TEXPLORE: Temporal Difference Reinforcement Learning for Robots and Time-Constrained Domains

Todd Hester

Learning Agents Research Group
Department of Computer Science
The University of Texas at Austin

Thesis Defense
December 3, 2012

Thesis Defense

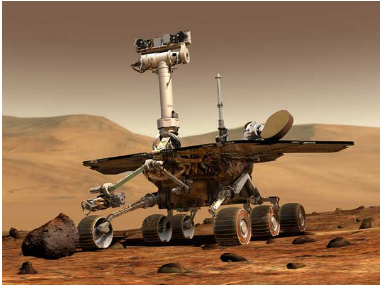
TEXPLORE: Temporal Difference Reinforcement Learning for Robots and Time-Constrained Domains

Todd Hester

Learning Agents Research Group
Department of Computer Science
The University of Texas at Austin

Thesis Defense
December 3, 2012

Robot Learning



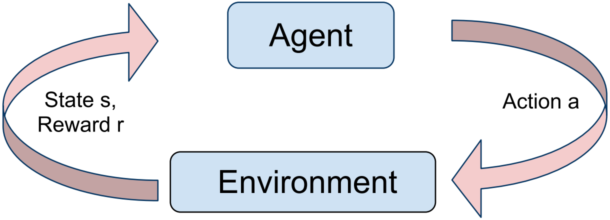
- Robots have the potential to solve many problems
- Moving from controlled to natural environments is difficult
- We need methods for them to learn and adapt to new situations

Robot Learning

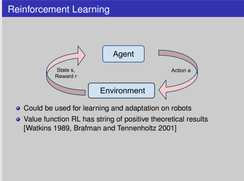
- Robots have the potential to solve many problems
- Moving from controlled to natural environments is difficult
- We need methods for them to learn and adapt to new situations

Robots have the potential to be very useful in society, by doing tasks that no one wants or is able to do. However, they are currently limited by the need to hand-code them for most tasks. Therefore, we need methods for them to learn from experience in the world.

Reinforcement Learning

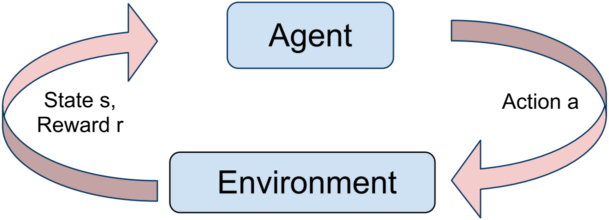


- Could be used for learning and adaptation on robots
- Value function RL has string of positive theoretical results [Watkins 1989, Brafman and Tennenholtz 2001]

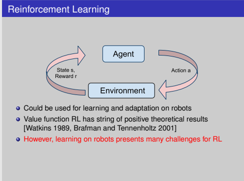


Reinforcement Learning is a paradigm for having an agent learn through interaction with the environment. In particular, value function RL has a long string of positive theoretical results that make it appear promising for learning on robots. However, learning on robots presents many challenges for RL.

Reinforcement Learning



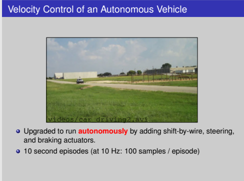
- Could be used for learning and adaptation on robots
- Value function RL has string of positive theoretical results [Watkins 1989, Brafman and Tennenholtz 2001]
- However, learning on robots presents many challenges for RL



Reinforcement Learning is a paradigm for having an agent learn through interaction with the environment. In particular, value function RL has a long string of positive theoretical results that make it appear promising for learning on robots. However, learning on robots presents many challenges for RL.

Velocity Control of an Autonomous Vehicle

- Upgraded to run **autonomously** by adding shift-by-wire, steering, and braking actuators.
- 10 second episodes (at 10 Hz: 100 samples / episode)



An example task for learning on a robot is learning to control the velocity of an autonomous vehicle. This example will be used throughout this presentation. Here the agent is controlling the gas and brake pedals of an autonomous vehicle, and learning to drive at different speeds.

Velocity Control

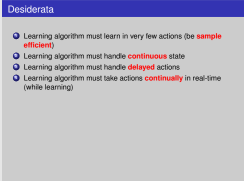
- State:
  - Current Velocity
  - Desired Velocity
  - Accelerator Pedal Position
  - Brake Pedal Position
- Actions:
  - Do nothing
  - Increase/decrease brake position by 0.1
  - Increase/decrease accelerator position by 0.1
- Reward: -10.0 * velocity error (m/s)

Velocity Control

- State:
  - Current Velocity
  - Desired Velocity
  - Accelerator Pedal Position
  - Brake Pedal Position
- Actions:
  - Do nothing
  - Increase/decrease brake position by 0.1
  - Increase/decrease accelerator position by 0.1
- Reward: -10.0 * velocity error (m/s)

Desiderata

1. Learning algorithm must learn in very few actions (be **sample efficient**)
2. Learning algorithm must handle **continuous** state
3. Learning algorithm must handle **delayed** actions
4. Learning algorithm must take actions **continually** in real-time (while learning)



Learning on a robot such as the car presents a number of challenges. Learning must be sample efficient, as the agent cannot take thousands or millions of actions to learn on a real robot. After that amount of time, the robot will have broken down or overheated. The agent must handle continuous state features. On many robots, rather than actions taking effect instantly, actuators have delay. On the autonomous vehicle, the brake is physically actuated with a cable pulling the pedal, causing significant delay before the brake moves to the desired position. Finally, we want the learning algorithm to continue acting in real-time as the agent is learning. It can not stop and think about the next action as the car approaches a red light or a vehicle in front of it stops.

Desiderata

1. Learning algorithm must learn in very few actions (be **sample efficient**)
2. Learning algorithm must handle **continuous** state
3. Learning algorithm must handle **delayed** actions
4. Learning algorithm must take actions **continually** in real-time (while learning)

Desiderata

- Learning algorithm must learn in very few actions (be **sample efficient**)
- Learning algorithm must handle **continuous** state
- Learning algorithm must handle **delayed** actions
- Learning algorithm must take actions **continually** in real-time (while learning)

Learning on a robot such as the car presents a number of challenges. Learning must be sample efficient, as the agent cannot take thousands or millions of actions to learn on a real robot. After that amount of time, the robot will have broken down or overheated. The agent must handle continuous state features. On many robots, rather than actions taking effect instantly, actuators have delay. On the autonomous vehicle, the brake is physically actuated with a cable pulling the pedal, causing significant delay before the brake moves to the desired position. Finally, we want the learning algorithm to continue acting in real-time as the agent is learning. It can not stop and think about the next action as the car approaches a red light or a vehicle in front of it stops.

Desiderata

1. Learning algorithm must learn in very few actions (be **sample efficient**)
2. Learning algorithm must handle **continuous** state
3. Learning algorithm must handle **delayed** actions
4. Learning algorithm must take actions **continually** in real-time (while learning)

Desiderata

- Learning algorithm must learn in very few actions (be **sample efficient**)
- Learning algorithm must handle **continuous** state
- Learning algorithm must handle **delayed** actions
- Learning algorithm must take actions **continually** in real-time (while learning)

Learning on a robot such as the car presents a number of challenges. Learning must be sample efficient, as the agent cannot take thousands or millions of actions to learn on a real robot. After that amount of time, the robot will have broken down or overheated. The agent must handle continuous state features. On many robots, rather than actions taking effect instantly, actuators have delay. On the autonomous vehicle, the brake is physically actuated with a cable pulling the pedal, causing significant delay before the brake moves to the desired position. Finally, we want the learning algorithm to continue acting in real-time as the agent is learning. It can not stop and think about the next action as the car approaches a red light or a vehicle in front of it stops.

Common Approaches

Algorithm	Citation	Sample Efficient	Real Time	Continuous	Delay
R-MAX	Brafman and Tennenholtz, 2001	Yes	No	No	No
Q-LEARNING	Watkins, 1989	No	Yes	No	No
with F.A.	Sutton and Barto, 1998	No	Yes	Yes	No
SARSA	Rummery and Niranjan, 1994	No	Yes	No	No
PILCO	Deisenroth and Rasmussen, 2011	Yes	No	Yes	No
NAC	Peters and Schaal 2008	Yes	No	Yes	No
BOSS	Asmuth et al., 2009	Yes	No	No	No
Bayesian DP	Strens, 2000	Yes	No	No	No
MBBE	Dearden et al., 2009	Yes	No	No	No
SPITI	Degris et al., 2006	Yes	No	No	No
MBS	Walsh et al., 2009	Yes	No	No	Yes
U-TREE	McCallum, 1996	Yes	No	No	Yes
DYNA	Sutton, 1990	Yes	Yes	No	No
DYNA-2	Silver et al., 2008	Yes	Yes	Yes	No
KWIK-LR	Strehl and Littman, 2007	Yes	No	Partial	No
FITTED R-MAX	Jong and Stone, 2007	Yes	No	Yes	No
DRE	Nouri and Littman 2010	Yes	No	Yes	No

Common Approaches

Algorithm	Citation	Sample Efficient	Real Time	Continuous	Delay
R-MAX	Brafman and Tennenholtz, 2001	Yes	No	No	No
Q-LEARNING	Watkins, 1989	No	Yes	No	No
with F.A.	Sutton and Barto, 1998	No	Yes	Yes	No
SARSA	Rummery and Niranjan, 1994	No	Yes	No	No
PILCO	Deisenroth and Rasmussen, 2011	Yes	No	Yes	No
NAC	Peters and Schaal 2008	Yes	No	Yes	No
BOSS	Asmuth et al., 2009	Yes	No	No	No
Bayesian DP	Strens, 2000	Yes	No	No	No
MBBE	Dearden et al., 2009	Yes	No	No	No
SPITI	Degris et al., 2006	Yes	No	No	No
MBS	Walsh et al., 2009	Yes	No	No	Yes
U-TREE	McCallum, 1996	Yes	No	No	Yes
DYNA	Sutton, 1990	Yes	Yes	No	No
DYNA-2	Silver et al., 2008	Yes	Yes	Yes	No
KWIK-LR	Strehl and Littman, 2007	Yes	No	Partial	No
FITTED R-MAX	Jong and Stone, 2007	Yes	No	Yes	No
DRE	Nouri and Littman 2010	Yes	No	Yes	No

There are a number of RL algorithms that have addressed some of these challenges at least partially, but not have addressed all four challenges together. In this thesis, I present TEXPLORE, the first algorithm to address all four challenges together. Of course, many of these algorithms were not attempting to address the problem of learning on robots. In particular, many of these methods focus on learning an optimal policy, which requires the agent to try every action from every state in the world, to guarantee it does not miss some high-rewarding state-action. We can look more at this issue through the sample complexity of exploration.

Common Approaches

Algorithm	Citation	Sample Efficient	Real Time	Continuous	Delay
R-MAX	Brafman and Tennenholtz, 2001	Yes	**No**	**No**	**No**
Q-LEARNING	Watkins, 1989	**No**	Yes	**No**	**No**
with F.A.	Sutton and Barto, 1998	**No**	Yes	Yes	**No**
SARSA	Rummery and Niranjan, 1994	**No**	Yes	**No**	**No**
PILCO	Deisenroth and Rasmussen, 2011	Yes	**No**	Yes	**No**
NAC	Peters and Schaal 2008	Yes	**No**	Yes	**No**
BOSS	Asmuth et al., 2009	Yes	**No**	**No**	**No**
Bayesian DP	Strens, 2000	Yes	**No**	**No**	**No**
MBBE	Dearden et al., 2009	Yes	**No**	**No**	**No**
SPITI	Degris et al., 2006	Yes	**No**	**No**	**No**
MBS	Walsh et al., 2009	Yes	**No**	**No**	Yes
U-TREE	McCallum, 1996	Yes	**No**	**No**	Yes
DYNA	Sutton, 1990	Yes	Yes	**No**	**No**
DYNA-2	Silver et al., 2008	Yes	Yes	Yes	**No**
KWIK-LR	Strehl and Littman, 2007	Yes	**No**	**Partial**	**No**
FITTED R-MAX	Jong and Stone, 2007	Yes	**No**	Yes	**No**
DRE	Nouri and Littman 2010	Yes	**No**	Yes	**No**
TEXPLORE	This thesis	Yes	Yes	Yes	Yes

Common Approaches

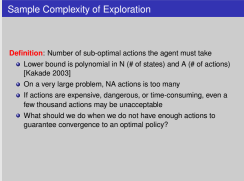
Algorithm	Citation	Sample Efficient	Real Time	Continuous	Delay
R-MAX	Brafman and Tennenholtz, 2001	Yes	No	No	No
Q-LEARNING	Watkins, 1989	No	Yes	No	No
with F.A.	Sutton and Barto, 1998	No	Yes	Yes	No
SARSA	Rummery and Niranjan, 1994	No	Yes	No	No
PILCO	Deisenroth and Rasmussen, 2011	Yes	No	Yes	No
NAC	Peters and Schaal 2008	Yes	No	Yes	No
BOSS	Asmuth et al., 2009	Yes	No	No	No
Bayesian DP	Strens, 2000	Yes	No	No	No
MBBE	Dearden et al., 2009	Yes	No	No	No
SPITI	Degris et al., 2006	Yes	No	No	No
MBS	Walsh et al., 2009	Yes	No	No	Yes
U-TREE	McCallum, 1996	Yes	No	No	Yes
DYNA	Sutton, 1990	Yes	Yes	No	No
DYNA-2	Silver et al., 2008	Yes	Yes	Yes	No
KWIK-LR	Strehl and Littman, 2007	Yes	No	Partial	No
FITTED R-MAX	Jong and Stone, 2007	Yes	No	Yes	No
DRE	Nouri and Littman 2010	Yes	No	Yes	No
TEXPLORE	This thesis	Yes	Yes	Yes	Yes

There are a number of RL algorithms that have addressed some of these challenges at least partially, but not have addressed all four challenges together. In this thesis, I present TEXPLORE, the first algorithm to address all four challenges together. Of course, many of these algorithms were not attempting to address the problem of learning on robots. In particular, many of these methods focus on learning an optimal policy, which requires the agent to try every action from every state in the world, to guarantee it does not miss some high-rewarding state-action. We can look more at this issue through the sample complexity of exploration.

