This chapter presents the TEXPLORE algorithm, which uses the architecture presented in the previous chapter. First, TEXPLORE’s model learning approach is presented in Section 4.1. TEXPLORE utilizes a factored model, making a separate prediction about the next value of each state feature and reward. It builds decision trees to model each feature, enabling it to generalize the effects of actions across states. In Section 4.1.1, I describe how TEXPLORE’s decision tree models can be extended to regression tree models to model domains with continuous state. Next, in Section 4.1.2 I describe how TEXPLORE’s trees can model domains with sensor or actuator delays by providing them with the agent’s previous $k$ actions as additional inputs. I describe how to modify TEXPLORE’s model for domains with dependent feature transitions in Section 4.1.3.

Section 4.2 presents TEXPLORE’s approach to performing limited, targeted exploration. In TEXPLORE’s approach, the agent acts greedily with respect to a random forest model, which aggregates multiple decision tree models together. This approach enables the agent to balance each of its hypotheses of the true dynamics of the domain in a natural way. Then, I describe how the various components presented in this chapter along with the architecture from Chapter 3 can be combined into the full TEXPLORE algorithm. Finally, I summarize the chapter in Section 6.5.

While the parallel architecture presented in the previous chapter enables TEXPLORE to operate in real time, the algorithm must learn the task with high sample efficiency. This objective requires the agent to learn a model of the transition and reward functions in the domain very quickly, and explore intelligently to improve that model. I present TEXPLORE’s model learning in the next section, and its exploration in Section 4.2.

4.1 Model Learning

To learn a high quality behavior in few samples, TEXPLORE must learn an accurate model of the domain quickly. Although tabular models are a common approach, they require the agent to take every action from each state once (or multiple times in stochastic domains), since they learn a prediction for each state-action separately. Instead, TEXPLORE uses supervised learning techniques to generalize the
effects of actions across states, as has been done by some previous algorithms (De-
gris et al., 2006; Jong and Stone, 2007). Since the relative transition effects of ac-
tions are similar across states in many domains, TEXPLORE follows the approach of
Leffler et al. (2007) and Jong and Stone (2007) in predicting relative transitions
rather than absolute outcomes. In this way, model learning becomes a supervised
learning problem with \((s, a)\) as the input and \(s' - s\) and \(r\) as the outputs to be
predicted. Model learning is sped up by the ability of the supervised learner to
make predictions for unseen or infrequently visited states.

Like Dynamic Bayesian Network (DBN) based RL algorithms (Guestrin et al.,
2002; Strehl et al., 2007; Chakraborty and Stone, 2011), the algorithm learns a
model of the factored domain by learning a separate prediction for each of the
state features and the reward, as shown in Algorithm 4.1. The MDP model
is made up of \(n\) models to predict each feature (\(featModel_{1}\) to \(featModel_{n}\))
and a model to predict reward (\(rewardModel\)). Each model can be queried
for a prediction for a particular state-action (\(featModel \Rightarrow QUERY(\langle s, a \rangle)\))
or updated with a new training experience (\(featModel \Rightarrow UPDATE(\langle s, a, out \rangle)\)). In
TEXPLORE, each of these models is a random forest, shown later in Algorithm 4.3.

Algorithm 4.1 shows TEXPLORE’s model learning algorithm. It starts by cal-
culating the relative change in the state \((s^{rel})\) on Line 12, then it updates the
model for each feature with the new transition on Line 14 and updates the reward
model on Line 16. Like DBN-based algorithms, TEXPLORE assumes that each of
the state variables transitions independently (however, I present an extension
for dependent feature transitions in Section 4.1.3). Therefore, the separate fea-
ture predictions can be combined to create a prediction of the complete state
vector. The agent samples a prediction of the value of the change in each feature
on Line 23 and adds this vector, \(s^{rel}\), to \(s\) to get a prediction of \(s'\). The agent
then samples a prediction of reward (Line 27) and these sampled predictions are
returned for planning with MCTS.

We tested the applicability of several different supervised learning methods to
the task of learning an MDP model in previous work (Hester and Stone, 2009a).
Decision trees, committees of trees, random forests, support vector machines, neu-
ral networks, nearest neighbor, and tabular models were compared on their ability
to predict the transition and reward models across three toy domains after being
given a random sample of experiences in the domain. Decision tree based models
(single decision trees, committees of trees, and random forests) consistently pro-
vided the best results. Decision trees generalize broadly and refine their predic-
tions to smaller regions as they learn. Starting with a broad representation and
refining it over time has been shown to be effective in other areas such as value
function approximation (Munos and Moore, 2002). Another reason decision trees
perform well is that in many domains, the state space can be split into regions with
similar dynamics. For example, on a vehicle, the dynamics can be split into differ-
ent regions corresponding to which gear the car is in. Another advantage of using
decision trees is that they can learn context-specific feature independence, meaning
that they can learn that a prediction is independent of some features given that
other features have specific values (Boutilier et al., 2000).