Structured Exploration for Reinforcement Learning

Nicholas K. Jong

Department of Computer Sciences
The University of Texas at Austin

December 1, 2010 / PhD Final Defense

Structured Exploration
for Reinforcement Learning

Nicholas K. Jong

Department of Computer Sciences
The University of Texas at Austin

December 1, 2010 / PhD Final Defense

Outline

Outline
Introduction
Exploration and Approximation
Exploration and Hierarchy
Conclusion

This thesis is really all about extending certain exploration mechanisms beyond the case of unstructured MDPs. Section 1 motivates RL and exploration. Section 2 extends R-MAX exploration to MDPs with continuous state spaces. Section 3 extends R-MAX exploration to environments with known hierarchical structure. Section 4 discusses some potential future directions and concludes.

Outline

Structured Exploration for Reinforcement Learning
2010-12-15
Introduction
Outline

Outline
Introduction
The Reinforcement Learning Problem
Reinforcement Learning Methods
Thesis Focus
Exploration and Approximation
Exploration and Hierarchy
Conclusion

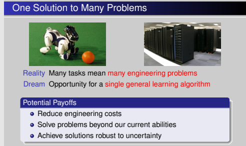
One Solution to Many Problems

Reality Many tasks mean many engineering problems

Dream Opportunity for a single general learning algorithm

Potential Payoffs

- Reduce engineering costs
- Solve problems beyond our current abilities
- Achieve solutions robust to uncertainty



The key appeal of Reinforcement Learning is the prospect of designing and developing a single learning algorithm that can solve many problems, in much the same way that any given human can learn many tasks. (Of course, the human must invest significant time and energy: exploration!) General mechanisms for learning therefore also appear to those interested in how humans learn, not just those interested in solving multiple problems with a single agent.

One Formalism for Many Problems

Environment

- Generate reward $r \in \mathbb{R}$ with expected value $R(s, a)$
- Generate next state $s' \in S$ with probability $P(s, a, s')$
- Using unknown reward and transition functions R and P

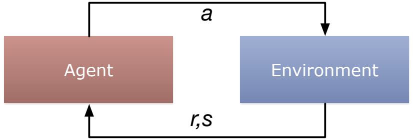
Goal Find a policy $\pi : S \to A$ that maximizes future rewards

RL seems almost too good to be true, but in humans we have an existence proof that general learning is possible. How can we formalize learning in domain-agnostic way? This thesis builds upon the modern field of Reinforcement Learning, which adopts the Markov Decision Process (MDP) formalism. Agents generate actions given states. Environments generate immediate rewards and successor states given actions (and the current state). The agent takes the output of the environment as feedback, and its goal is to find a policy that maps states to actions in a way that will maximize cumulative rewards.

One Formalism for Many Problems

Agent

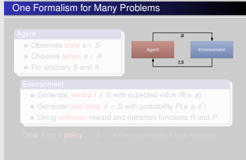
- Observes state $s \in S$
- Chooses action $a \in A$
- For arbitrary S and A



Environment

- Generate reward $r \in \mathbb{R}$ with expected value $R(s, a)$
- Generate next state $s' \in S$ with probability $P(s, a, s')$
- Using unknown reward and transition functions R and P

Goal Find a policy $\pi : S \to A$ that maximizes future rewards

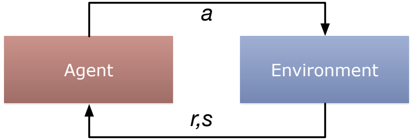


RL seems almost too good to be true, but in humans we have an existence proof that general learning is possible. How can we formalize learning in domain-agnostic way? This thesis builds upon the modern field of Reinforcement Learning, which adopts the Markov Decision Process (MDP) formalism. Agents generate actions given states. Environments generate immediate rewards and successor states given actions (and the current state). The agent takes the output of the environment as feedback, and its goal is to find a policy that maps states to actions in a way that will maximize cumulative rewards.

One Formalism for Many Problems

Agent

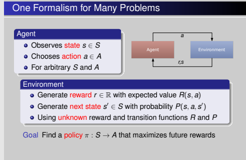
- Observes state $s \in S$
- Chooses action $a \in A$
- For arbitrary S and A



Environment

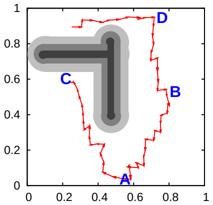
- Generate reward $r \in \mathbb{R}$ with expected value $R(s, a)$
- Generate next state $s' \in S$ with probability $P(s, a, s')$
- Using unknown reward and transition functions R and P

Goal Find a policy $\pi : S \to A$ that maximizes future rewards



RL seems almost too good to be true, but in humans we have an existence proof that general learning is possible. How can we formalize learning in domain-agnostic way? This thesis builds upon the modern field of Reinforcement Learning, which adopts the Markov Decision Process (MDP) formalism. Agents generate actions given states. Environments generate immediate rewards and successor states given actions (and the current state). The agent takes the output of the environment as feedback, and its goal is to find a policy that maps states to actions in a way that will maximize cumulative rewards.

Example: A Resource Gathering Simulation



Simulated Robot's Task

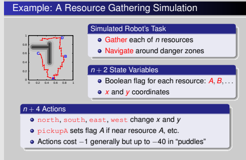
- Gather each of n resources
- Navigate around danger zones

$n+2$ State Variables

- Boolean flag for each resource: $A, B, \ldots$
- x and y coordinates

$n+4$ Actions

- `north`, `south`, `east`, `west` change x and y
- `pickupA` sets flag A if near resource A, etc.
- Actions cost -1 generally but up to -40 in "puddles"



Many of the experiments in this thesis use a simulated problem generalized from the PuddleWorld domain that appears in the RL literature. The agent lives in the unit square and must collect several resources from initially unknown locations. At each time step, the agent knows its x and y coordinates, as well as whether it has collected each resource. It can move in four different directions and attempt to collect any one of the resources that might be near the current coordinates. The environment contains costly puddle regions that the agent should learn to avoid. Note that to a RL algorithm, these actions are just unlabeled buttons, and the state variables comprise a vector of unlabeled numbers. The figure shows an optimal trajectory for a particular placement of four resources. After collecting the last resource, the agent starts over in a random location, but the resources don't move.

Evaluating Policies with Value Functions

The Bellman Equation

- State value depends on policy and action values
- Long-term value equals present value plus future value.

$$
\begin{aligned}
V^\pi(s) &= Q^\pi(s, \pi(s)) \\
Q^\pi(s,a) &= R(s,a) + \gamma \sum_{s'} P(s,a,s') V^\pi(s')
\end{aligned}
$$

Evaluating Policies with Value Functions
The Bellman Equation
State value depends on policy and action values
Long-term value equals present value plus future value.
$V^\pi(s) = Q^\pi(s, \pi(s))$
$Q^\pi(s,a) = R(s,a) + \gamma \sum_{s'} P(s,a,s') V^\pi(s')$

The MDP formalism gives us a way to evaluate a given policy, using the Bellman equations. The thesis adopts a matrix notation that expresses the Bellman equations more compactly than the traditional form, and this presentation adopts a visual representation intended to show the structure of the equation as a kind of circuit.

Given a policy, the long-term value $V^\pi(s)$ of a state is equal to the long-term value $Q^\pi(s, \pi(s))$ of executing the policy action at that state. This state-action value is equal to the immediate reward for that state-action plus the expected long-term value of the successor state for that state-action.

Evaluating Policies with Value Functions

The Bellman Equation

- State value depends on policy and action values
- Long-term value equals present value plus future value.

$$
\begin{aligned}
V^\pi &= \pi Q^\pi \\
Q^\pi &= R + \gamma P V^\pi
\end{aligned}
$$

The MDP formalism gives us a way to evaluate a given policy, using the Bellman equations. The thesis adopts a matrix notation that expresses the Bellman equations more compactly than the traditional form, and this presentation adopts a visual representation intended to show the structure of the equation as a kind of circuit.

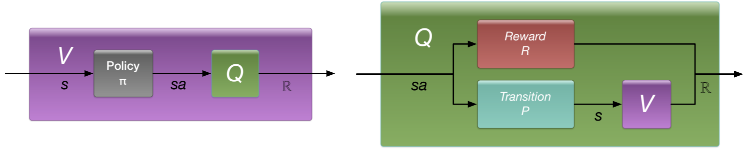
Given a policy, the long-term value $V^\pi(s)$ of a state is equal to the long-term value $Q^\pi(s, \pi(s))$ of executing the policy action at that state. This state-action value is equal to the immediate reward for that state-action plus the expected long-term value of the successor state for that state-action.