Sample Complexity of Exploration

Definition: Number of sub-optimal actions the agent must take

- Lower bound is polynomial in N (# of states) and A (# of actions) [Kakade 2003]
- On a very large problem, NA actions is too many
- If actions are expensive, dangerous, or time-consuming, even a few thousand actions may be unacceptable
- What should we do when we do not have enough actions to guarantee convergence to an optimal policy?



The sample complexity of exploration says that an agent must take every action from every state at least once to guarantee it can learn a near-optimal policy. However, on robots, this number if very large, and actions are very expensive. What can we do in this case?

Thesis Question

Thesis Question

How should an online reinforcement learning agent act in time-constrained domains?

Thesis Question

Thesis Question

How should an online reinforcement learning agent act in time-constrained domains?

Thesis Question

Thesis Question

How should an online reinforcement learning agent act in time-constrained domains?

- Takes actions continually at specified frequency (not batch mode)
- Concerned with reward during learning (not just final policy)

Thesis Question

Thesis Question

How should an online reinforcement learning agent act in time-constrained domains?

- Takes actions continually at specified frequency (not batch mode)
- Concerned with reward during learning (not just final policy)

Thesis Question

Thesis Question

How should an online reinforcement learning agent act in time-constrained domains?

- Agent has a limited number of time steps
- Not enough time steps to learn optimal policy without some assumptions

2013-05-16

Thesis Defense
└─Introduction
  └─Motivation
    └─Thesis Question

Thesis Question

Thesis Question

How should an online reinforcement learning agent act in time-constrained domains?

- Agent has a limited number of time steps
- Not enough time steps to learn optimal policy without some assumptions

Solution

Model-Based Method

- Learn transition and reward dynamics, then update value function using model
- Typically more sample-efficient than model-free approaches
- Can update action-values without taking real actions in the world

Model-Based Method

- Learn transition and reward dynamics, then update value function using model
- Typically more sample-efficient than model-free approaches
- Can update action-values without taking real actions in the world

We take a two-pronged approach to this problem. First, we want to use a model-based method because they are typically more sample-efficient than model-free approaches. However, they can take significant computation time, so we combine this model learning approach with a real-time architecture.

Solution

Model-Based Method

- Learn transition and reward dynamics, then update value function using model
- Typically more sample-efficient than model-free approaches
- Can update action-values without taking real actions in the world
- **But,** can take significant computation time

Model-Based Method

- Learn transition and reward dynamics, then update value function using model
- Typically more sample-efficient than model-free approaches
- Can update action-values without taking real actions in the world
- **But,** can take significant computation time

We take a two-pronged approach to this problem. First, we want to use a model-based method because they are typically more sample-efficient than model-free approaches. However, they can take significant computation time, so we combine this model learning approach with a real-time architecture.

Solution

Model-Based Method

- Learn transition and reward dynamics, then update value function using model
- Typically more sample-efficient than model-free approaches
- Can update action-values without taking real actions in the world
- **But,** can take significant computation time

Real-Time Architecture

- Parallelize model learning, planning, and acting onto 3 parallel threads
- Utilize an **anytime** sample-based planning algorithm

Solution

Model-Based Method

- Learn transition and reward dynamics, then update value function using model
- Typically more sample-efficient than model-free approaches
- Can update action-values without taking real actions in the world
- **But,** can take significant computation time

Real-Time Architecture

- Parallelize model learning, planning, and acting onto 3 parallel threads
- Utilize an **anytime** sample-based planning algorithm

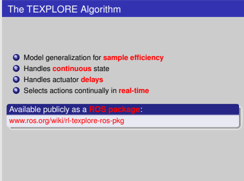
We take a two-pronged approach to this problem. First, we want to use a model-based method because they are typically more sample-efficient than model-free approaches. However, they can take significant computation time, so we combine this model learning approach with a real-time architecture.

The TEXPLORE Algorithm

1. Model generalization for **sample efficiency**
2. Handles **continuous** state
3. Handles actuator **delays**
4. Selects actions continually in **real-time**

Available publicly as a ROS package:

www.ros.org/wiki/rl-texplore-ros-pkg



The TEXPLORE algorithm addresses all four of these challenges. In addition, we have released it as a Robot Operating System (ROS) package, making it easy to integrate TEXPLORE into systems already running ROS.

Now that I have motivated the problem, in the next section I will provide some background. Then I will describe the TEXPLORE algorithm, followed by empirical evaluations of the algorithm. Then I will discuss some recent work on various approaches for exploration, before concluding.

Time-Constrained Domains

- For many practical problems, agent cannot take thousands of actions
- Actions may be expensive, time-consuming, or dangerous
- Agent does not have enough actions to guarantee it can learn an optimal policy
- Define domains that have this property as **time-constrained domains**

Sample Complexity of Exploration

Definition: Number of sub-optimal actions the agent must take

- Proven lower bound: $O(\frac{NA}{\epsilon(1-\gamma)} log(\frac{1}{\delta}))$
- For deterministic domains: $O(\frac{NA}{(1-\gamma)})$ [Kakade 2003]

Time-Constrained Domains

- For many practical problems, agent cannot take thousands of actions
- Actions may be expensive, time-consuming, or dangerous
- Agent does not have enough actions to guarantee it can learn an optimal policy
- Define domains that have this property as **time-constrained domains**

Sample Complexity of Exploration

Definition: Number of sub-optimal actions the agent must take

- Proven lower bound: $O(\frac{NA}{\epsilon(1-\gamma)} log(\frac{1}{\delta}))$
- For deterministic domains: $O(\frac{NA}{(1-\gamma)})$ [Kakade 2003]

In particular, we are going to focus on domains where the agent does not have enough actions to guarantee it can learn an optimal policy. We define these domains as **time-constrained domains**.

Time-Constrained Domains

- Lifetime L bounds the number of actions agent can take
- Time-Constrained if $L < 2NA$
- Two orders of magnitude less than lower bound
- The agent does **not** have enough time steps to learn the optimal policy without some additional assumptions about the domain
- **Assumption:** Transition and reward are **similar** across states

Domain	No. States	No. Actions	No. State-Actions	Min Bound Deterministic	Min Bound Stochastic	Maximum L
Taxi	500	6	3,000	300,000	1,050,000	6,000
Four Rooms	100	4	400	40,000	140,000	800
Two Rooms	51	4	204	20,400	72,400	408
Fuel World	39,711	8	317,688	31,768,800	111,190,800	635,376
Mountain Car	10,000	3	30,000	300,000	10,500,000	60,000
Puddle World	400	4	1,600	160,000	560,000	3,200
Cart-Pole Balancing	160,000	2	320,000	32,000,000	11,200,000	640,000

Time-Constrained Domains

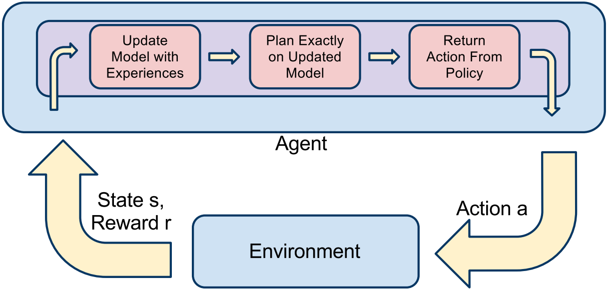
- Lifetime L bounds the number of actions agent can take
- Time-Constrained if $L < 2NA$
- Two orders of magnitude less than lower bound
- The agent does **not** have enough time steps to learn the optimal policy without some additional assumptions about the domain
- **Assumption:** Transition and reward are **similar** across states

Domain	No. States	No. Actions	No. State-Actions	Min Bound Deterministic	Min Bound Stochastic	Maximum L
Taxi	500	6	3,000	300,000	1,050,000	6,000
Four Rooms	100	4	400	40,000	140,000	800
Two Rooms	51	4	204	20,400	72,400	408
Fuel World	39,711	8	317,688	31,768,800	111,190,800	635,376
Mountain Car	10,000	3	30,000	300,000	10,500,000	60,000
Puddle World	400	4	1,600	160,000	560,000	3,200
Cart-Pole Balancing	160,000	2	320,000	32,000,000	11,200,000	640,000

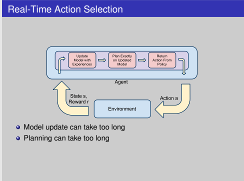
Learning in these time-constrained domains will require some assumptions. In this work, we assume that actions will have similar effects in similar states.

Now I will describe the TEXPLORE algorithm, starting with the real-time architecture.

Real-Time Action Selection



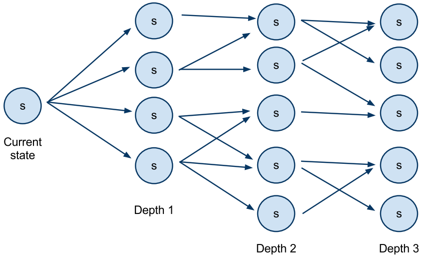
- Model update can take too long
- Planning can take too long

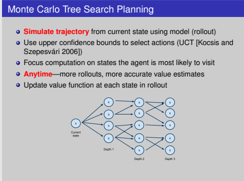


In a typical sequential model-based agent, the agent takes an action, updates its model with the new experience, plans exactly on the new model, and returns the optimal action from its planned policy. However, both the model update and planning step can take too long for a robot acting in the real world. Instead, we have developed a real-time architecture that resolves this issue.

Monte Carlo Tree Search Planning

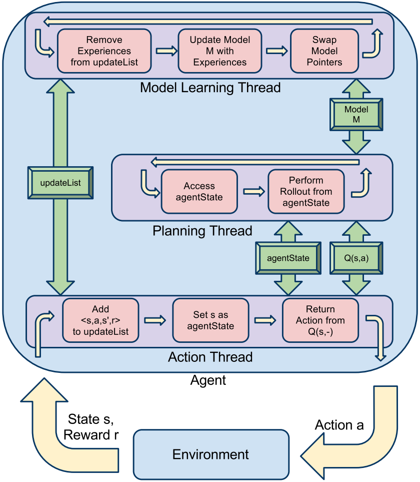
- **Simulate trajectory** from current state using model (rollout)
- Use upper confidence bounds to select actions (UCT [Kocsis and Szepesvári 2006])
- Focus computation on states the agent is most likely to visit
- **Anytime**—more rollouts, more accurate value estimates
- Update value function at each state in rollout





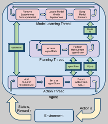
First, instead of exact planning with something such as value iteration, we are going to use Monte Carlo Tree Search to do approximate planning.

Real-Time Model Based Architecture (RTMBA)



- Model learning and planning on parallel threads
- Action selection **is not restricted** by their computation time
- Use sample-based planning (anytime)
- Mutex locks on shared data

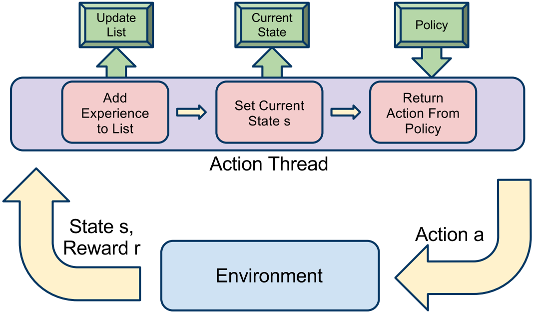
Real-Time Model Based Architecture (RTMBA)



- Model learning and planning on parallel threads
- Action selection **is not restricted** by their computation time
- Use sample-based planning (anytime)
- Mutex locks on shared data

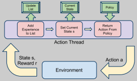
Second, we have created an architecture that places the model learning and planning on parallel threads, such that action selection can occur in real-time no matter how long model learning or planning take.

