measure. All three are therefore capable of expressing arbitrarily nonlinear surfaces, including approximations of covariance. Unlike other techniques compared here, MILES is a bag-only classifier rather than an instance classifier, but we include it because it is simple and performs well. We use the MISVM package of Doran and Ray (2014) for implementations of mi-SVM and MI-SVM. We use our own implementation of the MILES feature set with LIBLINEAR (Fan et al., 2008) as the SVM classifier,

using the L2-loss L1-regularization option, which differs from the L1-loss of standard MILES.

In testing the algorithms, we are primarily concerned with performance "in the wild." That is, we assume that new learning problems need to be addressed without the opportunity for extensive analysis and parameter tuning. We therefore hold all parameters constant for our learning algorithm. Specifically, we hold constant the progress window size A at 8, covariance prior weight w at 10^{-1}, max covariance condition c at 10^5, and both number of restarts N and number of aggregation bags sampled M at 4. We have chosen these parameters based upon exploratory investigations with some of the synthetic data sets, as well as the well-known Musk 1 data set (Dietterich et al., 1997).

Using a similar amount of exploration, we have also selected parameters C and γ for evaluation of SVM-based techniques. For C, we use a constant value of 10^4. We choose $\gamma = 1/2\sigma^2$ using a heuristic calculation for σ as the mean of the tenth and ninetieth percentiles of all (positive and negative) inter-point distances in the training set (e.g., as used as a starting point by Takeuchi et al., 2006).

Also, in exploration with DD and EM-DD, we found it necessary to filter the Musk data sets for meaningful learning results. Specifically, we divide the values of each feature dimension by their standard deviation. For consistency, we do this for all data sets for DD and EM-DD. We have not performed such filtering for other learning methods, although mechanisms such as the inverse Wishart prior used by our method provide a similar effect.

For our synthetic data sets, in addition to the heuristically-chosen SVM parameters, we also test against tuned SVM parameters. In these cases, we search across a grid of C values from 10^{-4} to 10^9 (stepping by powers of 10) and factors of our heuristically-chosen γ, from 2^{-7} to 2^7.

Our reported results use different random seeds for data generation, shuffling,

and/or stochastic learning than those used during exploratory evaluation or parameter selection.

3.3.1 Synthetic, Low-Dimensional Data Sets

Our method is designed for use in low-dimensional (often physical) settings with potentially covariant classification volumes. We have therefore designed multiple synthetic learning problems that exhibit interesting characteristics in two or three dimensions. Summarized in Table 3.1, these data sets consist of the following mixture distributions:

- **Covariant 1**, depicted in Figure 3.1, is a 2D problem where negative instances come from one of two isotropic Gaussians of standard deviation 1 and means of $(-2, 2)$ and $(2, -2)$, respectively. Positive instances come from a covariant Gaussian along the orthogonal diagonal, centered at $(0, 0)$. Eigenvalues of the positive distribution are 1 and $1/5^2$. All bags have 3 instances, where exactly 1 of the 3 is drawn from the positive distribution for each positive bag.

- **Covariant 2** is the same as Covariant 1, except that there is a single negative distribution, centered at $(0, 0)$ along with the positive distribution. This problem is therefore very noisy.

- **Color** is a 3D problem where each distribution represents a red, green, blue, or yellow color in RGB space. The distributions are based on photographs of printed color patterns, and thus represent data with real-world characteristics. Red, green, and blue are negative distributions. Yellow is the positive concept and is inherently covariant in RGB space. All bags contain 5 instances. Positive bags again contain exactly one instance drawn from the positive distribution.

	Dimensions	Covariant	# Pos Dists	# Neg Dists	Bag Size
Color	3D	Yes	1	3	5
Covariant 1	2D	Yes	1	2	3
Covariant 2	2D	Yes	1	1	3
Disjunctive	2D	No	2	1	10

Table 3.1: Summary of synthetic data sets. Each positive bag contains exactly one instance drawn from a positive distribution.

- **Disjunctive** is a non-covariant data set, designed to challenge our proposed approach by drawing positive instances from a non-compact set. A single negative, isotropic distribution is centered at $(0, 0)$ with a standard deviation of 1. The positive distribution is a mixture of two, isotropic Gaussian distributions, centered at $(\pm 3, 0)$, each with a standard deviation $\sqrt{1/2}$. All bags contain ten instances, filling up the negative space thoroughly. Positive bags contain exactly one instance drawn from either side of the mixture (both are equally probable).

For all synthetic data sets in this work, we generate 100 training bags (50 positive and 50 negative) and 100 test bags. Because the data is synthetic, rather than doing n-fold cross-validation, we generate 100 independent training and test sets for calculating statistics.

3.3.2 Standard MIL Data Sets

In addition to our synthetic cases, we also test using several standard data sets. Specifically, we use MIL data sets provided by the Weka project, including most of those evaluated by Foulds and Frank (2008). We specifically evaluate performance on the well-known data sets Musk 1 and Musk 2; other chemical classification sets Thioredoxin and Mutagenesis Atoms, Bonds, and Chains; the Corel image data sets Elephant, Fox, and Tiger; and the train direction classification problem East-West. From the Weka-distributed sample files, we exclude the Component, Function, and

Process data sets because they are very large. We also exclude two data sets for applicability reasons: Suramin, because some instances contain unspecified values and West-East, because it is a universal rather than an existential problem. Note also that any artificial tuning to Musk 1 performance imposed by our exploratory selection could have negative consequences on the remaining datasets.

Of the third-party data sets we test, the least number of dimensions (for Thioredoxin) is 8. Some data sets (such as Fox or Musk) have more than one hundred. Furthermore, we have no expectation in advance of which data sets are well represented by covariant volumes. As such, these standard sets are outside the designed use case of our method. Still, we include test results here for comparison with other MIL algorithms. For synthetic data sets, statistics are computed using stratified 10-fold cross validation.

3.3.3 Evaluation Method

We perform statistical comparisons using a standard two-sample, two-tailed t-test using $\alpha = 0.05$. For cases where the mean score of our method is better than that of another, we additionally apply Šidák correction for multiple comparisons:[1] $\alpha_{corrected} = 1 - (1 - \alpha)^{1/n}$. For our 9 comparisons, $\alpha_{corrected} \approx 0.00568$. When another method outperforms ours, we retain the original α. This has the effect of not overestimating the performance of our approach, whether our approach performs better or worse than another.

We report algorithm performance using the Peirce Skill Score (PSS), which penalizes merely reporting the most common label (Stephenson, 2000). In binary classification, this score is equivalent to the hit rate minus the false alarm rate. A PSS of 1 is perfect, 0 is random or majority class assignment, and -1 is a perfectly incorrect labeling. Many of our evaluated data sets have balanced or nearly-balanced

[1] Jensen and Cohen (2000) refer to this correction as Bonferroni adjustment.

labels. Therefore, higher PSS here often implies higher accuracy, but not in every case.

3.3.4 Results

As shown in Table 3.2, our method, labeled CovAgg, outperforms the other methods, with the exception of MI-SVM, on the three explicitly covariant data sets. Across these sets, MI-SVM performs about as well as our approach. As further shown in Table 3.3, the default parameter values for MI-SVM on these sets are already near the best performing parameters. Parameter tuning brings mi-SVM to nearly the same performance level. It therefore also seems that, even in noisy settings, our method extracts covariant volumes very effectively. As expected, Optimal ball ("OptBall" in the table), DD, and EM-DD fail to support covariance, as exhibited by the low performance.

On the other hand, all methods seem to do fairly well on the Disjunctive set, perhaps with the exception of EM-DD. While CovAgg performs below the level of some others, the difference isn't great. Most of the techniques, including CovAgg, extract volumes covering one of the two positive areas. Notably, mi-SVM and MILES-LL seem capable of extracting both areas. It is unsurprising that an SVM with an RBF kernel could accomplish this, because support vectors with radial influence can be chosen on both sides. Interestingly, MI-SVM, even with parameter tuning, is unable to find both positive regions.

Across the standard, third-party data sets, CovAgg performs approximately as well as the other methods. It does fall behind most techniques on Tiger and behind MI-SVM and MILES-LL on Elephant. The SVM techniques, on the other hand, perform poorly overall on the Mutagenesis data sets using the default parameters. That is, our heuristic SVM parameter selection works well for some problems, but tuning is needed for others.

	CovAgg	OptBall	DD	EM-DD	mi-SVM	MI-SVM	MILES-LL
Color	0.93 ± 0.01	$> 0.88 \pm 0.01$	$> 0.88 \pm 0.01$	$> 0.88 \pm 0.03$	$> 0.78 \pm 0.03$	$< 0.95 \pm 0.01$	$> 0.86 \pm 0.01$
Covariant 1	0.76 ± 0.02	$> 0.46 \pm 0.02$	$> 0.44 \pm 0.02$	$> 0.38 \pm 0.04$	$> 0.39 \pm 0.03$	0.73 ± 0.03	$> 0.24 \pm 0.03$
Covariant 2	0.27 ± 0.02	$> 0.13 \pm 0.02$	$> 0.06 \pm 0.02$	$> 0.10 \pm 0.02$	$> 0.11 \pm 0.02$	$> 0.21 \pm 0.02$	$> 0.07 \pm 0.01$
Disjunctive	0.27 ± 0.02	$< 0.33 \pm 0.02$	$< 0.31 \pm 0.02$	$> 0.19 \pm 0.04$	$< 0.62 \pm 0.02$	0.24 ± 0.06	$< 0.47 \pm 0.03$
Musk 1	0.61 ± 0.18	0.51 ± 0.22	0.71 ± 0.17	0.71 ± 0.19	0.64 ± 0.15	0.60 ± 0.21	0.63 ± 0.12
Musk 2	0.56 ± 0.11	0.49 ± 0.14	0.58 ± 0.23	0.69 ± 0.15	0.43 ± 0.18	0.49 ± 0.16	0.59 ± 0.19
Muta Atoms	0.54 ± 0.14	0.47 ± 0.19	0.34 ± 0.21	0.20 ± 0.20	> 0.00	$> 0.17 \pm 0.15$	$> 0.09 \pm 0.15$
Muta Bonds	0.46 ± 0.15	0.45 ± 0.20	0.36 ± 0.15	0.35 ± 0.15	> 0.00	> 0.00	$> 0.15 \pm 0.11$
Muta Chains	0.42 ± 0.14	0.31 ± 0.14	0.50 ± 0.18	0.23 ± 0.23	> 0.00	0.28 ± 0.15	0.47 ± 0.17
Thioredoxin	0.13 ± 0.14	0.30 ± 0.21	0.34 ± 0.28	0.18 ± 0.22	-	-	0.00
East West	0.00 ± 0.45	$< 0.60 \pm 0.35$	0.10 ± 0.50	0.20 ± 0.43	0.00	0.40 ± 0.47	0.30 ± 0.33
Elephant	0.46 ± 0.10	0.58 ± 0.12	$< 0.60 \pm 0.09$	0.50 ± 0.07	0.58 ± 0.13	$< 0.71 \pm 0.09$	$< 0.66 \pm 0.11$
Fox	0.15 ± 0.09	$> -0.06 \pm 0.08$	0.29 ± 0.13	0.25 ± 0.17	0.12 ± 0.11	0.10 ± 0.13	0.18 ± 0.11
Tiger	0.17 ± 0.14	0.23 ± 0.13	$< 0.47 \pm 0.15$	$< 0.44 \pm 0.14$	$< 0.58 \pm 0.11$	$< 0.66 \pm 0.10$	$< 0.52 \pm 0.11$

Table 3.2: Mean PSS, using constant/heuristic parameter choices for each learning method. Bounds, where variance occurs, represent 95% confidence intervals presuming a t-distribution. Inequalities, where present, indicate a statistical difference between CovAgg and the indicated method. The symbol '-' indicates aborted runs.

	mi-SVM*	MI-SVM*	MILES-LL*
Color	0.92 ± 0.01	$< 0.95 \pm 0.01$	$> 0.90 \pm 0.01$
Covariant 1	0.76 ± 0.02	0.78 ± 0.01	0.73 ± 0.02
Covariant 2	$> 0.20 \pm 0.02$	0.22 ± 0.03	$> 0.22 \pm 0.02$
Disjunctive	$< 0.65 \pm 0.02$	$< 0.35 \pm 0.06$	$< 0.56 \pm 0.02$

Table 3.3: Mean PSS, using SVM parameters chosen by grid search. Bounds represent 95% confidence intervals presuming a t-distribution. Inequalities, where present, indicate a statistical difference relative to CovAgg.

Overall, the SVM techniques compare in performance to ours on covariant problems, but only if tuned properly. Across the board, the simpler techniques (such as CovAgg and Optimal Ball) are less likely to completely degenerate, as do the SVM methods on the Mutagenesis data sets.

Further, Table 3.4 shows CPU user-space execution time. At least for these implementations, mi-SVM and MI-SVM are usually fast but are occasionally very slow at converging. In the case of Thioredoxin, the mi-SVM and MI-SVM runs failed to finish (cut off at over 24 CPU hours). Tuning SVM parameters requires additional time, unless this cost is amortized in prior analysis of a data set. (Note that we do not report running time for the tuned SVM results reported here.) Finally, we also find that DD, as reported by Zhang and Goldman (2001), can be very slow at times. It should also be noted that each training and testing of Weka-based implementations includes overhead Java VM startup time, which is probably most notable in the faster-running synthetic data sets.

3.4 Discussion

We have presented a method for multiple instance learning of covariant volumes that provides labels for indiviudal instances in a bag and uses a simple, parametric representation for the decision volume. Without performing problem-specific parameter tuning, our approach performs particularly well relative to other, standard

	CovAgg	OptBall	DD	EM-DD	mi-SVM	MI-SVM	MILES-LL
Color	0.47 ± 0.01	$< 1.18 \pm 0.02$	$< 4.23 \pm 0.16$	$< 1.34 \pm 0.02$	$< 6.04 \pm 1.14$	0.49 ± 0.02	$> 0.19 \pm 0.00$
Covariant 1	0.57 ± 0.02	$< 0.92 \pm 0.01$	$< 1.77 \pm 0.06$	$< 0.85 \pm 0.01$	$< 2.96 \pm 0.46$	$> 0.20 \pm 0.01$	$> 0.15 \pm 0.00$
Covariant 2	0.44 ± 0.01	$< 1.01 \pm 0.01$	$< 2.20 \pm 0.09$	$< 0.93 \pm 0.02$	$< 1.54 \pm 0.33$	$> 0.28 \pm 0.03$	$> 0.16 \pm 0.00$
Disjunctive	0.44 ± 0.01	$< 1.31 \pm 0.01$	$< 10.9 \pm 0.5$	$< 1.47 \pm 0.01$	$< 32.6 \pm 5.2$	$< 6.22 \pm 0.70$	$< 0.58 \pm 0.00$
Musk 1	10.2 ± 0.3	$> 2.25 \pm 0.07$	$< 16.6 \pm 4.0$	$< 22.5 \pm 5.4$	$> 0.56 \pm 0.15$	$> 0.43 \pm 0.08$	$> 0.32 \pm 0.02$
Musk 2	36.0 ± 5.3	$> 10.6 \pm 0.4$	$< 1358 \pm 527$	$< 265 \pm 111$	$< 4157 \pm 1244$	$< 1557 \pm 199$	$< 47.1 \pm 3.6$
Muta Atoms	0.75 ± 0.04	$< 2.29 \pm 0.05$	$< 7.31 \pm 0.73$	$< 1.96 \pm 0.07$	$< 9.74 \pm 1.77$	$< 5.52 \pm 0.72$	$< 1.33 \pm 0.05$
Muta Bonds	1.26 ± 0.08	$< 4.25 \pm 0.10$	$< 32.1 \pm 18.1$	$< 3.68 \pm 0.46$	$< 309 \pm 56$	$< 15.1 \pm 1.3$	$< 5.14 \pm 0.06$
Muta Chains	2.08 ± 0.14	$< 6.08 \pm 0.12$	$< 125 \pm 25$	$< 5.43 \pm 0.68$	$< 1775 \pm 321$	$< 27.4 \pm 5.4$	$< 9.30 \pm 0.17$
Thioredoxin	4.21 ± 0.49	$< 8.56 \pm 0.45$	$< 681 \pm 68$	$< 14.4 \pm 1.7$	-	-	$< 121 \pm 2$
East West	0.24 ± 0.01	$< 0.80 \pm 0.03$	$< 2.91 \pm 0.31$	$< 1.42 \pm 0.05$	$> 0.09 \pm 0.03$	$> 0.05 \pm 0.01$	$> 0.02 \pm 0.00$
Elephant	24.3 ± 2.9	$> 4.58 \pm 0.13$	$< 219 \pm 32$	$< 155 \pm 22$	$< 70.0 \pm 28.4$	$> 5.46 \pm 0.92$	$> 5.04 \pm 0.10$
Fox	16.8 ± 1.5	$> 4.27 \pm 0.04$	$< 126 \pm 17$	$< 92.0 \pm 34.3$	$< 86.8 \pm 26.8$	$> 7.44 \pm 1.95$	$> 3.50 \pm 0.02$
Tiger	19.3 ± 2.3	$> 4.59 \pm 0.16$	$< 72.9 \pm 14.5$	$< 70.4 \pm 20.2$	53.5 ± 26.5	$> 5.71 \pm 0.64$	$> 3.47 \pm 0.06$

Table 3.4: Mean training and prediction time in seconds, using constant/heuristic parameter choices for each learning method. Bounds, where variance occurs, represent 95% confidence intervals presuming a t-distribution. Inequalities, where present, indicate a statistical difference between CovAgg and the indicated method. The symbol '-' indicates aborted runs.

MIL approaches on problems in which the data exhibit covariant properties. Our approach also performs competitively on typical MIL data sets. As such, we believe our algorithm to be a viable and effective MIL approach, especially when robustness to covariance is required, and when parameter turning is not a practical option.

Going forward, we are interested in the application of MIL methods to learning in multi-attribute, relational settings (e.g., Blockeel and De Raedt 1998; Bodenhamer et al. 2009). In these contexts, the MIL classifier is applied repeatedly for each potential relational question, and must be efficient and robust in finding effective decision volumes. While algorithms such as MI-SVM could perform better for some specific problems, the additional time required to select problem-specific parameters and the potential complexity of the resulting decision volumes could yield these alternative approaches ineffective in the broader relational settings.

In future work, we expect to apply our method to non-Euclidean topologies, such as orientation in two or three dimensions. While we have emphasized *covariant* aggregation here, the technique is easily applicable to other forms of parametric distance.

Chapter 4

The Spatiotemporal Multidimensional Relational Framework

The world can be viewed in terms of objects and their relationships. In cluttered indoor environments, one object might be *on* or *beside* another. In a sports game, an opponent might be *between* two teammates. Key relationships are important to an agent when making decisions, and many potential relations need considered within the context of a task.

For its relational representation and learning capabilities, this work incorporates the Spatiotemporal Multidimensional Relational Framework (SMRF) of Bodenhamer (2014; see also Bodenhamer et al. 2009, Palmer et al. 2012). Primarily, this chapter serves as an overview of prior work by Bodenhamer. However, the covariant aggregation (CovAgg) multiple instance learning (MIL) algorithm, detailed in Chapter 3, forms a key component of the SMRF learning algorithm and is one of the primary contributions of this dissertation. I explain the role played by CovAgg later in this chapter.

4.1 SMRF Trees

SMRF expresses relational concepts in the form of decision trees. These trees assess the probability that a given collection (or *scene*) of objects contains an instance of a particular *target concept*. The objects have associated attributes, which can be either discrete or continuous scalar or multidimensional. Figure 4.1 shows the types of nodes that can appear in a SMRF tree:

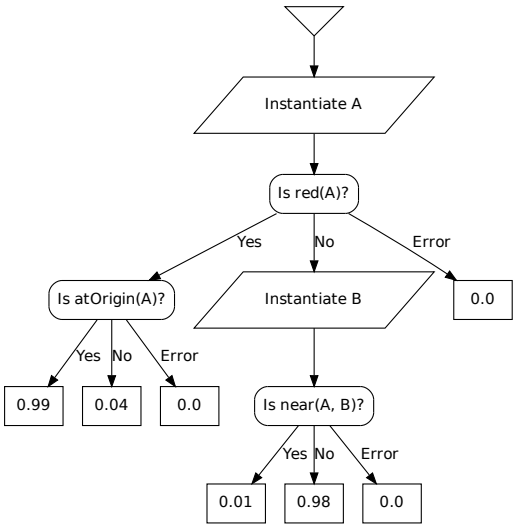


Figure 4.1: Hand-crafted SMRF tree encoding a disjunctive concept that identifies either a red object at the origin, or an object that is not red and has another object not near it. Predicates such as *red* and *near* are defined in terms of decision volumes in the RGB color and physical distance metric spaces, respectively.

- *Instantiation nodes*, shown as parallelograms, bind an object from a scene to a variable. An ordered sequence of such bindings is called an *instantiation sequence*, which is derived by a sequence of instantiation nodes down a particular branch of a tree.

- *Question nodes*, shown as rounded boxes, sort instantiation sequences down various branches of the tree, based on the attributes of the objects in that sequence. If a question is true for a particular instantiation sequence, it is sorted down the *Yes* branch, or if false then down the *No* branch. The *Error* branch is used for cases when an attribute is undefined or if not enough objects exist for instantiation. Different instantiation sequences from the same scene can be sorted down different branches.

- *Leaf nodes*, shown as rectangles, give the probability of membership in the target concept for instantiation sequences arriving there. The probability that a scene contains an instance of the target concept is the highest probability

assigned to any instantiation sequence from the scene.

In this way, SMRF trees resemble existentially quantified logical expressions. For example, the SMRF tree in Fig. 4.1, given some probability decision threshold, is equivalent to the following first-order logic expression:

$$\begin{aligned}\exists_A \big[(red(A) \quad &\wedge \quad atOrigin(A)) \vee \\ (\neg red(A) \quad &\wedge \quad \exists_B \left[\neg near(A, B) \wedge A \neq B \right]) \big].\end{aligned}$$

In SMRF, each object has attributes that can be multidimensional and continuous. Relations are implied as functions of attribute values. For example, objects may include a *position* attribute, as defined within some fixed coordinate frame. In addition, the position of one object may be described relative to that of another. Each object also has a distinct identity, and subsequent instantiations down a branch of a tree do not repeat objects earlier in the sequence (hence, $A \neq B$ in the example). Such sequential instantiations produce permutations of the objects. These are pruned, however, by the question nodes between them. Often, only a small subset of earlier instantiation sequences filter down to branches with additional instantiation nodes (Bodenhamer, 2014, Section 7.2.4).

The SMRF approach allows complex question node models to be defined in terms of the multidimensional attributes of the objects in the training data. While a decision surface in a multidimensional space can be represented in a variety of ways, we choose to define them in terms of three components:

- $\phi(I)$, a *mapping function*, which computes some quantity over a subset of the attributes of a subset of the objects in instantiation sequence I,
- $p(\bullet|\theta)$, a probability density function (pdf) defined over the codomain of the mapping function, and

- Θ, a probability density threshold, which determines the sorting at the question node.

A mapping function maps an instantiation sequence to a value in a metric space. It does this by selecting some number of objects out of an instantiation sequence, and performing a numeric operation on their attributes. For instance, a mapping function might compute the relative difference of the color attributes of the first and second objects in the sequence. Another mapping function might, as an identity function, simply return the value of the location attribute of the fourth object in the sequence.

For defining a decision boundary in Euclidean spaces, the Gaussian distribution is a convenient choice of pdf. Combined with a likelihood threshold, this defines an ellipsoidal volume in the space; points falling within this volume are considered as satisfying the question. For 2D orientation, we use a von Mises distribution (Mardia, 1975). The representative power of the SMRF approach is found in the different possible mapping functions that are available, and the ability to create an appropriate decision volume based on the training data. If, for example, the distance between two objects is a central aspect of the target concept, a "distance" mapping function allows this relationship to be expressed, and the pdf/threshold pair allows the appropriate distance between the objects to be modeled from the training data.

Formally, in the classification process, a question node computes $p(\phi(I)|\theta)$ for each instantiation sequence I sorted to that node. For each I, if $p(\phi(I)|\theta) \geq \Theta$, then I is sorted down the *Yes* branch. If $p(\phi(I)|\theta) < \Theta$, I is sorted down the *No* branch. If $\phi(I)$ is undefined for I (e.g., if insufficient objects exist for instantiation or if an attribute used by ϕ is absent), then I is sorted down the Error branch.

Algorithm 4.1 SMRF Learning Algorithm

Begin with a stub tree consisting of one leaf.
repeat
  Pick the leaf with best potential to improve L.
  for all n attempts each of all possible expansions **do**
    Create a new candidate tree with leaf expanded.
    Expand instantiations for new instantiation nodes.
    Calculate mapping values for all instances.
    Determine decision volume using L-aware CovAgg.
    Determine leaf node probabilities.
    Calculate p for likelihood ratio test.
  end for
  Select the candidate tree with the lowest p if $< \alpha_{corrected}$.
while the expansion was successful
Return the best tree accepted.

4.2 SMRF Learning Algorithm

The objective of the SMRF tree learning algorithm is to grow a tree that accurately predicts the label of novel object scenes and, in the process, distinguish which objects in each scene play an important role. Because scenes are probabilistically classified, the algorithm seeks to build a tree that maximizes likelihood over the training set. The likelihood (L) of a tree given the training data is defined as follows:

$$L = \prod_{S \in S^+} \Pr\left(\mathcal{W}(S)\right) \prod_{S \in S^-} \left(1 - \Pr\left(\mathcal{W}(S)\right)\right), \tag{4.1}$$

where S^+ and S^- denote the set of positive and negative scenes, respectively, labeled based on whether or not they contain an instance of the target concept. $\mathcal{W}(S)$ denotes the highest-probability leaf in the tree into which some instantiation sequence from scene S is sorted. $\Pr(\ell)$ denotes the probability value that leaf ℓ assigns to instantiation sequences that reach it. Of course, the learning algorithm operates in log likelihood space for numerical reasons.

The SMRF learning algorithm, overviewed in Algorithm 4.1, expands one leaf at

each iteration. Each expansion includes zero or more instantiation nodes followed by one question node. The set of mapping functions and rules governing their usage are configurable, in a somewhat similar fashion to the refinement specifications of TILDE (Blockeel and De Raedt, 1998). The best expansion is accepted if it passes a likelihood ratio test (Huelsenbeck and Crandall, 1997) with Šidák correction (called Bonferroni adjustment by Jensen and Cohen, 2000) for the repeated expansion attempts.

The construction of decision volumes (which consist of a pdf and likelihood threshold) in the SMRF learning algorithm differs from common relational learning methods such as TILDE in that SMRF directly addresses multidimensional continuous attributes. For a given expansion, the sampled mapping function ϕ under consideration (e.g., relative position), transforms tuples of objects from the current instantiation sequences into points in some metric space. Each positive or negative scene (from S^+ or S^-, respectively) contributes an unordered set of such points. As SMRF addresses existential questions, we wish to find the volume within this metric space containing at least one point from as many positive scenes and from as few negative scenes as possible. This is the common formulation of the multiple instance learning (MIL) problem (Dietterich et al., 1997). Because of our concern for potential covariance, we employ the covariant aggregation (CovAgg) learning algorithm as detailed in Chapter 3.

To use CovAgg for SMRF, we first need to (1) map the SMRF context into the CovAgg context and (2) alter the optimization metric used by CovAgg. To map SMRF into CovAgg, every instantiation sequence at a question node provides an instance point from its scene, which constitutes a MIL bag. For pragmatism, at the time a question volume is being learned, we entirely ignore other leaves in the tree. Error sequences also do not participate. We alter the metric used by CovAgg because of the probabilistic nature of SMRF classification. Specifically, we seek to

maximize L, as defined by Equation 4.1, rather than classification accuracy. As L is defined in terms of leaves, we presume the existence of virtual leaves during CovAgg learning. Instances within any particular volume are sorted to a virtual *Yes* leaf, and those outside to a virtual *No*. Then, the fraction of positive *scenes* represented inside the volume gives the probability for the virtual *Yes* leaf. Scenes entirely outside the volume determine the probability for the *No*. These probabilities allow calculation of L. Also, given a center and covariance for a decision volume, we can calculate $\log L$ for varying threshold distances efficiently in the same fashion as described for accuracy in Section 3.2.2. The adaptation of CovAgg to SMRF, in addition to the CovAgg learning algorithm itself, is a key contribution of this dissertation.