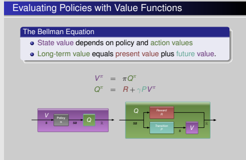
Evaluating Policies with Value Functions

The Bellman Equation

- State value depends on policy and action values
- Long-term value equals present value plus future value.

$$
\begin{aligned}
V^\pi &= \pi Q^\pi \\
Q^\pi &= R + \gamma P V^\pi
\end{aligned}
$$

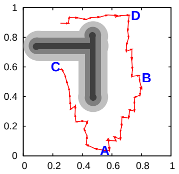




The MDP formalism gives us a way to evaluate a given policy, using the Bellman equations. The thesis adopts a matrix notation that expresses the Bellman equations more compactly than the traditional form, and this presentation adopts a visual representation intended to show the structure of the equation as a kind of circuit.

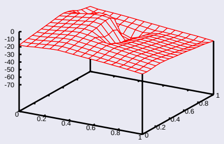
Given a policy, the long-term value $V^\pi(s)$ of a state is equal to the long-term value $Q^\pi(s, \pi(s))$ of executing the policy action at that state. This state-action value is equal to the immediate reward for that state-action plus the expected long-term value of the successor state for that state-action.

Example: An Optimal Value Function

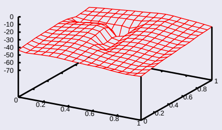


- Some policy achieves maximal V^*
- Planning algorithms compute V^* from R, P
- But RL algorithms don't know R and P

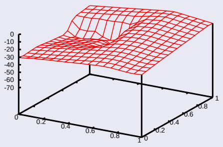
$V^*(x, y, \{C\})$

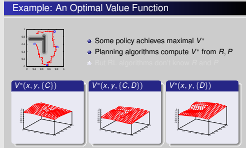


$V^*(x, y, \{C, D\})$



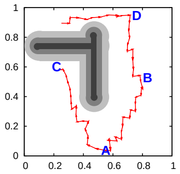
$V^*(x, y, \{D\})$





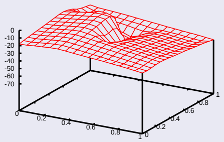
Planning algoritms can compute the optimal value function for the resource-gathering domain, given access to its reward and transition functions. In this four-resource instance, the state value function V is six-dimensional, but this slide shows some two-dimensional slices. For example, the first figure shows the value of each x and y coordinate assuming the only uncollected resource is resource C.

Example: An Optimal Value Function

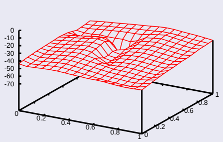


- Some policy achieves maximal V^*
- Planning algorithms compute V^* from R, P
- But RL algorithms don't know R and P

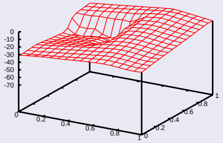
$V^*(x, y, \{C\})$

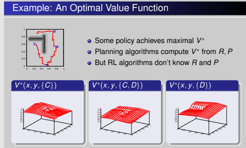


$V^*(x, y, \{C, D\})$



$V^*(x, y, \{D\})$



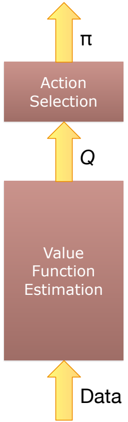


Planning algoritms can compute the optimal value function for the resource-gathering domain, given access to its reward and transition functions. In this four-resource instance, the state value function V is six-dimensional, but this slide shows some two-dimensional slices. For example, the first figure shows the value of each x and y coordinate assuming the only uncollected resource is resource C.

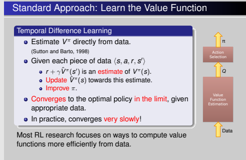
Standard Approach: Learn the Value Function

Temporal Difference Learning

- Estimate V^π directly from data.
  (Sutton and Barto, 1998)
- Given each piece of data $\langle s, a, r, s' \rangle$
  - $r + \gamma \hat{V}^\pi(s')$ is an estimate of $V^\pi(s)$.
  - Update $\hat{V}^\pi(s)$ towards this estimate.
  - Improve π.
- Converges to the optimal policy in the limit, given appropriate data.
- In practice, converges very slowly!



Most RL research focuses on ways to compute value functions more efficiently from data.



Most RL algorithms are model-free, meaning that estimate the optimal value function directly from data, without needing the true reward and transition functions or even constructing a model or estimate of these functions. The fact that it is possible to converge to optimal behavior in this fashion is somewhat remarkable and providing the impetus for the field of RL. However, these original algorithms only guaranteed convergence in the limit and by making certain key assumptions, such as that unbounded amounts of data for every state-action would become available.

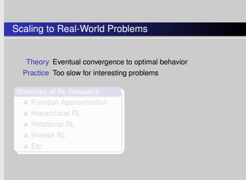
Scaling to Real-World Problems

Theory Eventual convergence to optimal behavior

Practice Too slow for interesting problems

Branches of RL Research

- Function Approximation
- Hierarchical RL
- Relational RL
- Inverse RL
- Etc.



The desire to apply RL to more practical problems gave rise to innumerable branches of RL research, all of which seek to improve the efficiency of RL. This thesis focuses on at least two such methods, function approximation (Section 2) and hierarchical decomposition (Section 3).

Scaling to Real-World Problems

Theory Eventual convergence to optimal behavior

Practice Too slow for interesting problems

Branches of RL Research

- Function Approximation
- Hierarchical RL
- Relational RL
- Inverse RL
- Etc.

Scaling to Real-World Problems
Theory Eventual convergence to optimal behavior
Practice Too slow for interesting problems
Branches of RL Research
Function Approximation
Hierarchical RL
Relational RL
Inverse RL
Etc.

The desire to apply RL to more practical problems gave rise to innumerable branches of RL research, all of which seek to improve the efficiency of RL. This thesis focuses on at least two such methods, function approximation (Section 2) and hierarchical decomposition (Section 3).

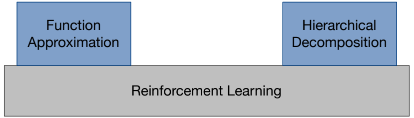
Scaling to Real-World Problems

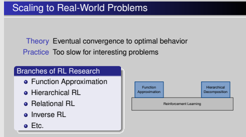
Theory Eventual convergence to optimal behavior

Practice Too slow for interesting problems

Branches of RL Research

- Function Approximation
- Hierarchical RL
- Relational RL
- Inverse RL
- Etc.





The desire to apply RL to more practical problems gave rise to innumerable branches of RL research, all of which seek to improve the efficiency of RL. This thesis focuses on at least two such methods, function approximation (Section 2) and hierarchical decomposition (Section 3).

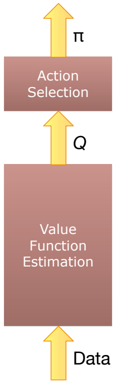
Exploration and Exploitation

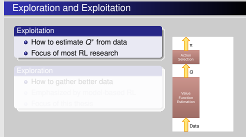
Exploitation

- How to estimate Q^* from data
- Focus of most RL research

Exploration

- How to gather better data
- Emphasized by model-based RL
- Focus of this thesis





The branch of RL at the core of this thesis is model-based RL, which underlies much of the research showing that it is possible to learn near-optimal policies most of the time given finite (and polynomial in certain parameters) interactions with the environment. The key idea behind such results is that it is not sufficient just to estimate the optimal value function given the currently available data. Instead, an efficient agent realizes that it should gather more data now to improve its estimate of the optimal value function later. In other words, agents should reason explicitly about the fact that they control the cycle of interaction with the environment and thereby determine the data from which they learn (active learning).

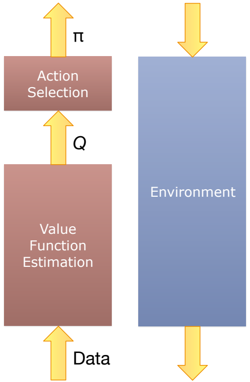
Exploration and Exploitation

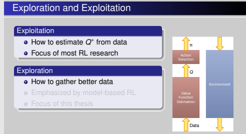
Exploitation

- How to estimate Q^* from data
- Focus of most RL research

Exploration

- How to gather better data
- Emphasized by model-based RL
- Focus of this thesis





The branch of RL at the core of this thesis is model-based RL, which underlies much of the research showing that it is possible to learn near-optimal policies most of the time given finite (and polynomial in certain parameters) interactions with the environment. The key idea behind such results is that it is not sufficient just to estimate the optimal value function given the currently available data. Instead, an efficient agent realizes that it should gather more data now to improve its estimate of the optimal value function later. In other words, agents should reason explicitly about the fact that they control the cycle of interaction with the environment and thereby determine the data from which they learn (active learning).