Action Thread



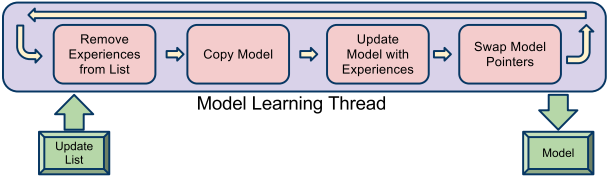
- Add experience, $\langle s, a, s', r \rangle$ to list of experiences to be added to model
- Set agent's current state for planning
- Return best action according to policy

Action Thread



- Add experience, $\langle s, a, s', r \rangle$ to list of experiences to be added to model
- Set agent's current state for planning
- Return best action according to policy

Model Learning Thread

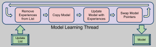


- Make a copy of current model
- Update model copy with new experiences from list (batch updates)
- Swap model pointers
- Repeat

2013-05-16

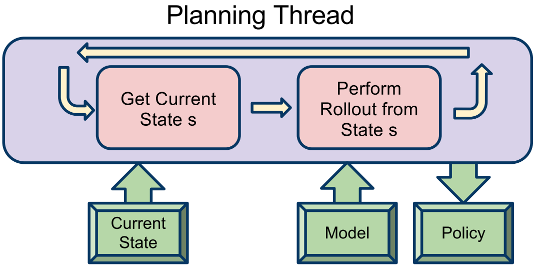
Thesis Defense
└─TEXPLORE
  └─Real-Time Architecture
    └─Model Learning Thread

Model Learning Thread



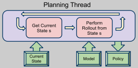
- Make a copy of current model
- Update model copy with new experiences from list (batch updates)
- Swap model pointers
- Repeat

Planning Thread



- Plan using a sample-based MCTS planner (i.e. UCT [Kocsis and Szepesvári 2006])
- Continually perform rollouts from agent's current state
- Rollouts from previous state can help

Planning Thread



- Plan using a sample-based MCTS planner (i.e. UCT [Kocsis and Szepesvári 2006])
- Continually perform rollouts from agent's current state
- Rollouts from previous state can help

Now I will describe TEXPLORE's approach to sample efficiency.

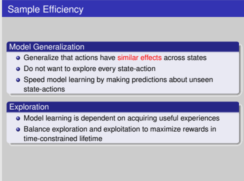
Sample Efficiency

Model Generalization

- Generalize that actions have similar effects across states
- Do not want to explore every state-action
- Speed model learning by making predictions about unseen state-actions

Exploration

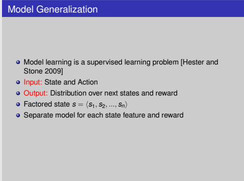
- Model learning is dependent on acquiring useful experiences
- Balance exploration and exploitation to maximize rewards in time-constrained lifetime



We take a two-pronged approach to sample efficiency. First, instead of using a tabular model that learns a separate prediction for each state-action, we want to generalize that actions have similar effects across states. Second, we need the agent to explore intelligently to get good training examples for its model.

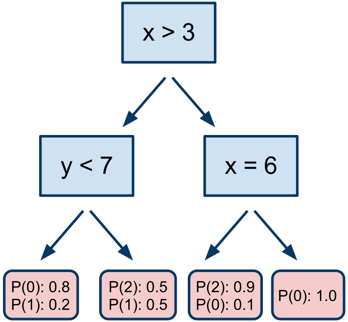
Model Generalization

- Model learning is a supervised learning problem [Hester and Stone 2009]
- Input: State and Action
- Output: Distribution over next states and reward
- Factored state $s = \langle s_1, s_2, ..., s_n \rangle$
- Separate model for each state feature and reward

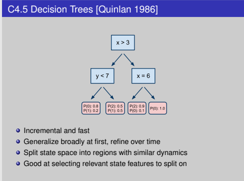


We treat model learning as a supervised learning problem.

C4.5 Decision Trees [Quinlan 1986]



- Incremental and fast
- Generalize broadly at first, refine over time
- Split state space into regions with similar dynamics
- Good at selecting relevant state features to split on



TEXPLORE uses decision trees to learn the model of the domain.

Relative Effects

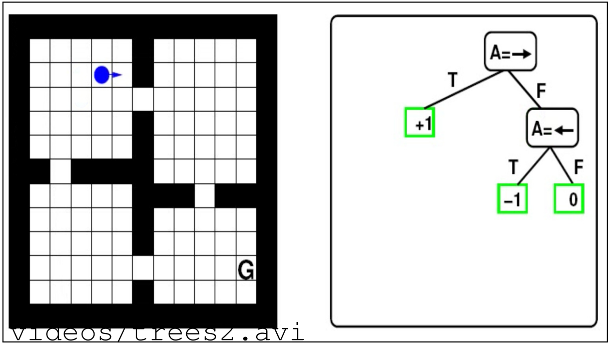
- Predict the **change in state**: $s^{rel} = s' - s$ rather than absolute next state s'
- Often actions have the **same effect across states**
- Previous work predicts relative effects [Jong and Stone 2007] [Leffler et al. 2007]

Relative Effects

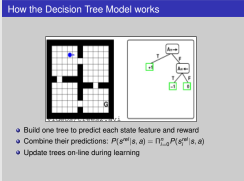
- Predict the **change in state**: $s^{rel} = s' - s$ rather than absolute next state s'
- Often actions have the **same effect across states**
- Previous work predicts relative effects [Jong and Stone 2007] [Leffler et al. 2007]

Rather than predict absolute next states, TEXPLORE predicts the change in the state as this often generalizes better across states.

How the Decision Tree Model works



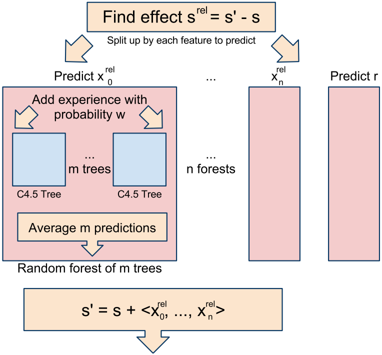
- Build one tree to predict each state feature and reward
- Combine their predictions: $P(s^{rel}|s,a) = \Pi_{i=0}^{n} P(s_i^{rel}|s,a)$
- Update trees on-line during learning

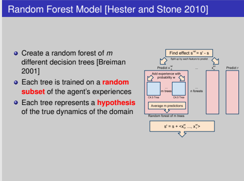


The agent builds the tree model incrementally while learning.

Random Forest Model [Hester and Stone 2010]

- Create a random forest of *m* different decision trees [Breiman 2001]
- Each tree is trained on a **random subset** of the agent's experiences
- Each tree represents a **hypothesis** of the true dynamics of the domain

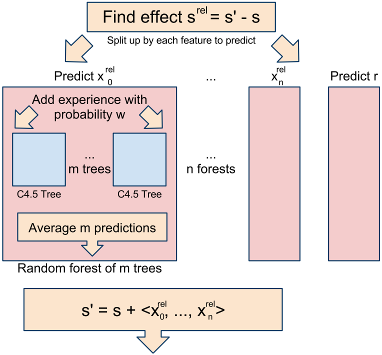


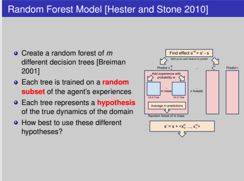


One possible problem with the decision tree model is that there are multiple ways to generalize the data, and the tree model could generalize it incorrectly. Instead, we want the agent to have knowledge of the different possibilities for the real dynamics of the world. Therefore, we have the agent learn a random forest model, where each tree in the forest is a different hypothesis of the true domain dynamics.

Random Forest Model [Hester and Stone 2010]

- Create a random forest of *m* different decision trees [Breiman 2001]
- Each tree is trained on a **random subset** of the agent's experiences
- Each tree represents a **hypothesis** of the true dynamics of the domain
- How best to use these different hypotheses?



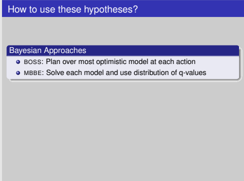


One possible problem with the decision tree model is that there are multiple ways to generalize the data, and the tree model could generalize it incorrectly. Instead, we want the agent to have knowledge of the different possibilities for the real dynamics of the world. Therefore, we have the agent learn a random forest model, where each tree in the forest is a different hypothesis of the true domain dynamics.

How to use these hypotheses?

Bayesian Approaches

- BOSS: Plan over most optimistic model at each action
- MBBE: Solve each model and use distribution of q-values



There are a number of approaches that use samples of a distribution over models in various ways. However, these approaches are trying to guarantee optimality. In contrast, we want our algorithm to be greedier and explore less. The main difference is that these other approaches ignore the models predicting bad outcomes, while we include that information to guide the agent where **not** to explore.

How to use these hypotheses?

Bayesian Approaches

- BOSS: Plan over most optimistic model at each action
- MBBE: Solve each model and use distribution of q-values

TEXPLORE

- Desiderata: Explore less, be greedier
- Plan on average of the predicted distributions
- Balance models that are optimistic with ones that are pessimistic

How to use these hypotheses?

Bayesian Approaches
- BOSS: Plan over most optimistic model at each action
- MBBE: Solve each model and use distribution of q-values

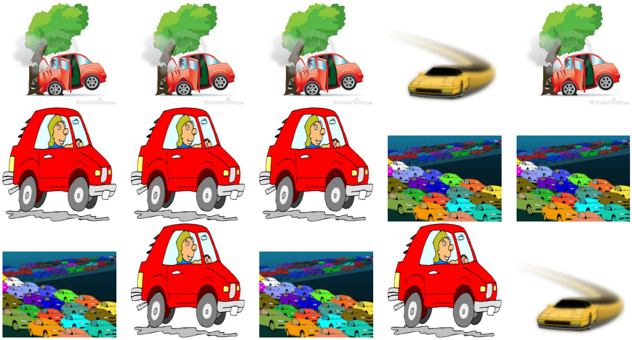
TEXPLORE
- Desiderata: Explore less, be greedier
- Plan on average of the predicted distributions
- Balance models that are optimistic with ones that are pessimistic

There are a number of approaches that use samples of a distribution over models in various ways. However, these approaches are trying to guarantee optimality. In contrast, we want our algorithm to be greedier and explore less. The main difference is that these other approaches ignore the models predicting bad outcomes, while we include that information to guide the agent where **not** to explore.

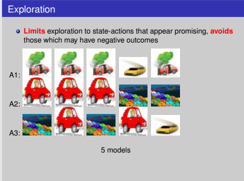
Exploration

- **Limits** exploration to state-actions that appear promising, **avoids** those which may have negative outcomes

A1:
A2:
A3:



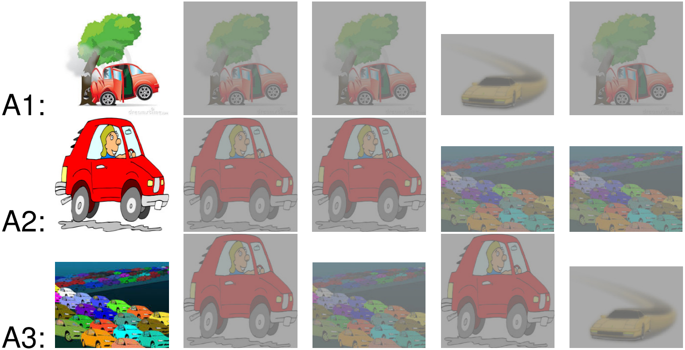
5 models



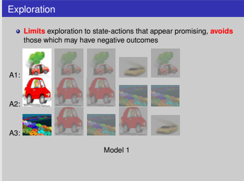
Imagine an agent navigating a city. At a particular intersection, it may have 3 actions: turn left, turn right, or go straight. It has different predictions of the outcomes of these actions from the 5 tree models in its random forest. While BOSS would plan using the most optimistic model for each action, taking the possibly dangerous first action, TEXPLORE incorporates the possibility of a very negative outcome and takes a different action.

Exploration

- **Limits** exploration to state-actions that appear promising, **avoids** those which may have negative outcomes



Model 1

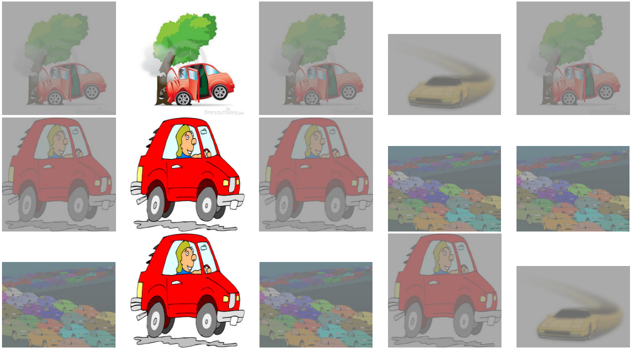


Imagine an agent navigating a city. At a particular intersection, it may have 3 actions: turn left, turn right, or go straight. It has different predictions of the outcomes of these actions from the 5 tree models in its random forest. While BOSS would plan using the most optimistic model for each action, taking the possibly dangerous first action, TEXPLORE incorporates the possibility of a very negative outcome and takes a different action.