Once the decision volume is in place, the instantiation sequences are sorted accordingly, and the leaf node probabilities are assigned. Algorithm 4.2 gives an overview of how this is done. Because each scene is assigned a probability based on leaf with max probability across all instantiation sequences for the scene, each scene participates in assigning probabilities for only one leaf. We first iterate across each leaf, tentatively choosing a probability for each based on the fraction of positive scenes with instantation sequences present. There is no additional effect if a scene has multiple sequences present. The leaf with the highest tentative probability has the assignment retained. We then repeat the process for remaining leaves, assigning a probability to only one leaf at a time. At each outer iteration, all instantantiation sequences from all scenes represented in the assigned leaf are removed from consideration in remaining leaves.

Bodenhamer (2014) presents further details of the SMRF tree learning algorithm. Bodenhamer also empirically demonstrates SMRF's advantage over TILDE and several MIL algorithms when addressing multidimensional, continuous attributes or when large numbers of distractor objects exist in scenes.

Algorithm 4.2 SMRF Leaf Probability Assignment

Let $\mathcal{L}_R$ be the set of leaves that need probabilities assigned, initialized to all leaves.
Let $\mathcal{S}_R$ be the set of scenes unassigned to a winning leaf, initialized to all scenes.
while $\mathcal{L}_R \neq \varnothing$ **do**
    for all leaves in $\mathcal{L}_R$ **do**
        Assign a tentative probability to the leaf using only scenes from $\mathcal{S}_R$.
    end for
    Let ℓ_W be the leaf from $\mathcal{L}_R$ with highest tentative probability.
    Assign the tentative probability of ℓ_W as its final probability.
    Remove ℓ_W from $\mathcal{L}_R$.
    Remove scenes at ℓ_W from $\mathcal{S}_R$.
end while

4.3 Mapping Functions

The core SMRF framework depends on a set of mapping functions to access data about objects in a scene. While the precise set of function can be specific to a domain or task, some are common for basic physical properties. In particular, all tasks using SMRF in this work make use of the following functions or some subset of them:

- Identity Location (A), which returns the vector location or position of the center of A, denoted here as x_A,
- Distance Location (A, B), which returns the distance between A and B, $\|x_B - x_A\|$,
- Difference Location (A, B), which returns the relative location between A and B, $x_B - x_A$,
- Identity Color (A), which returns the RGB 3-vector color of A, denoted here as c_A,
- Difference Color (A, B), which returns the relative color difference between A and B, $c_B - c_A$,

- Identity Extent (A), which returns the distance in each dimension from the center of A to the edge, somewhat corresponding to a radius,
- Identity Extent Ratio 2D (A), which is only defined for 2D space and which returns the ratio of of the extent of dimension 2 of A to that of dimension 1,
- Reframe 2D Unscaled Location (A, B, C), which returns the location of C in a coordinate frame where A is the origin and the x axis points toward B and which ensures against reflection in the reframe,
- Reframe 2D Scaled Location (A, B, C), which also reframes without reflection and additionally scales all dimensions equally in the new coordinate frame such that B is one unit from the origin, and
- Identity Orientation (A), which is also only defined for 2D space and which returns the angle of rotation of A.

The final four mapping functions listed here are tailored to 2D space, although variations for higher dimensionality could be defined.

Chapter 5

Learning Affordances by One-Step Action Outcomes

5.1 Introduction and Related Work

The SMRF framework, discussed in Chapter 4, provides a foundation on which to learn relational concepts about continuous, multidimensional worlds. In this chapter (based on Palmer et al., 2012), I use such relational learning for predicting the success or failure of agent actions, as this ability can form the component of a higher-level planning or learning system. As an example of such a planner, Stulp et al. (2012) learn position boundaries that predict success or failure of actions such as robotic grasping of objects on a table. Identification of probable action success can also be viewed as a form of affordance (Gibson, 1977). Stulp et al., however, do not address the learning of relations for arbitrary numbers of objects, except in certain *ad hoc* fashions. Many other recent works also address learning consequences of agent actions in continuous, multidimensional domains, often predicting detailed world state (e.g., Mugan and Kuipers, 2009; Modayil and Kuipers, 2008; Brunskill et al., 2009). However, such works commonly avoid arbitrary object relationships.

For addressing relational concerns, logical formulations are a common and traditional technique. Such representations are frequently purely discrete or, perhaps, include one-dimensional continuous attributes. Learning in such domains is also well established (Shen, 1993; Pasula et al., 2007; Wu and Givan, 2010). This includes the general field of relational reinforcement learning (RRL) which adapts reinforcement learning (RL) techniques to such relational, logical domains and representations (van

Otterlo, 2005). However, when applied to multidimensional continuous attributes, many of these approaches depend on hand-crafted predicates. For example, a predicate for $above(A, B)$ might help in predicting the outcome of agent actions such as $pour(A, B)$. This requires manual effort and may fail to consider the specific requirements of such concepts in a highly detailed and unstructured world. Among relational learners supporting continuous attributes, the TILDE algorithm of Blockeel and De Raedt (1998) makes use of entropy-based discretization. This has been employed recently for feature learning in RL (Kersting and Driessens, 2008) and imitation learning (Natarajan et al., 2011) contexts, but only for one-dimensional domains or for one-dimensional attributes (e.g., angles and distances).

Some approaches combine relational and multidimensional, continuous learning of world dynamics. These include Verbancsics and Stanley (2010), who evolve neural networks capable of learning physical relationships given a grid-based world representation of somewhat flexible resolution. With this representation, among other test domains, they learn to play a 2D simulated soccer keepaway game (Stone et al., 2006), scaling to more players in test than in training, even before additional learning. Another example of combined continuous and relational dynamics learning is that of Xu and Laird (2011), who learn separate continuous and relational models using a form of case-based reasoning and allow interaction between the two models using human-defined predicates.

In our work, we use the spatiotemporal multidimensional relational framework (SMRF) of Bodenhamer (2014) to predict the probability of binary outcomes of world dynamics, including those of parameterized actions. SMRF includes the ability to explicitly reason about specific object instances across multiple question nodes of a tree. This is not common to all multidimensional, continuous relational decision tree methods (e.g., Neville et al., 2003; McGovern et al., 2008), although conjunctive question nodes in the enhanced spatiotemporal relational probability tree (SRPT)

method of McGovern et al. (2014) allow asking a pair of questions about an object or set of objects. Like Džeroski et al. (2001), we use explicit object instance references to focus attention on action parameters. In the $pour(A, B)$ example, A and B identify certain participant objects in each use (or potential use) of this action.

Another key aspect of our work is that the SMRF learning algorithm is stochastic. Repeated training runs produce a variety of trees, and to further promote this, we use random subsampling of training data for each learned tree. Such variety can increase generalization accuracy (Breiman, 2001), although, unlike Breiman, we do not force variety by limiting the types of questions that a tree can ask. Given an ensemble (or *forest*) of SMRF trees, we apply the forest to new scenes to predict binary outcomes. In this way, the trees form a basis for representing the world and its dynamics.

In the remainder of this chapter, I describe the learning method in more detail, including differences in the earlier version of the SMRF learning algorithm, as applied in this work. I then show results for three different problems in 2D simulated physical worlds.

5.2 Method

Real-world physical relations are originally implicit in continuous, multidimensional data, and predicting action outcomes depends on detailed issues of world dynamics. We present a method for predicting binary outcomes of actions in such continuous, relational worlds. The core of our approach is based on Spatiotemporal Multidimensional Relational Framework (SMRF) trees (Bodenhamer et al., 2009; Bodenhamer, 2014), and we begin with an overview of this algorithm as it applied to the experiments in this chapter. We then address the use of action parameters within the SMRF framework and how a forest of SMRF trees forms a basis to learn outcome

prediction in new situations.

5.2.1 SMRF Learning

The work in this chapter (see Palmer et al., 2012) predates the version of the SMRF learning algorithm presented in Chapter 4 and differs primarily in that it uses a different method for constructing decision volumes than the covariant aggregation algorithm described in Chapter 3. Here, to learn volumes, we initially rank sampled volume centers using the well-known diverse density metric of Maron and Lozano-Pérez (1997):

$$\left(\prod_{S\in S^+} \max_{I\in\mathcal{I}^S} \Pr\left(\phi(I)\right)\right) \times \left(\prod_{S\in S^-} \left(1 - \max_{I\in\mathcal{I}^S} \Pr\left(\phi(I)\right)\right)\right),$$

where $\mathcal{I}^S$ denotes all instantiation sequences from scene S and $\Pr\left(\phi(I)\right)$ refers to the probability volume center and shape under consideration. Volume covariance still is initialized based on the covariance of points from all positive scenes. All pdf parameters are then refined iteratively in a fashion inspired by expectation maximization (EM; Dempster et al., 1977), using only one point at maximum pdf density from each positive scene. Following from Friedman (1977), we determine the pdf threshold Θ by the point that maximizes the Kolmorogov-Smirnov (KS) distance of positive and negative scenes, again counting only one point per scene.

5.2.2 Parameterized Actions

We apply SMRF to the learning of outcomes for parameterized actions. When learning the outcomes of such actions, we include in each training scene, not only the world state, but also the arguments of the action. For the $pour(A, B)$ example, training samples include the identification of A (the thing being poured from) and B (the thing being poured into).

To use this information when learning SMRF trees for a k-ary action, we constrain the objects selected for the first k instantiation nodes of any branch in the tree, in a fashion similar to the action argument binding of Džeroski et al. (2001). When predicting the outcomes of the same action in future situations, we again bind the action arguments to the first k instantiation nodes. Any additional instantiation nodes produce permutations of other objects in the scene per standard SMRF evaluation. For example, if the SMRF tree in Figure 4.1 were for some action $act(A)$, A would only be bound to the action argument, but all remaining objects could be bound to B.

5.2.3 Forest-Based Classification

Because question node expansions are sampled, tree growth is a stochastic process. Therefore, while each SMRF tree yields probabilities from a fixed set of leaves, multiple SMRF trees together, all learned for the same classification task, provide a richer representation. Ensemble techniques (aggregating individual classifiers) have also been shown to improve accuracy when applied to new data (Breiman, 2001).

The existential nature of SMRF concepts is another reason to use multiple trees. Sometimes absence, rather than existence, is what determines action success. For example, passing a ball in soccer is likely to be successful if *no* opponent is between the passer and receiver. Such *negative existential* concepts depend on *universal* quantification. For instance, $\neg\exists_X t(X)$ is logically equivalent to $\forall_X \neg t(X)$. Because SMRF currently has no support for universal quantification, we instead learn trees for both the original and negated labels.

Therefore, to predict world outcomes, we learn a set of n trees, each using a subset of data sampled from an original training set. Half of the trees are learned using the original labels and half using negated labels. Each tree provides a predicted probability of outcome for a scene. We can thereby transform any domain

state of arbitrary dimensionality (referring to objects and their attributes) into a new abstract state $s \in [0,1]^n$. An aggregate classifier can then be learned on new scenes converted into n-dimensional vectors. Specifically, we use support vector machine (SVM) classification because it is effective and has readily available implementations, such as LIBSVM (Chang and Lin, 2011). For the present work, we use only discrete classification from LIBSVM, although it and other techniques are also able to provide probability estimates, a capability that might be valuable for more advanced applications. Further, as shown to be effective by Breiman (2001), simple aggregate voting could also be employed. However, again, SVM implementations are readily available, and SVM learning is able to select relative weighting among available trees, according to the data.

5.3 Experimental Results

In this section, we present results for three problems: two in a 2D simulated physical blocks world and one in 2D simulated soccer. SMRF has been applied to concept learning in 3D domains as well (Bodenhamer et al., 2009; Bodenhamer, 2014). In all experiments, we provide the mapping functions to SMRF that are previously identified in Section 4.3: absolute and relative position, distance, reframed position (translated and rotated according to the location of two other objects), scaled reframed position (scaling the coordinate frame such that one unit is between the reference objects), absolute orientation, size, elongation, and individual and relative color. Some of these fit more naturally to some tasks than others, but are included in all cases to demonstrate that the algorithm is able to choose appropriately.

All data sets consist of between 560 and 1000 total scenes, depending on the task, and we subsample 500 scenes without replacement for all SMRF and SVM training sets. For each task, we learn 50 SMRF trees with original labeling and 50 with

negative labeling. This gives us a core set of learned representations. To observe variance resulting from the set of trees learned, we then subsample 25 positive and 25 negative trees, and perform 10-fold cross validation with LIBSVM using a second data set. We perform SMRF tree subsampling and SVM cross-validation 100 times. For the SVM, we use a linear kernel. Within each training set, we choose the SVM cost parameter C by means of internal 10-fold cross-validation, evaluating options for C by powers of 10. Our performance measure, which also drives the choice of C, is the Peirce Skill Score (PSS) (Stephenson, 2000), as described in Section 3.3.3. We base each PSS value on aggregate results across the 10 test folds.

5.3.1 Blocks World

We first demonstrate our method on two tasks in a simulated blocks world built using the JBox2D physics engine (Murphy, 2010). This world is governed by a hidden gravity vector. For both tasks, no agent actions occur after each initial state. Only gravity and collisions affect the outcome.

The first of these two experiments is inspired by Siegler's classic developmental cognitive psychology work on human learning of rules for predicting the outcome of placing a set of weights on a balance scale (Siegler, 1976). The dynamics question is whether the spatially distributed weights will cause the scale to tip left, tip right, or remain balanced. This task focuses on an unstable world state rather than a parameterized agent action, although similar active tasks could be imagined. Also, while SMRF's capabilities are different than Siegler's proposed human rules, this task still provides an interesting case for comparison between human developmental learning of world dynamics and that of our method.

Here, we simplify the problem to whether the scale tips left or right, and do not consider balanced cases. For classification, we arbitrarily label tipping left as true, effectively defining a *tipsLeft* predicate, which is to be learned. An example

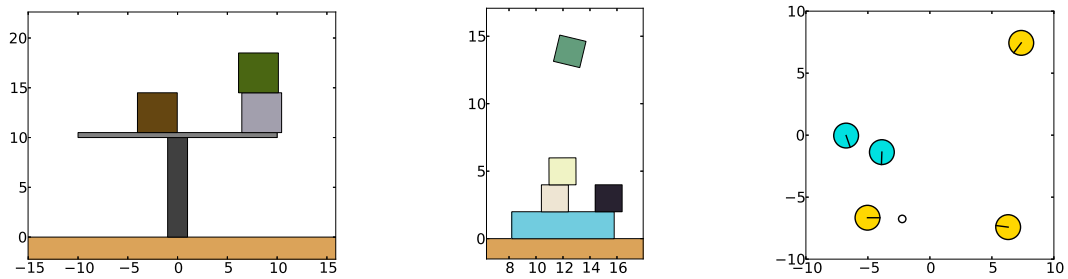


Figure 5.1: Example starting states for the three tasks evaluated. The balance scale (a) tips either left or right, where we arbitrarily label *tipsLeft* as true. The $dropOn(A, B)$ action (b) is successful if dropped block A comes to rest anywhere above B, here the elongated horizontal (cyan) block. The keepaway $pass(A, B)$ action (c) is successful if the "keepers" (yellow) retain possession of the ball (white) after the pass from A (here, lower left) to B (lower right).

initial state is shown in Fig. 5.1(a). We construct each initial scene with one to four randomly placed weights of equal size and mass along the horizontal beam. Weights that would otherwise overlap are stacked upward, approximately centered above the base of the stack. The constant-sized balance scale is always at the same location, and, while it clearly affects world dynamics, we leave it out of the scene description because it is constant. The absolute positions of weights directly indicate whether they are to the left or the right side.

To predict tipping left or right, learned SMRF trees commonly focus on the positions of the weights. For example, weights to the left suggest tipping left. Absolute position is indeed the most common mapping function in learned trees for this task. However, such weights cannot always be considered in isolation. With a small number of weights present, SMRF uses error branches, where failures to instantiate imply that there must be no counterweight on the opposite side. Relative mapping functions, especially distance, also appear in the trees. With more weights, high vertical positions imply large stacks, a heuristic perhaps usable by humans in approximate counting. Such issues are somewhat visible in the "heat map" shown in

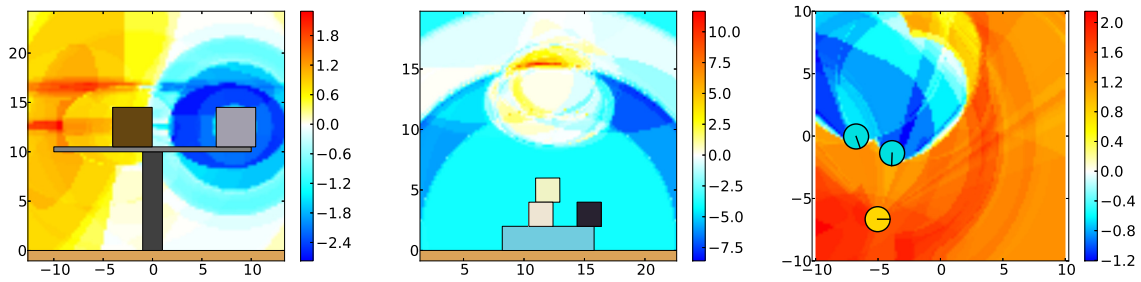


Figure 5.2: SVM decision values showing sample evaluations for the three experimental tasks, where value 0 is the decision boundary. For the balance scale (a), the value is for the scale tipping left rather than right, if a new block is added centering on varying locations. For $dropOn(A, B)$ (b), the value is for different drop locations of A. For $pass(A, B)$ (c), the value is for attempting to pass from the yellow player A to a receiver B at varying locations, where opponents are shown in blue.

Fig. 5.2(a). Also, note that, as all training scenes involve blocks properly stacked on the scale, no strict distinctions have been learned for areas outside such conditions. The narrow horizontal bands are indicative of the constant size of the weights. Only certain vertical positions appear in the data. Overall, the learned trees are effective for the learning scenario, and our method performs nearly perfectly in this task.

Our second blocks world task is for an agent action, $dropOn(A, B)$, where block A is dropped on support block B. This action is considered successful if the dropped block indeed comes to rest over the support, even if other blocks are stacked between them. An example initial state is shown in Fig. 5.1(b). We choose a uniformly randomly distributed length and horizontal position for the support block, within certain bounds, as well as its orientation. We place the block to be dropped with normally distributed horizontal and uniformly distributed vertical position relative to the top of the support. The block to be dropped is a square of constant size. Zero to three potentially interfering blocks are dropped according to the same distribution in advance of the block drop action to be classified.

In this experiment, relative position is crucial. Learned trees often ask whether

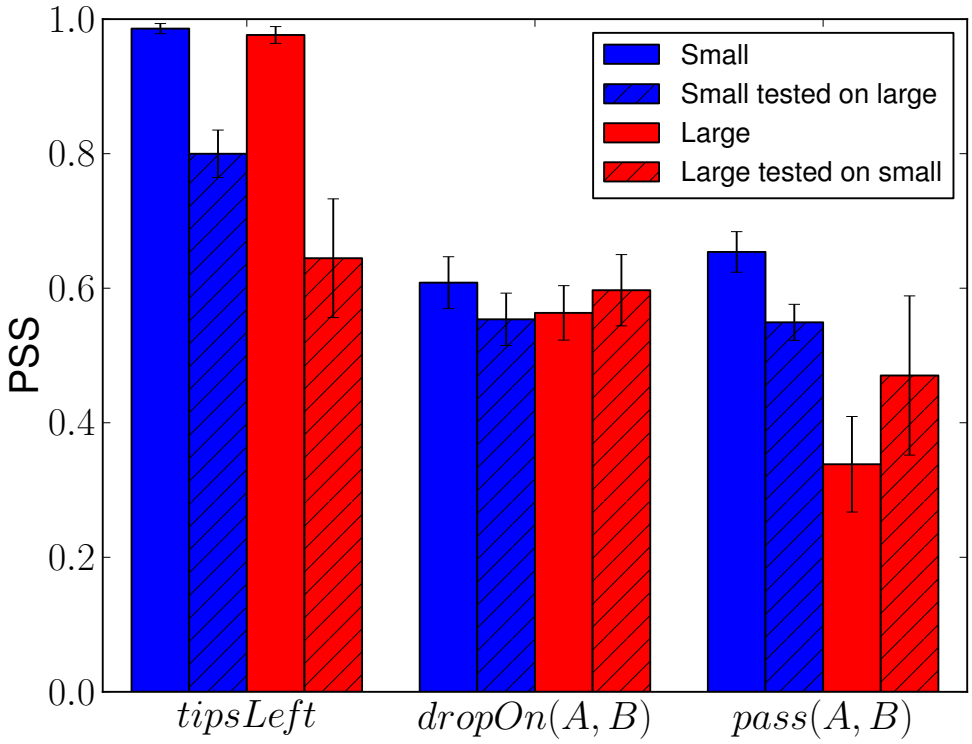


Figure 5.3: Mean and standard deviation of classification performance, including cross testing for scenes with different numbers of objects than occurred during training. Small scenes contain fewer objects, and large scenes contain more.

the dropped block is above the support block. Questions often follow about the elongation and orientation of the support. For branches identifying a narrow support block, questions constraining the dropped block's relative horizontal variance sometimes appear. Other structural forms of trees also occur, but often with similar kinds of questions. Rarely do trees instantiate additional interfering blocks. Such blocks seem to help as much as hinder, and seem more just to provide noise to the system. In Fig. 5.2(b), such noisy effects are visible. While the highest decision values are in a narrow horizontal band, areas predicting positive outcomes are scattered in a large region above the support block.

Both block problems generalize to higher numbers of objects in test scenes than in training data ($p \approx 0$ by one-sample t-tests), as shown in Fig. 5.3. In the balance scale case, we place five to six blocks for "large" scenes. Counting via error branches

does not scale to larger scenes, but classification scores remain strong, implying that positional questions are still helpful. For the drop action, large scenes contain four to five interfering blocks, and even for trees learned in the larger scenes, such interfering blocks are rarely instantiated.

5.3.2 Soccer Domain

Our second experimental domain is that of the RoboCup 2D soccer simulator, which at this time we use only for the keepaway benchmark task (Stone et al., 2006). While we do not attempt to scale to full soccer here, subtasks such as this have an eye toward learning some of the representations needed for the full game or other similar activities.

The keepaway game consists of two teams: keepers and takers. The keepers begin in possession of the ball, and an episode ends when the takers take possession or the ball goes out of bounds. The common configuration is 3 vs. 2, meaning three keepers and two takers, on a 20m by 20m playing area, as shown in the example state in Fig. 5.1(c). At each decision step, the keeper in control of the ball chooses either to hold the ball or to pass to a chosen teammate. All other players, including the takers, follow manually scripted behavior, as provided by Stone et al. (2006). We represent the pass action as $pass(A, B)$ where A is the player with the ball and B is the intended recipient of the pass. For the purposes of producing training experience, we select the random policy for the keeper with the ball, meaning a uniformly random choice among all available pass and hold actions. The objects in the world are the players themselves. In this work, we do not include the ball itself in the scene description, although the *player* with the ball is always identified by parameter A.

In keepaway, there is no inherent up or down, unlike the situation with gravity in the blocks world. On the other hand, opponents between passer and receiver are

important, and the scaled reframed position mapping function, placing the passer at the origin and the receiver at (1, 0), commonly appears in learned trees. In this coordinate frame, potential interceptors appear near the x axis between coordinates 0 and 1. Exact boundaries depend on world dynamics, including agent behavior. Because interceptors cause failure rather than success, identifying them depends on negative labeling. Although rare, some learned trees consider multiple interceptors down the same branch. Fig. 5.2(c) shows inherent existential queries of each interceptor. Interestingly, with large SMRF training sets (such as the 500 scenes used here), we also see many learned trees for positive labeling, focusing initially on the position of the passer or the relative position of the receiver.

For keepaway, "large" scenes are based on 4 vs. 3 play on a 25m by 25m field. As for the blocks world, we again transfer somewhat to the larger game without retraining ($p \approx 0$). We show performance in Fig. 5.3. Perhaps surprisingly, prediction is actually *better* on 4 vs. 3 play when transfering from 3 vs. 2 than when learning directly in the larger game ($p \approx 0$ by two-sample t-test). This seems to be due to a frequent inability of the algorithm to learn any tree expansions in 4 vs. 3 beyond the initial tree with one leaf. Such a tree provides only an unconditioned probability and is therefore uninformative about world state. Because so many trees learned in 4 vs. 3 are uninformative, a larger sampling of trees is necessary to ensure that at least some informative trees are included. Fig. 5.4 shows this effect, including the wide variance depending on the quantity of informative trees sampled. Note, however, that in 3 vs. 2 keepaway, PSS saturates quickly with relatively small forests. Other learning contexts evaluated in this chapter have similarly quick saturation.

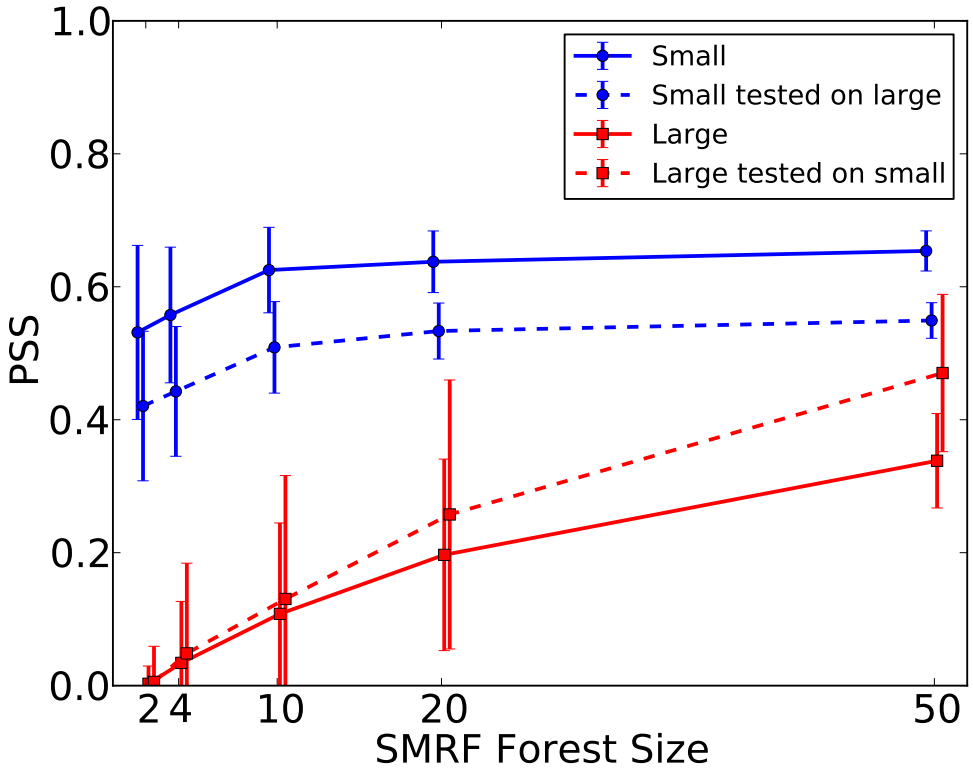


Figure 5.4: Mean and standard deviation of performance as a function of forest size (number of trees) for the $pass(A, B)$ action. Half of the trees in each forest are for positive labeling and half for negative (i.e., varying from 1×2 through 25×2).