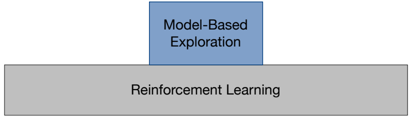
Exploration and Exploitation

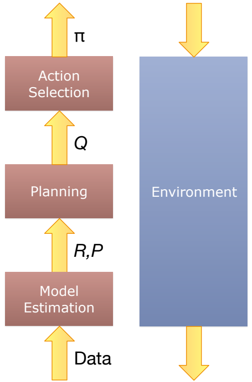
Exploitation

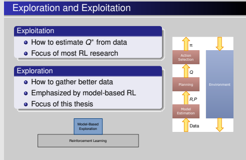
- How to estimate Q^* from data
- Focus of most RL research

Exploration

- How to gather better data
- Emphasized by model-based RL
- Focus of this thesis

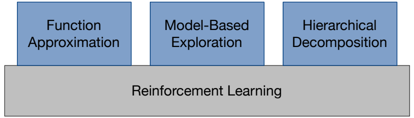






The branch of RL at the core of this thesis is model-based RL, which underlies much of the research showing that it is possible to learn near-optimal policies most of the time given finite (and polynomial in certain parameters) interactions with the environment. The key idea behind such results is that it is not sufficient just to estimate the optimal value function given the currently available data. Instead, an efficient agent realizes that it should gather more data now to improve its estimate of the optimal value function later. In other words, agents should reason explicitly about the fact that they control the cycle of interaction with the environment and thereby determine the data from which they learn (active learning).

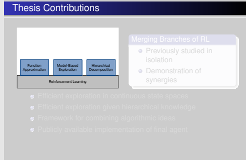
Thesis Contributions



Merging Branches of RL

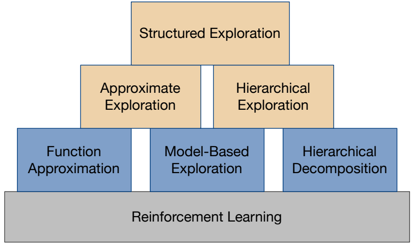
- Previously studied in isolation
- Demonstration of synergies

- Efficient exploration in continuous state spaces
- Efficient exploration given hierarchical knowledge
- Framework for combining algorithmic ideas
- Publicly available implementation of final agent



Surprisingly, the three branches on which this thesis focuses are fairly independent. Few researchers have tried to combine the ideas from multiple of these branches. The primary contribution of this thesis is the integration of these ideas, thereby showing their synergies and extending the reach of RL. The source code for an agent combining all these technologies is publicly available at `http://library.rl-community.org/wiki/Fitted_R-MAXQ`.

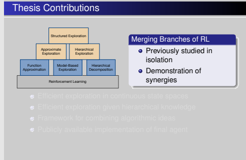
Thesis Contributions



Merging Branches of RL

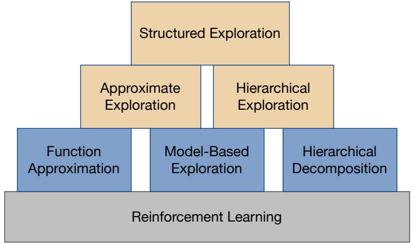
- Previously studied in isolation
- Demonstration of synergies

- Efficient exploration in continuous state spaces
- Efficient exploration given hierarchical knowledge
- Framework for combining algorithmic ideas
- Publicly available implementation of final agent



Surprisingly, the three branches on which this thesis focuses are fairly independent. Few researchers have tried to combine the ideas from multiple of these branches. The primary contribution of this thesis is the integration of these ideas, thereby showing their synergies and extending the reach of RL. The source code for an agent combining all these technologies is publicly available at `http://library.rl-community.org/wiki/Fitted_R-MAXQ`.

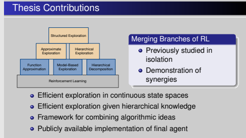
Thesis Contributions



Merging Branches of RL

- Previously studied in isolation
- Demonstration of synergies

- Efficient exploration in continuous state spaces
- Efficient exploration given hierarchical knowledge
- Framework for combining algorithmic ideas
- Publicly available implementation of final agent



Surprisingly, the three branches on which this thesis focuses are fairly independent. Few researchers have tried to combine the ideas from multiple of these branches. The primary contribution of this thesis is the integration of these ideas, thereby showing their synergies and extending the reach of RL. The source code for an agent combining all these technologies is publicly available at `http://library.rl-community.org/wiki/Fitted_R-MAXQ`.

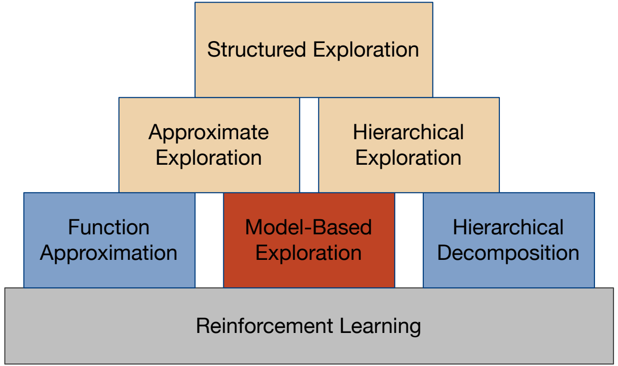
Outline

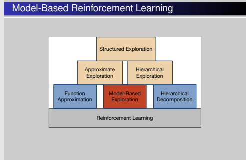
1. Introduction
2. Exploration and Approximation
   - Model-Based Exploration
   - Generalization in Large State Spaces
   - The Fitted R-MAX Algorithm
3. Exploration and Hierarchy
4. Conclusion

Outline

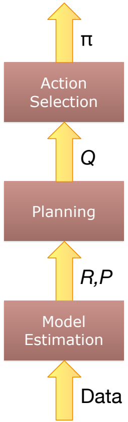
Introduction
Exploration and Approximation
Model-Based Exploration
Generalization in Large State Spaces
The Fitted R-MAX Algorithm
Exploration and Hierarchy
Conclusion

Model-Based Reinforcement Learning





Model-Based Reinforcement Learning

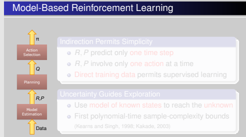


Indirection Permits Simplicity

- R, P predict only one time step
- R, P involve only one action at a time
- Direct training data permits supervised learning

Uncertainty Guides Exploration

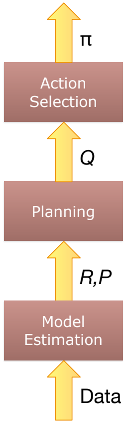
- Use model of known states to reach the unknown
- First polynomial-time sample-complexity bounds (Kearns and Singh, 1998; Kakade, 2003)



Why construct a model of the environment (and then plan given the model) when it is possible just to estimate the value function directly? One key benefit is that model-estimation is a supervised learning process, permitting the use of well-understood techniques, and the model may be intrinsically easier to learn than the value function. In particular, each entry in the value function folds together all future time steps and all possible actions; for each state-action, the model only predicts one time step ahead and only involves one action at a time.

Furthermore, uncertainty in the model provides a rational basis for exploration. High variance at a given state-action implies that additional data at that state-action will lower the variance. Such reasoning underlies many of the theoretical guarantees currently available for RL.

Model-Based Reinforcement Learning

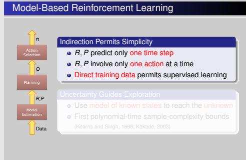


Indirection Permits Simplicity

- R, P predict only one time step
- R, P involve only one action at a time
- Direct training data permits supervised learning

Uncertainty Guides Exploration

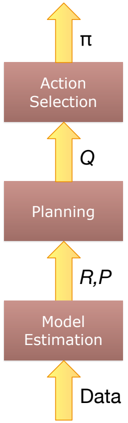
- Use model of known states to reach the unknown
- First polynomial-time sample-complexity bounds (Kearns and Singh, 1998; Kakade, 2003)



Why construct a model of the environment (and then plan given the model) when it is possible just to estimate the value function directly? One key benefit is that model-estimation is a supervised learning process, permitting the use of well-understood techniques, and the model may be intrinsically easier to learn than the value function. In particular, each entry in the value function folds together all future time steps and all possible actions; for each state-action, the model only predicts one time step ahead and only involves one action at a time.

Furthermore, uncertainty in the model provides a rational basis for exploration. High variance at a given state-action implies that additional data at that state-action will lower the variance. Such reasoning underlies many of the theoretical guarantees currently available for RL.