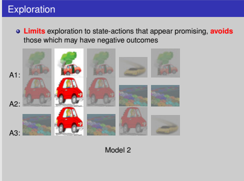
Exploration

- **Limits** exploration to state-actions that appear promising, **avoids** those which may have negative outcomes

A1:
A2:
A3:



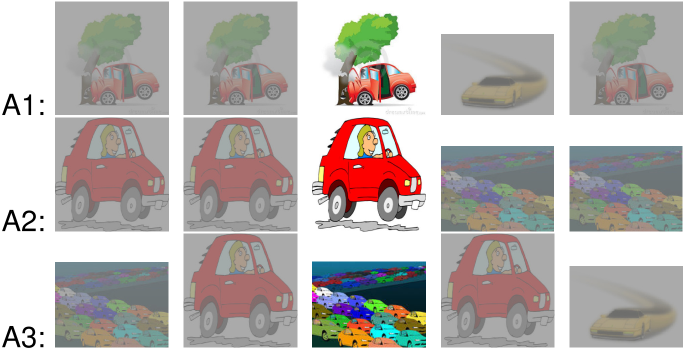
Model 2



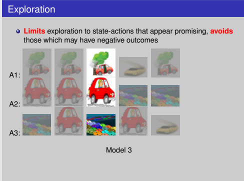
Imagine an agent navigating a city. At a particular intersection, it may have 3 actions: turn left, turn right, or go straight. It has different predictions of the outcomes of these actions from the 5 tree models in its random forest. While BOSS would plan using the most optimistic model for each action, taking the possibly dangerous first action, TEXPLORE incorporates the possibility of a very negative outcome and takes a different action.

Exploration

- **Limits** exploration to state-actions that appear promising, **avoids** those which may have negative outcomes



Model 3

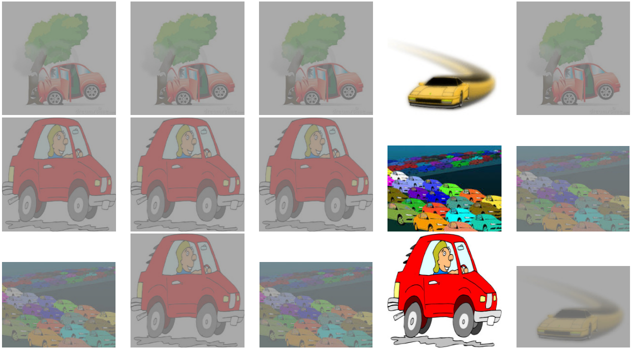


Imagine an agent navigating a city. At a particular intersection, it may have 3 actions: turn left, turn right, or go straight. It has different predictions of the outcomes of these actions from the 5 tree models in its random forest. While BOSS would plan using the most optimistic model for each action, taking the possibly dangerous first action, TEXPLORE incorporates the possibility of a very negative outcome and takes a different action.

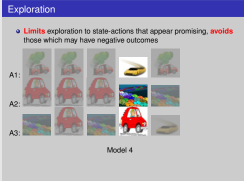
Exploration

- **Limits** exploration to state-actions that appear promising, **avoids** those which may have negative outcomes

A1:
A2:
A3:



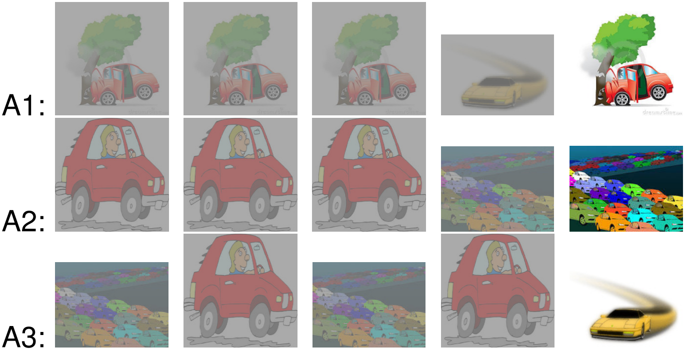
Model 4



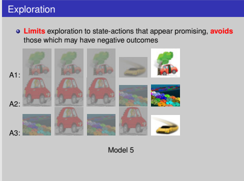
Imagine an agent navigating a city. At a particular intersection, it may have 3 actions: turn left, turn right, or go straight. It has different predictions of the outcomes of these actions from the 5 tree models in its random forest. While BOSS would plan using the most optimistic model for each action, taking the possibly dangerous first action, TEXPLORE incorporates the possibility of a very negative outcome and takes a different action.

Exploration

- **Limits** exploration to state-actions that appear promising, **avoids** those which may have negative outcomes



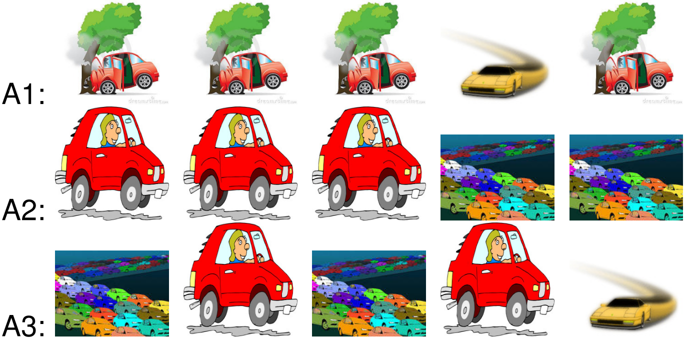
Model 5



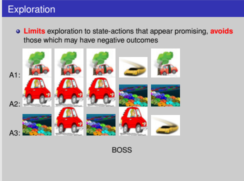
Imagine an agent navigating a city. At a particular intersection, it may have 3 actions: turn left, turn right, or go straight. It has different predictions of the outcomes of these actions from the 5 tree models in its random forest. While BOSS would plan using the most optimistic model for each action, taking the possibly dangerous first action, TEXPLORE incorporates the possibility of a very negative outcome and takes a different action.

Exploration

- **Limits** exploration to state-actions that appear promising, **avoids** those which may have negative outcomes



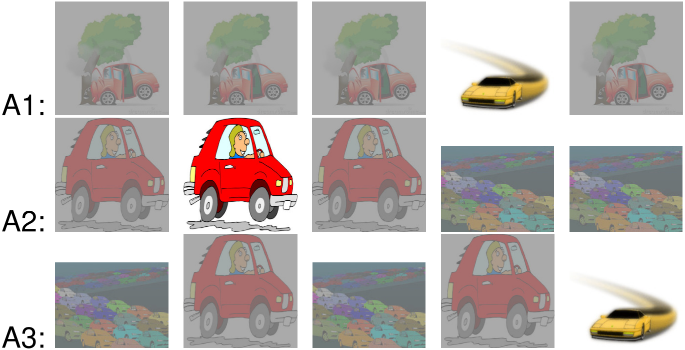
BOSS



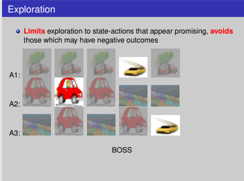
Imagine an agent navigating a city. At a particular intersection, it may have 3 actions: turn left, turn right, or go straight. It has different predictions of the outcomes of these actions from the 5 tree models in its random forest. While BOSS would plan using the most optimistic model for each action, taking the possibly dangerous first action, TEXPLORE incorporates the possibility of a very negative outcome and takes a different action.

Exploration

- **Limits** exploration to state-actions that appear promising, **avoids** those which may have negative outcomes



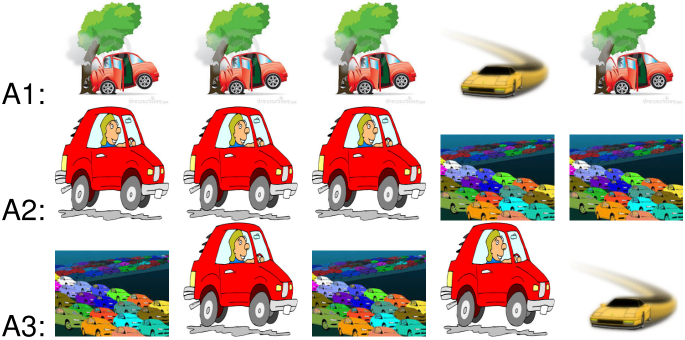
BOSS



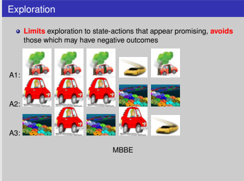
Imagine an agent navigating a city. At a particular intersection, it may have 3 actions: turn left, turn right, or go straight. It has different predictions of the outcomes of these actions from the 5 tree models in its random forest. While BOSS would plan using the most optimistic model for each action, taking the possibly dangerous first action, TEXPLORE incorporates the possibility of a very negative outcome and takes a different action.

Exploration

- **Limits** exploration to state-actions that appear promising, **avoids** those which may have negative outcomes



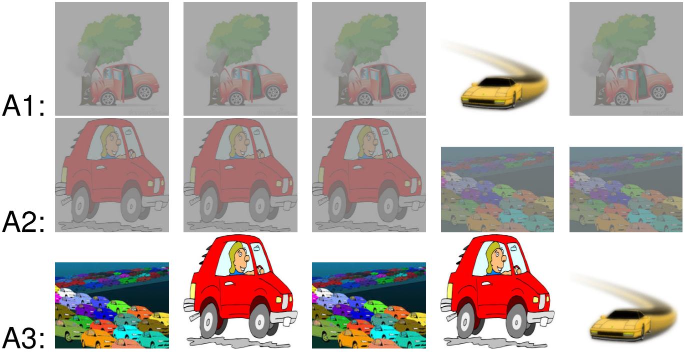
MBBE



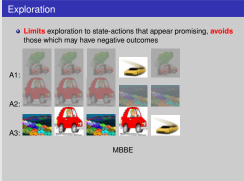
Imagine an agent navigating a city. At a particular intersection, it may have 3 actions: turn left, turn right, or go straight. It has different predictions of the outcomes of these actions from the 5 tree models in its random forest. While BOSS would plan using the most optimistic model for each action, taking the possibly dangerous first action, TEXPLORE incorporates the possibility of a very negative outcome and takes a different action.

Exploration

- **Limits** exploration to state-actions that appear promising, **avoids** those which may have negative outcomes



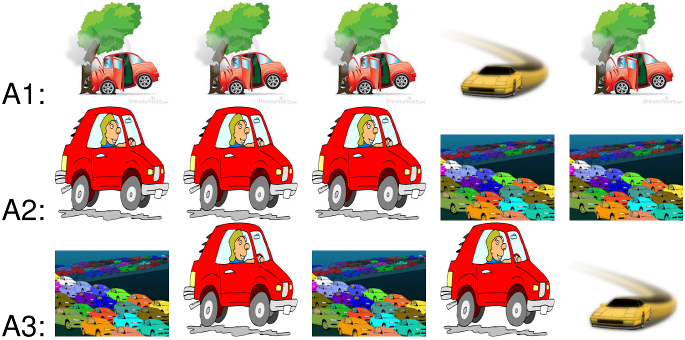
MBBE



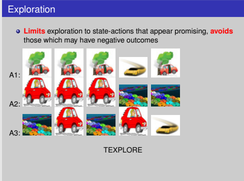
Imagine an agent navigating a city. At a particular intersection, it may have 3 actions: turn left, turn right, or go straight. It has different predictions of the outcomes of these actions from the 5 tree models in its random forest. While BOSS would plan using the most optimistic model for each action, taking the possibly dangerous first action, TEXPLORE incorporates the possibility of a very negative outcome and takes a different action.

Exploration

- **Limits** exploration to state-actions that appear promising, **avoids** those which may have negative outcomes



TEXPLORE

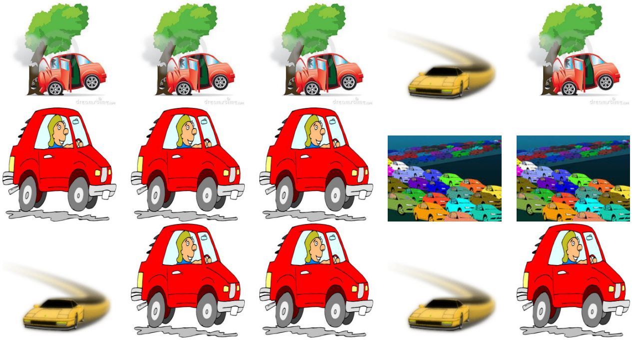


Imagine an agent navigating a city. At a particular intersection, it may have 3 actions: turn left, turn right, or go straight. It has different predictions of the outcomes of these actions from the 5 tree models in its random forest. While BOSS would plan using the most optimistic model for each action, taking the possibly dangerous first action, TEXPLORE incorporates the possibility of a very negative outcome and takes a different action.