5.4 Discussion and Conclusion

To function in an unstructured world, agents must understand the consequences of their actions. In this chapter, we have demonstrated a robust method for action prediction on continuous tasks with arbitrary numbers of objects and implicit relations between them. This method learns trees using a technique similar to existentially quantified logic but which supports stochastic outcomes and also avoids the need for hand-crafted predicates. We employ a forest of trees to transform an object-attribute state representation of arbitrary dimensionality into a robust, fixed-dimension space usable in standard vector-based machine learning algorithms. We have successfully applied our method, using a fixed set of mapping functions,

to simulated domains with substantially different dynamics. We have also shown scalability, without retraining, to scenes with both more and fewer objects than are present in the training data.

Returning to Siegler's work on human prediction of balance scale outcomes (Siegler, 1976), SMRF's tree expansion process is somewhat reminiscent of Siegler's progression to more advanced rules. However, Siegler suggests that humans use state features outside the kinds of mapping functions that we can currently apply in SMRF. For example, his rules include direct counting of blocks on each side of the scale. Both for group attributes, such as quantity, and for the many other varieties of potentially useful mapping functions, the number of compared tree expansions could increase substantially. We use a wide variety of mapping functions here, to good effect, but each carries a cost in the form of the multiple comparisons correction needed for tests of the significance of learned tree expansions. Learning time also increases with more mapping functions. Therefore, to increase flexibility, additional sampling and pruning techniques for mapping functions might be valuable.

Finally, going forward in the next chapter, we extend this framework for use in actual agent control.

Chapter 6

SMRF-Based Representation Policy Iteration

6.1 Introduction

In the preceding chapter, I present a mechanism for learning to predict success or failure of individual actions in relational domains. Here, I present a mechanism using SMRF to learn relational representations for agent decision making in multistep reinforcement learning (RL) tasks. In this work, I approach RL from a model-free, Q-learning perspective. To support arbitrary reward structures in continuous spaces, the Q function must be approximated via some kind of regression mechanism. However, SMRF learning algorithms currently incorporate only binary classification. Here, I propose the use of a forest of SMRF trees to provide a fixed-size feature space for Q function regression, not unlike the use of SMRF forests in the previous chapter. For learning policies, I use the least squares policy iteration (LSPI) method of Lagoudakis and Parr (2003), which includes the least squares temporal difference Q (LSTDQ) learning algorithm for Q function regression, as detailed in Chapter 2. Finally, I employ an instantation of the representation policy iteration (RPI) framework of Mahadevan (2005b), to interleave representation and policy learning. Because I use SMRF as the representation learning algorithm, I term the overall method SMRF-RPI.

The high-level SMRF-RPI technique I present here, in addition to using SMRF as its relational feature learner, also differs from existing relational RL (RRL) feature-learning techniques in at least four ways: (1) I employ multiple rounds of represen-

tation and policy learning between iterations of experience gathering, (2) I employ a *wrapper* feature selection technique for the stochastically learned trees, (3) I extract continuous, radial features from SMRF question nodes for use in the state-action feature vector, and (4) I present a mechanism for continuous actions.

On repeated learning, Parr et al. (2007) learn multiple basis functions in RPI fashion on a single batch of experience, but this is outside the relational setting. In contrast, Wu and Givan (2010) generate a new batch of experience for each relational feature learned.

For feature selection, various well-known techniques exist (Guyon and Elisseeff, 2003), and techniques safeguarding against overfit are used in RRL, such as standard tree pruning or depth limitations in TILDE (Blockeel and De Raedt, 1998). However, with the stochastic nature of SMRF tree learning and the noisy labels of stochastic domains, SMRF-RPI opts for learning multiple possible trees and then selecting from them. Concerning the feature space itself, the relational regression tree algorithm ReMauve (Vens et al., 2007) is at least one example of extracting continuous features for regression at leaf nodes. Specifically, ReMauve uses linear regression on the attributes used in query nodes, but it does not address multi-dimensional continuous attributes in the same way as SMRF. As for ReMauve, SMRF-RPI uses continuous features to be able to fit real-valued functions with less overall model complexity.

Turning to the action space, continuous actions have seen some attention in RL (e.g., Smart and Kaelbling, 2000; van Hasselt and Wiering, 2007; van Hasselt, 2012) but are less explored for the relational setting. Zaragoza and Morales (2009) interpolate continuous actions for RRL from discrete actions, and while Kulick et al. (2013) include a continuous action, it is not used for RRL. In contrast, SMRF-RPI evaluates and selects continuous actions directly as part of the RRL process.

Going forward in this chapter, I explain in detail the SMRF-RPI method, in-

cluding further explanation of the key points of distinction from other techniques.

6.2 Action Types

Fundamental to SMRF-RPI is the action-oriented view it makes of world state. As in Section 5.2.2, SMRF-RPI binds action arguments to the initial instantiation nodes of a SMRF tree. However, different types of actions might have different numbers of parameters and different meanings for them. For example, in soccer, a player might choose either to hold a ball or to pass it to another player. Passing a ball includes the additional information of which player is the intended recipient. In TILDE or similar techniques, actions are part of the asserted propositions for a scene. SMRF-RPI, however, represents each action type as a different set of trees, so as to promote the significance of the distinction and to guide the learning process.

SMRF-RPI also makes a slight modification to the tree learning process used in Chapter 5, with respect to action parameters. Ordinarily, SMRF requires asking immediate questions in connection with every new instantiation node in a branch. For example, there is no value in adding a new instantiation node for variable B unless the added question also immediately addresses the new variable. The question *might* also address variables added in previous expansions (e.g., variable A), but the new question is *required* to reference the newly added instantiation node. However, SMRF-RPI dismisses this requirement for instantiation nodes that represent action parameters. It does this because objects instantiated at these nodes have already been identified. For example, when placing one block on another, perhaps the block being moved is not as interesting as other blocks near the destination, depending on the task at hand. Therefore, it might make sense to ask questions about the other blocks first.

Algorithm 6.1 SMRF-RPI

```
repeat ▷ Experience loop
    Gather test experience, as desired.
    Gain training experience.
    repeat ▷ RPI loop
        for all action types do
            Apply binary labels by Bellman error.
            Learn n SMRF trees.
            Learn n SMRF trees with negated labels.
            for all learned trees do
                Apply LSPI for each tree with old representation.
                Calculate F-test p for scaled Bellman error of each.
            end for
            Select the tree with lowest p, if p < α_corrected.
            Otherwise, no trees are selected.
        end for
        Learn a new policy using LSPI with all selected trees.
    until no new trees have been selected
until converged, or as desired
```

6.3 Learning Algorithm

The goal of SMRF-RPI is to learn a set of trees for each action type, and weights for features extracted from the trees to yield an agent policy. Algorithm 6.1 presents an overview of the SMRF-RPI algorithm.

At each iteration of the outer "experience" loop, SMRF-RPI first gathers experience. Training experience commonly begins with random actions, usually either uniformly selected from all possibilities, or first by a multinomial distribution to weight certain action types over others. Training experience could, however, also include expert demonstration for more guided learning (Schaal, 1996; Argall et al., 2009). After the first iteration, training experience is guided more by the agent's own policy, although with continued exploration, such as by the simple ϵ-greedy method (Sutton and Barto, 1998).

Following collection of experience, SMRF-RPI repeats a cycle of representation

and policy learning (the "RPI loop"), until no new representations are accepted. I define no formal criterion for ending the experience loop, although a convergence test could be employed, or one could simply repeat the loop a fixed number of times.

The representation learning phase begins once training experience has been gathered. Depending on the configuration, each learning phase might include examples from only the most recent experience or perhaps all iterations or some subset. Experience contributes a set of (s, a, r, s', D) SMDP examples, as discussed in Chapter 2. To learn representations, the experience is divided by action type, and a set of candidate trees is learned for each action type. Because SMRF is a binary classifier, to learn a SMRF tree, we must first provide a binary label for each example. In Chapter 5, I do this based on immediate success or failure of the action. In generalizing to full reinforcement learning, I use the common practice of labeling each SMDP sample by Bellman error (BE):

$$BE = r + \gamma^D Q_\pi(s', a') - Q_\pi(s, a).$$

For the present work, to learn SMRF trees correlated with BE, I label based on whether an example is above or below the mean BE for that action type. Once binary labels have been assigned to the training data, SMRF-RPI learns a set of trees with both that labeling and negated versions of the labels, just as in Chapter 5. The latter allows the representation of negative existential concepts, which is equivalent to universal quantification. Also, note that SMRF itself may fail to find a meaningful expansion and output merely a "stub" without question nodes. Because these stub trees do not make any distinctions, SMRF-RPI discards them, and therefore the total number of candidate trees learned for an action type might be less than the number attempted, including none at all.

Once a set of candidate trees has been learned for an action type, SMRF-RPI

then selects at most a single tree to aggregate into the representation for that action type. It is possible, and eventually likely, that none of the trees is selected. Among other matters, a useful tree is both linearly independent from existing features and also effective at reducing overall BE. Because the tree learning phase ignores exact BE values, the tree selection phase addresses fit at a more precise level by attempting to learn a new policy including each candidate tree individually, one at a time. In feature selection literature, this is known as a *wrapper* method.[1]

Before testing each tree, SMRF-RPI first calculates the sum squared BE for a policy learned using the current representation. (For the first round of learning, no previous features exist, and the policy is learned using only one bias feature for each action type.) Then, each candidate tree is tested by aggregating it alone into the current representation, learning a new policy, and calculating the new sum squared BE. Because different features might result in substantially different magnitudes for the Q values themselves (especially early in learning when representation is lacking), I also scale BE by the standard deviation of the Q values. Finally, an F-test is performed for each new aggregate representation, adjusting for the number of degrees of freedom in the tree and the total number of trees attempted. The tree with the lowest p-value is selected if it falls below a corrected α threshold.

Policy learning itself is simply an application of the Least Squares Policy Iteration (LSPI) method of Lagoudakis and Parr (2003), as discussed in Chapter 2. Optionally, diagonal regularization can be applied to condition the A matrix. As mentioned above, policy learning, using LSPI, forms part of the current feature selection method. LSPI is also used for learning the final policy of the iteration, using all selected trees (at most one for each action type). See Figure 6.1 for a simple

[1] Alternative *filter* methods could be employed, such as by using feature correlation with labels or various successful information theoretic techniques (Brown et al., 2012). Mahadevan (2007) also suggests basis adaptation even within the LSPI loop itself, such as by discarding individual features with low weights. Of course, standard dimensionality reduction techniques such as principle component analysis (PCA) could also be used.

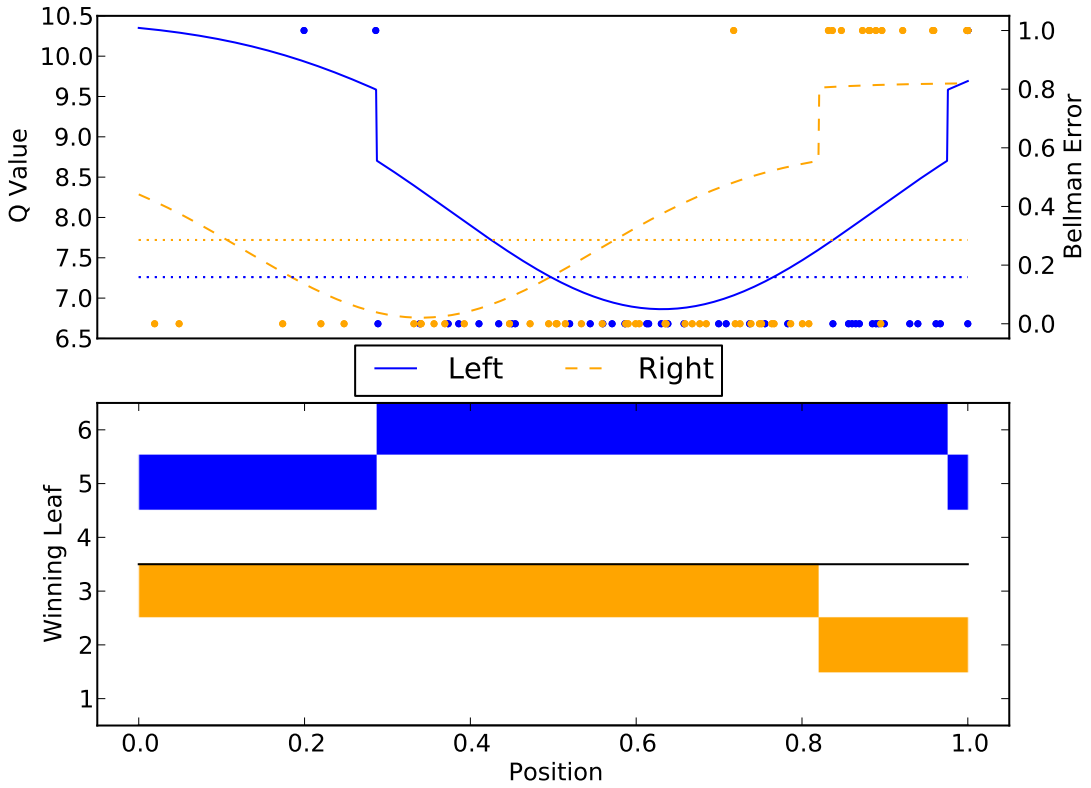


Figure 6.1: SMRF-RPI policy after one iteration of learning for an example 1D corridor task, where the agent is rewarded for reaching a 0.1 unit zone at either edge. The curves at top show Q values for each action. The policy is given by the action with maximum Q for each horizontal position. Individual points show the Bellman error (BE) for each training example. Dotted lines show mean BE for each action. The bars at bottom show regions for which different leaves are active.

example of a policy learned after one iteration.

As described in Algorithm 6.1, the subloop of representation and policy learning continues so long as new SMRF trees are learned and accepted for any action type. The motivation for this is that, if the training experience represents a large fraction of relevant state-action space, multiple iterations of RPI learning might benefit the learning process, even without new experience. For example, if an agent is rewarded for reaching some point in a maze, at first the only BE indicates when the goal is reached. After this, a learned Q function has higher values also for the states and actions that *lead* to the goal. In cases where learning is limited to the most recent

batch of experience, it is vital that later agent experience continue to fill relevant areas of the state-action space, and this is dependent on learning approximately useful policies at each iteration.

After all learning ends for an iteration of the outer loop, it repeats, first gathering new experience again. If a valuable policy *has* been learned, it may explore new areas that present new opportunities for learning. Over multiple iterations, it is expected that the number of new trees learned and/or the complexity of the learned trees will be reduced.

6.4 Features from SMRF Trees

The overall SMRF-RPI framework is independent of the actual features extracted from the SMRF trees themselves. In Chapter 5, I use SMRF probability predictions as features, yielding one real-valued feature for each tree. This might be appropriate in some cases, including for prediction of binary success or failure. On the other hand, two leaves in the same tree with similar probabilities might represent very different configurations. For SMRF-RPI, I instead identify each leaf as its own binary feature. One tree then contributes a vector of mutually exclusive binary values.[2] A forest effectively provides a set of overlapping tilings reminiscent of CMAC representations (Sutton and Barto, 1998).

However, unlike CMAC, SMRF-RPI begins with no tilings defined and instead needs to *learn* bins for every region of interest. As a result, bootstrapping representation could be inefficient. Therefore, I also propose a set of nonlinear, continuous features extracted automatically from relevant state features. Specifically, SMRF question nodes reference exactly those continuous state variables found to be relevant in predicting binary labels. However, rather than using the raw attributes

[2]This feature set is strictly richer than the probabilities used in Chapter 5, because feature weights corresponding to leaf probabilities would yield the probability-based feature space.

(for example, relative location of two objects), I instead use the probability density from the pdf model at the question node. In Euclidean space, as detailed in Chapter 3, this is a Gaussian distribution with arbitrary covariance defined by the key points. By directly using pdf densities from question nodes, SMRF-RPI incorporates a form of RBF representation in addition to the CMAC bins of the leaves. Figure 6.1 demonstrates the effects of these nonlinear features on both the learned Q values and policy.

This leads us to the same question addressed by Vens et al. (2007): If multiple instantiation sequences from a scene define the label for the scene, which one should be used? In the case of soccer, for example, we might have multiple opponents within a decision volume relevant to the decision of whether or not we should pass the ball that direction. Which opponent or opponents should provide the continuous features? LSPI and other similar approximation techniques require a *fixed* number of features, and the purpose of the SMRF forest representation is to provide such a fixed-length feature vector. Therefore, because the number of opponents might vary, it is not possible to present a separate feature for each one. One could average the values, but that might dilute key information. Vens et al. (2007) present a method capable of selecting representative instances or aggregate functions of them. For SMRF-RPI, I simply choose the most representative instance from the winning leaf. Specifically, each instantiation sequence has a pdf density associated with it for each question node. A higher density indicates a more representative instance. Treating each density as independent, given the instance, I take the product of all densities for an instantation sequence. The sequence with the highest product (calculated in log space) is the representative, and its densities are used in the feature vector. Non-winning leaves contribute 0 for each question above them in addition to contributing the binary 0 for the leaf itself.

In all, a tree contributes one feature for each leaf and one feature for each question

node above each leaf (traversing rootward). The feature set for each action type consists of all trees accepted in the learning process for that action type and also a single constant bias feature. The entire feature action-state feature vector is composed of the features for all action types. When computing $\phi(s, a)$, features for action types not matching a are set to zero.

6.5 Continuous Action Spaces

While SMRF-RPI works directly in the continuous state spaces derived from object attributes, I so far have discussed only discrete actions. That is, if all actions are parameterized with respect to objects in the scene, and there is a finite number of objects, then there is a finite number of possible actions as well. This makes choosing the action with maximum Q (as shown for example in Equation 2.2) a straightforward process. However, some tasks may require actions not directly related to the enumerated objects. For example, to set a table, a robot needs to be able to choose where to put a plate when the table is empty or where to put a spoon relative to the plate.

Certainly, many techniques exist for choosing continuous actions, either using direct continuous prediction, optimizing across Q values, or some combination of the two (e.g., DeJong, 1994; Davies et al., 1998; Palmer and Goodrich, 2002; Xu and Laird, 2011; Kulick et al., 2013). For the present work, SMRF-RPI accommodates continuous action spaces through the use of *virtual objects*. These entities represent attributes corresponding to potential actions. For example, the table-setting robot imagines a potential plate location as if it were an entity in the scene. Such virtual objects can then be instantiated and queried by SMRF just as for observed objects. Any appropriate optimization algorithm might be employed for choosing attributes (such as position) for virtual objects. Given a Q function, these attributes are the

input, and the Q value is the output. Depending on the representation, the Q function might be generally or locally nondifferentiable in addition to containing local maxima. For the current scope, I employ a simple beam search across sampled actions, where the action sampler is a parameter of the task at hand.

Chapter 7

Experimental Results

7.1 Domains

SMRF-RPI is concerned with multistep tasks in continuous, multidimensional domains with multiple objects and interesting relationship between them, much like many concerns in the real world. In this chapter, I test SMRF-RPI in multiple simulated domains. I begin with a propositional 1D corridor domain, for convenient testing and analysis. I then follow up with well-known domains having relational concerns, specifically those already presented in Chapter 5, those of block stacking and soccer. Here, I present different (and multistep) tasks for the block stacking and soccer domains, appropriate for the evaluation of SMRF-RPI.

7.2 SMRF-RPI and Experimental Configuration

For these experiments, I use the same mapping functions as listed in Section 5.3. Except where otherwise noted, I gather 500 episodes of training (as well as test) experience per iteration of SMRF-RPI, SMRF receives a maximum of 500 randomly sampled scenes for each tree (without replacement), and LSPI receives a maximum of 10^5 SMDP examples, also sampled without replacement. For SMRF tree learning, only the latest batch of training experience is used, but all batches are available for sampling for LSPI. For the CovAgg MIL algorithm, I use the same parameter values as mentioned in Section 3.3, except that the number of aggregation bags sampled M depends on the number of positive bags, varying between 4 and 6. For LSPI,

I enforce a condition number on matrix A of no greater than 10^5, using singular values for the calculation. At each iteration of representation learning, two trees with original labels and two with negated labels are attempted for each action type. I use $\alpha = 0.05$ for the F-test on tree selection, adjusted to 0.0125 by Bonferroni correction for the four trees per action type. Additionally, as some tree features often go unused (particularly for error branches), features with no data yield rows and columns of zero for LSPI. I trim these entirely from the computation and assign weights of zero to these features. For statistical purposes, each experiment includes 30 independent runs of SMRF-RPI.

Throughout this chapter, I report rewards as mean *undiscounted* total reward per episode. In the case of the keepaway benchmark task, specifically, and in other RL research, this is common, even when discounted future rewards are used during the learning process.

Also, throughout results in this chapter, all statistical comparison tests consist of one-tailed, two-sample t-tests, unless otherwise stated. When comparing methods, I most often compare performance after the first iteration of learning when demonstrating which learns faster. I also commonly compare method performance after the last iteration to demonstrate which ultimately achieves higher reward. For the SMRF-RPI learning process itself, I also compare final performance to the random performance before learning, to demonstrate that learning as occured.

7.3 Corridor Domain

7.3.1 Domain and Task Description

1D "chain walk" or "corridor" domains, whether discrete (Lagoudakis and Parr, 2003) or continuous (Kersting and Driessens, 2008), are common testbeds for reinforcement learning because of their simple nature. The corridor I use is unit-length

and continuous, with rewards given at either end. The agent begins at a uniformly selected position and can move either left or right at any time step. The mean step distance is 0.1 units, with a environmental noise effect added by a Gaussian of standard deviation 0.05. The agent receives 0 reward for ending an action at any location other than a 0.1 unit window on either side of the corridor. In an edge window, it receives a reward of 1. Episodes are of 10 equal-duration time steps. Even once reaching an edge, the agent can continue to take more actions. An optimal policy consists of going left when already left of position 0.5 and going right otherwise. As this world consists of a fixed number of variables (specifically, 1), it is not relational in nature. However, it provides an opportunity to test convergence of SMRF-RPI in an easily diagnosable domain.

7.3.2 Corridor Results

Because this is a straightforward task with only a single 1D variable, I expect that sufficient experience and appropriate features allow learning an optimal policy. Figure 7.1 shows a sample Q function and resulting policy for the corridor task, after just one RPI iteration. In this first round, all Bellman error (BE) is either 0 or 1, depending on the immediate reward achieved, where Q values before learning are initialized to 0. After this round, Q values are no longer constant and BE is no longer binary. Also after the first round, mean BE for each action is often near 0, because LSPI produces Q functions with low BE magnitude. Later RPI iterations, including after further experience iterations, continue to refine the Q function, although certain regions (as defined by tree leaves) fit better than others. Figure 7.2 shows algorithm performance as a function of experience iteration, for different experience batch sizes. Here, with a single batch of 500 episodes, SMRF-RPI consistently learns a policy that is approximately optimal. This optimal policy achieves mean total reward of about 8.68, which is not significantly higher than that achieved by

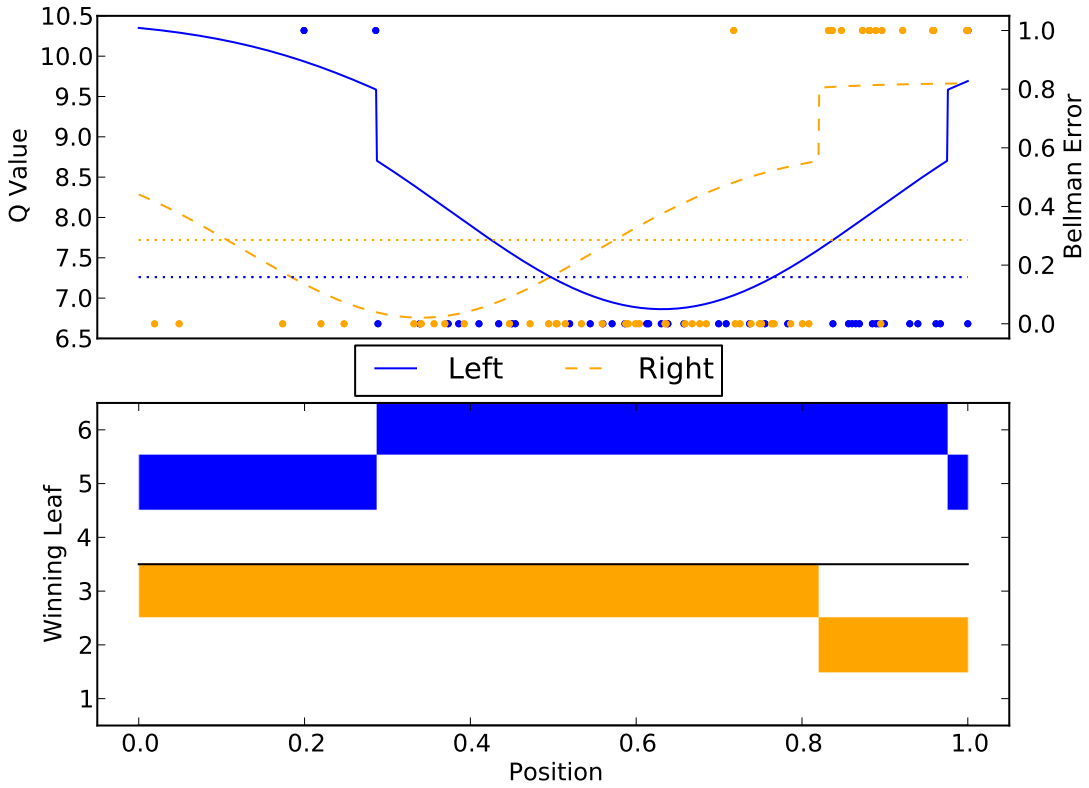


Figure 7.1: SMRF-RPI policy after one iteration of learning for an example 1D corridor task, where the agent is rewarded for reaching a 0.1 unit zone at either edge. The curves at top show Q values for each action. The policy is given by the action with maximum Q for each horizontal position. Individual points show the Bellman error (BE) for each training example. Dotted lines show mean BE for each action. The bars at bottom show regions for which different leaves are active. This figure is repeated from Figure 6.1 for convenience.

the SMRF-RPI learned policy ($p < 0.5$). Note that the lower standard deviation for batches of 500 episodes is due in part to taking the mean across many more episodes. I include the demonstration of 10-episode batches to show the effect of experience on the learning process.

Figure 7.1 also shows how a single tree for each action type can produce nearly optimal policies on this task. Here, the decision boundary near 0.5 is solely due to the nonlinear pdf density features. Immediate reward is only visible near edges, but the key decision point is the center. With binary features (bits) alone, more RPI iterations are needed to find larger regions of positive Q near the edges and to

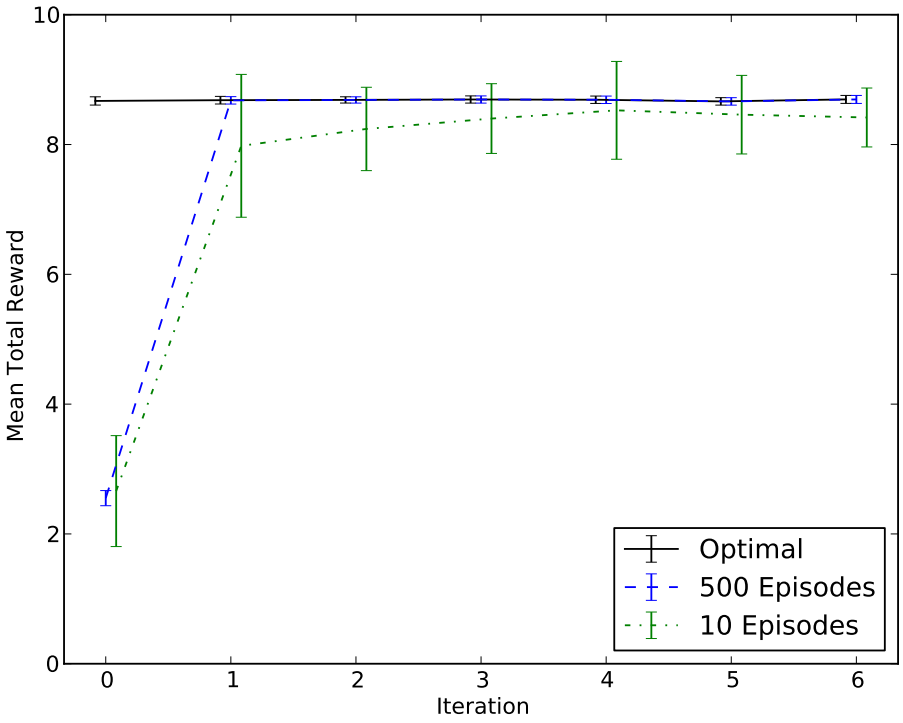


Figure 7.2: Mean total reward, across 30 independent runs, for different numbers of episodes per iteration for the corridor task. Error bars show standard deviation across runs.

gradually learn distinctions near the middle. Because of the multiple iterations of learning for each batch of experience, SMRF-RPI with bits alone does not perform significantly worse than with the density features ($p < 0.5$ at the final iteration), as shown in Figure 7.3. However, Figure 7.4 shows that using bits alone performs well at the cost of more total tree complexity, as measured by the total number of leaves across all trees ($p < 10^{-11}$ at the final iteration). Note that in both cases, while later iterations add fewer features than earlier iterations, some still are learned. Among other matters, this can be due to SMRF-RPI's attempt to match the Q function itself and not just to learn an optimal policy. In other words, the policy converges before the Q function does.