Model-Based Reinforcement Learning

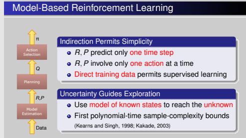


Indirection Permits Simplicity

- R, P predict only one time step
- R, P involve only one action at a time
- Direct training data permits supervised learning

Uncertainty Guides Exploration

- Use model of known states to reach the unknown
- First polynomial-time sample-complexity bounds (Kearns and Singh, 1998; Kakade, 2003)



Why construct a model of the environment (and then plan given the model) when it is possible just to estimate the value function directly? One key benefit is that model-estimation is a supervised learning process, permitting the use of well-understood techniques, and the model may be intrinsically easier to learn than the value function. In particular, each entry in the value function folds together all future time steps and all possible actions; for each state-action, the model only predicts one time step ahead and only involves one action at a time.

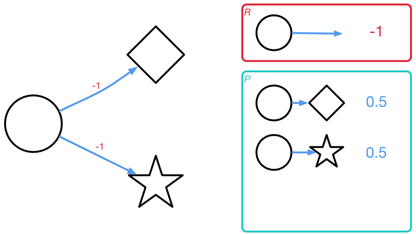
Furthermore, uncertainty in the model provides a rational basis for exploration. High variance at a given state-action implies that additional data at that state-action will lower the variance. Such reasoning underlies many of the theoretical guarantees currently available for RL.

Simple and Efficient Learning with R-max

(Moore and Atkeson, 1993; Brafman and Tennenholtz, 2002)

Maximum-Likelihood Estimation

- Straightforward in finite state spaces
- Unreliable with small sample sizes

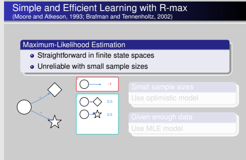


Small sample sizes

Use optimistic model

Given enough data

Use MLE model



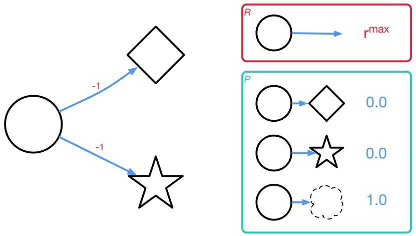
This thesis builds upon a simple model-based RL algorithm known as R-MAX, which acknowledges that the standard maximum-likelihood estimate of the transition function is unreliable given a small sample size. For example, if we have executed the blue action from the circle state twice, we might estimate a 50-50 chance of ending up in either the diamond or star states the next time we try this state-action. But if star is a very bad state, we might be afraid to try this state-action again, even if it's actually optimal. R-MAX employs an optimistic model whenever the amount of data for a given state-action falls beneath a certain threshold. This optimistic model gives an immediate reward equal to some upper bound on the value function, and it transitions to some artificial terminal state (instead of any existing state, whose values are unknown). R-MAX only reverts to the MLE model given a large enough sample size, in this case 5.

Simple and Efficient Learning with R-max

(Moore and Atkeson, 1993; Brafman and Tennenholtz, 2002)

Maximum-Likelihood Estimation

- Straightforward in finite state spaces
- Unreliable with small sample sizes

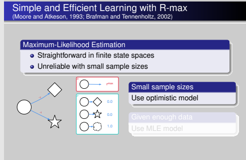


Small sample sizes

Use optimistic model

Given enough data

Use MLE model



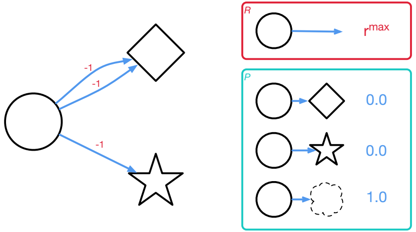
This thesis builds upon a simple model-based RL algorithm known as R-MAX, which acknowledges that the standard maximum-likelihood estimate of the transition function is unreliable given a small sample size. For example, if we have executed the blue action from the circle state twice, we might estimate a 50-50 chance of ending up in either the diamond or star states the next time we try this state-action. But if star is a very bad state, we might be afraid to try this state-action again, even if it's actually optimal. R-MAX employs an optimistic model whenever the amount of data for a given state-action falls beneath a certain threshold. This optimistic model gives an immediate reward equal to some upper bound on the value function, and it transitions to some artificial terminal state (instead of any existing state, whose values are unknown). R-MAX only reverts to the MLE model given a large enough sample size, in this case 5.

Simple and Efficient Learning with R-max

(Moore and Atkeson, 1993; Brafman and Tennenholtz, 2002)

Maximum-Likelihood Estimation

- Straightforward in finite state spaces
- Unreliable with small sample sizes

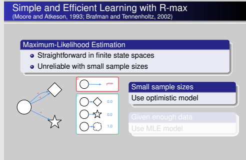


Small sample sizes

Use optimistic model

Given enough data

Use MLE model



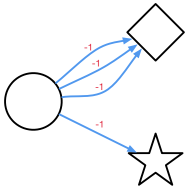
This thesis builds upon a simple model-based RL algorithm known as R-MAX, which acknowledges that the standard maximum-likelihood estimate of the transition function is unreliable given a small sample size. For example, if we have executed the blue action from the circle state twice, we might estimate a 50-50 chance of ending up in either the diamond or star states the next time we try this state-action. But if star is a very bad state, we might be afraid to try this state-action again, even if it's actually optimal. R-MAX employs an optimistic model whenever the amount of data for a given state-action falls beneath a certain threshold. This optimistic model gives an immediate reward equal to some upper bound on the value function, and it transitions to some artificial terminal state (instead of any existing state, whose values are unknown). R-MAX only reverts to the MLE model given a large enough sample size, in this case 5.

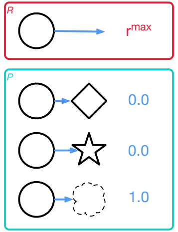
Simple and Efficient Learning with R-max

(Moore and Atkeson, 1993; Brafman and Tennenholtz, 2002)

Maximum-Likelihood Estimation

- Straightforward in finite state spaces
- Unreliable with small sample sizes



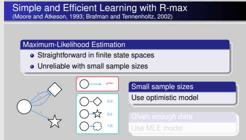


Small sample sizes

Use optimistic model

Given enough data

Use MLE model

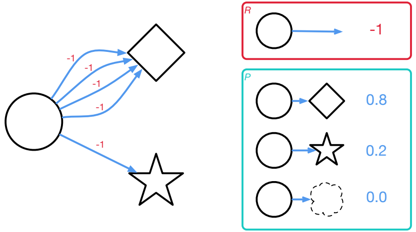


This thesis builds upon a simple model-based RL algorithm known as R-MAX, which acknowledges that the standard maximum-likelihood estimate of the transition function is unreliable given a small sample size. For example, if we have executed the blue action from the circle state twice, we might estimate a 50-50 chance of ending up in either the diamond or star states the next time we try this state-action. But if star is a very bad state, we might be afraid to try this state-action again, even if it's actually optimal. R-MAX employs an optimistic model whenever the amount of data for a given state-action falls beneath a certain threshold. This optimistic model gives an immediate reward equal to some upper bound on the value function, and it transitions to some artificial terminal state (instead of any existing state, whose values are unknown). R-MAX only reverts to the MLE model given a large enough sample size, in this case 5.

Simple and Efficient Learning with R-max
(Moore and Atkeson, 1993; Brafman and Tennenholtz, 2002)

Maximum-Likelihood Estimation

- Straightforward in finite state spaces
- Unreliable with small sample sizes

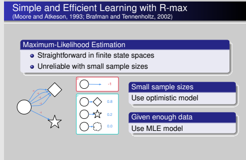


Small sample sizes

Use optimistic model

Given enough data

Use MLE model



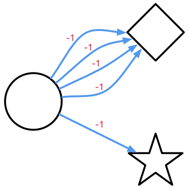
This thesis builds upon a simple model-based RL algorithm known as R-MAX, which acknowledges that the standard maximum-likelihood estimate of the transition function is unreliable given a small sample size. For example, if we have executed the blue action from the circle state twice, we might estimate a 50-50 chance of ending up in either the diamond or star states the next time we try this state-action. But if star is a very bad state, we might be afraid to try this state-action again, even if it's actually optimal. R-MAX employs an optimistic model whenever the amount of data for a given state-action falls beneath a certain threshold. This optimistic model gives an immediate reward equal to some upper bound on the value function, and it transitions to some artificial terminal state (instead of any existing state, whose values are unknown). R-MAX only reverts to the MLE model given a large enough sample size, in this case 5.