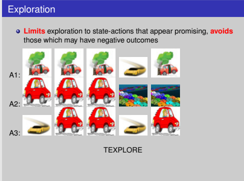
Exploration

- **Limits** exploration to state-actions that appear promising, **avoids** those which may have negative outcomes

A1:
A2:
A3:



TEXPLORE



Imagine an agent navigating a city. At a particular intersection, it may have 3 actions: turn left, turn right, or go straight. It has different predictions of the outcomes of these actions from the 5 tree models in its random forest. While BOSS would plan using the most optimistic model for each action, taking the possibly dangerous first action, TEXPLORE incorporates the possibility of a very negative outcome and takes a different action.

Seed experiences

- Do not want to start from scratch learning on robots [Smart and Kaelbling 2002]
- Provide a few **example transitions** to initialize model
- Example transitions could come from human experience
- Avoid having the agent explore **every** state-action for unusual states

Seed experiences

- Do not want to start from scratch learning on robots [Smart and Kaelbling 2002]
- Provide a few **example transitions** to initialize model
- Example transitions could come from human experience
- Avoid having the agent explore **every** state-action for unusual states

Rather than starting completely from scratch on go-to-goal type tasks, we give the agent some example transitions to initialize its model and jump-start learning.

Now I will show TEXPLORE's approach to domains with continuous state.

Continuous State

Problems

- Make continuous predictions
- Plan over continuous state space

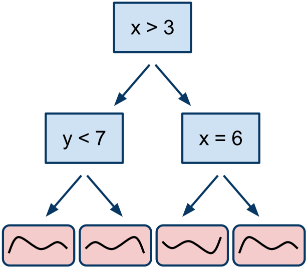
Continuous State

Problems

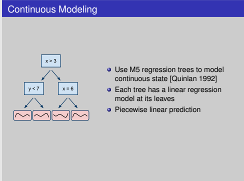
- Make continuous predictions
- Plan over continuous state space

Handling continuous state brings up two problems: making continuous predictions, and planning over a continuous state space.

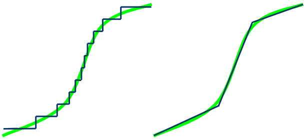
Continuous Modeling



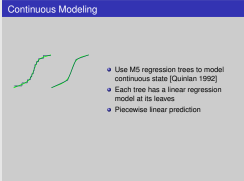
- Use M5 regression trees to model continuous state [Quinlan 1992]
- Each tree has a linear regression model at its leaves
- Piecewise linear prediction



To make continuous predictions, we replace the C4.5 decision trees with regression trees, which have a linear regression in each leaf. This enables the agent to better model the data and possibly learn with fewer experiences.

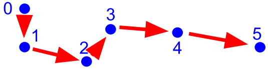


- Use M5 regression trees to model continuous state [Quinlan 1992]
- Each tree has a linear regression model at its leaves
- Piecewise linear prediction

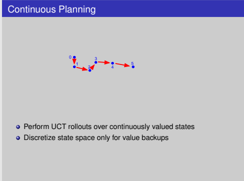


To make continuous predictions, we replace the C4.5 decision trees with regression trees, which have a linear regression in each leaf. This enables the agent to better model the data and possibly learn with fewer experiences.

Continuous Planning

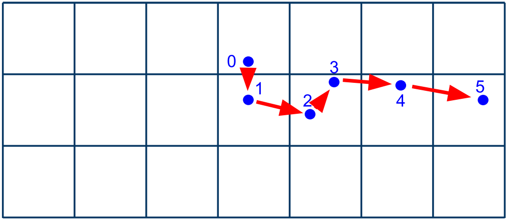


- Perform UCT rollouts over continuously valued states
- Discretize state space only for value backups



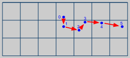
We perform the UCT planning rollouts over the continuously valued states. We only require a discretized state space for updating the state-values from the planner. This approach is much better than discretizing the state first, as UCT can plan multiple steps through a large discrete state.

Continuous Planning



- Perform UCT rollouts over continuously valued states
- Discretize state space only for value backups

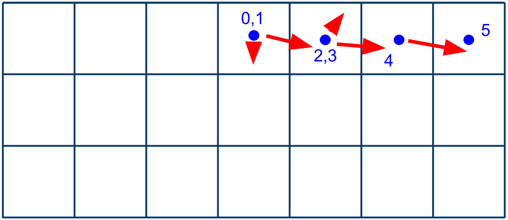
Continuous Planning



- Perform UCT rollouts over continuously valued states
- Discretize state space only for value backups

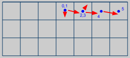
We perform the UCT planning rollouts over the continuously valued states. We only require a discretized state space for updating the state-values from the planner. This approach is much better than discretizing the state first, as UCT can plan multiple steps through a large discrete state.

Continuous Planning



- Perform UCT rollouts over continuously valued states
- Discretize state space only for value backups
- Know where in discretized state you are

Continuous Planning



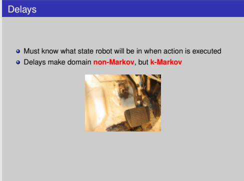
- Perform UCT rollouts over continuously valued states
- Discretize state space only for value backups
- Know where in discretized state you are

We perform the UCT planning rollouts over the continuously valued states. We only require a discretized state space for updating the state-values from the planner. This approach is much better than discretizing the state first, as UCT can plan multiple steps through a large discrete state.

Now I will describe TEXPLORE's approach to handling delay.

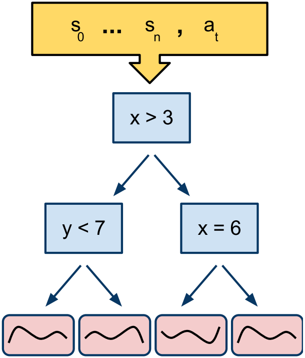
Delays

- Must know what state robot will be in when action is executed
- Delays make domain **non-Markov**, but **k-Markov**

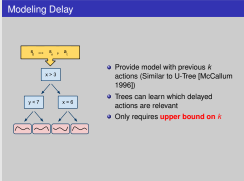


In RL, it is typically assumed that domains are Markov, meaning you can predict the next state from only the last state and action. However, with delay, the domains become k-Markov, as you need the previous k states or actions to predict the next state.

Modeling Delay

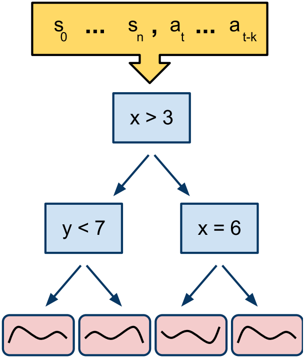


- Provide model with previous *k* actions (Similar to U-Tree [McCallum 1996])
- Trees can learn which delayed actions are relevant
- Only requires **upper bound on *k***

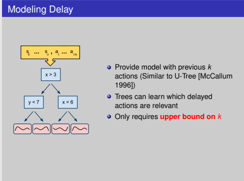


To model delay, we give the decision tree the previous k actions. The decision tree can then learn which delayed action is relevant, or can even learn more complex delays that are not a static time delay.

Modeling Delay

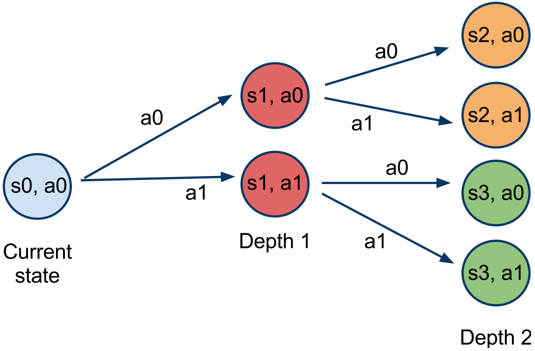


- Provide model with previous *k* actions (Similar to U-Tree [McCallum 1996])
- Trees can learn which delayed actions are relevant
- Only requires **upper bound on *k***

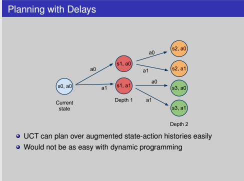


To model delay, we give the decision tree the previous k actions. The decision tree can then learn which delayed action is relevant, or can even learn more complex delays that are not a static time delay.

Planning with Delays



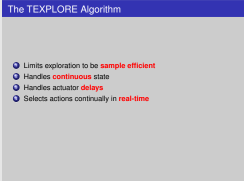
- UCT can plan over augmented state-action histories easily
- Would not be as easy with dynamic programming



UCT can easily track the history of actions during its rollout.

The TEXPLORE Algorithm

1. Limits exploration to be **sample efficient**
2. Handles **continuous** state
3. Handles actuator **delays**
4. Selects actions continually in **real-time**



I have now presented the entire TEXPLORE algorithm. In addition, it can model domains with dependent feature transitions, but that is not covered in these slides. It is fully detailed in the dissertation.

The TEXPLORE Algorithm

1. Limits exploration to be **sample efficient**
2. Handles **continuous** state
3. Handles actuator **delays**
4. Selects actions continually in **real-time**
5. Models domains with **dependent feature transitions**

The TEXPLORE Algorithm

- Limits exploration to be **sample efficient**
- Handles **continuous** state
- Handles actuator **delays**
- Selects actions continually in **real-time**
- Models domains with **dependent feature transitions**

I have now presented the entire TEXPLORE algorithm. In addition, it can model domains with dependent feature transitions, but that is not covered in these slides. It is fully detailed in the dissertation.

I have now shown the entire TEXPLORE algorithm. In this section, I will present a comparison of each component of TEXPLORE against other state-of-the-art approaches, before showing the entire algorithm learning to control the autonomous vehicle.

Autonomous Vehicle

- Upgraded to run **autonomously** by adding shift-by-wire, steering, and braking actuators.
- Vehicle runs at 10 Hz.
- Agent **must** provide commands at this frequency.

Autonomous Vehicle

- Upgraded to run **autonomously** by adding shift-by-wire, steering, and braking actuators.
- Vehicle runs at 10 Hz.
- Agent **must** provide commands at this frequency.

Again, these evaluations will be on the autonomous vehicle, where the agent is controlling the pedals and attempting to learn to drive at different speeds.

Velocity Control Task

- State:
  - Current Velocity
  - Desired Velocity
  - Accelerator Pedal Position
  - Brake Pedal Position
- Actions:
  - Do nothing
  - Increase/decrease brake position by 0.1
  - Increase/decrease accelerator position by 0.1
- Reward: -10.0 * velocity error (m/s)

Velocity Control Task

- State:
  - Current Velocity
  - Desired Velocity
  - Accelerator Pedal Position
  - Brake Pedal Position
- Actions:
  - Do nothing
  - Increase/decrease brake position by 0.1
  - Increase/decrease accelerator position by 0.1
- Reward: -10.0 * velocity error (m/s)

Simulated Velocity Control Task

- Experiments performed **in simulation**
- 10 second episodes (100 samples)
- **Random starting and target velocity** chosen each episode
- Time-Constrained Lifetime is $436,150$ actions ($4,361$ episodes)
- **No seed experiences**
- Brake is controlled by a **PID** controller

Simulated Velocity Control Task

- Experiments performed **in simulation**
- 10 second episodes (100 samples)
- **Random starting and target velocity** chosen each episode
- Time-Constrained Lifetime is 436, 150 actions (4, 361 episodes)
- **No seed experiences**
- Brake is controlled by a **PID** controller

The first evaluations are in simulation. The agent controls the vehicle for 10 second episodes, where each episode has a random starting and target velocity. The brake still has a significant delay in the simulation.

First, we evaluate TEXPLORE's sample efficiency.

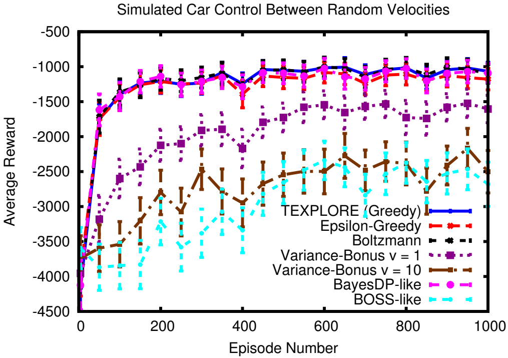
Exploration Comparisons using TEXPLORE's model

1. TEXPLORE
2. ϵ-greedy exploration ($\epsilon = 0.1$)
3. Boltzmann exploration ($\tau = 0.2$)
4. VARIANCE-BONUS Approach $v = 1$ [Deisenroth & Rasmussen 2011]
5. VARIANCE-BONUS Approach $v = 10$
6. Bayesian DP-like Approach (use sampled model for 1 episode) [Strens 2000]
7. BOSS-like Approach (use optimistic model) [Asmuth et al. 2009]

First five approaches use **TEXPLORE's model**

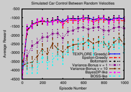
We compare against other state-of-the-art exploration approaches used in conjunction with TEXPLORE's model. All the methods get the benefit of model generalization.