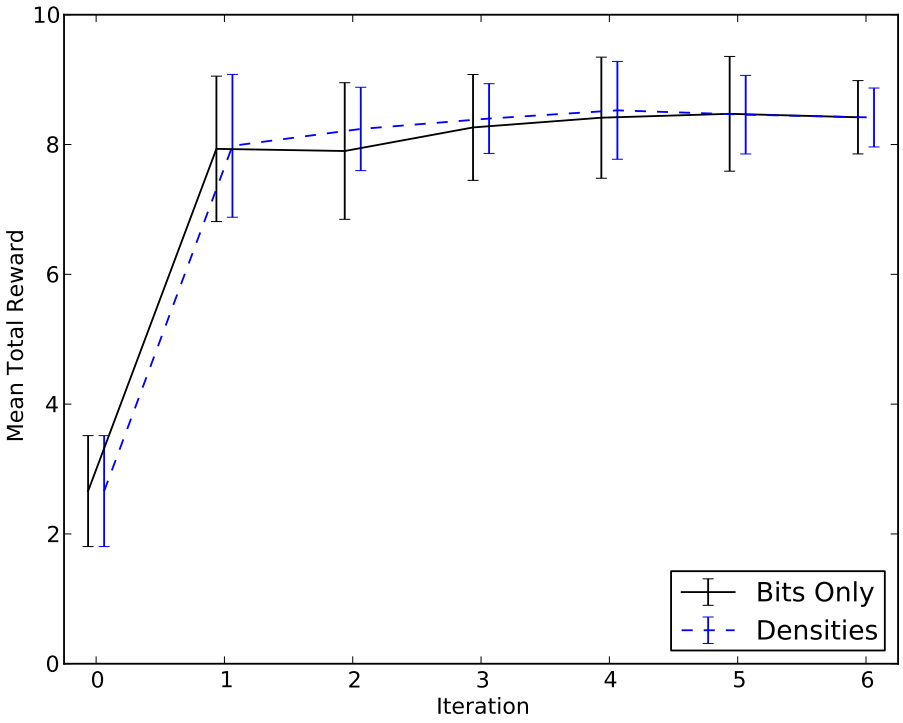


Figure 7.3: Mean total reward for representations of pure bits and for pdf densities with bits for the corridor task, using 10 episodes per iteration. Error bars show standard deviation across runs.

In a further experimental configuration, in order to demonstrate the value of the wrapper selection method discussed in Chapter 6, I evaluate tree selection without this mechanism. Instead, I select the tree with minimum p value from the SMRF learning process itself. If SMRF-RPI's representation learning loop produces at least one tree, then at least one tree is kept. In this configuration, at least to a certain limit, SMRF-RPI never finishes the first loop of representation and policy learning. SMRF keeps learning new trees, perhaps due to having enough noise in a stochastic domain that inhibits perfect fit and also due to SMRF having access only to a binary approximation of BE error from which to work. In any case, this demonstrates the value of the wrapper selection process, which can reject trees that fail to reduce BE

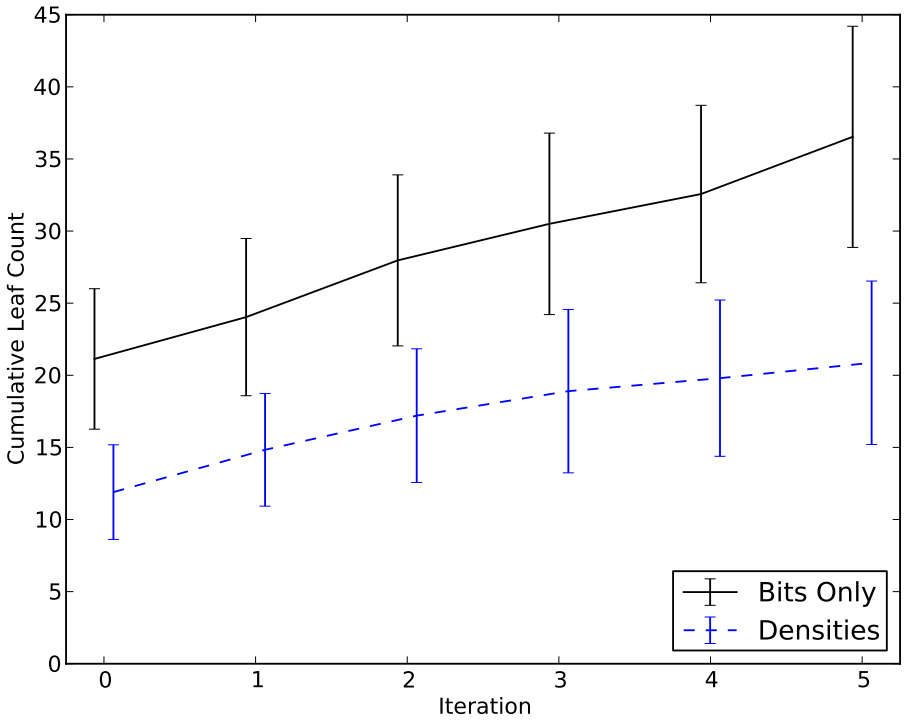


Figure 7.4: Cumulative leaf count for representations of pure bits and for pdf densities with bits for the corridor task, using 10 episodes per iteration. Error bars show standard deviation across runs.

error.

7.4 Blocks World Domain

A very traditional domain for the study of artificial intelligence is that of the blocks world (Slaney and Thiébaux, 2001). Block arrangement can represent a simplified form of many manipulation tasks and is very relational in nature. Historically, the blocks world is entirely symbolic. The world state is often defined by predicates such as $on(A, B)$ with derived predicates such as $clear(A)$ when no block is on A. A special constant called *table* exists on which any number of blocks may be placed. The primary action is $put(A, B)$, which puts A on T, given, for example,

the precondition $clear(A) \wedge clear(B)$ or if $B = table$.

While fruitful research still exists in symbolic blocks world domains, recent research also exists in physical block stacking domains, both simulated and real world (Pasula et al., 2007; Lang and Toussaint, 2010; Toussaint et al., 2010; Kulick et al., 2013). This also includes two-dimensional dynamics simulations (e.g., Zhang and Renz, 2014).

For my present work, I consider a 2D simulated blocks world (Palmer, 2015) based on the JBox2D dynamics engine (Murphy, 2010), with a variety of rectangular block proportions and sizes. Specific properties affecting dynamics (such as friction) and user interaction (for example, pull force being proportional to mass and grasps being more stable closer to block centers) have been selected for a somewhat predictable world with simple, but rich, pointer interaction capabilities. Having been tailored for human interface, the world dynamics are also compelling for automatic agent interaction as well. The software implementing the 2D blocks world here is the same as that used in Chapter 5, although, in this chapter, I address different tasks.

Specifically, I test SMRF-RPI on three tasks: (1) "high towers," (2) color grouping, and (3) arch building. The high towers task is titled and described by Lang and Toussaint (2010), and based on an earlier task by Pasula et al. (2007). The goal of this task is to build stacks as high as possible. The color grouping task has the goal of grouping blocks with similar colors. It is based on the "desktop clearance" of Lang and Toussaint (2010), although I address the matter of multidimensional, continuous color in my version. Finally, the arch building task, inspired by the classic work of Winston (1970), requires building arches out of three blocks, where arch quality is based on the area under the arch.

The blocks worlds of Pasula et al. (2007) and Lang and Toussaint (2010) are 3D rather than 2D. Both include blocks of different sizes. Lang and Toussaint (2010)

use two different sizes, and they include both cubes and spheres. Pasula et al. (2007) include only spheres although still of "varying size." In contrast with these domains, I include non-square blocks, such that block orientation matters more. Also, whereas the other works use separate $grab(A)$ and $puton(B)$ actions, I use the more traditional $put(A, B)$ to unify the two. This puts less emphasis on the deictic context emphasized by Pasula et al. (2007), as my focus is on other matters. Also, I emphasize 2D because of the ease of human analysis and interaction, although SMRF is designed to work in higher dimensions as well.

In all tasks, selected high-level actions (options) end after all block translational and rotational velocities drop below a certain absolute magnitude. 100 time steps occur per simulated second. SMDP discounting occurs by simulated second; although fast-finishing actions aren't of highest importance here, they are preferred to otherwise equivalently positive but slower actions. Because the duration of an episode is potentially many seconds, and because some of these tasks receive reward only at the end, I use discount factor $\gamma = 0.99$ for this domain. (For other domains, I use $\gamma = 0.9$.)

Also, while I present some amount of detail on each of these tasks, many of the choices are also somewhat arbitrary. High-level purposes exist for the inclusion of each task and its structure, of course, but the details here are often merely for reference.

7.4.1 High Towers Task

The high towers task is meant to be somewhat straightforward but still relational. The reward is the mean height of the center points of all blocks. A small block on a large one yields greater reward than a large on a small. Before each episode, the width and height for each block is chosen randomly to be between 1 and 10 units. The width is considered to be the greater of the two, and I use a representation of

orientation that acknowledges the rotational symmetry of the blocks. Specifically, angles near 0 are horizontal and near π are vertical. Five blocks are dropped to start each episode, drawing the x coordinate from a uniform distribution between -30 and 30 units, where 0 is the center of the table.

The following options are available:

- *put*(A, B): Puts block A on block B, by first grasping block A at its center point. After grasping the block, the option chooses a height to lift the block of 5 units of clearance above the highest block other than A, and it lifts the grasped block to that height. It then moves the block horizontally to center over the x coordinate of B, lowers the block until it has a clearance of 2 units, then drops the block. There might be other blocks above B. If so, these determine the clearance rather than B itself. Even A is not required to be the top of the stack; others above it might be carried or, more often, thrown as the option moves A. For the initial grasp, 2D Gaussian noise of standard deviation 0.5 in each dimension is added to the intended grasp point, retrying if the grasp point falls outside the block. This noise model implies the grasp position of smaller blocks might be closer to the edges than the center, causing the grasp to be less stable on average. All goal positions, intermediate and final, have a 2D Gaussian noise component with standard deviation 1.

- *rotate*(A, B): Because grasps further from the center are less stable, this action can position the grasp near one edge and lift to change the rotation of the block. First, the option determines if the block is primarily vertical or horizontal to determine its grasp point. It chooses a point straight out from the center 0.7 of the way to the edge. The block is then lifted twice the distance needed for clearance of its lowest corner from the height where it started. It is then lowered and dropped as for *put*. 2D Gaussian noise is added to the grasp

point, with standard deviation of 0.025 times the distance in units from the target grasp point from the center. A fair amount of precision is required for the action to be usually successful. The same noise model as for *put* is applied to all goal positions.

For this task, I use a constant episode duration of 5 actions and also a shaping reward in accordance with Ng et al. (1999). Specifically, the agent receives a reward which is the difference in average block height before and after each the option, which might be positive (for an increase) or negative.

Random exploration actions select from a multinomial of 0.75 for *put* and 0.25 for rotate. Once an action type is selected, all possible instantiations of that action are selected from uniformly.

High Towers Results

To demonstrate agent behavior in the high towers task, Figure 7.5 shows two example episodes from the same relatively high performing policy at the last test experience iteration, where block identification numbers increment across episodes. The first example is a case where the agent stacks all blocks in an effective, though suboptimal, fashion within the time limit. While not depicted, the agent's action at the first time step is to place block 38 on block 39. Then, in choosing to place block 37 on block 39, it topples the existing stack. Overall, this results in an improved final configuration, although by chance. In placing the remaining blocks, however, it begins with block 36, which is a better choice than the other remaining options. The final total reward for this episode is about 14.98.

The second example episode is less effective in its result. Not shown here, the agent repeatedly attempts to place block 57 on block 59, failing three times due to environmentally imposed noise. It finally succeeds and then chooses to stack block 56. The final total reward is about 6.17. These two examples suggest that somewhat

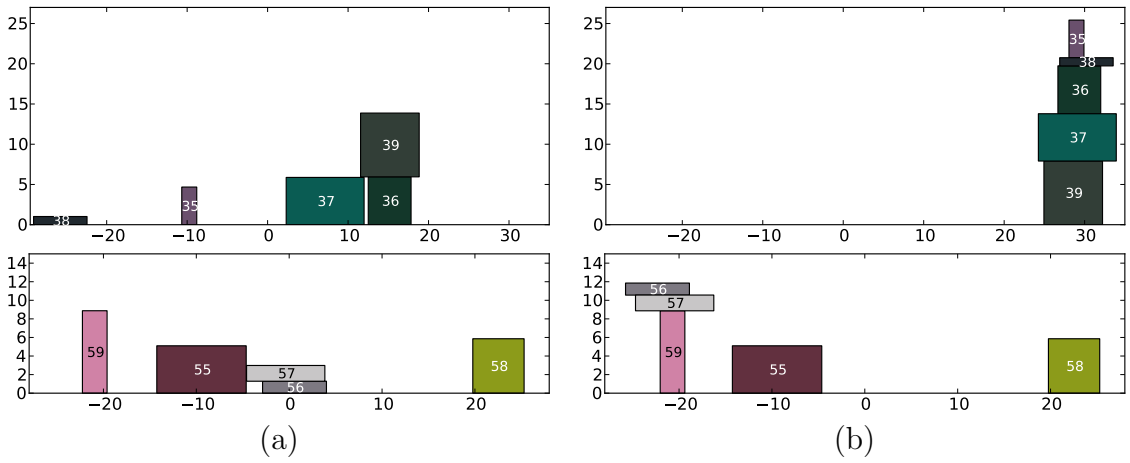


Figure 7.5: Two examples episodes for the high towers task, showing (a) the starting state and (b) the ending state for each.

unpredictable policies can be learned, perhaps due in part to the amount of noise in the dynamics of the environment. Still, the policies can be effective in some cases. Also, for both example episodes, the agent attempts to place blocks on the highest existing stack at every time step.

Looking at additional episodes, higher performing SMRF-RPI policies emphasize creating a single stack. It is difficult to analytically determine an optimal policy in this noisy and time-constrained setting, but the single stack might be a reasonable strategy. Even with the time limit, analysis suggests that *rotate* might occasionally be put to good use. However, while observed SMRF-RPI policies do sometimes use *rotate*, it does not seem to be used effectively. Occasionally, degenerate cycles arise where a policy rotates a single block repeatedly while other meaningful stacking possibilities still exist. Perhaps the agent uses this as a form of "no op" when it fails to observe more useful alternatives. Also, SMRF-RPI seems rarely to rotate blocks that would cause other blocks to fall.

To help provide intuition about learned representations, Figure 7.6 shows the learned tree selected for *put* in a first RPI iteration. The first question tests for a block whose center is less than about 6.7 units in altitude (where mean block edge

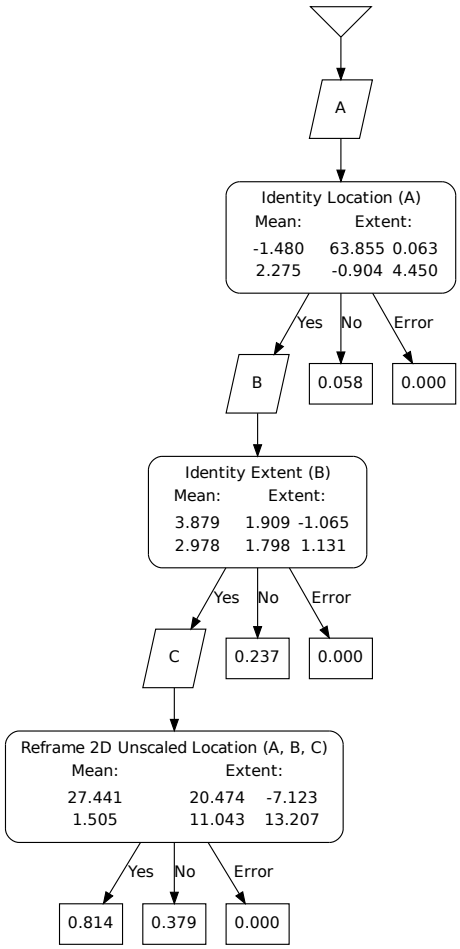


Figure 7.6: Example tree for $put(A, B)$ on the high towers task, where the first question is effectively both one-dimensional and an inequality, due to the nature of the task. Later questions express more covariance. Extent columns show scaled eigenvectors.

length is 5.5 units), irrespective of the x coordinate. SMRF location questions are multidimensional, but by supplying a large variance in the x direction, it effectively makes this coordinate irrelevant. Also, because blocks exist only at positive altitudes, this question effectively becomes an inequality. The second question asks for a relatively large block for the destination. The larger size is always used for the first dimension of extent (which, here, is half the length of a side), and the maximum extent used in this task is 5 units. The final question asks about a third block beyond the potential destination. This might be due either to overfit or a subtle

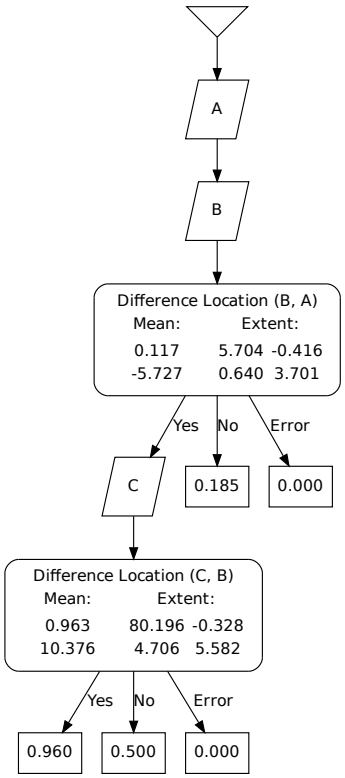


Figure 7.7: Example tree for "not" *rotate*(A) on the high towers task, learned on negated labels. The first question identifies other blocks above A. Such blocks could presumably be at risk of being toppled. Extent columns show scaled eigenvectors.

effect of task dynamics.

Figure 7.7 shows the learned tree selected for "not" *rotate* (i.e., using negated labels) in the same first RPI iteration as for the *put* tree. This tree identifies the existence of other blocks vertically aligned with and above the block A, the one being rotated. Specifically, the first question asks if B is within about 6 units of A horizontally and from about 2 to about 9.5 units above it, because the altitude of A minus that of B is negative. The second question asks if B is about 5 to 16 units above another block C, though without regard to the horizontal difference. In the highest probability leaf, B is at risk of being toppled from a high altitude, which would provide a negative reward to the agent. While this tree makes some sense, it presumably only helps to avoid mistakes rather than to constructively build

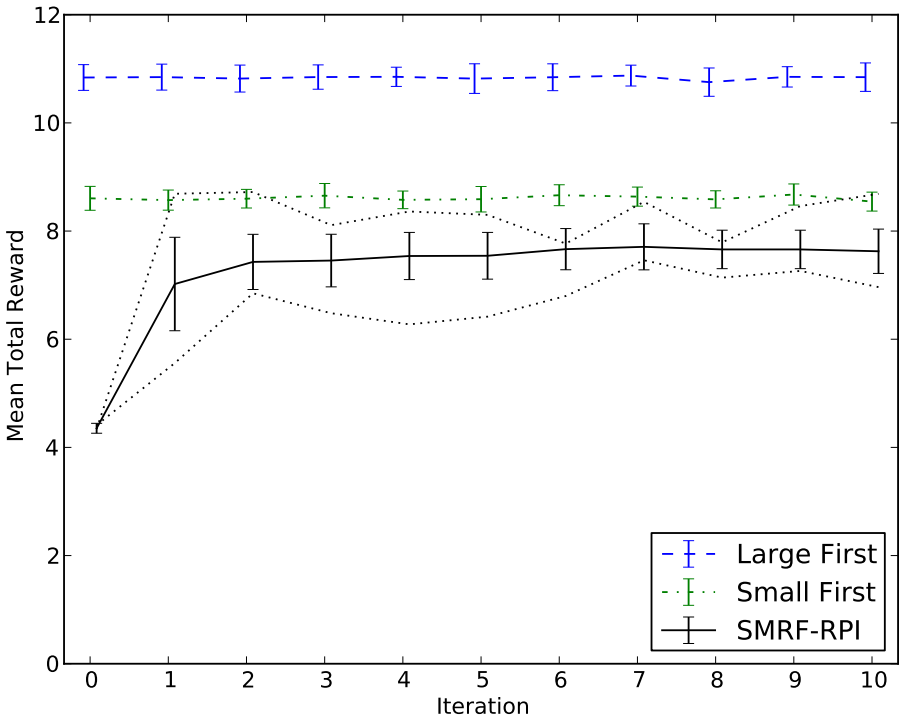


Figure 7.8: Mean total reward, across 30 independent runs, for high towers task for 500 episodes per iteration. The "large first" and "small first" policies both are fixed and hand-coded. Dotted lines around SMRF-RPI results show the *test* performance of the runs with the maximum and minimum *training* performance for SMRF-RPI after each iteration. Error bars show standard deviation across runs.

higher towers. Trees learned in the second RPI iteration for this run are shown and discussed in Appendix A.

Looking at overall performance, SMRF-RPI is still expected to consistently learn beyond random behavior in this task, despite the task being complicated over previous incarnations by including the *rotate* option. Figure 7.8 shows mean total reward for this task. SMRF-RPI does learn to improve on its initial random play ($p < 10^{-43}$, according to a one-tailed, two-sample t-test, as also used for all other statistical comparisons in this chapter, unless stated otherwise).

In addition to SMRF-RPI, this figure shows results for two manually designed

policies. The "large first" policy tries to put the largest blocks at the bottom of the stack, knowing that this will result in more blocks later at a higher altitude. The "small first" policy starts by moving the smallest blocks first. Both place blocks onto the highest point available, thereby attempting a single stack of blocks. Neither uses *rotate*, although it could result in raising a block's center. SMRF-RPI mean performance is below both the large first ($p < 10^{-40}$) and small first ($p < 10^{-16}$) manual policies. However, the high end of the SMRF-RPI runs seems perhaps to border on the mean performance of the small first policy. This is visible in the dotted lines of Figure 7.8, which show the *test* performance of the runs with the maximum and minimum *training* performance. As a common theme throughout the results in this chapter, some SMRF-RPI runs actually do learn better policies than others. To test this, I subtract the overall mean test reward for each experience iteration from the corresponding mean training and test reward for each SMRF-RPI run, to allow grouping together of all experience iterations for comparison. Yielding 30 points per experience iteration, the correlation between training and test performance for SMRF-RPI for this task is $R = 0.81$ ($p < 10^{-77}$). In contrast, variance in fixed policy performance is due entirely to noise in the environment.

7.4.2 Color Grouping Task

The color grouping task is an opportunity to require the use of both color and position in a natural context. While inspired by the "desktop clearance" task from Lang and Toussaint (2010), the color grouping task here is substantially different. First, rather than the single explicit goal (exact grouping), the reward here is continuous. Related to this, color matching here is also multidimensional and continuous. I further complicate the grouping process in the means of determining color similarity. In the SMRF learning process, I use an RGB colorspace, in the same fashion as Bodenhamer (2014). However, reward calculation for this task uses the industry

standard CIEDE2000 color distance formula (Luo et al., 2001), which is designed to approximate human perception of color differences. This formula is neither axis aligned with nor a linear transformation of RGB space. Further, CIEDE2000 does not correspond to a proper metric space at all (Ridolfi et al., 2010; Kinsman et al., 2012). Therefore, for the present work, this task is an opportunity to observe how RGB color might be still be used for modeling a complicated and inconsistent, but human-oriented, transformation of the multidimensional space.

The initial state of each episode begins with five blocks, in the same fashion as for the high towers task. This task, like high towers, uses a relative score from option start to end for a shaping reward. The overall score for a group of individually isolated blocks is zero. A stack of blocks might receive a positive or negative score. Stacks are defined by any group of blocks that include some horizontal overlap between the midpoint of one block and the horizontal extrema of any other block in the stack. For any stack, each block's RGB color is converted to the CIE Lab color space (which also forms the foundation of the CIEDE2000 distance formula), and the mean Lab color is calculated. The block who's color is most similar to the mean, according to CIEDE2000, is the prototype member. All other blocks in the stack add a score based on their CIEDE2000 distance to the prototype.

The score for a block is based on the following formula

$$score_{linear} = \begin{cases} (\Theta_{mid} - distance)/\Theta_{mid} & \text{if } distance \leq \Theta_{mid} \\ \max(-1, (\Theta_{mid} - distance)/(\Theta_{max} - \Theta_{mid})) & \text{otherwise,} \end{cases}$$

$$score = \sin\left(score_{linear}\, \pi/2\right),$$

where I choose $\Theta_{mid} = 30$ and $\Theta_{max} = 120$ for CIEDE2000. In other words, distances less than 30 contribute a positive score, and distances greater a negative. Distance 30 is near the 25^{th} percentile of distances sampled (and the median at about 45). There

is no maximum CIEDE2000 distance, but using both random sampling and evenly spaced grids of various resultions in RGB space, the effective maximum distance for this domain is approximately 120. The linear spaces between 0 and 30 and between 30 and 120 are adjusted to a sine curve for smoothness at the boundary and to group the highs and lows somewhat.

In addition to $put(A, B)$, the additional following option is available:

- $isolate(A)$: This option takes a block and places it separately from other blocks. Commonly, in blocks world domains, this might be accomplished by $put(A, table)$. However, to simplify the view of relevant objects, I exclude the table from the scene description. Therefore, we need a special option to supporting isolating blocks. Random initial arrangements and other complicated interactions sometimes result in blocks being stacked without agent control. This provides a chance to separate blocks that don't belong in any stack. This option is implemented the same as for $put(A, B)$, except for the choice of the goal x position. To choose the goal, it first determines the horizontal extent of the block and looks for a gap between existing blocks with enough space for a buffer zone of 2 units on either side. If no such gap exists, the block is positioned to either the left or right of all existing blocks, whichever is closest to the center of the table. If the block was already isolated, the action merely delays a mean of 1 simulation second, with Gaussian deviation of 0.2 and a resampling minimum of 0.5.

Random exploration for this task is that same as for high towers, except that *isolate* takes the place of *rotate*.

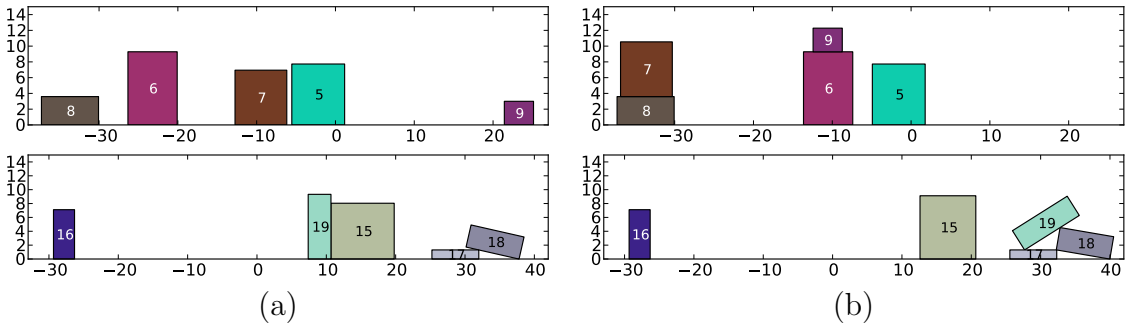


Figure 7.9: Two examples episodes for the color grouping task, showing (a) the starting state and (b) the ending state for each.