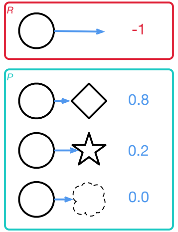
Simple and Efficient Learning with R-max

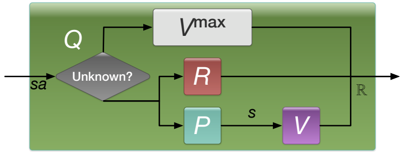
(Moore and Atkeson, 1993; Brafman and Tennenholtz, 2002)

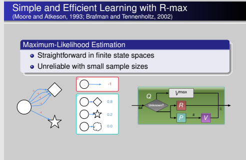
Maximum-Likelihood Estimation

- Straightforward in finite state spaces
- Unreliable with small sample sizes









This thesis builds upon a simple model-based RL algorithm known as R-MAX, which acknowledges that the standard maximum-likelihood estimate of the transition function is unreliable given a small sample size. For example, if we have executed the blue action from the circle state twice, we might estimate a 50-50 chance of ending up in either the diamond or star states the next time we try this state-action. But if star is a very bad state, we might be afraid to try this state-action again, even if it's actually optimal. R-MAX employs an optimistic model whenever the amount of data for a given state-action falls beneath a certain threshold. This optimistic model gives an immediate reward equal to some upper bound on the value function, and it transitions to some artificial terminal state (instead of any existing state, whose values are unknown). R-MAX only reverts to the MLE model given a large enough sample size, in this case 5.

Challenges for Model-Based Reinforcement Learning

Computational Complexity

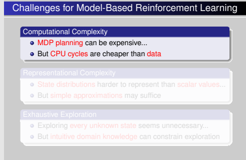
- MDP planning can be expensive...
- But CPU cycles are cheaper than data

Representational Complexity

- State distributions harder to represent than scalar values...
- But simple approximations may suffice

Exhaustive Exploration

- Exploring every unknown state seems unnecessary...
- But intuitive domain knowledge can constrain exploration



Given the benefits of model-based RL, why does the vast majority of RL research rely on model-free methods? First, the computational cost of planning given a model tends to be relatively high. However, the limiting factor in most practical applications is data, not CPU time.

Second, defining representations and algorithms that involve transition models, which specify distributions over successor states, seems more complex than working with value functions, which specify scalar values. However, models may be easier than value functions to approximate adequately, since they only need to be accurate for one action at a time and for one time step.

Finally, algorithms such as R-MAX that employ model-based exploration have a reputation for exploring every reachable state-action too aggressively. A key contribution of this thesis is a demonstration that such exploration can be reduced used model generalization and hierarchical constraints.

Challenges for Model-Based Reinforcement Learning

Computational Complexity

- MDP planning can be expensive...
- But CPU cycles are cheaper than data

Representational Complexity

- State distributions harder to represent than scalar values...
- But simple approximations may suffice

Exhaustive Exploration

- Exploring every unknown state seems unnecessary...
- But intuitive domain knowledge can constrain exploration

Challenges for Model-Based Reinforcement Learning
Computational Complexity
MDP planning can be expensive...
But CPU cycles are cheaper than data
Representational Complexity
State distributions harder to represent than scalar values...
But simple approximations may suffice
Exhaustive Exploration
Exploring every unknown state seems unnecessary...
But intuitive domain knowledge can constrain exploration

Given the benefits of model-based RL, why does the vast majority of RL research rely on model-free methods? First, the computational cost of planning given a model tends to be relatively high. However, the limiting factor in most practical applications is data, not CPU time.

Second, defining representations and algorithms that involve transition models, which specify distributions over successor states, seems more complex than working with value functions, which specify scalar values. However, models may be easier than value functions to approximate adequately, since they only need to be accurate for one action at a time and for one time step.

Finally, algorithms such as R-MAX that employ model-based exploration have a reputation for exploring every reachable state-action too aggressively. A key contribution of this thesis is a demonstration that such exploration can be reduced used model generalization and hierarchical constraints.

Challenges for Model-Based Reinforcement Learning

Computational Complexity

- MDP planning can be expensive...
- But CPU cycles are cheaper than data

Representational Complexity

- State distributions harder to represent than scalar values...
- But simple approximations may suffice

Exhaustive Exploration

- Exploring every unknown state seems unnecessary...
- But intuitive domain knowledge can constrain exploration

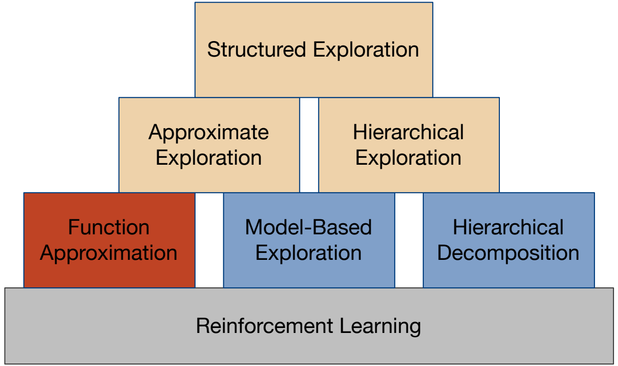
Challenges for Model-Based Reinforcement Learning
Computational Complexity
MDP planning can be expensive...
But CPU cycles are cheaper than data
Representational Complexity
State distributions harder to represent than scalar values...
But simple approximations may suffice
Exhaustive Exploration
Exploring every unknown state seems unnecessary...
But intuitive domain knowledge can constrain exploration

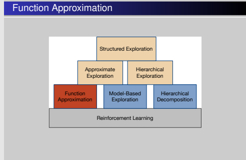
Given the benefits of model-based RL, why does the vast majority of RL research rely on model-free methods? First, the computational cost of planning given a model tends to be relatively high. However, the limiting factor in most practical applications is data, not CPU time.

Second, defining representations and algorithms that involve transition models, which specify distributions over successor states, seems more complex than working with value functions, which specify scalar values. However, models may be easier than value functions to approximate adequately, since they only need to be accurate for one action at a time and for one time step.

Finally, algorithms such as R-MAX that employ model-based exploration have a reputation for exploring every reachable state-action too aggressively. A key contribution of this thesis is a demonstration that such exploration can be reduced used model generalization and hierarchical constraints.

Function Approximation





Function Approximation

Problem Exact representation of V requires a parameter for each state. Many environments have infinite states!

Key Idea

Represent V^π using a small number of parameters.

- Examples
  - The weights of a neural network
  - Coefficients of some basis functions: $V^\pi = \sum_i w_i^\pi \phi_i$
- Generalization of values
  - Changing $V^\pi(s)$ changes one or more parameters.
  - Each parameter influences the value of several states.

One prerequisite for learning in real-world settings is the ability to cope with infinite state spaces. Research into function approximation extends model-free RL methods by replacing exact representations of the value function, which store each state's value independently, which a function parameterized by a small number of parameters. This approach effectively reduces the degrees of freedom of the estimated value function, introducing some regularization. In particular, adjusting the value function given data at one state tends to improve the value function for other states that depend on the same parameters. This effect both facilitates successful applications of RL to continuous state spaces as well as invalidating many theoretical convergence guarantees.

Function Approximation

Problem Exact representation of V requires a parameter for each state. Many environments have infinite states!

Key Idea

Represent V^π using a small number of parameters.

- Examples
  - The weights of a neural network
  - Coefficients of some basis functions: $V^\pi = \sum_i w_i^\pi \phi_i$
- Generalization of values
  - Changing $V^\pi(s)$ changes one or more parameters.
  - Each parameter influences the value of several states.

One prerequisite for learning in real-world settings is the ability to cope with infinite state spaces. Research into function approximation extends model-free RL methods by replacing exact representations of the value function, which store each state's value independently, which a function parameterized by a small number of parameters. This approach effectively reduces the degrees of freedom of the estimated value function, introducing some regularization. In particular, adjusting the value function given data at one state tends to improve the value function for other states that depend on the same parameters. This effect both facilitates successful applications of RL to continuous state spaces as well as invalidating many theoretical convergence guarantees.

Function Approximation

Problem Exact representation of V requires a parameter for each state. Many environments have infinite states!

Key Idea

Represent V^π using a small number of parameters.

- Examples
  - The weights of a neural network
  - Coefficients of some basis functions: $V^\pi = \sum_i w_i^\pi \phi_i$
- Generalization of values
  - Changing $V^\pi(s)$ changes one or more parameters.
  - Each parameter influences the value of several states.

One prerequisite for learning in real-world settings is the ability to cope with infinite state spaces. Research into function approximation extends model-free RL methods by replacing exact representations of the value function, which store each state's value independently, which a function parameterized by a small number of parameters. This approach effectively reduces the degrees of freedom of the estimated value function, introducing some regularization. In particular, adjusting the value function given data at one state tends to improve the value function for other states that depend on the same parameters. This effect both facilitates successful applications of RL to continuous state spaces as well as invalidating many theoretical convergence guarantees.