Sample Efficiency Results



- Adding ϵ-greedy, Boltzmann, or Bayesian DP-like exploration **does not** improve performance

Sample Efficiency Results



- Adding ϵ-greedy, Boltzmann, or Bayesian DP-like exploration **does not** improve performance

Comparing with other models

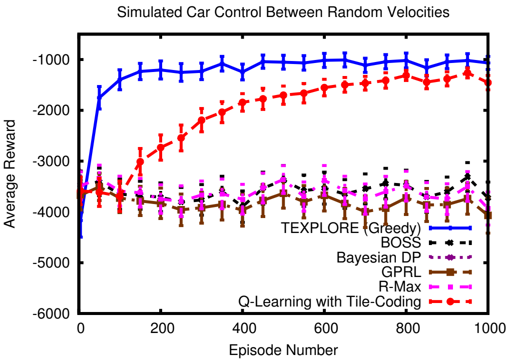
1. BOSS (Sparse Dirichlet prior) [Asmuth et al. 2009]
2. Bayesian DP (Sparse Dirichlet prior) [Strens 2000]
3. PILCO (Gaussian Process Regression model) [Deisenroth & Rasmussen 2011]
4. R-MAX (Tabular model) [Brafman & Tennenholtz 2001]
5. Q-LEARNING using tile-coding [Watkins 1989]

Comparing with other models

- BOSS (Sparse Dirichlet prior) [Asmuth et al. 2009]
- Bayesian DP (Sparse Dirichlet prior) [Strens 2000]
- PILCO (Gaussian Process Regression model) [Deisenroth & Rasmussen 2011]
- R-MAX (Tabular model) [Brafman & Tennenholtz 2001]
- Q-LEARNING using tile-coding [Watkins 1989]

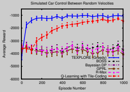
Next, we compared against other methods with different types of models.

Sample Efficiency Results



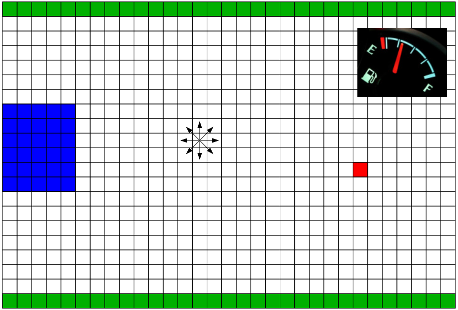
- TEXPLORE accrues **significantly more rewards** than all the other methods after episode 24 ($p < 0.01$).

Sample Efficiency Results

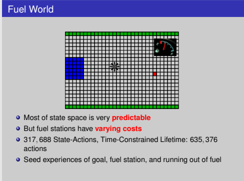


- TEXPLORE accrues **significantly more rewards** than all the other methods after episode 24 ($p < 0.01$).

Fuel World



- Most of state space is very **predictable**
- But fuel stations have **varying costs**
- $317,688$ State-Actions, Time-Constrained Lifetime: $635,376$ actions
- Seed experiences of goal, fuel station, and running out of fuel



We created a domain called **Fuel World** to further examine exploration. In it, the agent starts in the middle left of the domain and is trying to reach a terminal state in the middle right of the domain. The agent's state vector, s, is made up of three features: its ROW, COL, and FUEL. Each step the agent takes reduces its fuel level by 1. If the fuel level reaches 0, the episode terminates with reward -400. There are fuel stations along the top and bottom row of the domain that increase the agent's fuel level by 20. The fuel stations have costs varying from -10 to -30 reward. The idea is that the white states are all easily predictable, while the fuel stations are more interesting. However, even with the fuel stations, some are more relevant than others.

Comparison methods

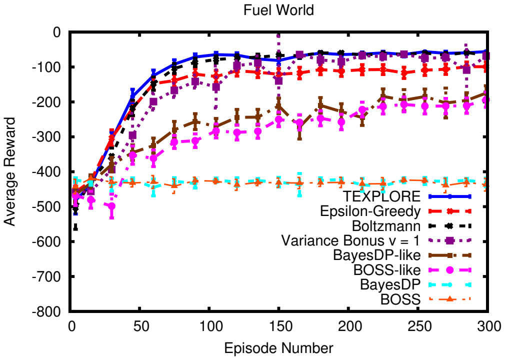
1. TEXPLORE (Greedy w.r.t. aggregate model)
2. ϵ-greedy exploration ($\epsilon = 0.1$)
3. Boltzmann exploration ($\tau = 0.2$)
4. VARIANCE-BONUS Approach $v = 10$ [Deisenroth & Rasmussen 2011]
5. Bayesian DP-like Approach (use sampled model for 1 episode) [Strens 2000]
6. BOSS-like Approach (use optimistic model) [Asmuth et al. 2009]
7. BOSS (Sparse Dirichlet prior) [Asmuth et al. 2009]
8. Bayesian DP (Sparse Dirichlet prior) [Strens 2000]

Comparison methods

- TEXPLORE (Greedy w.r.t. aggregate model)
- ϵ-greedy exploration ($\epsilon = 0.1$)
- Boltzmann exploration ($\tau = 0.2$)
- VARIANCE-BONUS Approach $v = 10$ [Deisenroth & Rasmussen 2011]
- Bayesian DP-like Approach (use sampled model for 1 episode) [Strens 2000]
- BOSS-like Approach (use optimistic model) [Asmuth et al. 2009]
- BOSS (Sparse Dirichlet prior) [Asmuth et al. 2009]
- Bayesian DP (Sparse Dirichlet prior) [Strens 2000]

We compare with the same state-of-the-art approaches as before.

Fuel World Results

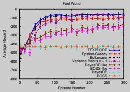


- TEXPLORE learns the **fastest** and **accrues the most cumulative reward** of any of the methods.
- TEXPLORE learns the task **within the time-constrained lifetime** of 635, 376 steps.

2013-05-16

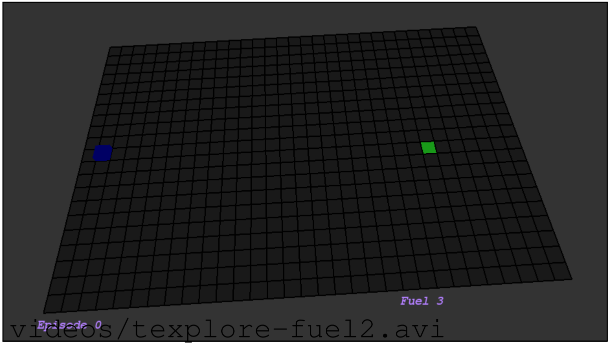
Thesis Defense
└─Empirical Evaluation
  └─Sample Efficiency
    └─Fuel World Results

Fuel World Results

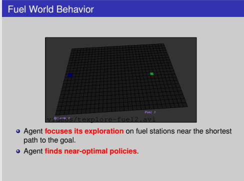


- TEXPLORE learns the **fastest** and **accrues the most cumulative reward** of any of the methods.
- TEXPLORE learns the task **within the time-constrained lifetime** of 635,376 steps.

Fuel World Behavior



- Agent **focuses its exploration** on fuel stations near the shortest path to the goal.
- Agent **finds near-optimal policies**.



After a few early episodes where TEXPLORE must wander around exploring, it starts exploring intelligent. Each episode, it tries a different fuel station on the way to the goal. It quickly converges to the optimal fuel station.

Now I present comparisons on handling continuous state.

Continuous Models

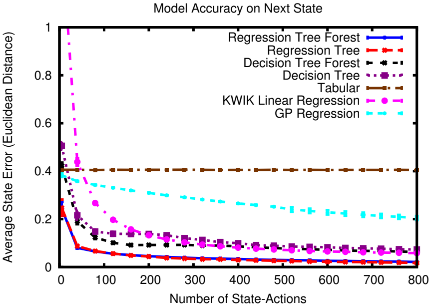
1. Regression Tree Forest (TEXPLORE Default)
2. Single Regression Tree
3. Decision Tree Forest
4. Single Decision Tree
5. Tabular Model
6. KWIK Linear Regression [Strehl and Littman 2007]
7. Gaussian Process Regression (PILCO model) [Deisenroth & Rasmussen 2011]

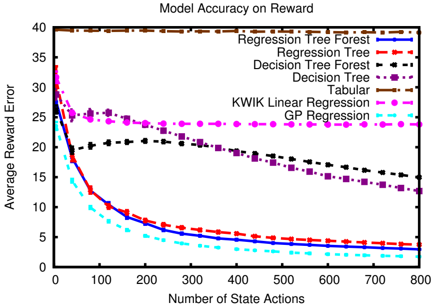
Continuous Models

1. Regression Tree Forest (TEXPLORE Default)
2. Single Regression Tree
3. Decision Tree Forest
4. Single Decision Tree
5. Tabular Model
6. KWIK Linear Regression [Strehl and Littman 2007]
7. Gaussian Process Regression (PILCO model) [Deisenroth & Rasmussen 2011]

We compare with variants of TEXPLORE's model as well as state-of-the-art methods for modeling continuous domains.

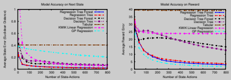
Continuous Model Accuracy





- **Regression tree forest and single regression tree** have significantly less error than all the other models in predicting the next state ($p < 0.001$).
- For reward, regression tree is significantly better than all models but GP regression after 205 state-actions ($p < 0.001$).

Continuous Model Accuracy



- **Regression tree forest and single regression tree** have significantly less error than all the other models in predicting the next state ($p < 0.001$).
- For reward, regression tree is significantly better than all models but GP regression after 205 state-actions ($p < 0.001$).

Each model is trained on random experiences from the domain and tested on its ability to predict 10,000 random experiences from the domain. The state error is the average Euclidean distance between the most likely predicted state and the true most likely next state.

Now we evaluate TEXPLORE's approach to handling delay.

Methods for Delays

Model-Based Simulated (MBS) [Walsh et al. 2009]

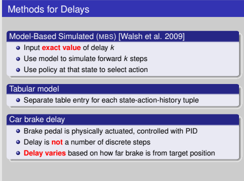
- Input **exact value** of delay k
- Use model to simulate forward k steps
- Use policy at that state to select action

Tabular model

- Separate table entry for each state-action-history tuple

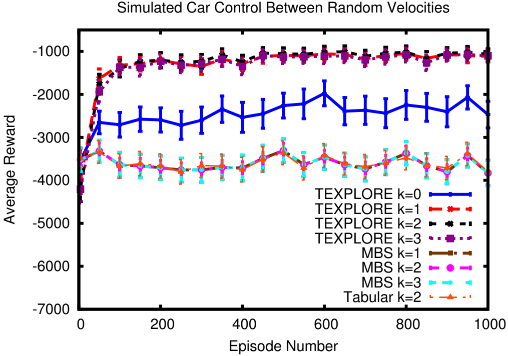
Car brake delay

- Brake pedal is physically actuated, controlled with PID
- Delay is **not** a number of discrete steps
- **Delay varies** based on how far brake is from target position



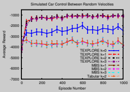
There are a few comparisons for handling delay. MBS requires the exact delay as an input, however the delay on the car brake is not an exact or constant number of discrete steps.

Handling Action Delays



- TEXPLORE with $k = 1, 2$, or 3 all perform **significantly better** than than using no delay ($k = 0$) ($p < 0.005$).
- These approaches are **significantly better** than using another approach to handling delay ($p < 0.005$).

Handling Action Delays



- TEXPLORE with $k = 1, 2$, or 3 all perform **significantly better** than than using no delay ($k = 0$) ($p < 0.005$).
- These approaches are **significantly better** than using another approach to handling delay ($p < 0.005$).

We evaluate TEXPLORE's real-time architecture.

Real-Time Action Selection Methods

Using TEXPLORE's model

1. RTMBA (TEXPLORE)
2. Real Time Dynamic Programming (RTDP) [Barto et al. 1995]
3. Parallel Value Iteration
4. Value Iteration

Other methods

1. Dyna [Sutton 1990]
2. Q-Learning with tile-coding [Watkins 1989]

Real-Time Action Selection Methods

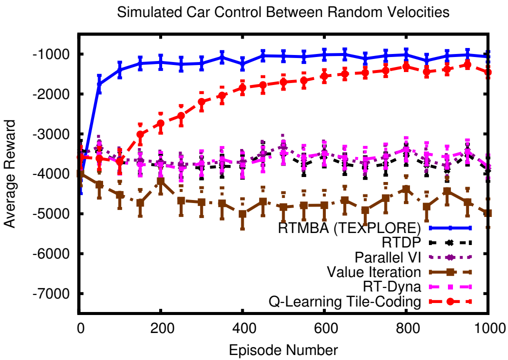
Using TEXPLORE's model

- RTMBA (TEXPLORE)
- Real Time Dynamic Programming (RTDP) [Barto et al. 1995]
- Parallel Value Iteration
- Value Iteration

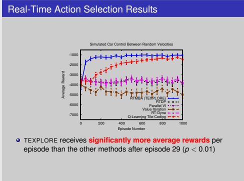
Other methods

- Dyna [Sutton 1990]
- Q-Learning with tile-coding [Watkins 1989]

Real-Time Action Selection Results



- TEXPLORE receives **significantly more average rewards** per episode than the other methods after episode 29 ($p < 0.01$)



It is particularly difficult for the non-real-time methods to model the domain as the amount of time that goes by between sensor readings varies for them.