Color Grouping Results

Figure 7.9 shows learned behavior with example episodes from the final test experience iteration of a relatively high performing SMRF-RPI policy. The first example is the first episode of this test experience. In this example, the agent successfully builds two groups, leaving out block 5 in the end, whose color does seem distinct from either group. The final action of the episode is, in fact, to isolate block 5, although it is already alone. In this sense, the agent seems to use *isolate* appropriately as a "no op" in some cases. Within the episode some ineffective behavior occurs; the agent first places block 6 on block 9 and then block 9 on block 6. In this case, no harm results, but similar behavior in other episodes may have hurt performance. The final total reward for this episode is about 1.61.

The second example is the third episode of the same policy as before. This example demonstrates how particular block arrangements can hamper effective performance. Block 19 is not the only one that has moved here. In fact, the agent moves both block 19 and block 15 multiple times during the episode, but the interference of block 18 results in unstable stacking. The SMRF mapping functions in use have no clear way to appreciate exactly when blocks overlap, and apparently the situation here is too subtle. In the end, only blocks 19 and 17 are considered to

be in the same group, and the final total reward is about 0.28.

Looking at additional behavior, SMRF-RPI policies do sometimes isolate blocks from groups when they do not match well. It is also fairly common to use *isolate* as a "no op" just as in one of the examples examined above. Degenerate policies use *isolate* far too often. Higher performing policies also sometimes fail to group blocks of similar colors, or sometimes even group colors effectively then subsequently separate them. Some poor matches are occasionally made as well. Still, overall, higher performing policies do often effectively group similar colors.

As an example of learned representations, Figure 7.10 shows the tree learned for the *put* action in the first RPI iteration from the first of 30 independent runs of SMRF-RPI for this task. The first question asks if the two blocks have a similar color, using a clearly non-axis-aligned covariance matrix. Also, oddly, the leaf with highest probability exists down the *No* branch of the relative color question. Rather, the highest probability leaf results from a question about the configuration of two other blocks. On closer inspection, the number of original scenes matched for the high probability leaf is low. Perhaps these cases might have been due to situations where nonmatching blocks were knocked out of a group.

Figure 7.11 shows the tree learned for "not" *isolate* (i.e., with negated labels) in the same RPI iteration and run as the previous tree. There is a difference color question here that seeks at least some nonuniformity in the green axis. The questions about block configuration are enigmatic. In particular, the reframe question is asking about a block D, which is far from the one being isolated, where block A is also somewhat near the center of the table and low down.

Looking at overall performance, Figure 7.12 shows mean total reward for the color grouping task. As expected, SMRF-RPI is able to able to improve behavior over random play ($p < 10^{-19}$). On the other hand, SMRF-RPI performs worse than a manually designed policy ($p < 10^{-33}$ at the final iteration). The manual policy

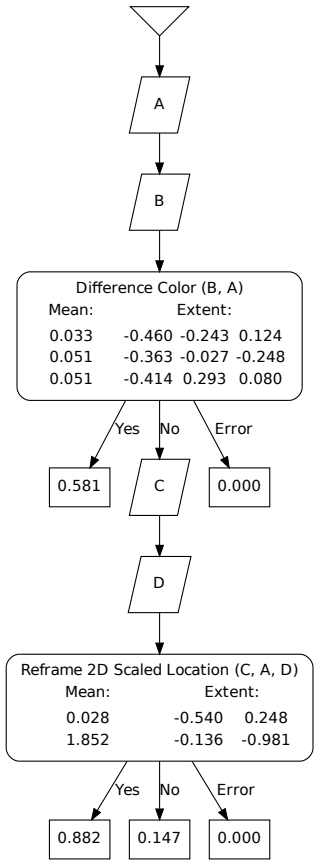


Figure 7.10: Example tree for $put(A, B)$ on the CIEDE2000 color grouping task, where the first question asks about color similarity. Extent columns show scaled eigenvectors.

has access to an accurate list of block groups, according to the scoring rules, and understands the idea of a prototype member of each group. However, as SMRF has no direct understanding of CIEDE2000, the manual policy determines each group's prototype and distances between colors using RGB space rather than by the CIEDE2000 formula. After determining a prototype for each group, the manual policy first looks to see if any block in any group is too dissimilar from the prototype, and isolates it if so. When isolating a block, the manual policy ignores the vertical position within the group. Once no groups with dissimilar blocks remain (and commonly none exist to begin with), the manual policy then finds already isolated blocks and chooses the best group to put each in. The manual policy presumes

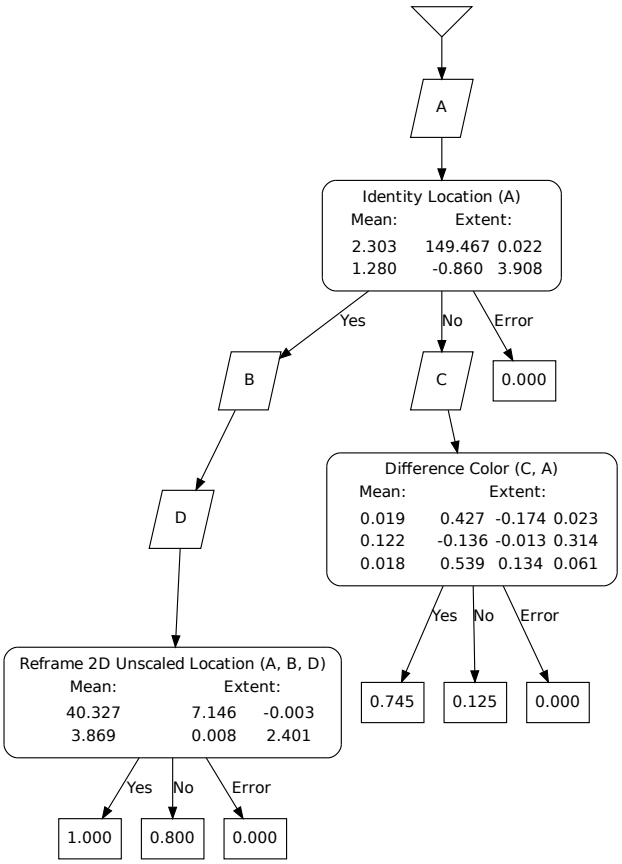


Figure 7.11: Example tree for "not" $isolate(A)$ on the CIEDE2000 color grouping task, learned on negated labels. The color question expresses similarity, but with some nonuniformity in the green axis. Extent columns show scaled eigenvectors.

an RGB distance threshold of 0.5 between positive and negative rewards, which is approximately the 25^{th} percentile of RGB distances when measuring each axis of RGB from 0 to 1. In the end, while both SMRF-RPI and the manual policy use RGB calculations, the manual policy is very aware of exact grouping and the concept of a prototype within each group. Although SMRF can ask about other blocks approximately vertically aligned with a target destination, it has no ability to reason directly about groups nor key members of them. It is therefore unsurprising that this manual policy outperforms SMRF-RPI.

Still, as mentioned above, SMRF-RPI does learn somewhat how to perform this task. However, during the learning process, around experience iteration 6, some runs

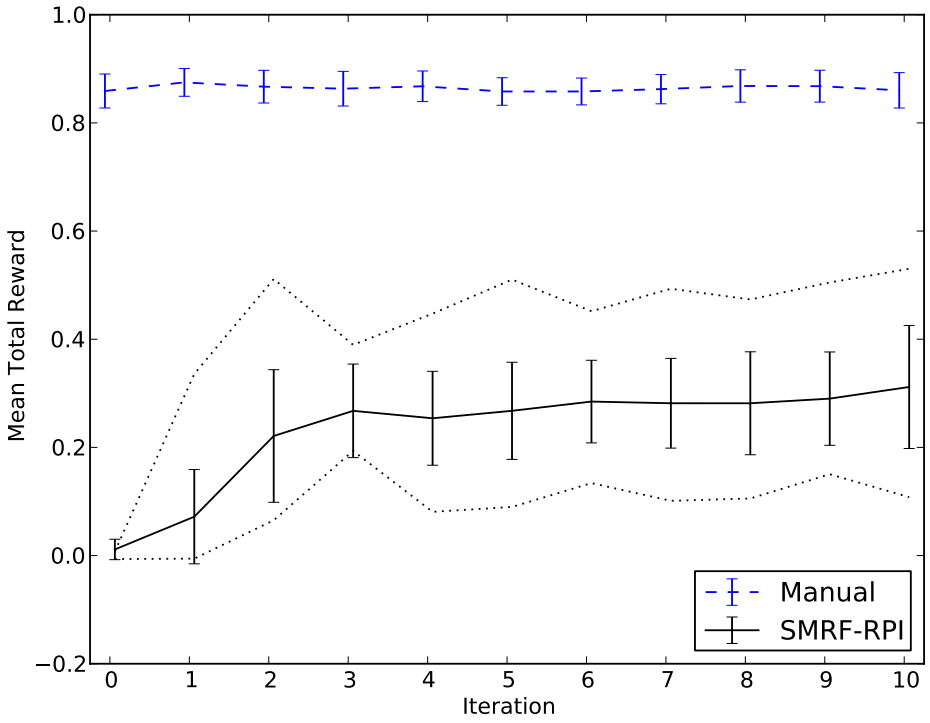


Figure 7.12: Mean total reward, across 30 independent runs, for the CIEDE2000 color grouping task for 500 episodes per iteration. Dotted lines around SMRF-RPI results show the *test* performance of the runs with the maximum and minimum *training* performance for SMRF-RPI after each iteration. Error bars show standard deviation across runs.

performing many more RPI iterations than usual, creating a large number of trees, which are perceived by SMRF-RPI as reducing the TD error and are therefore kept. Because of its non-metric nature, I hypothesize that this is due to the CIEDE2000 distance formula. To test this, I also perform a version of the task with scoring based on RGB distance. This version of the task uses an RGB distance of 0.5 for the cutoff between positive and negative scores. Figure 7.13 shows the performance of SMRF-RPI and the same manual policy (which now always correctly identifies group prototypes and score cutoffs) on the RGB color grouping task. However, as before, SMRF-RPI does improve its performance through learning ($p < 10^{-19}$), and

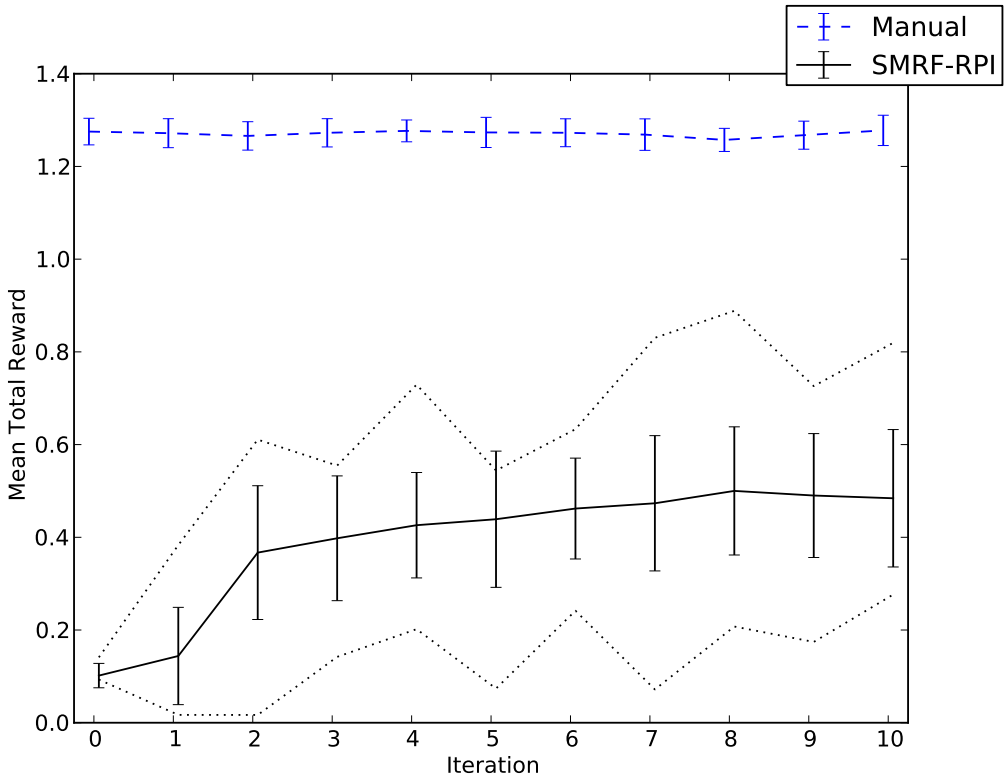


Figure 7.13: Mean total reward, across 30 independent runs, for the RGB color grouping task for 500 episodes per iteration. Dotted lines around SMRF-RPI results show the *test* performance of the runs with the maximum and minimum *training* performance for SMRF-RPI after each iteration. Error bars show standard deviation across runs.

the manual policy outperforms SMRF-RPI ($p < 10^{-35}$).

Too many factors exist for simple comparison of reward achieved between the two versions of the task. However, it still makes sense to compare representation size, which motivates the RGB version. Figure 7.14 shows mean cumulative leaf counts for both versions of the task. By the final experience iteration, the CIEDE2000 version has a larger mean total leaf count ($p < 0.05$). Only some of the runs suffer from the apparently degenerate sublearning, but they are enough to offset the overall mean.

As this task is primarily concerned with color *distance* rather than relative colors,

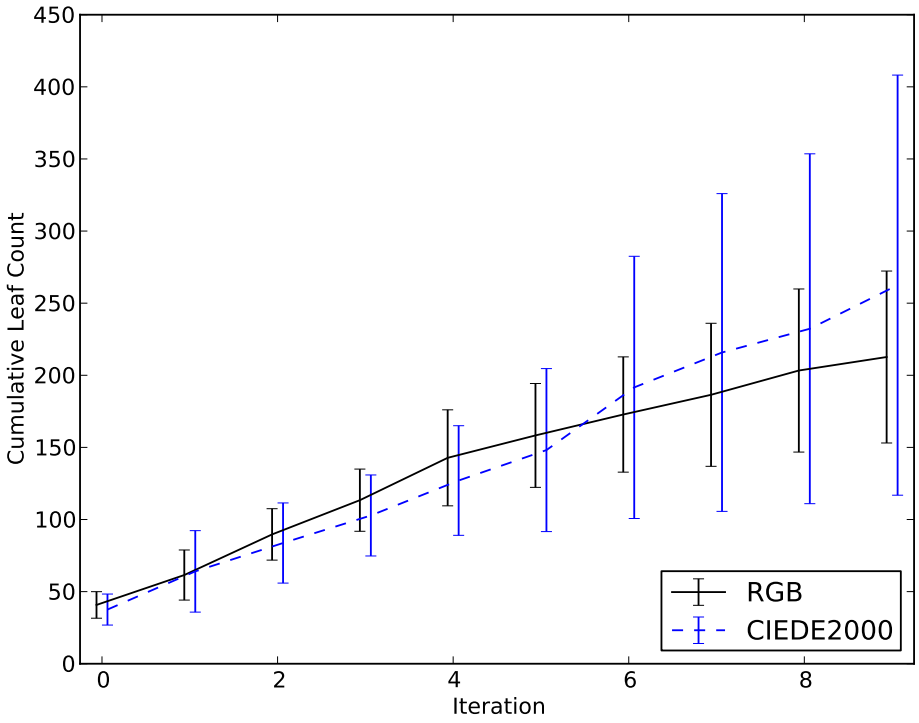


Figure 7.14: Mean cumulative leaf counts, across 30 independent runs, for the CIEDE2000 and RGB color grouping tasks. Error bars show standard deviation across runs.

there might be value in including a mapping function for color distance in SMRF tree learning here. However, following Bodenhamer (2014), I do not include color distance among available mapping functions (as listed in Section 4.3). Given performance of the manual policies, it seems likely that both versions of the task could take advantage of such a function, especially in that it has fewer degrees of freedom than relative color (having only 1 dimension rather than 3). It would also be interesting to compare the use of a color distance function in learned representations between the RGB and CIEDE2000 versions of this task.

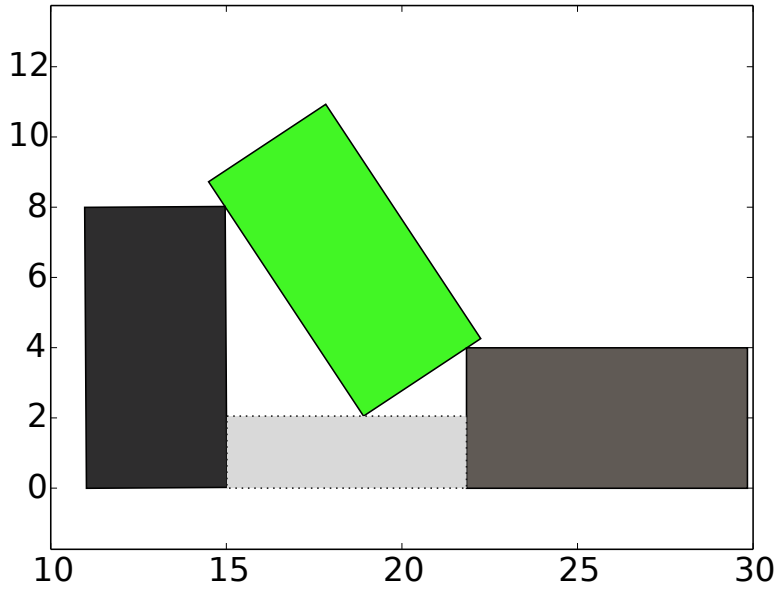


Figure 7.15: Example arch with scored area in light gray with dotted border.

7.4.3 Arch Building Task

My final blocks world task is building arch configurations, as inspired by the classic symbolic-level work of Winston (1970). Most substantially different from the other two blocks tasks in this chapter, I allow arbitrary x coordinate placement of blocks. Also, unlike the other blocks tasks here, building an arch can inherently require multiple steps to achieve a goal. I define an arch as a block (the *top*) entirely above rather than resting on the table, with no other block under its center point, and touching two other blocks (the *supports*) whose midpoints are less than the altitude of that of the top block. To encourage arch quality, different arches can receive different rewards. If an arch has been built, the score is equal to the open axis-aligned rectangular area under the arch, as shown in Figure 7.15. Taller, wider arches (if successful!) with flat tops score best.

For this task, I always drop three blocks, each 8 units wide and 4 units high.

While dropped at uniformly random rotations, world dynamics result in most of the blocks resting horizontally at first, although some initially come to rest in an upright pose.

For this task, I allow the *rotate* option, as previously described, in addition to the following options:

- *move*(A, X): Moves A to the given x coordinate, following the same behavior pattern as for *put*(A, B). This is a continuous action, in that any goal coordinate can be chosen.

- *done*: This option performs no action within the scene. It merely terminates the task after a mean 1 simulation second delay, with deviation 0.2 and a resampling minimum of 0.5. Further, during the learning process, the agent must use this action to declare the task done in order to perceive any reward. This action provides an opportunity for the agent to model more directly the goal of the task as well as to demonstrate full lookahead in the learning process. It also provides a chance for the agent to say when it thinks moving blocks is likely to do more harm than good.

The action sampling process described in Section 6.5 is customized for this continuous action space. Further, the action sampler is different for training and learning than for testing. The continuous space prevents sampling all possible actions. For test behavior, I initially sample (1) moving each block between each other pair of blocks, (2) moving to the same x as (on top of) each other block, (3) closing 0.5 of the distance toward each other block, and (4) moving 0.1 farther away from each, counting from centers. After this initial sampling, later iterations of exploration beam search make additional steps of 0.5 toward or 0.1 away from each other block. Random exploration for test trials use a multinomial probabilities as for the high stacks task, with a probability of 0.1 for *done*, 0.9×0.75 for *move*, and 0.9×0.25

for *rotate*.

For training experience and in the learning process, the sampling is more directed toward arch-building outcomes, to provide more learning opportunity. In the first iteration, instead of the choices provided by the testing sampler, the training sampler chooses positions nearby other blocks (specifically, 0.75 times sum of the current horizontal size of the two), such that a successful arch might be built in the next round. It also samples cases of putting a block between two others, as for the testing sampler. Later iterations of action sampling need only relative offsets and therefore proceed the same as for the testing sampler.

In addition to the training action sampler, I also use a custom exploration policy for this task rather than random behavior. This policy is used wherever random behavior is used in other tasks during training sessions. This policy has competence in building successful arches; some of its failures are purposely coded just to provide variety. Specifically, it expects 3 or more blocks, chooses one to be the arch top and two others to be the supports. As episode duration might be limited, it selects an existing horizontal block, if present, to be the top. All else equal, it prioritizes block selection from left to right, according to their current positions. It rotates horizontal supports upright, although, with probability 0.5, it skips the rotation. It places supports at just the right distance apart, with a buffer zone of 1 unit on each side, for the chosen top. If any arch as been built, it selects *done* with probability 0.5. Finally, with probability 0.1, it selects at random from training-sampled actions, using the same multinomial as for test trials, rather than following the expert behavior at all. The careful training policy (and training sampler) somewhat resemble an expert teacher, allowing the learner to see examples of successful (and unsuccessful) behavior. Alternatives to careful demonstration could include active learning (Cohn et al., 1996; Settles, 2009; Kulick et al., 2013) or intrinsic motivation (Singh et al., 2004; Oudeyer et al., 2007; Schmidhuber, 2010; Barto, 2013), but such

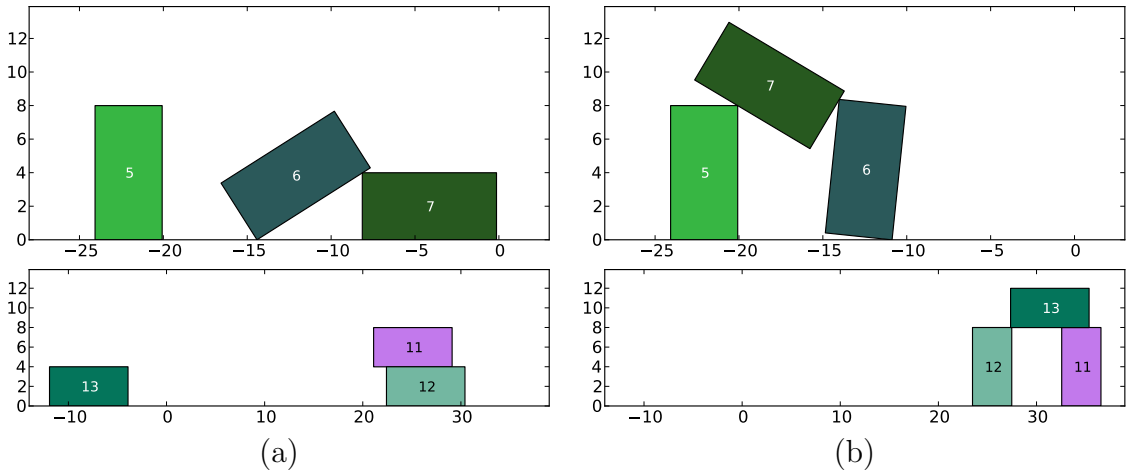


Figure 7.16: Two examples episodes for the arch building task, showing (a) the starting state and (b) the ending state for each.

are beyond the scope of this work.

Arch Building Results

Figure 7.16 shows learned behavior with example episodes from test experience iteration 3 of the highest performing SMRF-RPI policy at that iteration. The first example is the first episode of this test experience. In this example, the agent builds an arch after three actions and then declares the task done. The target x coordinate for placing the top block is about -16.7, but environmental noise seems to have resulted in it being placed farther to the left. The resulting diagonal top results in a final total reward of about 28.36. Encouragingly, SMRF-RPI demonstrates that it can use *rotate* in forward-looking fashion for this task.

The second example shown is the third episode from the same test experience iteration. In this case, the agent again successfully builds an arch, and, again, environmental noise results in the top being less centered than planned. (The agent here attempts to place the top at an x of about 30.0.) For this episode, the agent uses all five actions before building the arch, leaving no chance to declare the arch

done. In learning, the agent would perceive no reward. In test, however, this episode receives a total reward of about 40.66.

Looking at additional episodes, better performing SMRF-RPI policies commonly attempt to build upright arches. This includes using blocks that happen to be upright, or else setting blocks up if needed. Policies often make unhelpful choices, too, and the reason is often unclear. One common, unhelpful pattern I observed is an unwillingness to call an arch done if the supports were somewhat close together. Instead of attempting to separate the supports (perhaps due to short episode duration), it just repeatedly moves the top block to approximately the same position. For the first episode in Figure 7.16, the supports are somewhat well spaced. Perhaps the agent called this arch done because it failed to perceive the actual issues resulting in lower reward (i.e., the tilted top and support). Across the 500 test episodes of iteration 3 of the top-performing SMRF-RPI policy, it successfully ends 53% of episodes having built an arch. Of these successes, the mean total reward is about 40.87. Therefore, it seems possible that the key factor in the overall performance isn't the quality of built arches but rather the episodes with no arch at all.

As an example of learned representations, Figure 7.17 shows the learned tree selected for *done* in a first RPI iteration for this task. The high probability leaf is active when two blocks are about 9 units apart in x and when another block is about 10 units off the ground, at any x coordinate. This definition of an arch makes sense when there are only ever 3 blocks in a scene, which is the case for the task used here. Also in this case, as expected, the only tree learned on first RPI iteration is for *done*. On the second iteration of RPI, still before another batch of experience, SMRF-RPI adds another tree for *done* and one for *move*, both learned on negated labels. These trees are shown and discussed in Appendix A.

Considering overall performance, Figure 7.18 shows the mean total reward of SMRF-RPI as well as that of both a fixed policy closely matching the training policy

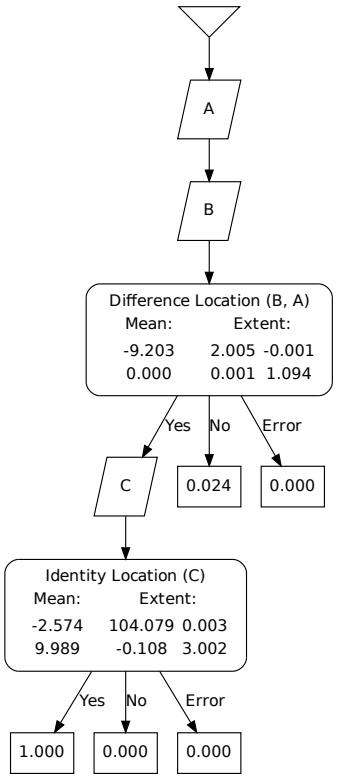


Figure 7.17: Example tree for *done* on the arch task, which asks about two blocks spaced beside each other and another at an appropriate altitude to form an arch top. Extent columns show scaled eigenvectors.

and an additional manual policy that excludes the noisy actions of the trainer. As discussed earlier, because of the wide-ranging nature of the task, the training policy is not purely random. Rather, it is a semi-expert demonstration, still with occasional noise added. The careful manual policy, on the other hand, makes no purposeful errors. Because of the block size used in this task, the theoretical maximum score is just shy of 64. However, due to noise in the *move* and *rotate* options and also the imposed time step limit, it becomes more difficult to construct a policy that achieves near that maximum.