Function Approximation

Problem Exact representation of V requires a parameter for each state. Many environments have infinite states!

Key Idea

Represent V^π using a small number of parameters.

- Examples
  - The weights of a neural network
  - Coefficients of some basis functions: $V^\pi = \sum_i w_i^\pi \phi_i$
- Generalization of values
  - Changing $V^\pi(s)$ changes one or more parameters.
  - Each parameter influences the value of several states.

Function Approximation
Problem Exact representation of V requires a parameter for each state. Many environments have infinite states!
Key Idea
Represent V^π using a small number of parameters.
Examples
The weights of a neural network
Coefficients of some basis functions: $V^\pi = \sum_i w_i^\pi \phi_i$
Generalization of values
Changing $V^\pi(s)$ changes one or more parameters.
Each parameter influences the value of several states.

One prerequisite for learning in real-world settings is the ability to cope with infinite state spaces. Research into function approximation extends model-free RL methods by replacing exact representations of the value function, which store each state's value independently, which a function parameterized by a small number of parameters. This approach effectively reduces the degrees of freedom of the estimated value function, introducing some regularization. In particular, adjusting the value function given data at one state tends to improve the value function for other states that depend on the same parameters. This effect both facilitates successful applications of RL to continuous state spaces as well as invalidating many theoretical convergence guarantees.

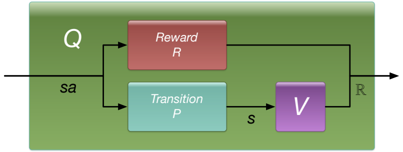
Fitted Value Iteration
(Gordon, 1995)

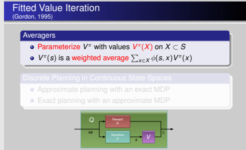
Averagers

- Parameterize V^{π} with values $V^{\pi}(X)$ on $X \subset S$
- $V^{\pi}(s)$ is a weighted average $\sum_{x \in X} \phi(s, x) V^{\pi}(x)$

Discrete Planning in Continuous State Spaces

- Approximate planning with an exact MDP
- Exact planning with an approximate MDP





This thesis investigates a form of function approximation originally studied in the context of planning with continuous-state MDPs. For the family of function approximations known as averagers, the parameters of the continuous-state function are the values of the function over some small subset $X \subset S$. For states outside X, the value is approximated as a weighted average of the values over X.

A key benefit of this approximation is that a standard planning algorithm, value iteration, always remains stable despite the approximate nature of the value function. Gordon (1995) proved this stability by showing that computing the approximate value function in this manner for the original exact MDP is equivalent to computing the exact value function for a derived MDP, in which the transition function incorporates the averaging approximation. In particular, the transition function of the derived MDP assumes that every action always transitions to states in X.

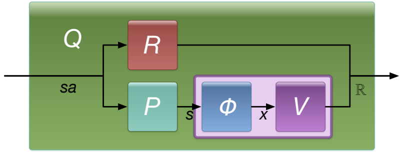
Fitted Value Iteration
(Gordon, 1995)

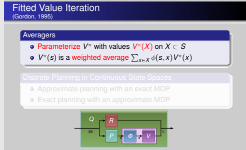
Averagers

- Parameterize V^π with values $V^\pi(X)$ on $X \subset S$
- $V^\pi(s)$ is a weighted average $\sum_{x \in X} \phi(s,x) V^\pi(x)$

Discrete Planning in Continuous State Spaces

- Approximate planning with an exact MDP
- Exact planning with an approximate MDP





This thesis investigates a form of function approximation originally studied in the context of planning with continuous-state MDPs. For the family of function approximations known as averagers, the parameters of the continuous-state function are the values of the function over some small subset $X \subset S$. For states outside X, the value is approximated as a weighted average of the values over X.

A key benefit of this approximation is that a standard planning algorithm, value iteration, always remains stable despite the approximate nature of the value function. Gordon (1995) proved this stability by showing that computing the approximate value function in this manner for the original exact MDP is equivalent to computing the exact value function for a derived MDP, in which the transition function incorporates the averaging approximation. In particular, the transition function of the derived MDP assumes that every action always transitions to states in X.

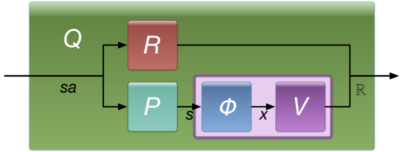
Fitted Value Iteration
(Gordon, 1995)

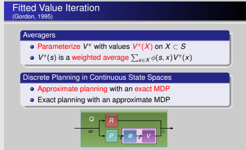
Averagers

- Parameterize V^π with values $V^\pi(X)$ on $X \subset S$
- $V^\pi(s)$ is a weighted average $\sum_{x \in X} \phi(s,x) V^\pi(x)$

Discrete Planning in Continuous State Spaces

- Approximate planning with an exact MDP
- Exact planning with an approximate MDP





This thesis investigates a form of function approximation originally studied in the context of planning with continuous-state MDPs. For the family of function approximations known as averagers, the parameters of the continuous-state function are the values of the function over some small subset $X \subset S$. For states outside X, the value is approximated as a weighted average of the values over X.

A key benefit of this approximation is that a standard planning algorithm, value iteration, always remains stable despite the approximate nature of the value function. Gordon (1995) proved this stability by showing that computing the approximate value function in this manner for the original exact MDP is equivalent to computing the exact value function for a derived MDP, in which the transition function incorporates the averaging approximation. In particular, the transition function of the derived MDP assumes that every action always transitions to states in X.

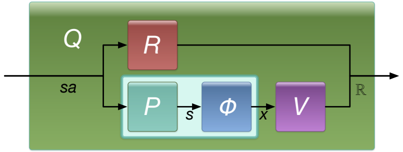
Fitted Value Iteration
(Gordon, 1995)

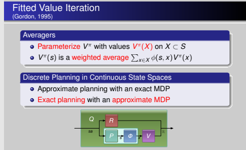
Averagers

- Parameterize V^π with values $V^\pi(X)$ on $X \subset S$
- $V^\pi(s)$ is a weighted average $\sum_{x \in X} \phi(s,x) V^\pi(x)$

Discrete Planning in Continuous State Spaces

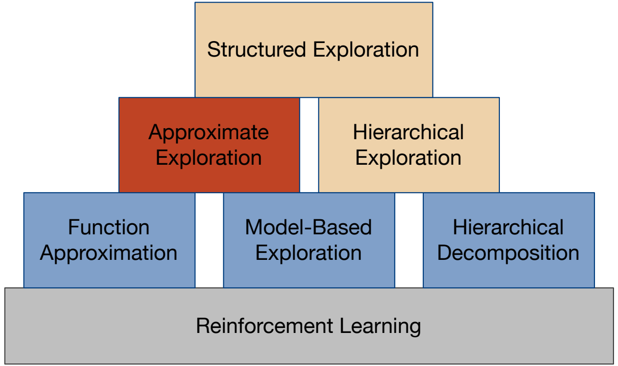
- Approximate planning with an exact MDP
- Exact planning with an approximate MDP

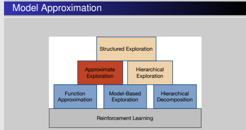




This thesis investigates a form of function approximation originally studied in the context of planning with continuous-state MDPs. For the family of function approximations known as averagers, the parameters of the continuous-state function are the values of the function over some small subset $X \subset S$. For states outside X, the value is approximated as a weighted average of the values over X. A key benefit of this approximation is that a standard planning algorithm, value iteration, always remains stable despite the approximate nature of the value function. Gordon (1995) proved this stability by showing that computing the approximate value function in this manner for the original exact MDP is equivalent to computing the exact value function for a derived MDP, in which the transition function incorporates the averaging approximation. In particular, the transition function of the derived MDP assumes that every action always transitions to states in X.