Now we evaluate the complete algorithm on the actual vehicle.

On the physical vehicle

- But, does it work on the actual vehicle?

On the physical vehicle

- But, does it work on the actual vehicle?

For running on the actual vehicle, we limited ourselves to a single starting and target velocity.

On the physical vehicle

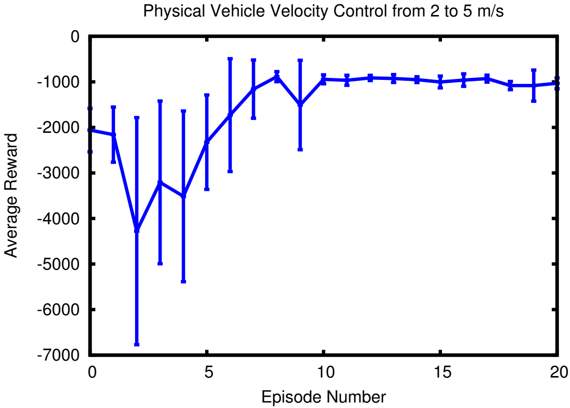
- 5 trials, starting at 2 m/s, target of 5 m/s.
- Time-constrained lifetime: $33,550$ steps, or 335 episodes.
- No seed experiences

On the physical vehicle

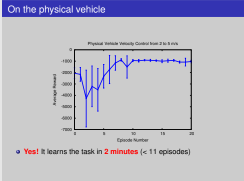
- 5 trials, starting at 2 m/s, target of 5 m/s.
- Time-constrained lifetime: 33, 550 steps, or 335 episodes.
- No seed experiences

For running on the actual vehicle, we limited ourselves to a single starting and target velocity.

On the physical vehicle



- **Yes!** It learns the task in **2 minutes** (< 11 episodes)



The agent learns the task within two minutes! The first few episodes, the agent takes random actions and stays near the starting velocity. The new couple episodes, the agent explores, trying out slamming on the brakes or the gas. Then the agent starts tracking the target velocity, improving until it converges around episode 10.

After presenting both the TEXPLORE algorithm and thorough empirical evaluations of it, we present some exploration extensions to TEXPLORE for various domains.

Characterization of Domains

Haystack Domains

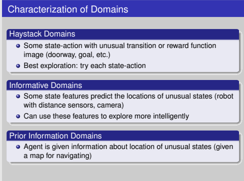
- Some state-action with unusual transition or reward function image (doorway, goal, etc.)
- Best exploration: try each state-action

Informative Domains

- Some state features predict the locations of unusual states (robot with distance sensors, camera)
- Can use these features to explore more intelligently

Prior Information Domains

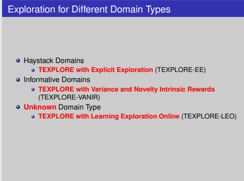
- Agent is given information about location of unusual states (given a map for navigating)



We classify RL domains into three classes, based on what type of information the agent has to bias its exploration.

Exploration for Different Domain Types

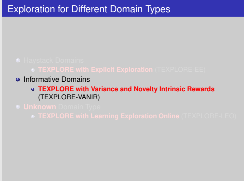
- Haystack Domains
  - **TEXPLORE with Explicit Exploration** (TEXPLORE-EE)
- Informative Domains
  - **TEXPLORE with Variance and Novelty Intrinsic Rewards** (TEXPLORE-VANIR)
- **Unknown** Domain Type
  - **TEXPLORE with Learning Exploration Online** (TEXPLORE-LEO)



We have developed different exploration extensions for TEXPLORE for each of these classes of domains. We have also developed an approach called LEO which learns on-line which exploration is best for a particular domain. In this presentation, I only discuss TEXPLORE-VANIR, but the others are fully detailed in the dissertation.

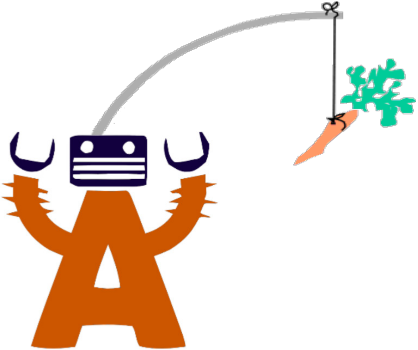
Exploration for Different Domain Types

- Haystack Domains
  - **TEXPLORE with Explicit Exploration** (TEXPLORE-EE)
- Informative Domains
  - **TEXPLORE with Variance and Novelty Intrinsic Rewards** (TEXPLORE-VANIR)
- **Unknown** Domain Type
  - **TEXPLORE with Learning Exploration Online** (TEXPLORE-LEO)

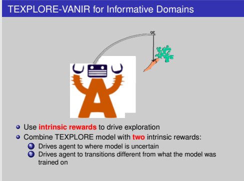


We have developed different exploration extensions for TEXPLORE for each of these classes of domains. We have also developed an approach called LEO which learns on-line which exploration is best for a particular domain. In this presentation, I only discuss TEXPLORE-VANIR, but the others are fully detailed in the dissertation.

TEXPLORE-VANIR for Informative Domains

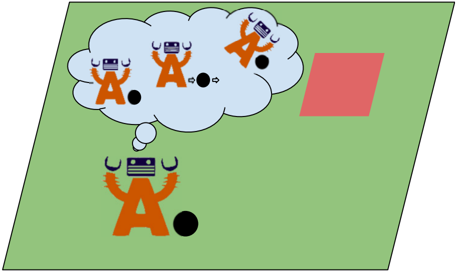


- Use **intrinsic rewards** to drive exploration
- Combine TEXPLORE model with **two** intrinsic rewards:
  1. Drives agent to where model is uncertain
  2. Drives agent to transitions different from what the model was trained on



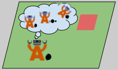
TEXPLORE-VANIR is meant for informative domains where the agent has some more informative or descriptive features about its state in the domain.

Variance Intrinsic Reward



- Reward where model is **uncertain**
- Calculate a **measure of variance**:
  $D(s,a) = \sum_{i=1}^{n} \sum_{j=1}^{m} \sum_{k=1}^{m} D_{KL}(P_j(x_i^{rel}|s,a) || P_k(x_i^{rel}|s,a))$
- Add intrinsic reward proportional to variance measure:
  $R(s,a) = \mathbf{v} D(s,a)$

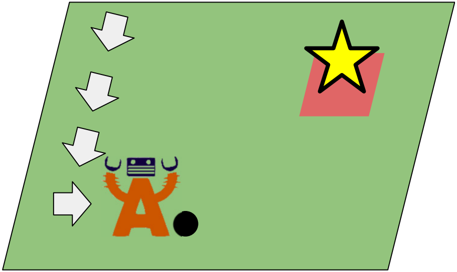
Variance Intrinsic Reward



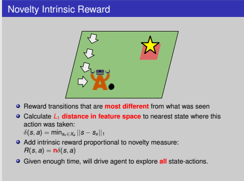
- Reward where model is **uncertain**
- Calculate a **measure of variance**:
  $D(s,a) = \sum_{i=1}^{n} \sum_{j=1}^{m} \sum_{k=1}^{m} D_{KL}(P_j(x_i^{rel}|s,a) || P_k(x_i^{rel}|s,a))$
- Add intrinsic reward proportional to variance measure:
  $R(s,a) = \mathbf{v} D(s,a)$

First, we provide an intrinsic reward for states where the agent's tree models differ in their predictions.

Novelty Intrinsic Reward

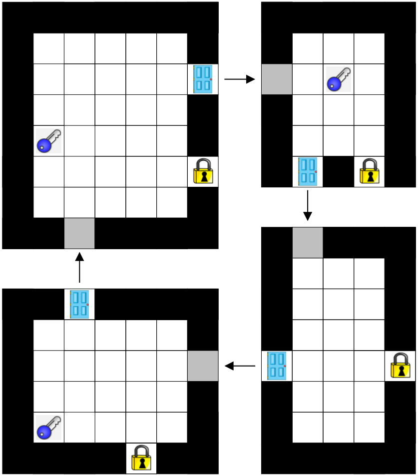


- Reward transitions that are **most different** from what was seen
- Calculate L_1 **distance in feature space** to nearest state where this action was taken:
  $\delta(s, a) = \min_{s_x \in X_a} ||s - s_x||_1$
- Add intrinsic reward proportional to novelty measure:
  $R(s, a) = \mathbf{n}\delta(s, a)$
- Given enough time, will drive agent to explore **all** state-actions.

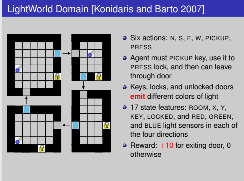


However, there may be states where all the trees in the forest agree incorrectly, because these states are very different from anything the trees were trained on. Therefore, we also provide intrinsic rewards for states that are the most different from the visited states.

LightWorld Domain [Konidaris and Barto 2007]



- Six actions: N, S, E, W, PICKUP, PRESS
- Agent must PICKUP key, use it to PRESS lock, and then can leave through door
- Keys, locks, and unlocked doors **emit** different colors of light
- 17 state features: ROOM, X, Y, KEY, LOCKED, and RED, GREEN, and BLUE light sensors in each of the four directions
- Reward: $+10$ for exiting door, 0 otherwise



The agent has light sensors that tell the agent the intensity of the light in each of the four directions. The light sensors makes the location of objects unique, as those are the only states with maximum light intensity in all four directions. Thus, the agent's novelty intrinsic reward should drive it towards the objects. This task would be very difficult for an agent exploring randomly.

Comparison Methods

TEXPLORE Model

1. TEXPLORE-VANIR with v = 1, n = 3
2. External Rewards Only (TEXPLORE)
3. Bonus for regions with more competence progress (similar to IAC [Baranes and Oudeyer 2009])
4. Bonus for regions with higher prediction errors
5. Explore state-actions with fewer than m visits (R-MAX [Brafman and Tennenholtz 2001])

Tabular Model

1. External Rewards Only
2. R-MAX (explore state-actions with fewer than m visits)

2013-05-16

Comparison Methods

TEXPLORE Model

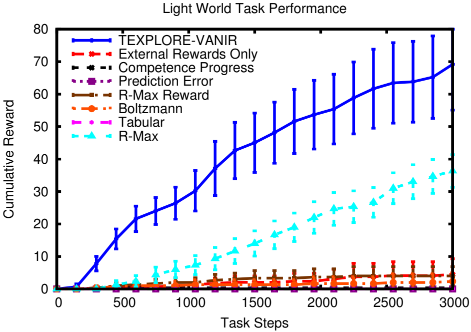
1. TEXPLORE-VANIR with v = 1, n = 3
2. External Rewards Only (TEXPLORE)
3. Bonus for regions with more competence progress (similar to IAC [Baranes and Oudeyer 2009])
4. Bonus for regions with higher prediction errors
5. Explore state-actions with fewer than m visits (R-MAX [Brafman and Tennenholtz 2001])

Tabular Model

1. External Rewards Only
2. R-MAX (explore state-actions with fewer than m visits)

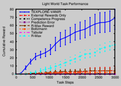
We compare with some other common exploration approaches, both using the TEXPLORE model and a tabular model.

Task Performance



- TEXPLORE-VANIR receives **significantly more** cumulative rewards ($p < 0.001$).

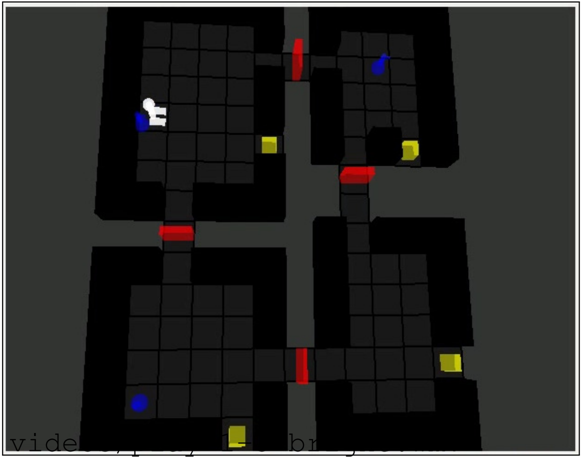
Task Performance



- TEXPLORE-VANIR receives **significantly more** cumulative rewards ($p < 0.001$).