Figure 7.18 demonstrates that SMRF-RPI has learned to improve its performance on this task ($p < 10^{-14}$), with some individual runs performing much better than the mean after the third iteration of learning. Some runs even outperform

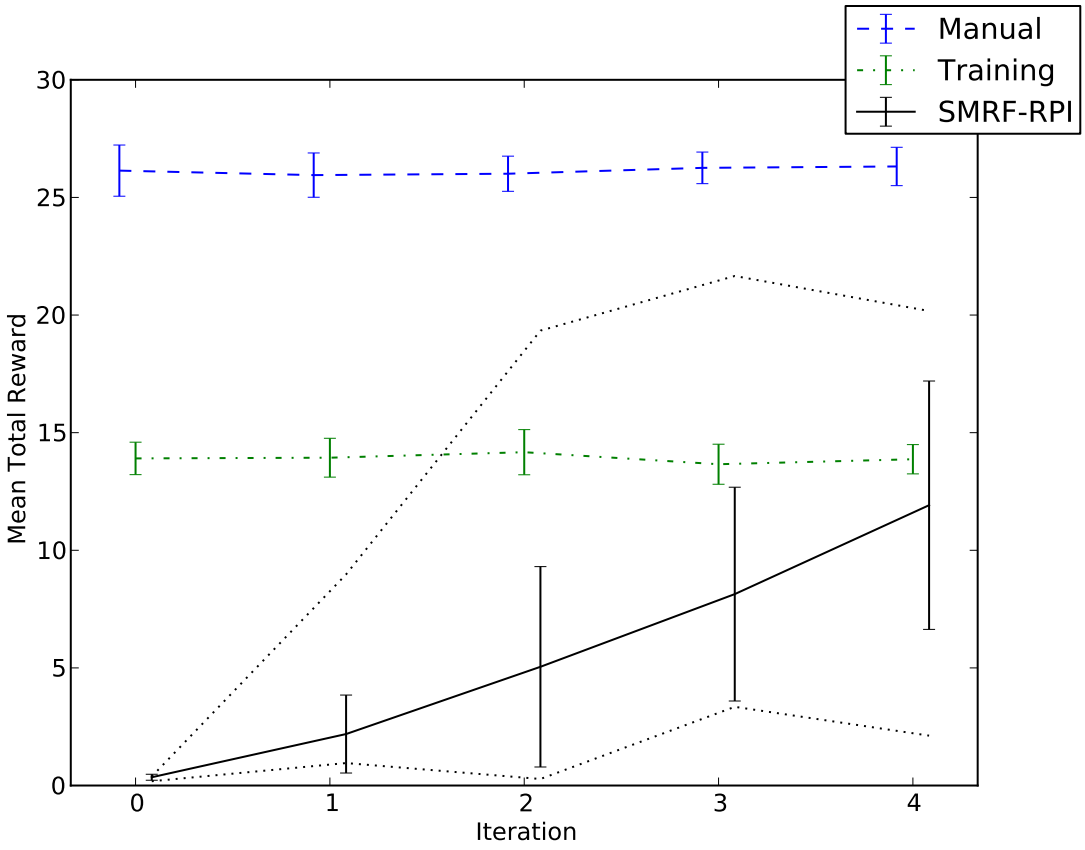


Figure 7.18: Mean total reward, across 30 independent runs, for arch building task for 500 episodes per iteration. The "manual" and "training" policies both are fixed and hand-coded. Dotted lines around SMRF-RPI results show the *test* performance of the runs with the maximum and minimum *training* performance for SMRF-RPI after each iteration. The final iteration consists of only 29 runs, as one run has not finished as of the time of writing. Error bars show standard deviation across runs.

the training policy mean, although, as of experience iteration 4, the training mean leads over the SMRF-RPI mean ($p < 0.05$). Still, the SMRF-RPI mean at experience iteration 4 is above that at iteration 3 ($p < 0.01$), and it seems likely that the mean may continue to rise in later experience iterations. However, as the number of SMRF representations also rises, and with the need to sample actions in continuous space, the learning process runs slowly. Therefore, later iterations have not been completed to determine asymptotic performance. The speed bottleneck is currently in the SMRF tree query process, and preliminary attempts at higher speed

SMRF query implementation suggest that large scalar speed improvements are very possible.

7.5 Soccer Domain

7.5.1 Keepaway Task

As in Chapter 5, my final domain for testing SMRF-RPI uses the third-party 2D-RoboCup-based keepaway benchmark task (Stone et al., 2006). Here, I employ the standard multistep task, which represents a portion of the full game of 2D RoboCup simulated soccer. In the common formulation of this task, 3 "keepers" and 2 "takers" compete on a 20m $\times$ 20m field. Each time step represents 0.1 simulation seconds. The keepers begin with possession of the ball. If the ball ever leaves the playing bounds or the takers take possession of the ball (for at least 4 time steps), the episode ends. The total reward is the duration of the episode in time steps, and the reward for any one action is the geometrically discounted sum of time units occuring during the action.

Also of note, in this dissertation, I use only the fully observable version of the task. Another common variation of the task, which I do not investigate here, limits the information visible to the players by restricting their field of vision.

All players are controlled by human-designed policies, except for the keeper closest to the ball when a new option is to be selected. Options include the following:

- $hold(A)$: Player A attempts to keep individual control of the ball. This option involves kicking the ball nearby so as to prepare for a subsequent kick. Alternatively, if an opponent is nearby (within 5m), this option results in a kick nearby but away from the nearest opponent.

- $pass(A, B)$: Player A attempts to kick the ball to player B.

World dynamics include some noise, such that attempted kicks do not always go where planned. Heuristic control algorithms, including option implementation, are unchanged from original sources, although I maintain a fork that continues to compile against current RoboCup Soccer Server code, incorporates the CMAC SARSA learning code of Stone et al. (2006), and has some minor modifications for bug fixes and conveniences in plugging in third-party SMDP agent implementations (Palmer, 2011).

The standard keepaway task includes a standard feature vector, which consists of 13 variables for the 3 vs. 2 case, sorting keepers by distance from the keeper with the ball:

- the distance from each player to the center of playing field,
- the distance from each player to the keeper with the ball,
- the distance from each keeper to its closest opponent, and
- the minimum angle for each teammate between the keeper with the ball and any opponent.

Standard actions are integers, indexed according to the distance to each teammate; action 0 always holds, action 1 always passes to the nearest teammate, and so on. In contrast, my raw relational attributes identify players directly and include the position, orientation, and team color of each player. Alternatively, team membership could be represented by a categorical variable, but I choose against this option to restrain the scope of SMRF algorithm for my present work. Further, this set of variables corresponds to blocks world variables in a straightforward fashion and reduces the amount of custom configuration needed.

In keepaway, the random policy is simply a uniform selection of available actions. In the common 3 vs. 2 configuration, this includes holding the ball or passing to one

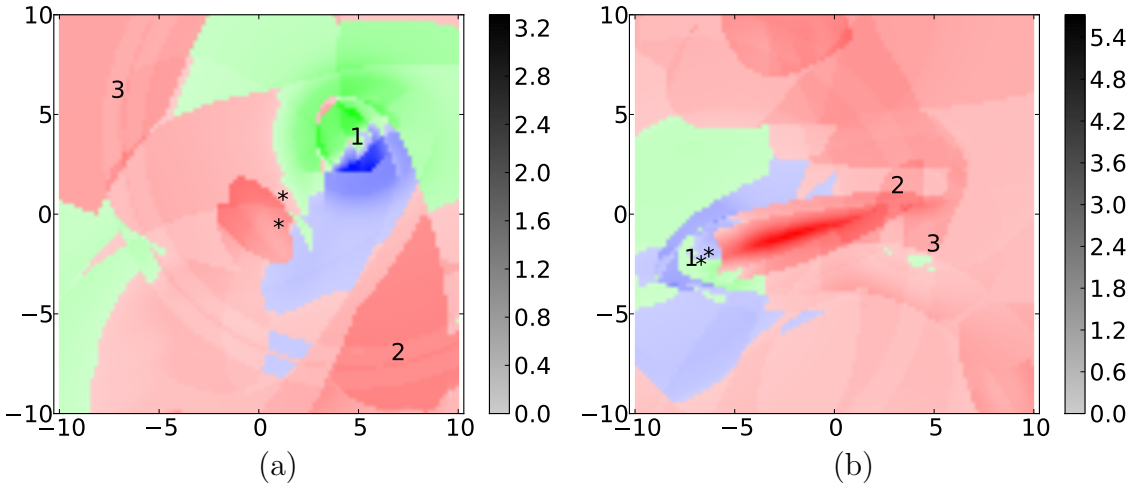


Figure 7.19: Two slices of an example policy learned by SMRF-RPI for the keepaway task. The number 1 indicates the keeper with the ball, and the asterisks indicate taker locations. The color darkness shows Q value difference from the winning action to that with next highest Q. Each position corresponds to changing the center of the position of the takers while retaining keeper poses unchanged. Red regions indicate where *hold* is the preferred action, green indicates *pass* to teammate 2, and blue indicates *pass* to teammate 3.

or the other of the two teammates.

7.5.2 Keepaway Results

In 3 vs. 2 keepaway, the takers are coded to both chase the ball rather than to split apart. Because of simulation dynamics, without two opponents nearby, a keeper can hold on to the ball indefinitely. As a result of this taker behavior, a good keeper policy involves holding the ball until takers come too close. At this point, the keeper with the ball should pass to the teammate who is most open. The standard manual policy for keepaway follows this heuristic (Stone et al., 2006). Higher performing SMRF-RPI policies also follow this general pattern, as seen by example configurations of an example learned policy in Figure 7.19. The exact boundaries are noisy, but this policy shows that when takers are near the keeper with the ball, it chooses to pass to the most open opponent, while at farther the

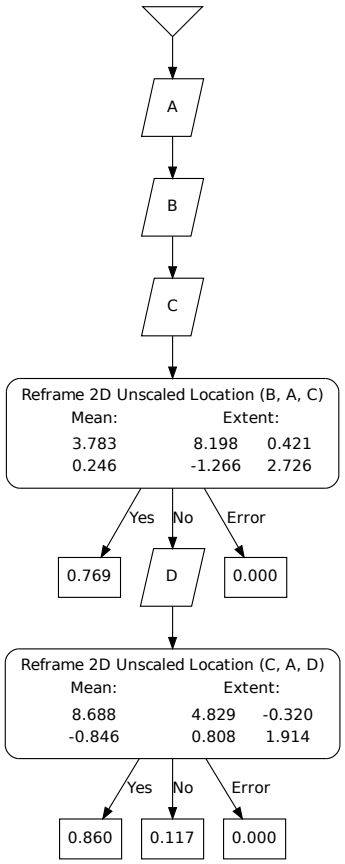


Figure 7.20: Example tree for "not" $pass(A, B)$ on the keepaway task, learned on negated labels. The first question identifies another player aligned somewhat with the vector between the passer and receiver. Extent columns show scaled eigenvectors.

distances, the agent often chooses to hold on to the ball.

As an example of learned representations, Figure 7.20 shows the "not" *pass* tree learned on negated labels in the first RPI iteration of one of the 30 independent runs of SMRF-RPI for this task. For SMRF reframe mapping functions (as detailed in Section 4.3), the first parameter is the origin, so the first question asks if there is a another player C in a region centered about 3.8 units toward the passer from the potential receiver. The region extends much more in the direction toward the passer than in the reframed y axis. The second question invokes another participant D where C does *not* refer to someone in the former region. This could either be an

example of overfit or an actual case of attention payed to subtle task dynamics. For this run, no *hold* representation is learned until the second RPI iteration, which is still before the next batch of experience. Trees learned in the second RPI iteration for this run are shown and discussed in Appendix A.

Performance Comparison

As described previously, RoboCup keepaway is a standard benchmark task. However, because of ambiguities in exact software configuration, for comparison with SMRF-RPI performance, I report on my own runs of the CMAC Sarsa learning software provided by Stone et al., rather than referencing previously published results. Figure 7.21 shows mean performance for both learning algorithms for 50 episodes per batch. As the Sarsa learning algorithm for the CMAC representation is an online learner, each "batch" consists of 50 episodes of online learning. The test performance shown is a snapshot of the policy before each new round of online learning begins. Also, SMRF-RPI uses all experience from a batch from all keepers, but the standard CMAC Sarsa implementation has each keeper learn independently from just its own experience. To put the algorithms on equal footing, I implement an experience sharing mechanism so that all keepers learn from each other's experience online. Unlike the report by Taylor et al. (2006), I find that learning does go much faster when the agents share learning. Still, from Figure 7.21, it is clear that in these early stages, SMRF-RPI clearly outperforms CMAC Sarsa, given the same number of episodes ($p < 10^{-7}$ at iteration 1, and comparisons through iteration 5 yield smaller p-values than this). SMRF is able to draw ellipsoids covering key regions, whereas CMAC tilings need to update weights for small regions at a time. This seems a likely cause for the initial lead of SMRF-RPI. After 10 iterations, however, the difference is no longer significant ($p < 0.1$).

Giving CMAC Sarsa more time to learn, Figure 7.22 shows a different configu-

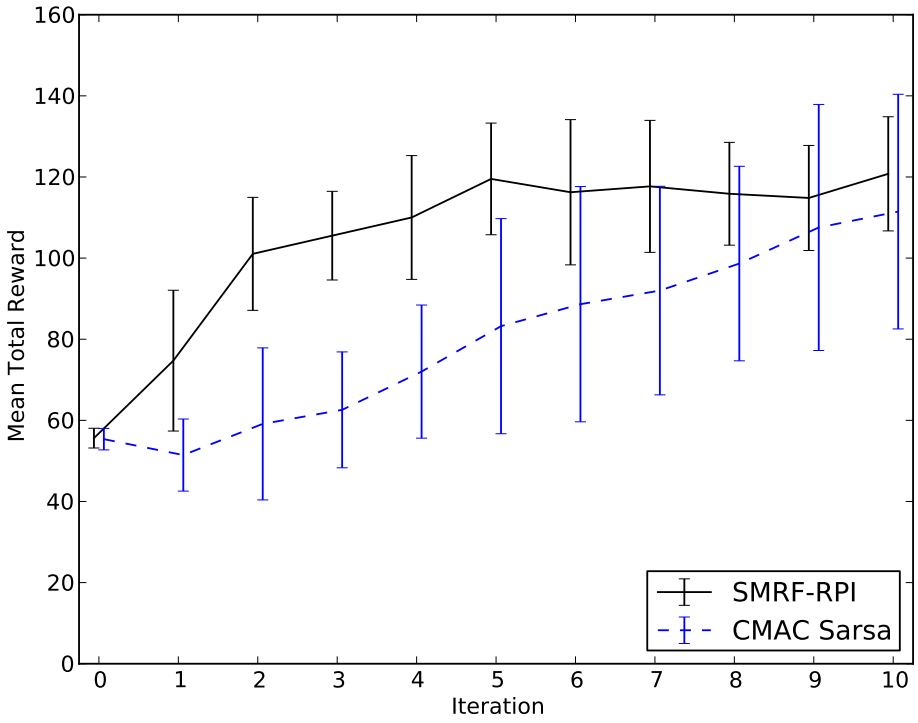


Figure 7.21: Mean total reward, across 30 independent runs, for both SMRF-RPI and CMAC Sarsa learning for 50 episodes per iteration. Error bars show standard deviation across runs.

ration at 500 episodes per batch. From the Sarsa perspective, where learning occurs online, this is merely 10 times the experience, and therefore provides a "zoomed out" view. Because SMRF-RPI actually works in batches, there could be additional, subtle effects. In particular, the entire first training batch of 500 SMRF-RPI episodes is entirely random play.

As stated earlier, higher performing SMRF-RPI policies often *are* better, rather than the performance being due merely to noise in execution. Subtracting out the mean test performance at each experience iteration, the SMRF-RPI runs for this task have high correlation between training and test performance ($R \approx 0.85$, $p < 10^{-91}$). In contrast, for CMAC Sarsa the training-test correlation, while significant, is lower

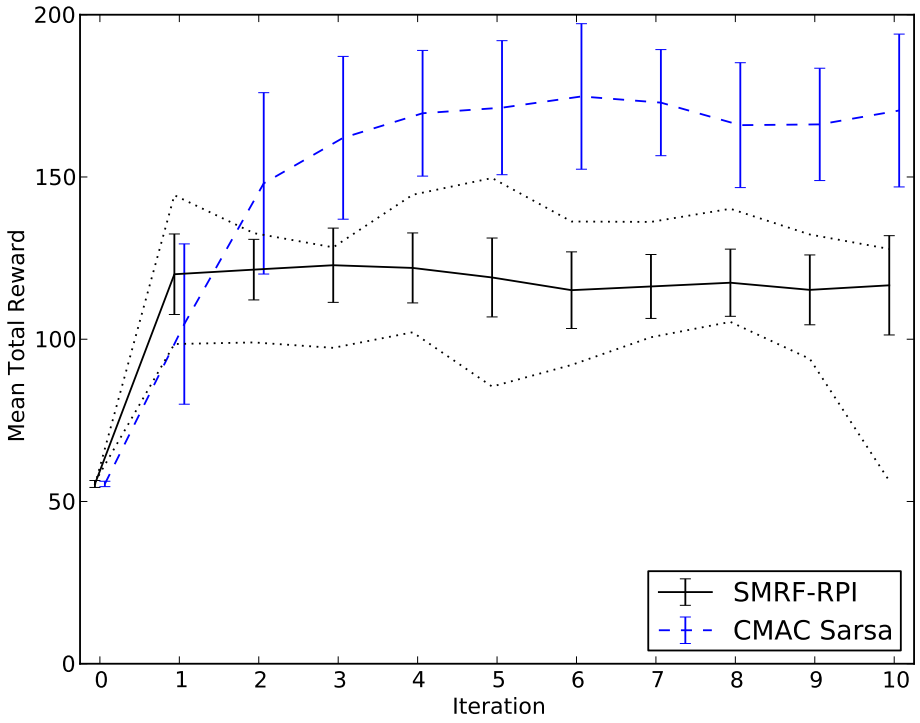


Figure 7.22: Mean total reward, across 30 independent runs, for both SMRF-RPI and CMAC Sarsa learning for 500 episodes per iteration. Dotted lines around SMRF-RPI results show the *test* performance of the runs with the maximum and minimum *training* performance for SMRF-RPI after each iteration. Error bars show standard deviation across runs.

($R \approx 0.20$, $p < 10^{-3}$). The better SMRF-RPI policies approach the lower end of the CMAC Sarsa range, as demonstrated by dotted margins in Figure 7.22. Still, this figure shows enough learning time to clearly demonstrate that CMAC Sarsa asymptotes at a higher level than SMRF-RPI ($p < 10^{-14}$). This leads to the question of why SMRF-RPI learns faster early on but then levels off too soon.

Potential explanations include: (1) that the policies SMRF-RPI learns early on prevent it from behaving in a fashion to learn better policies later, (2) that Sarsa is more effective than LSPI for policy learning, and (3) that SMRF-RPI learning fails to home in on key distinctions that enable higher performance. Other possibilities

may exist, but I investigate these explanations here.

As one method for investigating combined effects of both experience and RL algorithm, I conduct an experiment using the Sarsa implementation of Stone et al. (2006) with fixed, previously learned SMRF representations. No further representation learning occurs in this experiment. Specifically, I use representations after two iterations of SMRF-RPI with 500 episodes per batch. Each of 30 runs using SMRF representations in this experiment uses representations from a corresponding run with results shown in Figure 7.22. Further, this experiment uses the standard configuration where each agent learns independently, rather than sharing experience. Figure 7.23 shows the results of this experiment, where the horizontal axis is simulation time rather than corresponding to the number of episodes. Using simulation time is more common in previous work with keepaway when portraying the online learning process. Overall, this figure clearly shows the same general trend as before; the representations from SMRF-RPI learn quickly, outperforming CMAC at first ($p < 10^{-8}$), but are then surpassed ($p < 10^{-8}$ at the final time here, although the CMAC performance has yet to reach its asymptote). Of note, with SMRF representations, the online Sarsa learners almost immediately develop a certain level of competence. This suggests that a fair amount of domain knowledge is already encoded in the SMRF representations. That the representations have value for learning new policy weights also suggests that transfer of representations learned by SMRF-RPI might have value between similar tasks or domains. Also note, however, that the asymptote here for SMRF representations is below that of full SMRF-RPI, suggesting perhaps that Sarsa's performance is below that of LSPI. This experiment also suggests that perhaps the cause of both SMRF-RPI's higher initial performance and lower asymptotic performance is in the representations rather than in the RL method.

When comparing the performance of SMRF-RPI and CMAC Sarsa in this regard,

it is worth noting that for CMAC Sarsa, the features for standard keepaway learning (as detailed in Section 7.5.1), the CMAC tile sizes for each feature, and the number of tilings have all been hand selected for this task (Stone et al., 2005). Further, it is commonly understood in machine learning applications that feature engineering has clear impact on performance (Provost and Kohavi, 1998; Domingos, 2012). In contrast, SMRF-RPI must learn its own representations in a fashion general enough that the same algorithm can be applied to a variety of different tasks, including the other tasks addressed in this chapter. Compared to hand-selected features for each task, SMRF-RPI is bootstrapping more of its own learning process and relying less on human intelligence.

As a final comparison with CMAC Sarsa learning on keepaway, I also test transfer learning from the 3 vs. 2 case to 4 vs. 3. This equates to "small tested on large" case in Chapter 5. Figure 7.24 shows the results. For SMRF-RPI, this experiment tests untouched transfer from 3 vs. 2 representation and policies. For CMAC Sarsa, this experiment tests learning online from scratch as before. No transfer case exists for SMRF-RPI for iteration 0 because there is no policy to transfer. The asymptote SMRF-RPI achieves when learning from 500 episodes of random play is also seen in the 4 vs. 3 case. Importantly, transfered SMRF-RPI policies outperform CMAC Sarsa again in the early iterations ($p < 10^{-18}$ at iteration 1), despite having been learned in a different configuration of the task. Also, while, as in 3 vs. 2, CMAC Sarsa does eventually overtake SMRF-RPI ($p < 10^{-15}$), the highest-performing SMRF-RPI policies are closer to mean CMAC Sarsa performance than before. As stated previously, transfer to differing numbers of objects is one of the key benefits of a relational approach. For SMRF-RPI, this is the only transfer case I test, although results in Chapter 5 suggest that this expected benefit might be seen in other tasks as well. Of course, without carrying out additional experiments, it is hard to predict the exact quality of transfer for other tasks.

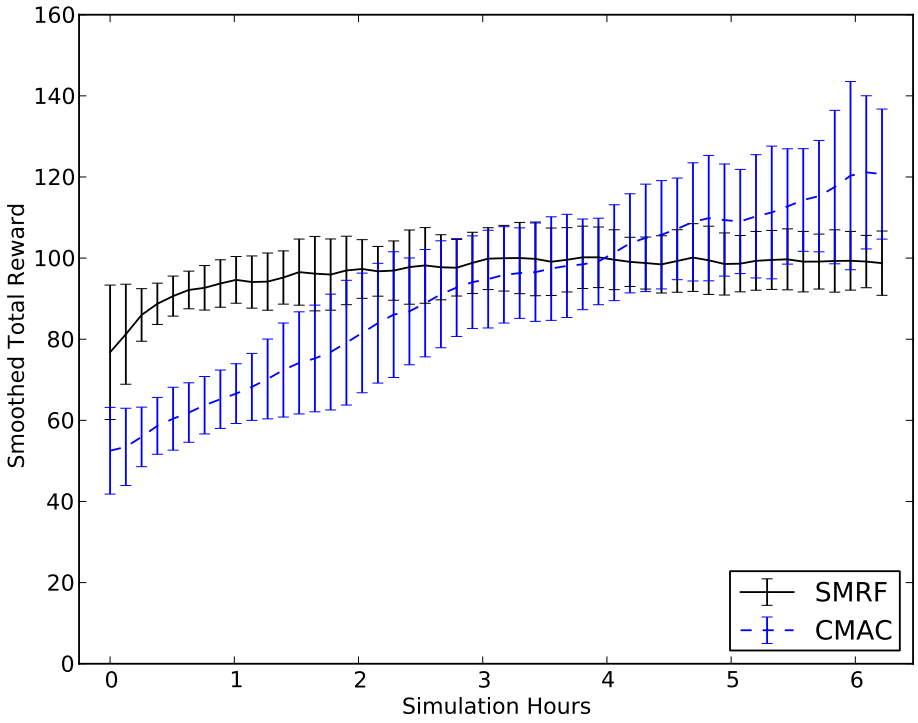


Figure 7.23: Smoothed total reward for Sarsa learning, by simulation time rather than episodes, for previously learned SMRF representations and for standard CMAC tilings. Rewards are smoothed by applying an acausal Gaussian kernel with standard deviation of 0.1 simulation hours for evenly spaced intervals, for each individual run. Error bars show standard deviation across runs.

SMRF-RPI Variants

One of the features of SMRF-RPI is the RPI loop that iterates multiple times for each batch of experience. Analytically, in some cases, this inner RPI loop is required for learning about multiple action types from a single batch of experience. For example, in the arch building task, this is required for learning representations for any action other than *done*. This is because there is no Bellman error to speak of for other actions until meaningful Q values exist for *done*.