Model Approximation





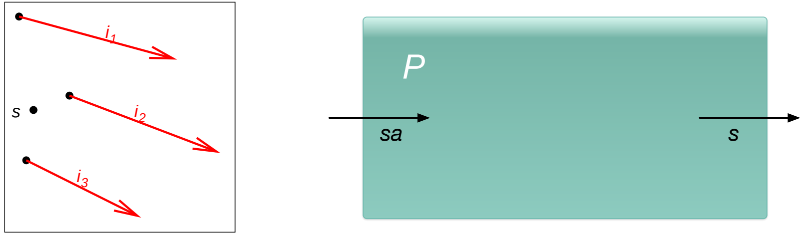
Model Approximation

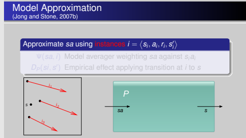
(Jong and Stone, 2007b)

Approximate *sa* using instances $i = \langle s_i, a_i, r_i, s_i' \rangle$

$\Psi(sa, i)$ Model averager weighting *sa* against $s_i a_i$

$D_P(si, s')$ Empirical effect applying transition at i to s





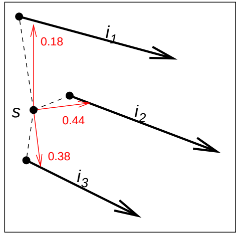
This thesis applies averaging approximation to the model, not just the value function. Assume that we want to define the transition function for a given state-action *sa*, using our available data, stored as a set of instances, one for each time step. These instances play the same role that the state set X plays in value approximation: we approximate the result of state-action *sa* using some weighted average over instances. To generate a random successor state for *sa*, we randomly choose an instance i, weighted by the function $\psi(sa, i)$. Then we apply to s whatever relative (vector) effect we observed at instance i. The approximation transition matrix composes the approximate "transition" to an instance with the instances' deterministic observed transitions, in much the same way that the derived transition function in fitted value iteration composes the given MDP's transition function with the approximate "transition" to a state in X.

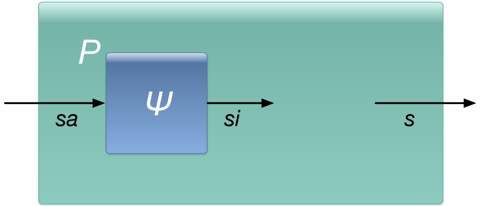
Model Approximation
(Jong and Stone, 2007b)

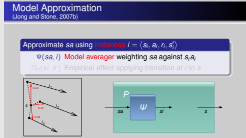
Approximate sa using instances $i = \langle s_i, a_i, r_i, s_i' \rangle$

$\Psi(sa, i)$ Model averager weighting sa against $s_i a_i$

$D_P(si, s')$ Empirical effect applying transition at i to s







This thesis applies averaging approximation to the model, not just the value function. Assume that we want to define the transition function for a given state-action sa, using our available data, stored as a set of instances, one for each time step. These instances play the same role that the state set X plays in value approximation: we approximate the result of state-action sa using some weighted average over instances. To generate a random successor state for sa, we randomly choose an instance i, weighted by the function $\psi(sa, i)$. Then we apply to s whatever relative (vector) effect we observed at instance i. The approximation transition matrix composes the approximate "transition" to an instance with the instances' deterministic observed transitions, in much the same way that the derived transition function in fitted value iteration composes the given MDP's transition function with the approximate "transition" to a state in X.

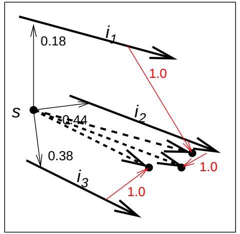
Model Approximation

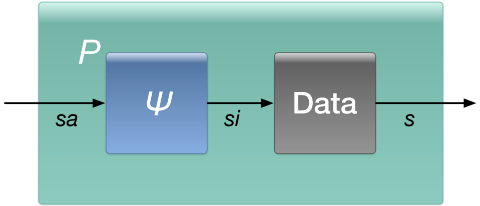
(Jong and Stone, 2007b)

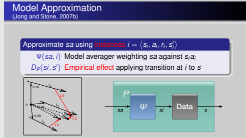
Approximate sa using instances $i = \langle s_i, a_i, r_i, s'_i \rangle$

$\Psi(sa, i)$ Model averager weighting sa against $s_i a_i$

$D_P(si, s')$ Empirical effect applying transition at i to s







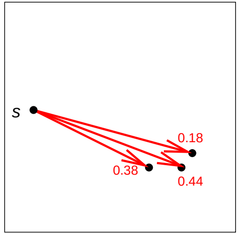
This thesis applies averaging approximation to the model, not just the value function. Assume that we want to define the transition function for a given state-action sa, using our available data, stored as a set of instances, one for each time step. These instances play the same role that the state set X plays in value approximation: we approximate the result of state-action sa using some weighted average over instances. To generate a random successor state for sa, we randomly choose an instance i, weighted by the function $\psi(sa, i)$. Then we apply to s whatever relative (vector) effect we observed at instance i. The approximation transition matrix composes the approximate "transition" to an instance with the instances' deterministic observed transitions, in much the same way that the derived transition function in fitted value iteration composes the given MDP's transition function with the approximate "transition" to a state in X.

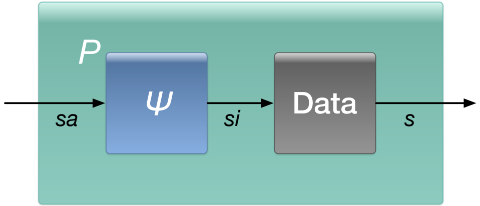
Model Approximation
(Jong and Stone, 2007b)

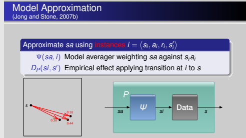
Approximate *sa* using instances $i = \langle s_i, a_i, r_i, s'_i \rangle$

$\Psi(sa, i)$ Model averager weighting *sa* against $s_i a_i$

$D_P(si, s')$ Empirical effect applying transition at *i* to *s*



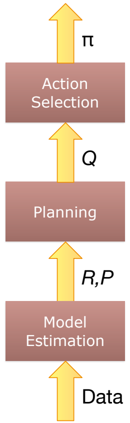


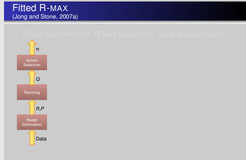


This thesis applies averaging approximation to the model, not just the value function. Assume that we want to define the transition function for a given state-action *sa*, using our available data, stored as a set of instances, one for each time step. These instances play the same role that the state set *X* plays in value approximation: we approximate the result of state-action *sa* using some weighted average over instances. To generate a random successor state for *sa*, we randomly choose an instance *i*, weighted by the function $\psi(sa, i)$. Then we apply to *s* whatever relative (vector) effect we observed at instance *i*. The approximation transition matrix composes the approximate "transition" to an instance with the instances' deterministic observed transitions, in much the same way that the derived transition function in fitted value iteration composes the given MDP's transition function with the approximate "transition" to a state in *X*.

Fitted R-MAX
(Jong and Stone, 2007a)

Model approximation, R-MAX exploration, value approximation

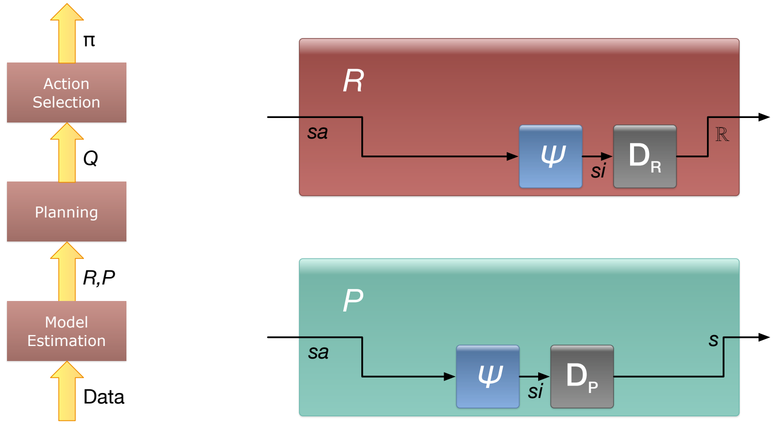


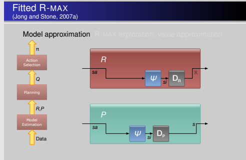


This thesis contributes the Fitted R-MAX algorithm, which modifies R-MAX only by changing the model estimate it uses for planning. Instead of maximum-likelihood estimation, Fitted R-MAX uses the averager-based approximation described on the previous slide. On top of that baseline model, it applies the same modifications to the Bellman equations that R-MAX applies, followed by the same modifications introduced by fitted value iteration to cope with the unbounded number of successor states in the model.