The best exploration depends on both the domain and the type of model being learned.

TEXPLORE-VANIR Exploration



- **Novelty** rewards draw agent to objects and corners.
- **Variance** rewards make it explore using objects.

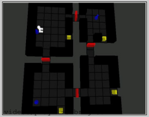
2013-05-16

Thesis Defense

└─Exploration

└─TEXPLORE-VANIR Exploration

TEXPLORE-VANIR Exploration



- **Novelty** rewards draw agent to objects and corners.
- **Variance** rewards make it explore using objects.

After presenting TEXPLORE's exploration extensions, I will present related and future work before concluding.

Related Work: Sample Efficient Model Learning

- **SPITI** [Degris et al. 2006]
  - Learn decision tree models for each feature
  - Used ϵ-greedy exploration
- **AMBI** [Jong and Stone 2007]
  - Instance-based model with relative effects
  - R_{max} bonus for state regions with few visits
- **PILCO** [Deisenroth and Rasmussen 2011]
  - Use Gaussian Process regression to model dynamics
  - Exploration based on variance of GP predictions
  - Batch mode, agent is provided reward model

Related Work: Sample Efficient Model Learning

- SPITI [Degris et al. 2006]
  - Learn decision tree models for each feature
  - Used ϵ-greedy exploration
- AMBI [Jong and Stone 2007]
  - Instance-based model with relative effects
  - R_{max} bonus for state regions with few visits
- PILCO [Deisenroth and Rasmussen 2011]
  - Use Gaussian Process regression to model dynamics
  - Exploration based on variance of GP predictions
  - Batch mode, agent is provided reward model

There are a number of methods focused on sample efficient model learning, which learn models that generalize the effects of actions across states in different ways.

Related Work: Bayesian RL

- Offers optimal solution to exploration problem [Duff 2003]
- Computationally intractable
- Many approximate solutions:
  - Tie model parameters together [Poupart et al. 2006]
  - Sample from model distributions [Strens 2000, Asmuth et al. 2009]
  - Learn Bayesian optimal policy over time [Kolter and Ng 2009]

Related Work: Bayesian RL

- Offers optimal solution to exploration problem [Duff 2003]
- Computationally intractable
- Many approximate solutions:
  - Tie model parameters together [Poupart et al. 2006]
  - Sample from model distributions [Strens 2000, Asmuth et al. 2009]
  - Learn Bayesian optimal policy over time [Kolter and Ng 2009]

Bayesian RL appears to be a promising approach for addressing the sample efficiency issue. However, the optimal solution is computationally intractable, and must be approximated.

Related Work: Continuous State

- **KWIK Linear Regression** [Strehl and Littman 2007]
  - Linear regression model with prediction confidence
  - Only for linearly parametrized domains
- **FITTED-R-MAX** [Jong and Stone 2007]
  - R-MAX style algorithm in continuous state
  - Use fitted value iteration [Gordon 1995]

Related Work: Continuous State

- **KWIK Linear Regression** [Strehl and Littman 2007]
  - Linear regression model with prediction confidence
  - Only for linearly parametrized domains
- **FITTED-R-MAX** [Jong and Stone 2007]
  - R-MAX style algorithm in continuous state
  - Use fitted value iteration [Gordon 1995]

Related Work: Sensor and Actuator Delay

- **Model Based Simulation** (MBS) [Walsh et al. 2009]
  - Provide domain's delay, k
  - Simulate k steps ahead in model, take best action for this state
- **U-TREE** [McCallum 1996]
  - Build decision trees for representing value function
  - Split on previous actions to handle delays

Related Work: Sensor and Actuator Delay

- **Model Based Simulation** (MBS) [Walsh et al. 2009]
  - Provide domain's delay, k
  - Simulate k steps ahead in model, take best action for this state
- **U-TREE** [McCallum 1996]
  - Build decision trees for representing value function
  - Split on previous actions to handle delays

Related Work: Real-Time Architecture

- **Dyna** Framework [Sutton 1990, 1991]
  - Do Bellman updates on random states using model when not action
  - Still uses tabular model, assumes model update takes insignificant time
- Combining sample-based planning with model-based method
  - With UCT [Silver et al. 2008], With FSSS [Walsh et al. 2010]
  - Neither places a time restriction on model update or planning
- **Real-Time Dynamic Programming** RTDP [Barto et al. 1995]
  - Similar to UCT, but full backups at each state

Related Work: Real-Time Architecture

- **Dyna** Framework [Sutton 1990, 1991]
  - Do Bellman updates on random states using model when not action
  - Still uses tabular model, assumes model update takes insignificant time
- Combining sample-based planning with model-based method
  - With UCT [Silver et al. 2008], With FSSS [Walsh et al. 2010]
  - Neither places a time restriction on model update or planning
- **Real-Time Dynamic Programming** RTDP [Barto et al. 1995]
  - Similar to UCT, but full backups at each state

- **Helicopter Control** [Ng et al. 2003]
  - Learn helicopter model from experiences acquired from human expert
  - Computation performed off-line
- **PILCO** [Deisenroth and Rasmussen 2011]
  - Use Gaussian Process regression for model learning and planning
  - Very sample efficient in learning to control cart-pole
  - Takes 10 minutes of computation for every 2.5 seconds of experience
- **POWER** [Kober and Peters 2008]
  - Policy search for parameterized motor primitives
  - Only for episodic tasks

Related Work: Robot Learning

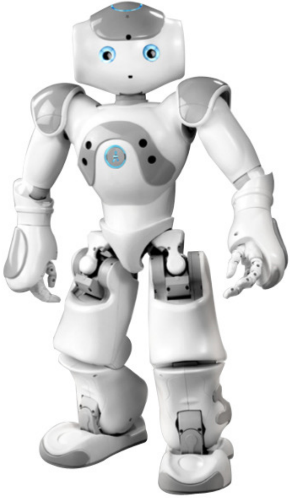
- **Helicopter Control** [Ng et al. 2003]
  - Learn helicopter model from experiences acquired from human expert
  - Computation performed off-line
- **PILCO** [Deisenroth and Rasmussen 2011]
  - Use Gaussian Process regression for model learning and planning
  - Very sample efficient in learning to control cart-pole
  - Takes 10 minutes of computation for every 2.5 seconds of experience
- **POWER** [Kober and Peters 2008]
  - Policy search for parameterized motor primitives
  - Only for episodic tasks

1. Introduction
2. TEXPLORE
3. Empirical Evaluation
4. Exploration
5. Conclusion
   - Related Work
   - Future Work
   - Conclusion

Now I show some directions for future work.

Future Work: Continuous Actions

- Many robots could utilize **continuous actions** (angles, velocities)
- Regression tree model can make predictions on the basis of continuous actions
- Could utilize work on UCT-like planning in continuous actions spaces (HOOT) using continuous bandit algorithms [Bubeck et al. 2011; Mansley et al. 2011; Weinstein and Littman 2012]



Future Work: Continuous Actions

- Many robots could utilize **continuous actions** (angles, velocities)
- Regression tree model can make predictions on the basis of continuous actions
- Could utilize work on UCT-like planning in continuous actions spaces (HOOT) using continuous bandit algorithms [Bubeck et al. 2011; Mansley et al. 2011; Weinstein and Littman 2012]

One challenge that this work does not address is utilizing continuous actions on robots. TEXPLORE could be extended to do so by taking advantage of the recent work using the HOO algorithm.

Future Work: Opponent Modeling

- Initialize each tree in forest with **possible opponent strategy**
- Could be from experience with past opponents
- Explore to determine which type of opponent you are playing

Future Work: Opponent Modeling

- Initialize each tree in forest with **possible opponent strategy**
- Could be from experience with past opponents
- Explore to determine which type of opponent you are playing

The different tree models in TEXPLORE's random forest could each model different possible opponents when playing a game.

Future Work: Lifelong Robot Learning

- **Goal:** Act and learn in environment over lifetime, performing many tasks
- Handle large and complex state space: **Make algorithm more parallel**
- Generalize knowledge to new tasks: **Find best state representation**

Future Work: Lifelong Robot Learning

- **Goal:** Act and learn in environment over lifetime, performing many tasks
- Handle large and complex state space: **Make algorithm more parallel**
- Generalize knowledge to new tasks: **Find best state representation**

A longer term goal for this work is to address the problem of lifelong robot learning. This problem will require the algorithm to handle a large and complex state space, generalize knowledge to new tasks, and be able to represent a lifetime of knowledge.

Contributions

- The TEXPLORE algorithm
- MDP Models that generalize action effects
- Targeted Exploration
- Real-Time Architecture
- ROS RL Interface
- Empirical Evaluation

Contributions

- The TEXPLORE algorithm
- MDP Models that generalize action effects
- Targeted Exploration
- Real-Time Architecture
- ROS RL Interface
- Empirical Evaluation

Contributions

- The TEXPLORE algorithm
  - Limits exploration to be **sample efficient**
  - Handles **continuous** state
  - Handles actuator **delays**
  - Selects actions continually in **real-time**
  - Models domains with **dependent feature transitions**
- MDP Models that generalize action effects
- Targeted Exploration
- Real-Time Architecture
- ROS RL Interface
- Empirical Evaluation

Contributions

- The TEXPLORE algorithm
  - Limits exploration to be **sample efficient**
  - Handles **continuous** state
  - Handles actuator **delays**
  - Selects actions continually in **real-time**
  - Models domains with **dependent feature transitions**
- MDP Models that generalize action effects
- Targeted Exploration
- Real-Time Architecture
- ROS RL Interface
- Empirical Evaluation

Contributions

- The TEXPLORE algorithm
- MDP Models that generalize action effects
    - Random forests of regression trees
- Targeted Exploration
- Real-Time Architecture
- ROS RL Interface
- Empirical Evaluation

Contributions

- The TEXPLORE algorithm
- MDP Models that generalize action effects
  - Random forests of regression trees
- Targeted Exploration
- Real-Time Architecture
- ROS RL Interface
- Empirical Evaluation

Contributions

- The TEXPLORE algorithm
- MDP Models that generalize action effects
- Targeted Exploration
  - Average predictions of each tree in random forest
  - Use intrinsic rewards for **informative** domains
- Real-Time Architecture
- ROS RL Interface
- Empirical Evaluation

Contributions

- The TEXPLORE algorithm
- MDP Models that generalize action effects
- Targeted Exploration
  - Average predictions of each tree in random forest
  - Use intrinsic rewards for **informative** domains
- Real-Time Architecture
- ROS RL Interface
- Empirical Evaluation

Contributions

- The TEXPLORE algorithm
- MDP Models that generalize action effects
- Targeted Exploration
- Real-Time Architecture
  - Maintain **sample efficiency** of model-based methods
  - While acting in **real-time**
- ROS RL Interface
- Empirical Evaluation

2013-05-16

Contributions

- The TEXPLORE algorithm
- MDP Models that generalize action effects
- Targeted Exploration
- Real-Time Architecture
  - Maintain **sample efficiency** of model-based methods
  - While acting in **real-time**
- ROS RL Interface
- Empirical Evaluation

Contributions

- The TEXPLORE algorithm
- MDP Models that generalize action effects
- Targeted Exploration
- Real-Time Architecture
- ROS RL Interface
  - Enables **easy integration** of RL on robots using ROS
- Empirical Evaluation

Contributions

- The TEXPLORE algorithm
- MDP Models that generalize action effects
- Targeted Exploration
- Real-Time Architecture
- ROS RL Interface
  - Enables **easy integration** of RL on robots using ROS
- Empirical Evaluation

Contributions

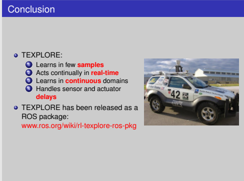
- The TEXPLORE algorithm
- MDP Models that generalize action effects
- Targeted Exploration
- Real-Time Architecture
- ROS RL Interface
- Empirical Evaluation
  - Real-time learning while running on-board **physical robot**

Contributions

- The TEXPLORE algorithm
- MDP Models that generalize action effects
- Targeted Exploration
- Real-Time Architecture
- ROS RL Interface
- Empirical Evaluation
  - Real-time learning while running on-board **physical robot**

Conclusion

- TEXPLORE:
  1. Learns in few **samples**
  2. Acts continually in **real-time**
  3. Learns in **continuous** domains
  4. Handles sensor and actuator **delays**
- TEXPLORE has been released as a ROS package: www.ros.org/wiki/rl-texplore-ros-pkg



The TEXPLORE algorithm is the first algorithm to address all four **RL for Robotics Challenges** together. By addressing these four challenges, TEXPLORE is applicable to many real-world problems and especially many robot control problems. I demonstrated TEXPLORE's success in addressing each challenge on the problem of controlling the velocity of an autonomous vehicle by manipulating the throttle and brake of the vehicle. This work presents an important step towards making RL generally applicable to a wide range of such challenging robotics problems. In addition, TEXPLORE's release as a ROS package enables easy application to robots already running ROS.