In this section, I empirically investigate the consequences of this feature for the keepaway task. Figure 7.25 shows the mean total reward for different configurations

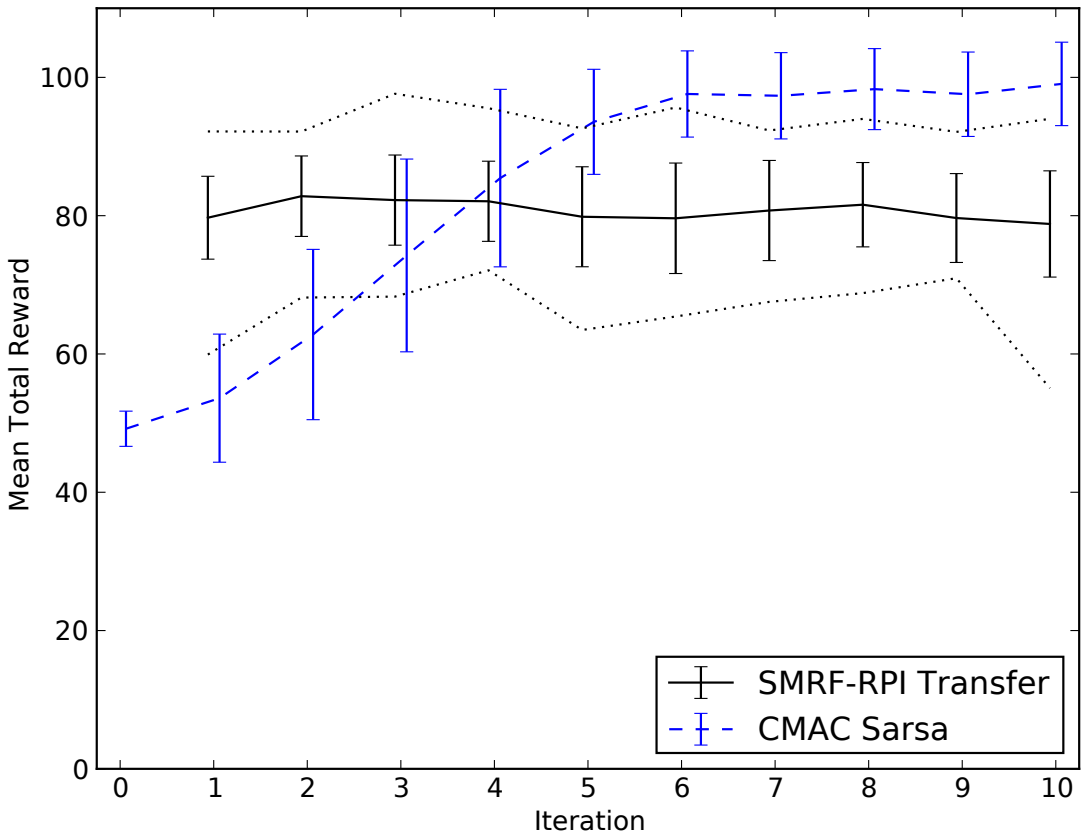


Figure 7.24: Mean total reward, across 30 independent runs, for SMRF-RPI transfer learning from 3 vs. 2 keepaway to 4 vs. 3 keepaway, with no additional learning, compared with CMAC Sarsa learning from scratch on 4 vs. 3 keepaway, using batches of 500 episodes each. Dotted lines around SMRF-RPI results show the *test* performance of the runs with the maximum and minimum *training* performance for SMRF-RPI after each iteration. Error bars show standard deviation across runs. Random play (iteration 0) results are missing for SMRF-RPI, because there is no policy to transfer.

of SMRF-RPI. In the legend, "multiple" and "single" iterations refer to the RPI iterations for a single iteration of experience. Also, "single tree" refers to a configuration where only one tree for positive and only one for negative is attempted for a single RPI iteration at each experience iteration. The main result here is that multiple RPI iterations help substantially for SMRF-RPI to take advantage of a batch of 500 episodes of random play. Specifically, after the first batch of experience, the "Multiple Iterations – 500 Episodes" configuration outperforms each other

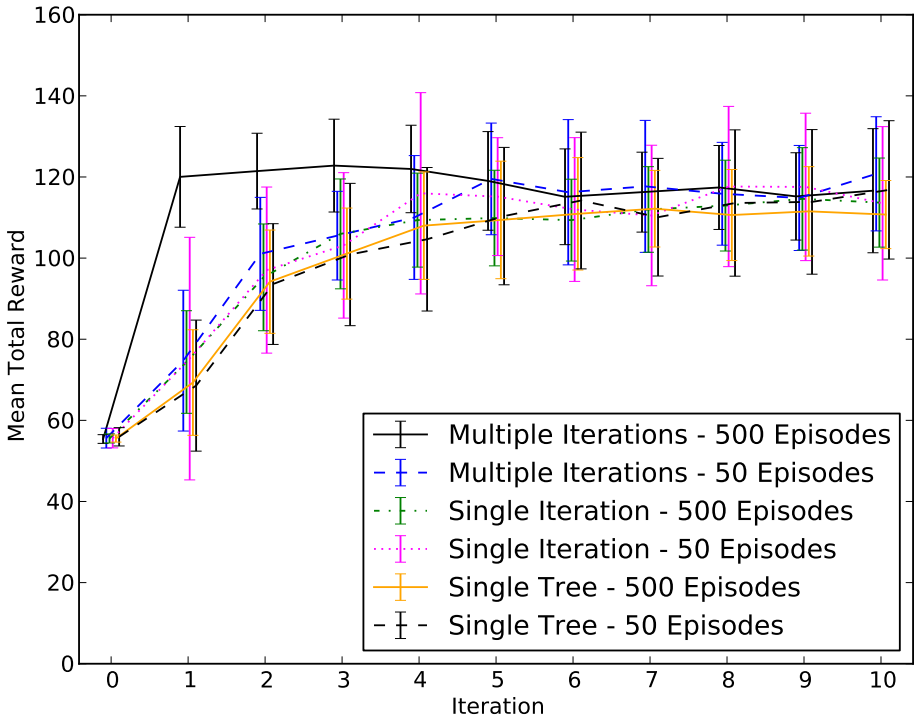


Figure 7.25: Mean total reward across 30 independent runs, for various configurations of SMRF-RPI on keepaway task. Error bars show standard deviation across runs.

configuration ($p < 10^{-8}$ for the least significant comparison, after correcting for multiple comparisons). By the final experience iteration, however, the "Multiple Iterations – 500 Episodes" configuration no longer has a significant lead over any other configuration ($p < 0.05$ at lowest, which is not significant after correcting for multiple comparisons).

Looking at cumulative leaf counts in Figure 7.26, the multiple RPI iterations versions both start and end with more leaves than the single RPI iteration versions ($p < 10^{-10}$ in the least significant case). Leaf count matters minimally because of execution speed, but concerns about excessive dimensionality are well understood in machine learning. Given equivalent performance, it is common to prefer simpler

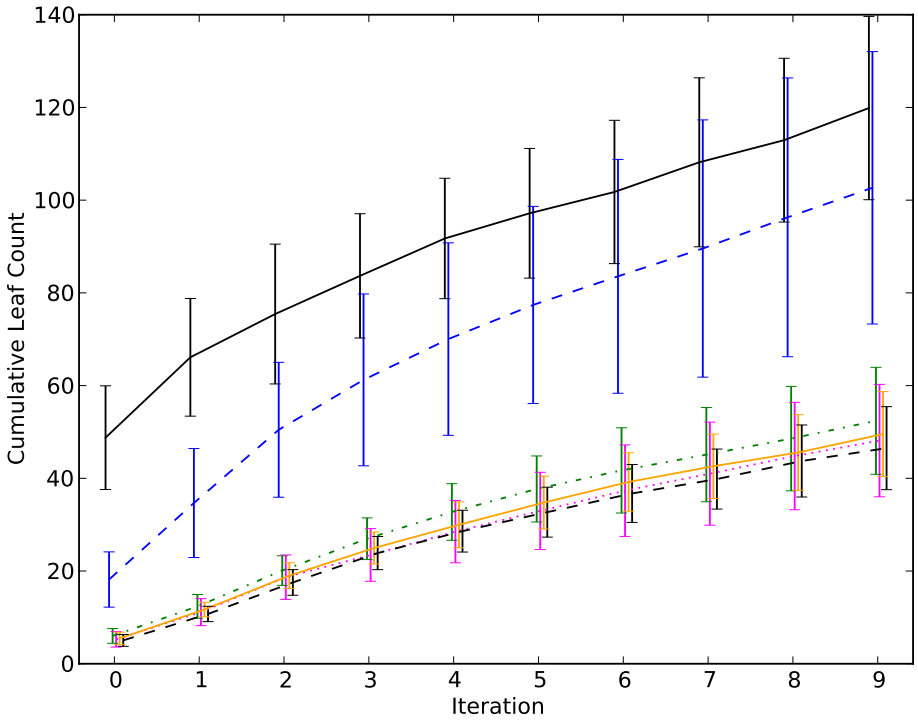


Figure 7.26: Cumulative leaf count, across 30 independent runs, for various configurations of SMRF-RPI on keepaway task. Error bars show standard deviation across runs.

representations. Results here show that repeated RPI can indeed help with performance, but some fine tuning to task needs might be helpful if representation size is truly a concern.

Chapter 8

Conclusion

8.1 Discussion

8.1.1 Reinforcement Learning in Implicitly Relational Worlds

Many real-world tasks require agents to interact with objects in the environment. Such objects might be items on a table, players in a team sport, or any others relevant to a task. Objects also have relationships between them. For example, one object might be *on* another, or an opponent might be *between* two teammates. As humans, we can describe these situations in discrete terms, but the relationships originally are implicit in the relative physical measures of the objects. What counts as "between" for one task might be different than for another. The number of objects in a situation, as well as the roles played by particular objects, might also change from time to time. This prevents immediate use of a simple, fixed list of variables with consistent meaning.

Instead, for an agent to perform tasks in such environments, it must be able to identify key variables from these objects and relations. Further, it should be possible for an agent to learn from its own experience how to perceive the world to accomplish its tasks. Reinforcement learning (RL) allows agents to learn how to accomplish multistep tasks from a scalar reward signal. Relational learning works with object relationships and also scales to differing numbers of objects by supporting quantifiers from first-order logic. For example, in soccer, does there exist a potential interceptor for the pass I want to make? This question is relevant no

matter how many players are on the field. Relational reinforcement learning (RRL) combines RL and relational learning (see, e.g., Džeroski et al., 2001; van Otterlo, 2005, 2012).

However, traditional relational learning methods do not emphasize continuous, multidimensional environments. Often, they rely on hand-crafted predicates (e.g., Pasula et al., 2007) or address continuous variables only in one dimension at a time (e.g., Blockeel and De Raedt, 1998). Those methods which seek to learn relations from physical data sometimes do so without the ability to reason over quantified variables during the relation learning process (e.g., Kulick et al., 2013).

In this dissertation, I present SMRF-RPI, a method for RRL in continuous, multidimensional domains. SMRF-RPI is an instantiation of the representation policy iteration (RPI) framework of Mahadevan (2005b), where the representation learning capabilities are provided by the spatiotemporal multidimensional relational framework (SMRF) of Bodenhamer (2014). In particular, SMRF-RPI emphasizes domains where no explicit relations exist in the data. Rather, objects have a variety of attributes, such as location, color, size, and orientation, and relations between objects must be inferred from relative measures. Using such attributes, the SMRF learning algorithm builds probabilistic decision trees for binary, existential classification. SMRF-RPI provides class labels based on the Bellman error (BE) of sample agent experience (see Parr et al., 2007; Wu and Givan, 2010). By these means, SMRF-RPI builds its own grounded, n-ary predicates from physical data in the process of learning to accomplish tasks.

Further, within SMRF-RPI, I present a method for sampling and evaluating actions with continuous parameters. Specifically, SMRF-RPI uses a beam search based on the learned Q function, which evaluates potential actions for a particular world state. For example, an agent might select from continuous goal locations for a pick-and-place task. My method differs from the few existing examples of continuous

actions for RRL. For example, Zaragoza and Morales (2009) use interpolation from discretized actions, and Kulick et al. (2013) present a continuous action for use in constructing examples for active learning of grounded predicates, but not in their relational planning task. In contrast, SMRF-RPI searches for possible actions in continuous space during the process of relational reinforcement learning.

Also, building on the earlier work of Bodenhamer (2014), I show in this work the effective use of ternary, reframe mapping functions. Other relational learning methods sometimes restrict relative measures to functions of two objects (e.g., Jetchev et al., 2013). Because objects might take on arbitary roles, permutations of object assignments need to be considered. SMRF's tree nature allows for pruning of these object *instantiation sequences* down different branches, thereby reducing the total number of permutations that needs to be considered in practice. This reduces the cost of functions of more than two parameters. Further, the tree nature allows for multiple questions of differing arity. Alternatively, to consider all relative measures in a single vector space, merely allowing for ternary relations requires all permutations of three objects to be present, even if in the end, for some cases, a binary relationship is all that matters. The vector space would also need to be of higher dimensionality to accommodate all the relative measures of varying arity under consideration. Trees simplify these concerns by dividing the larger question space into parts.

8.1.2 Experimental Results

In this dissertaton, I have demonstrated application of SMRF-RPI to a variety of tasks and domains with minimal tailoring of algorithm parameters. In addition to a 1D corridor domain, I test SMRF-RPI in three separate tasks in a simulated, physical 2D "blocks world" domain: building high towers, grouping blocks by similar colors, and building arches. The blocks world tasks address a variety of physical and

color relationships. Among these, the arch building task requires multiple steps of lookahead and also involves a *move* action with a continuous horizontal goal location parameter.

Further, using the standard RoboCup keepaway benchmark task (Stone et al., 2006), I have shown that SMRF-RPI can be sample efficient in learning initial representations. I compare the performance of SMRF-RPI with the existing online Sarsa learning algorithm coupled with a CMAC tiling representation and a hand-designed feature vector (Stone et al., 2005, 2006). SMRF-RPI outperforms CMAC Sarsa in the early stages of learning, but CMAC Sarsa asymptotes at a higher level. Experiments suggest that both differences are due to the representations in use. SMRF-RPI learns ellipsoidal representations of much lower dimensionality overall the the full CMAC tilings. However, the grid tilings can allow for more careful fine tuning.

Of note, the CMAC grid sizes and tilings, in addition to the features themselves, are also human-designed and tailored (Stone et al., 2005). On the other hand, SMRF-RPI bootstraps its own representation in a fashion that is also applicable to other domains, including for the blocks world tasks presented in this dissertation. As a result, it requires less human engineering effort to adapt to new tasks. This increased adaptability is a key benefit to my approach. I also show that the representations and policies learned by SMRF-RPI in 3 vs. 2 play transfer immediately to the 4 vs. 3 setting, outperforming the early learning of CMAC Sarsa. Such transfer a result of the existential nature of SMRF representations.

Finally, I demonstrate the effectiveness of SMRF-RPI's repeated iterations of representation policy learning between each batch of experience. In the arch building task, some runs substantially increase their performance using experience from the initial training demonstration. In the first iteration of RPI, the only Bellman error results from correct claims that an arch is completed, by use of the *done* action. Only

after this RPI interation does Bellman error indicate what actions and conditions lead to the state of a completed arch. A single batch of experience allows these multiple steps of learning, and mean performance across runs also continues to improve with further iterations of experience. In the case of the keepaway benchmark task, when a batch of experience is sufficiently large, multiple iterations of RPI are necessary to make full use of the available experience.

8.1.3 Additional Contributions

For SMRF to learn discrete relations from physical data, it needs a method for determining decision boundaries in multidimensional space. Mapping functions in SMRF extract a set of vectors for each scene from object instantiation sequences. For example, this set might be the relative positions of all pairs of objects, and the question becomes which of these is relevant to the task at hand? The scene as a whole is labeled, based on Bellman error for the case of SMRF-RPI, but the individual instance vectors are not. This existential classification problem is the common formulation of multiple instance learning (MIL, see Dietterich et al., 1997; Maron and Lozano-Pérez, 1997). In this dissertation, I present a novel MIL algorithm called covariant aggregation. In comparison with other MIL algorithms, covariant aggregation performs robustly with minimal parameter tuning, while also providing support for instance label prediction and simple yet covariance-sensitive decision boundaries. I have also empirically demonstrated its consistently fast execution in comparison with other capable methods. Covariant aggregation provides the decision volume learning method within the SMRF tree learning algorithm, and robustness and speed are important when repeatedly learning decision volumes for a variety of potential tree expansions.

Finally, as a prelude and potential complement to relational reinforcement learning, I also present a method for learning binary outcomes of actions in relational

settings. Commonly, this is question of action success or failure. For example, will a pass from player A to player B be successful? If so, the pass might be viewed as an *affordance* (Gibson, 1977) available to player A. Therefore, these predictive models indicate to an agent when particular actions might lead to successful outcomes in a decision-making process. I also demonstrate transfer of learned models to tasks with both more and fewer objects than present in the learning process.

8.2 Future Work

There are a number of potential directions for taking this work forward. This includes both focused performance matters in addition to big picture ideas. SMRF-RPI's performance, while encouraging, still asymptotes below optimal in all but the 1D corridor task, and some policies learned by SMRF-RPI perform better than others. Therefore, in this section, I begin with potential directions for improving performance. Next, looking at the larger picture, I discuss potential investigations into transfer of representations between tasks, as well as possible avenues for increasing the richness of the SMRF's representational capabilities.

First, there would be clear value in finding ways to achieve high performance more consistently. Differences between SMRF and CMAC representations in the keepaway task suggest that SMRF-RPI might need to draw finer distinctions in state space. Several other relational reinforcement learning algorithms use direct regression for approximating the Q function (Driessens et al., 2001; Driessens and Ramon, 2003; Driessens and Džeroski, 2005; Gärtner et al., 2003). In contrast, SMRF-RPI assigns binary labels before learning a tree. If the actual Bellman error value were present for each scene, then different branches could learn question nodes associated specifically with the subset of instantiation sequences present in each branch. Sophisticated regression might also include continuous models in the tree

itself (e.g., Vens et al., 2007). Another possible direction is that of policy gradient learning, which allows for stochastic policies and, thus, a smoother gradient in the learning process. Kersting and Driessens (2008) suggest that this allows for more consistent learning of higher-performing policies in their experiments.

Also, other mechanisms for feature selection might be appropriate to explore. Filter feature selection techniques (e.g., Guyon and Elisseeff, 2003; Brown et al., 2012) might be less processing intensive than, and at least as effective as, the wrapper method I use in this work. Also, no feature selection in this work actually evaluates trees on their performance in actual task execution. Rather, their effectiveness is inferred for the future based on previous experience. Feature selection based on actual performance relates to evolutionary computation methods (e.g., Girgin and Preux, 2008b; Verbancsics and Stanley, 2010), although the representation and policy learning components could still employ standard reinforcement learning techniques. Of course, testing direct performance of potential features might be less sample efficient than my current technique.

Looking beyond this matter, several other areas of RRL seem valuable for future research. In this dissertation, I have tested transfer only between differing numbers of objects. There could also be value in representation transfer between different types of tasks, such as for the *put* action, from the high towers to the color grouping task. Transfer might also apply to different agents in the same task. For example, the $pass(A, B)$ action in soccer contains information not only about passer A but also about receiver B. Representations and perhaps even Q functions learned for passing the ball might inform teammates about the best places to go to receive the ball.

There could also be value in addressing fundamental relational capabilities. For example, SMRF currently has no direct mechanism for mixed existential and universal quantification, whereas TILDE (Blockeel and De Raedt, 1998) at any branch

can ask about negative existence. A related topic is that of grouping (aggregation) and deconstruction. That is, object-hood can be somewhat arbitrary. A group of people or objects might matter as a single entity in a scene, or perhaps part of some object (edges, corners, etc.) might be best addressed independently. When addressing groups, universals and existentials become more interesting. For example, one might ask, are all the wheels on car X in good shape? This concern might also be visible in the color grouping task in Chapter 7 of this present work. The field of multi-relational data mining (MRDM) suggests approaches to this subject (Knobbe et al., 1999; Blockeel and Sebag, 2003). However, the intersection of MRDM and RL seems mostly unexplored.

Blockeel and Sebag (2003) also raise the important question of efficiency as the size of a search space increases. The space of questions and functions of object attributes is intractable. Bias and efficient search are vital. On this matter, both SMRF and TILDE prohibit arbitrary lookahead in tree growth (multiple, sequential questions added simultaneously) because of the exponential growth. On the other hand, some questions can only be answered in original attribute space by lookahead (see, e.g., Blockeel and De Raedt, 1997; Struyf et al., 2006). Lifelong learning (e.g., Thrun and Mitchell, 1995; Hawasly and Ramamoorthy, 2013) is perhaps one approach to learning the kinds of biases necessary to address these concerns. On another matter of efficiency, while SMRF attempts to prune instantiations before combinatorial expansion, some questions nodes must still filter large quantities of instantiation sequences. While only a small spatial area might be of interest, all instantiation sequences present must be tested in the current algorithm. Spatial indexing (e.g., Jensen et al., 2004) is a promising approach to make such queries more efficient.

Bibliography

Andrews, S., Tsochantaridis, I., and Hofmann, T. (2002). Support vector machines for multiple-instance learning. In *Advances in Neural Information Processing Systems (NIPS)*, pages 577–584.

Argall, B. D., Chernova, S., Veloso, M., and Browning, B. (2009). A survey of robot learning from demonstration. *Robotics and Autonomous Systems*, 57:469–483.

Auer, P. and Ortner, R. (2004). A boosting approach to multiple instance learning. In *European Conference on Machine Learning (ECML)*, volume 3201 of *Lecture Notes in Computer Science (LNCS)*, pages 63–74. Springer Berlin / Heidelberg.

Barto, A. G. (2013). Intrinsic motivation and reinforcement learning. In *Intrinsically Motivated Learning in Natural and Artificial Systems*, pages 17–47. Springer Berlin Heidelberg.

Bellman, R. (1957a). A Markovian decision process. *Indiana University Mathematics Journal (IUMJ)*, 6(4):679–684.

Bellman, R. E. (1957b). *Dynamic Programming*. Princeton University Press.

Bengio, Y. (2009). Learning deep architectures for AI. *Foundations and Trends in Machine Learning*, 2(1):1–127.

Blockeel, H. and De Raedt, L. (1997). Lookahead and discretization in ilp. In *Inductive Logic Programming*, volume 1297 of *Lecture Notes in Computer Science (LNCS)*, pages 77–84. Springer Berlin Heidelberg.

Blockeel, H. and De Raedt, L. (1998). Top-down induction of first-order logical decision trees. *Artificial Intelligence*, 101(1–2):285–297.

Blockeel, H., De Raedt, L., and Ramon, J. (1998). Top-down induction of clustering trees. In *International Conference on Machine Learning (ICML)*, pages 55–63.

Blockeel, H. and Sebag, M. (2003). Scalability and efficiency in multi-relational data mining. *ACM SIGKDD Explorations Newsletter*, 5(1):17–30.

Bodenhamer, M. (2014). *Learning Relational Concepts with the Spatiotemporal Multidimensional Relational Framework*. PhD dissertation, University of Oklahoma.

Bodenhamer, M., Bleckley, S., Fennelly, D., Fagg, A. H., and McGovern, A. (2009). Spatio-temporal multi-dimensional relational framework trees. In *IEEE International Conference on Data Mining (ICDM) Workshop on Spatial and Spatiotemporal Data Mining (SSTDM)*, pages 564–570.

Breiman, L. (2001). Random forests. *Machine Learning*, 45(1):5–32.

Brown, G., Pocock, A., Zhao, M.-J., and Luján, M. (2012). Conditional likelihood maximization: A unifying framework for information theoretic feature selection. *Journal of Machine Learning Research (JMLR)*, 13:27–66.

Brunskill, E., Leffler, B. R., Li, L., Littman, M. L., and Roy, N. (2009). Provably efficient learning with typed parametric models. *Journal of Machine Learning Research (JMLR)*, 10:1955–1988.

Chang, C.-C. and Lin, C.-J. (2011). LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2(3):27:1–27:27. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

Chen, Y., Bi, J., and Wang, J. Z. (2006). MILES: Multiple-instance learning via embedded instance selection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(12):1931–1947.

Cohn, D. A., Ghahramani, Z., and Jordan, M. I. (1996). Active learning with statistical models. *Journal of Artificial Intelligence Research (JAIR)*, 4:129–145.

Davies, S., Ng, A. Y., and Moore, A. (1998). Applying online search techniques to continuous-state reinforcement learning. In *AAAI Conference on Artificial Intelligence (AAAI)*, pages 753–760.

De Raedt, L. and Kersting, K. (2008). Probabilistic inductive logic programming. In De Raedt, L., Frasconi, P., Kersting, K., and Muggleton, S., editors, *Probabilistic Inductive Logic Programming*, volume 4911, pages 1–27. Springer Berlin / Heidelberg.

DeJong, G. F. (1994). Learning to plan in continuous domains. *Artificial Intelligence*, 65(1):71–141.

Dempster, A. P., Laird, N. M., and Rubin, D. B. (1977). Maximum likelihood estimation from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society, Series B*, 39(1):1–38.

Dietterich, T. G., Lathrop, R. H., and Lozano-Pérez, T. (1997). Solving the multiple instance learning problem with axis-parallel rectangles. *Artificial Intelligence*, 89(1–2):31–71.

Domingos, P. (2012). A few useful things to know about machine learning. *Communications of the ACM*, 55(10):78–87.

Doran, G. and Ray, S. (2014). A theoretical and empirical analysis of support vector machine methods for multiple-instance classification. *Machine Learning*, 97(1–2):79–102.

Driessens, K. and Džeroski, S. (2005). Combining model-based and instance-based learning for first order regression. In *International Conference on Machine Learning (ICML)*, pages 193–200.

Driessens, K. and Ramon, J. (2003). Relational instance based regression for relational reinforcement learning. In *International Conference on Machine Learning (ICML)*, pages 123–130.

Driessens, K., Ramon, J., and Blockeel, H. (2001). Speeding up relational reinforcement learning through the use of an incremental first order decision tree learner. In *European Conference on Machine Learning (ECML)*, volume 2167 of *Lecture Notes in Computer Science (LNCS)*, pages 97–108. Springer Berlin Heidelberg.

Džeroski, S., De Raedt, L., and Driessens, K. (2001). Relational reinforcement learning. *Machine Learning*, 43:7–52.

Fagg, A. H. (1993). Developmental robotics: A new approach to the specification of robot programs. In Bekey, G. A. and Goldberg, K. Y., editors, *Neural Networks in Robotics*, volume 202 of *The Springer International Series in Engineering and Computer Science*, pages 459–486. Springer.

Fan, R.-E., Chang, K.-W., Hsieh, C.-J., Wang, X.-R., and Lin, C.-J. (2008). LIBLINEAR: A library for large linear classification. *Journal of Machine Learning Research (JMLR)*, 9:1871–1874.

Fikes, R. E. and Nilsson, N. J. (1971). STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2:189–208.

Foulds, J. and Frank, E. (2008). Revisiting multiple-instance learning via embedded instance selection. In *Australasian Joint Conference on Artificial Intelligence*, volume 5360 of *Lecture Notes in Computer Science (LNCS)*, pages 300–310. Springer Berlin Heidelberg.

Foulds, J. and Frank, E. (2010). A review of multi-instance learning assumptions. *The Knowledge Engineering Review*, 25(1):1–25.

Friedman, J. H. (1977). A recursive partitioning decision rule for nonparametric classification. *IEEE Transactions on Computers*, C-26(4):404–408.

Gärtner, T., Driessens, K., and Ramon, J. (2003). Graph kernels and Gaussian processes for relational reinforcement learning. In Horváth, T. and Yamamoto, A., editors, *Inductive Logic Programming (ILP)*, volume 2835 of *Lecture Notes in Computer Science (LNCS)*, pages 146–163. Springer Berlin Heidelberg.

Gelman, A., Carlin, J. B., Stern, H. S., Dunson, D. B., Vehtari, A., and Rubin, D. B. (2013). *Bayesian Data Analysis*. Chapman and Hall/CRC.

Gibson, J. J. (1977). The theory of affordances. In Shaw, R. and Bransford, J., editors, *Perceiving, Acting, and Knowing: Toward an Ecological Psychology*, pages 67–82. Lawrence Erlbaum Associates.

Girgin, S. and Preux, P. (2008a). Basis expansion in natural actor critic methods. In *European Workshop on Reinforcement Learning (EWRL)*, volume 5323 of *Lecture Notes in Computer Science (LNCS)*, pages 110–123.

Girgin, S. and Preux, P. (2008b). Feature discovery in reinforcement learning using genetic programming. In *European Conference on Genetic Programming (EuroGP)*, volume 4971 of *Lecture Notes in Computer Science (LNCS)*, pages 218–229. Springer Berlin / Heidelberg.

Guyon, I. and Elisseeff, A. (2003). An introduction to variable and feature selection. *Journal of Machine Learning Research (JMLR)*, 3:1157–1182.

Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., and Witten, I. H. (2009). The WEKA data mining software: An update. *SIGKDD Explorations*, 11(1):10–18.

Hawasly, M. and Ramamoorthy, S. (2013). Lifelong transfer learning with an option hierarchy. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1341–1346.

Hornik, K., Stinchcombe, M., and White, H. (1989). Multilayer feedforward networks are universal approximators. *Neural Networks*, 2:359–366.

Huelsenbeck, J. P. and Crandall, K. A. (1997). Phylogeny estimation and hypothesis testing using maximum likelihood. *Annual Review of Ecology and Systematics*, 28:437–466.

Jensen, C. S., Lin, D., and Ooi, B. C. (2004). Query and update efficient B+-tree based indexing of moving objects. In *International Conference on Very Large Data Bases (VLDB)*, pages 768–779.

Jensen, D. D. and Cohen, P. R. (2000). Multiple comparisons in induction algorithms. *Machine Learning*, 38(3):309–338.

Jetchev, N., Lang, T., and Toussaint, M. (2013). Learning grounded relational symbols from continuous data for abstract reasoning. In *International Conference on Robotics and Automation (ICRA), Workshop on Autonomous Learning*.

Kandemir, M. and Hamprecht, F. A. (2014). Instance label prediction by Dirichlet process multiple instance learning. In *Uncertainty in Artificial Intelligence (UAI)*.

Kersting, K. and Driessens, K. (2008). Non-parametric policy gradients: A unified treatment of propositional and relational domains. In *International Conference on Machine Learning (ICML)*, pages 456–463.

Kinsman, T., Fairchild, M., and Pelz, J. (2012). Color is not a metric space. In *Western New York Image Processing Workshop (WNYIPW)*, pages 37–40. IEEE.

Knobbe, A. J., Siebes, A., and van der Wallen, D. (1999). Multi-relational decision tree induction. In *European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD)*, volume 1704 of *Lecture Notes in Computer Science (LNCS)*, pages 378–383. Springer Berlin Heidelberg.

Kolter, J. Z. and Ng, A. Y. (2009). Regularization and feature selection in least-squares temporal difference learning. In *International Conference on Machine Learning (ICML)*, pages 521–528.

Konidaris, G. D., Kaelbling, L. P., and Lozano-Pérez, T. (2014). Constructing symbolic representations for high-level planning. In *AAAI Conference on Artificial Intelligence (AAAI)*, pages 1932–1940.

Konidaris, G. D., Osentoski, S., and Thomas, P. (2011). Value function approximation in reinforcement learning using the Fourier basis. In *AAAI Conference on Artificial Intelligence (AAAI)*, pages 380–385.

Kulick, J., Toussaint, M., Lang, T., and Lopes, M. (2013). Active learning for teaching a robot grounded relational symbols. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1451–1457.

Lagoudakis, M. G. and Parr, R. (2003). Least-squares policy iteration. *Journal of Machine Learning Research (JMLR)*, 4:1107–1149.

Lang, T. and Toussaint, M. (2010). Planning with noisy probabilistic relational rules. *Journal of Artificial Intelligence Research (JAIR)*, 39:1–49.

Luo, M. R., Cui, G., and Rigg, B. (2001). The development of the CIE 2000 colour-difference formula: CIEDE2000. *Color Research and Application*, 26(5):340–350.

Mahadevan, S. (2005a). Proto-value functions: Developmental reinforcement learning. In *International Conference on Machine Learning (ICML)*, pages 553–560.

Mahadevan, S. (2005b). Representation policy iteration. In *Uncertainty in Artificial Intelligence (UAI)*, pages 372–379.

Mahadevan, S. (2007). Proto-value functions: A Laplacian framework for learning representation and control in Markov decision processes. *Journal of Machine Learning Research (JMLR)*, 8:2169–2231.

Mahadevan, S. and Liu, B. (2010). Basis function construction from power series expansions of value functions. In *Advances in Neural Information Processing Systems (NIPS)*, pages 1540–1548.

Mardia, K. V. (1975). Statistics of directional data. *Journal of the Royal Statistical Society, Series B*, 37(3):249–393.

Maron, O. and Lozano-Pérez, T. (1997). A framework for multiple-instance learning. In *Advances in Neural Information Processing Systems (NIPS)*, pages 570–576.

McGovern, A., Gagne, David J., I., Williams, J. K., Brown, R. A., and Basara, J. B. (2014). Enhancing understanding and improving prediction of severe weather through spatiotemporal relational learning. *Machine Learning*, 95(1):27–50.

McGovern, A., Hiers, N. C., Collier, M., Gagne II, D. J., and Brown, R. A. (2008). Spatiotemporal relational probability trees. In *IEEE International Conference on Data Mining (ICDM)*, pages 935–940.

Modayil, J. and Kuipers, B. (2008). The initial development of object knowledge by a learning robot. *Robotics and Autonomous Systems*, 56(11):879–890.

Mugan, J. and Kuipers, B. (2009). Autonomously learning an action hierarchy using a learned qualitative state representation. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1011–1016.

Muggleton, S. (1991). Inductive logic programming. *New Generation Computing*, 8(4):295–318.

Murphy, D. (2010). JBox2D: A Java physics engine. `http://www.jbox2d.org/`. Accessed August 2, 2010.

Natarajan, S., Joshi, S., Tadepalli, P., Kersting, K., and Shavlik, J. (2011). Imitation learning in relational domains: A functional-gradient boosting approach. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1414–1420.

Neville, J., Jensen, D., Friedland, L., and Hay, M. (2003). Learning relational probability trees. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 625–630.

Ng, A. Y., Harada, D., and Russell, S. (1999). Policy invariance under reward transformations: Theory and application to reward shaping. In *International Conference on Machine Learning (ICML)*, pages 278–287.

Oudeyer, P.-Y., Kaplan, F., and Hafner, V. V. (2007). Intrinsic motivation systems for autonomous mental development. *IEEE Transactions on Evolutionary Computation*, 11(2):265–286.

Palmer, T. J. (2011). Keepaway updates repository. `https://github.com/tjpalmer/keepaway`.

Palmer, T. J. (2015). Stackiter: 2D blocks world stacking simulator/toy for research. `https://github.com/tjpalmer/stackiter`.

Palmer, T. J., Bodenhamer, M., and Fagg, A. H. (2012). Learning to predict action outcomes in continuous, relational environments. In *IEEE International Conference on Development and Learning and on Epigenetic Robotics (ICDL-EpiRob)*. Electronically published.

Palmer, T. J. and Goodrich, M. A. (2002). Satisficing anytime action search for behavior-based voting. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 1013–1018.

Parr, R., Li, L., Taylor, G., Painter-Wakefield, C., and Littman, M. L. (2008). An analysis of linear models, linear value-function approximation, and feature selection for reinforcement learning. In *International Conference on Machine Learning (ICML)*, pages 752–759.

Parr, R., Painter-Wakefield, C., Li, L., and Littman, M. (2007). Analyzing feature generation for value-function approximation. In *International Conference on Machine Learning (ICML)*, pages 737–744.

Pasula, H. M., Zettlemoyer, L. S., and Kaelbling, L. P. (2007). Learning symbolic models of stochastic domains. *Journal of Artificial Intelligence Research (JAIR)*, 29:309–352.

Provost, F. and Kohavi, R. (1998). Guest editors' introduction: On applied research in machine learning. *Machine Learning*, 30(2–3):127–132.

Ridolfi, L., Gattass, M., and Lopes, H. (2010). Investigating Euclidean mappings for CIEDE2000 color difference formula. In *IS&T Color and Imaging Conference (CIC)*, pages 327–333.

Schaal, S. (1996). Learning from demonstration. In *Advances in Neural Information Processing Systems (NIPS)*, pages 1040–1046.

Schmidhuber, J. (2010). Formal theory of creativity, fun, and intrinsic motivation (1990–2010). *IEEE Transactions on Autonomous Mental Development*, 2(3):230–247.

Schmidhuber, J. (2015). Deep learning in neural networks: An overview. *Neural Networks*, 61:85–117.

Settles, B. (2009). Active learning literature survey. Technical Report Computer Sciences Technical Report 1648, University of Wisconsin–Madison.

Shen, W.-M. (1993). Discovery as autonomous learning from the environment. *Machine Learning*, 12:143–165.

Siegler, R. S. (1976). Three aspects of cognitive development. *Cognitive Psychology*, 8(4):481–520.

Singh, S., Barto, A. G., and Chentanez, N. (2004). Intrinsically motivated reinforcement learning. In *Advances in Neural Information Processing Systems (NIPS)*, pages 1281–1288.

Slaney, J. and Thiébaux (2001). Blocks World revisited. *Artificial Intelligence*, 125:119–153.

Smart, W. D. and Kaelbling, L. P. (2000). Practical reinforcement learning in continuous spaces. In *International Conference on Machine Learning (ICML)*, pages 903–910.

Stephenson, D. B. (2000). Use of the "odds ratio" for diagnosing forecast skill. *Weather and Forecasting*, 15(2):221–232.

Stone, P., Kuhlmann, G., Taylor, M. E., and Liu, Y. (2006). Keepaway soccer: From machine learning testbed to benchmark. In Bredenfeld, A., Jacoff, A., Noda, I., and Takahashi, Y., editors, *RoboCup 2005: Robot Soccer World Cup IX*, volume 4020 of *Lecture Notes in Computer Science (LNCS)*, pages 93–105. Springer Berlin / Heidelberg.

Stone, P., Sutton, R. S., and Kuhlmann, G. (2005). Reinforcement learning for robocup soccer keepaway. *Adaptive Behavior*, 13(3):165–188.

Struyf, J., Davis, J., and Page, D. (2006). An efficient approximation to lookahead in relational learners. In *European Conference on Machine Learning (ECML)*, volume 4212 of *Lecture Notes in Computer Science (LNCS)*, pages 775–782. Springer Berlin Heidelberg.

Stulp, F., Fedrizzi, A., Mösenlechner, L., and Beetz, M. (2012). Learning and reasoning with action-related places for robust mobile manipulation. *Journal of Artificial Intelligence Research (JAIR)*, 43:1–42.

Sutton, R. S. and Barto, A. G. (1998). *Reinforcement Learning: An Introduction*. The MIT Press.

Sutton, R. S., Precup, D., and Singh, S. (1999). Between MDPs and SMDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112:181–211.

Takeuchi, I., Le, Q. V., Sears, T. D., and Smola, A. J. (2006). Nonparametric quantile estimation. *Journal of Machine Learning Research (JMLR)*, 7:1231–1264.

Taylor, M. E., Whiteson, S., and Stone, P. (2006). Comparing evolutionary and temporal difference methods in a reinforcement learning domain. In *Conference on Genetic and Evolutionary Computation (GECCO)*, pages 1321–1328. ACM.

Thrun, S. and Mitchell, T. M. (1995). Lifelong robot learning. *Robotics and Autonomous Systems*, 15(1–2):25–46.

Toussaint, M., Plath, N., Lang, T., and Jetchev, N. (2010). Integrated motor control, planning, grasping and high-level reasoning in a blocks world using probabilistic inference. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 385–391.

van Hasselt, H. (2012). Reinforcement learning in continuous state and action spaces. In Wiering, M. and van Otterlo, M., editors, *Reinforcement Learning*, volume 12 of *Adaptation, Learning, and Optimization*, pages 207–251. Springer Berlin Heidelberg.

van Hasselt, H. and Wiering, M. A. (2007). Reinforcement learning in continuous action spaces. In *IEEE International Symposium on Approximate Dynamic Programming and Reinforcement Learning (ADPRL)*, pages 272–279.

van Otterlo, M. (2005). A survey of reinforcement learning in relational domains. Technical report, Centre for Telematics and Information Technology (CTIT) University of Twente.

van Otterlo, M. (2012). Solving relational and first-order logical Markov decision processes: A survey. In Wiering, M. and van Otterlo, M., editors, *Reinforcement Learning: State-of-the-Art*, volume 12, pages 253–292. Springer Berlin / Heidelberg.

Vens, C., Ramon, J., and Blockeel, H. (2007). ReMauve: A relational model tree learner. In Muggleton, S., Otero, R., and Tamaddoni-Nezhad, A., editors, *Inductive Logic Programming (ILP)*, volume 4455 of *Lecture Notes in Computer Science (LNCS)*, pages 424–438. Springer Berlin Heidelberg.

Verbancsics, P. and Stanley, K. O. (2010). Evolving static representations for task transfer. *Journal of Machine Learning Research (JMLR)*, 11:1737–1769.

Watkins, C. J. C. H. and Dayan, P. (1992). Q-learning. *Machine Learning*, 8:279–292.

Winston, P. H. (1970). Learning structural descriptions from examples. PhD thesis AITR-231, Massachusetts Institute of Technology (MIT).

Wu, J.-H. and Givan, R. (2010). Automatic induction of Bellman-error features for probabilistic planning. *Journal of Artificial Intelligence Research (JAIR)*, 38:687–755.

Xu, J. Z. and Laird, J. E. (2011). Combining learned discrete and continuous action models. In *AAAI Conference on Artificial Intelligence (AAAI)*, pages 1449–1454.

Zaragoza, J. H. and Morales, E. F. (2009). A two-stage relational reinforcement learning with continuous actions for real service robots. In *Mexican International Conference on Artificial Intelligence (MICAI)*, volume 5845 of *Lecture Notes in Computer Science (LNCS)*, pages 337–348. Springer Berlin Heidelberg.

Zhang, P. and Renz, J. (2014). Qualitative spatial representation and reasoning in Angry Birds: The extended rectangle algebra. In *International Conference on Principles of Knowledge Representation and Reasoning (KR)*.

Zhang, Q. and Goldman, S. A. (2001). EM-DD: An improved multiple-instance learning technique. In *Advances in Neural Information Processing Systems (NIPS)*, volume 2, pages 1073–1080.

Zhao, H., Cheng, J., Jiang, J., and Tao, D. (2013). Multiple instance learning via distance metric optimization. In *IEEE International Conference on Image Processing (ICIP)*, pages 2617–2621.

Appendix A

Additional Example Trees

This appendix demonstrates additional example trees learned by SMRF-RPI for some of the tasks discussed in Chapter 7. Example trees provide some intuition into the variety of representations learned by SMRF-RPI, although this still is a very small sampling of the many trees learned during experimental evaluation. Of note, the trees here are selected from arbitrary runs of SMRF-RPI. They do not necessarily represent either high performing, low performing, or typical cases.

A.1 High Towers Task

For the high towers task, Figures A.1 and A.2 show the trees learned in the second RPI iteration following the example trees in Section 7.4.1. Interestingly, both of these trees begin by asking about objects other than the action parameters. The $put(A, B)$ tree in Figure A.1 asks no questions about the parameters at all. While the size (extent) of object C can say nothing directly about A or B, there might be sequences of activity that cause the size or configuration of other blocks to be relevant. The $rotate(A)$ tree in Figure A.2 does ask a second question about the relative position of rotated block A and another block. The question places B above A just as for the *rotate* tree in Section 7.4.1, but in this case, there is a wide horizontal variance. Perhaps this tree in some ways allows refinement of the region described by the earlier tree.

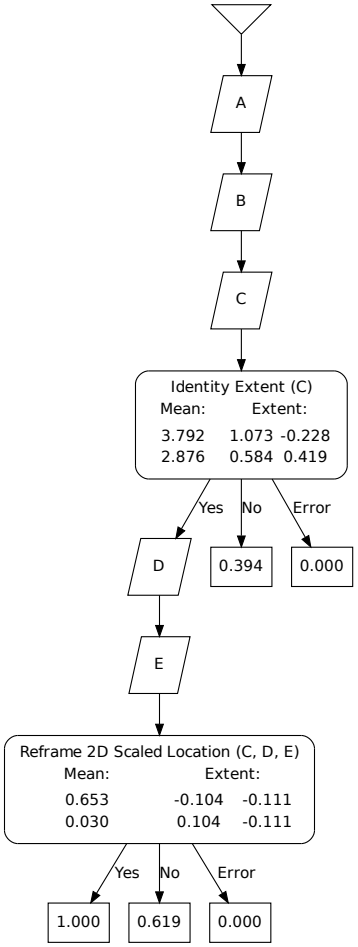


Figure A.1: Example tree for $put(A, B)$ on the high towers task for a second RPI iteration. In this case, the tree asks no questions about the action parameters at all. Extent columns show scaled eigenvectors.

A.2 Arch Building Task

For the arch building task, Figures A.3 and A.4 show the trees learned in the second RPI iteration following the example trees in Section 7.4.3. In this iteration, a "not" *done* tree, shown in Figure A.3, is included along with the previous tree for representing *done*, such that features from both trees are concatenated in the full representation. The new "not" *done* tree asks one question about all three blocks in the scene. The configuration space described by this reframe question likely refines the somewhat open-ended arch definition described by the *done* tree in Section 7.4.3.

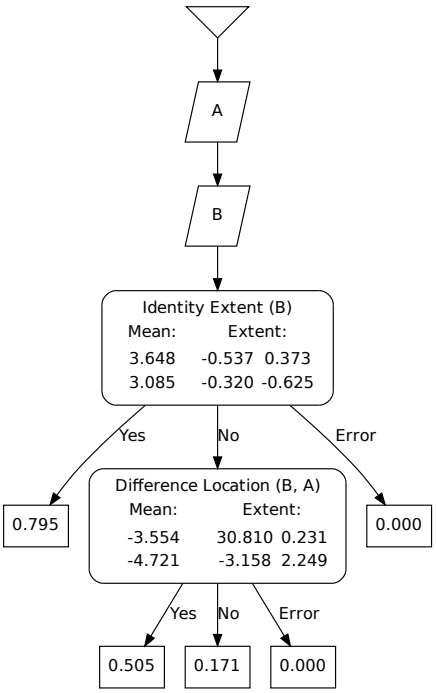


Figure A.2: Example tree for $rotate(A)$ on the high towers task for a second RPI iteration. While the tree begins asking about a block other than parameter A, it follows up the first *No* branch by asking about the relative position with A. Extent columns show scaled eigenvectors.

Because a *done* tree has already been added in the first RPI iteration, Bellman error now allows for learning the consequences of other actions. The newly learned and selected "not" $move(A, B)$ tree in Figure A.4, like the additional trees for the high stacks task above, asks only about blocks other than the one being moved. It does ask one question about target location B, but not at first. The first questions ask about whether an unmoved block C is horizontal, or at least not vertical. Perhaps this is also preventative. Moving one block when another is horizontal could risk destabilizing an existing arch. For blocks C that are vertical, additional questions are asked about both unmoved blocks that allow further understanding of the context in which the action is being taken.

In the third RPI iteration, a $rotate(A)$ tree is finally learned for the task, in addition to another *done* tree and another "not" $move(A, B)$ tree. These are shown

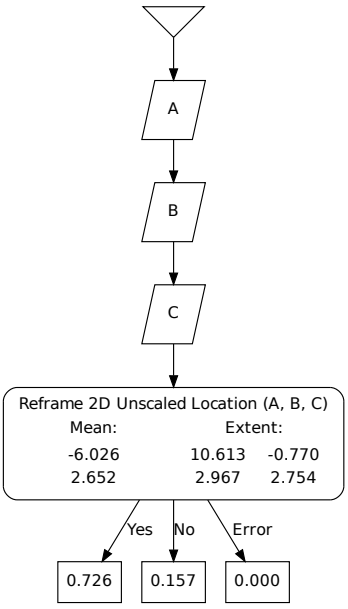


Figure A.3: Example tree for "not" *done* on the arch building task for a second RPI iteration, learned on negated labels. This tree asks about all three blocks with a single reframe question. Extent columns show scaled eigenvectors.

in Figures A.5, A.6, and A.7, respectively. The $rotate(A)$ tree, in Figure A.5, does not ask directly about the orientation of A. Instead, this tree looks for another block horizontally offset and whose center is about 2 units above A. All blocks for this task are the same size and shape, and this offset is the right amount to imply that A is horizontal and B is vertical. In this case, rotating A gives a second upright support. However, a higher probability leaf exists when some B *not* matching the previous condition is instead at a great distance (some 38 to 49 units) from A. Such a distance could be due to the random original placement at the beginning of the episode. Only when there exists no B matching either above condition is the third block queried.

The new *done* tree, in Figure A.6, asks for one block offset both horizontally and vertically from another, an apparent refinement of the arch condition. The new "not" $move(A, B)$, in Figure A.7, tree first examines whether A is horizontal. Note that, for its branch, instantiation C here represents the target move location, the

same as B does for its branch. Both are the second instantiation in their respective branches. When A is horizontal, the arrangement of both other blocks relative to the target location is considered. When A is vertical, there is merely a check to see if another block is near (some 4.5 to 8 units from) the target.

A.3 Keepaway Task

For the keepaway benchmark task, Figures A.8 and A.9 show the trees learned in the second RPI iteration following the example trees in Section 7.5.1. In the first RPI iteration, a "not" $pass(A, B)$ has already been learned, which provides additional lookahead into the consequences of hold actions. This is perhaps what allows learning of the "not" $hold(A)$ tree shown in Figure A.8. The scaled reframe suggests that a player C is much closer than B to player A, who has the ball. In the 20 $\times$ 20 meter field, C is no more than 3 meters away from A and possibly much closer. Usually, only an opponent would be so close to the active player. The radial density feature extracted from the question node also theoretically allows a continuous measure of *how* far the opponent is. Perhaps a scaled *distance* function would be more appropriate than the scaled location used here, but such a mapping function is not provided to the learning agent in this work.

Figure A.9 shows an additional learned "not" *pass* tree that is also selected at the second RPI iteration. As for the tree learned in the previous iteration (and discussed in Section 7.5.1), this tree seems to emphasize potential interceptors and perhaps refines the previous representation.

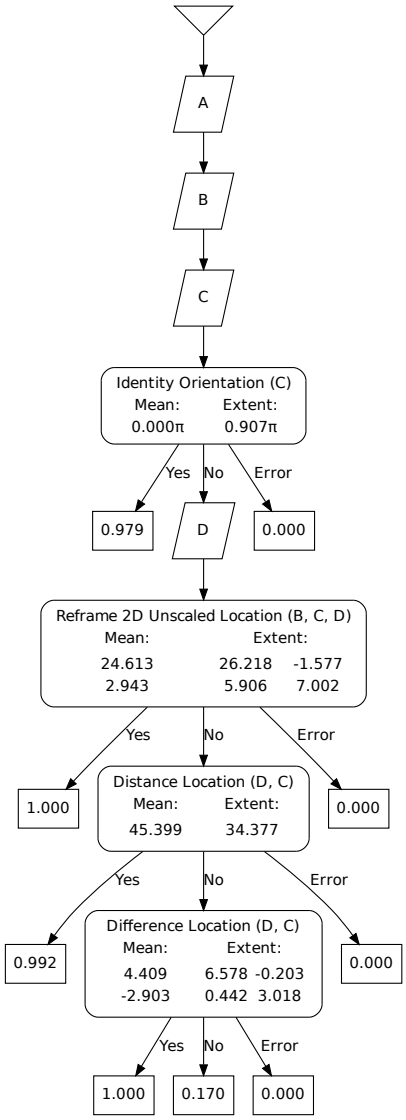


Figure A.4: Example tree for "not" $move(A, B)$ on the arch building task for a second RPI iteration, learned on negated labels. This tree emphasizes configuration of the blocks not being moved, although it also considers target location B. Extent columns show scaled eigenvectors.

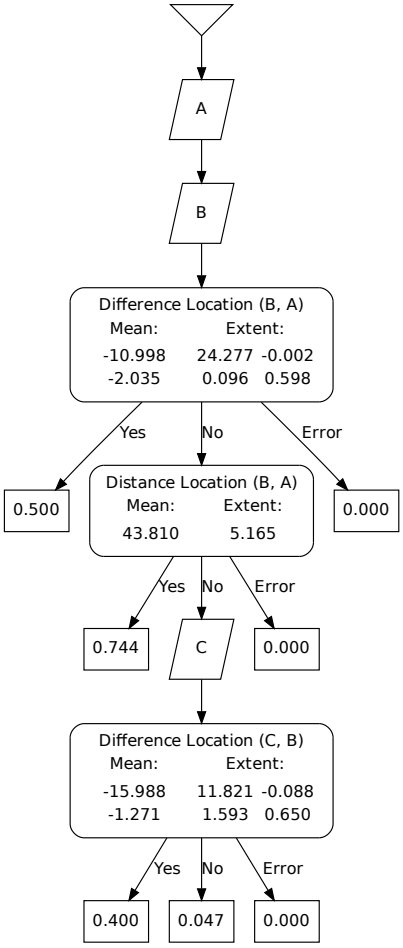


Figure A.5: Example tree for *rotate*(A) on the arch building task for a third RPI iteration. The first question asks about horizontal offset and also implies that A is horizontal and B vertical for the first *Yes* branch, by use of the vertical offset amount. Extent columns show scaled eigenvectors.

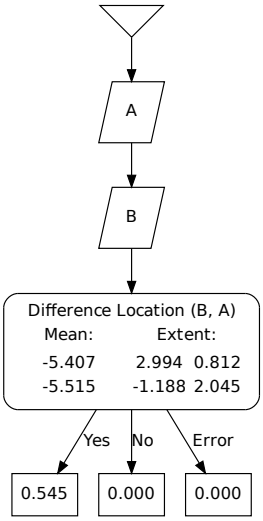


Figure A.6: Example tree for *done* on the arch building task for a third RPI iteration. This tree asks about one block offset both horizontally and vertically from another. Extent columns show scaled eigenvectors.

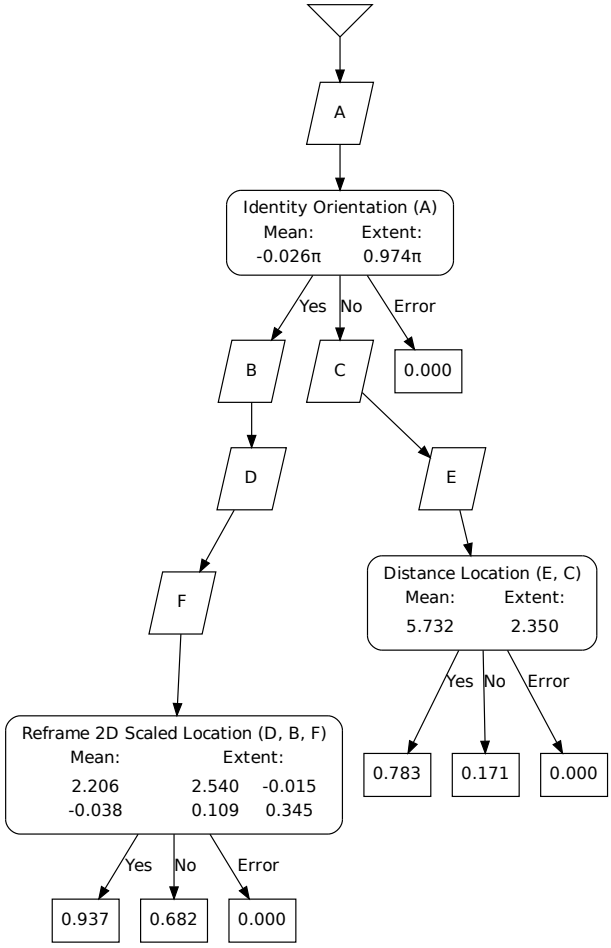


Figure A.7: Example tree for "not" $move(A, B)$ on the arch building task for a third RPI iteration, learned on negated labels. This tree has two different conditions, depending on whether A is horizontal or vertical. Either B or C represents the second parameter, depending on the branch. Extent columns show scaled eigenvectors.

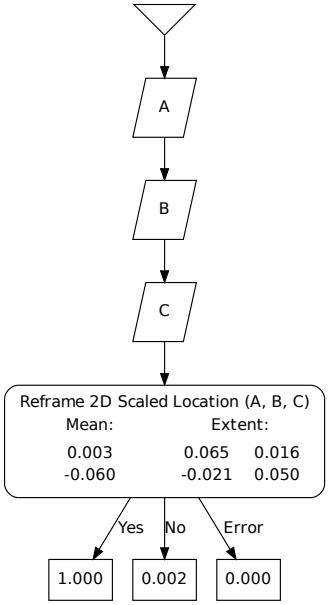


Figure A.8: Example tree for "not" $hold(A)$ on the keepaway task for a second RPI iteration, learned on negated labels. The scaled relative position suggests that another player, commonly an opponent, is very near player A, who has the ball. Extent columns show scaled eigenvectors.

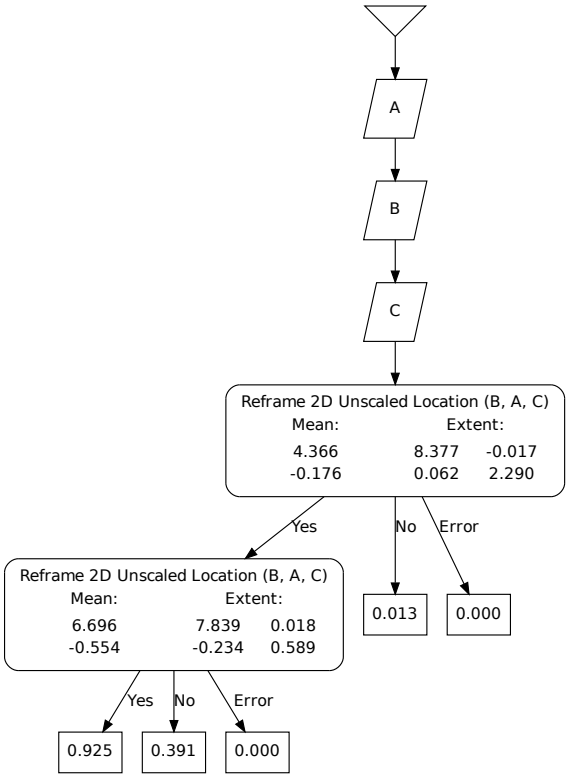


Figure A.9: Example tree for "not" $pass(A, B)$ on the keepaway task for a second RPI iteration, learned on negated labels. These questions focus on potential interceptors in somewhat different, but overlapping, regions. Extent columns show scaled eigenvectors.