Fitted R-MAX
(Jong and Stone, 2007a)

Model approximation, R-MAX exploration, value approximation

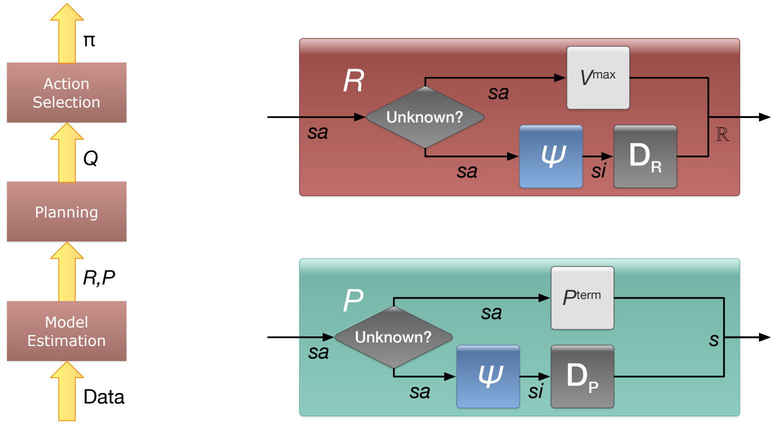


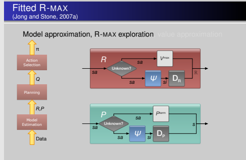


This thesis contributes the Fitted R-MAX algorithm, which modifies R-MAX only by changing the model estimate it uses for planning. Instead of maximum-likelihood estimation, Fitted R-MAX uses the averager-based approximation described on the previous slide. On top of that baseline model, it applies the same modifications to the Bellman equations that R-MAX applies, followed by the same modifications introduced by fitted value iteration to cope with the unbounded number of successor states in the model.

Fitted R-MAX
(Jong and Stone, 2007a)

Model approximation, R-MAX exploration, value approximation

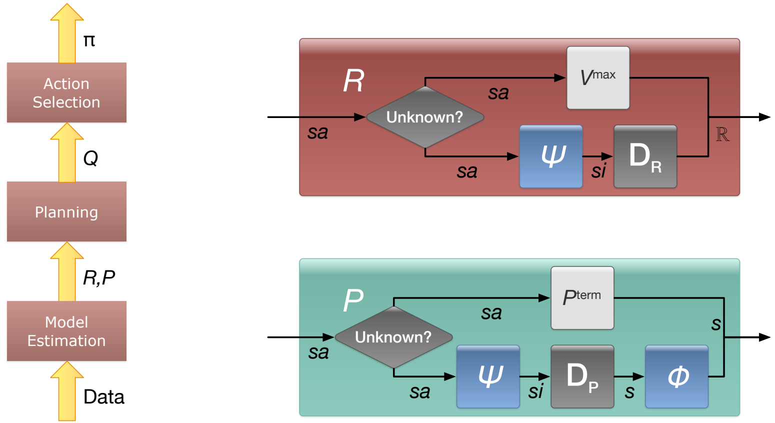


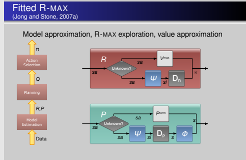


This thesis contributes the Fitted R-MAX algorithm, which modifies R-MAX only by changing the model estimate it uses for planning. Instead of maximum-likelihood estimation, Fitted R-MAX uses the averager-based approximation described on the previous slide. On top of that baseline model, it applies the same modifications to the Bellman equations that R-MAX applies, followed by the same modifications introduced by fitted value iteration to cope with the unbounded number of successor states in the model.

Fitted R-MAX
(Jong and Stone, 2007a)

Model approximation, R-MAX exploration, value approximation





This thesis contributes the Fitted R-MAX algorithm, which modifies R-MAX only by changing the model estimate it uses for planning. Instead of maximum-likelihood estimation, Fitted R-MAX uses the averager-based approximation described on the previous slide. On top of that baseline model, it applies the same modifications to the Bellman equations that R-MAX applies, followed by the same modifications introduced by fitted value iteration to cope with the unbounded number of successor states in the model.

An Instance of Fitted R-MAX

Model Averager

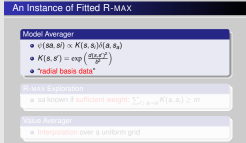
- $\psi(sa, si) \propto K(s, s_i)\delta(a, s_a)$
- $K(s, s') = \exp\left(\frac{d(s,s')^2}{b^2}\right)$
- "radial basis data"

R-MAX Exploration

- sa known if sufficient weight: $\sum_{i \mid a_i = a} K(s, s_i) \geq m$

Value Averager

- Interpolation over a uniform grid



The experiments in the thesis use one concrete instantiation of this algorithm. The averager it uses to weight instances for a given state-action sa applies a Gaussian kernel over every instance i with a matching action $a_i = a$, assuming a distance function defined over the state space. The R-MAX exploration mechanism considers a state-action known if the combined weight across all instances exceeds some threshold. The averager used to approximate the value function interpolates over a uniform grid over the state space.

An Instance of Fitted R-MAX

Model Averager

- $\psi(sa, si) \propto K(s, s_i)\delta(a, s_a)$
- $K(s, s') = \exp\left(\frac{d(s,s')^2}{b^2}\right)$
- "radial basis data"

R-MAX Exploration

- sa known if sufficient weight: $\sum_{i \mid a_i = a} K(s, s_i) \geq m$

Value Averager

- Interpolation over a uniform grid

An Instance of Fitted R-MAX
Model Averager
$\psi(sa, si) \propto K(s, s_i)\delta(a, s_a)$
$K(s, s') = \exp\left(\frac{d(s,s')^2}{b^2}\right)$
"radial basis data"
R-MAX Exploration
sa known if sufficient weight: $\sum_{i \mid a_i = a} K(s, s_i) \geq m$
Value Averager
Interpolation over a uniform grid

The experiments in the thesis use one concrete instantiation of this algorithm. The averager it uses to weight instances for a given state-action sa applies a Gaussian kernel over every instance i with a matching action $a_i = a$, assuming a distance function defined over the state space. The R-MAX exploration mechanism considers a state-action known if the combined weight across all instances exceeds some threshold. The averager used to approximate the value function interpolates over a uniform grid over the state space.

An Instance of Fitted R-MAX

Model Averager

- $\psi(sa, si) \propto K(s, s_i)\delta(a, s_a)$
- $K(s, s') = \exp\left(\frac{d(s,s')^2}{b^2}\right)$
- "radial basis data"

R-MAX Exploration

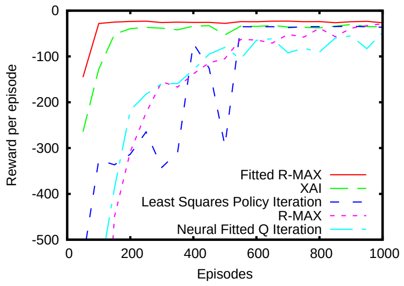
- sa known if sufficient weight: $\sum_{i \mid a_i = a} K(s, s_i) \geq m$

Value Averager

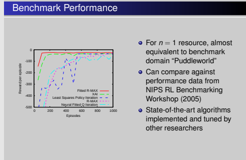
- Interpolation over a uniform grid

The experiments in the thesis use one concrete instantiation of this algorithm. The averager it uses to weight instances for a given state-action *sa* applies a Gaussian kernel over every instance *i* with a matching action $a_i = a$, assuming a distance function defined over the state space. The R-MAX exploration mechanism considers a state-action known if the combined weight across all instances exceeds some threshold. The averager used to approximate the value function interpolates over a uniform grid over the state space.

Benchmark Performance



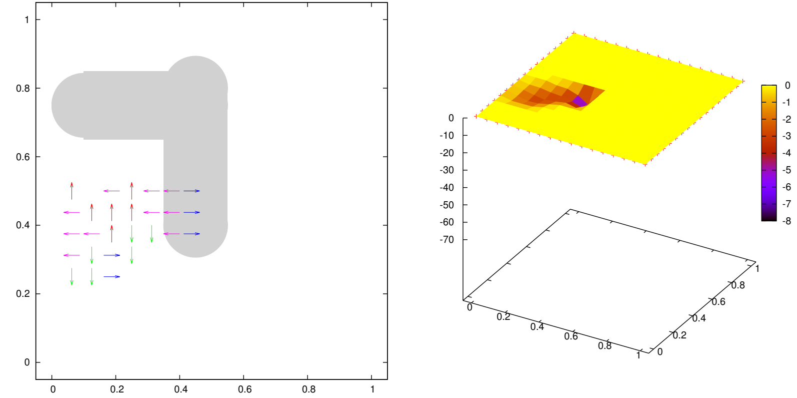
- For $n = 1$ resource, almost equivalent to benchmark domain "Puddleworld"
- Can compare against performance data from NIPS RL Benchmarking Workshop (2005)
- State-of-the-art algorithms implemented and tuned by other researchers

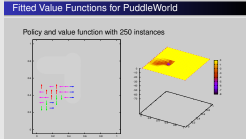


Fitted R-MAX was developed shortly after the NIPS RL Benchmarking Workshop, allowing a comparison with other contemporary RL algorithms implemented and optimized by other researchers. Michael Littman was kind enough to grant access to the raw data submitted to the workshop, which includes the average reward per episode after every 50 episodes in the PuddleWorld domain, which underlies the resource-gathering domain used in this thesis. Fitted R-MAX achieves near-optimal behavior within the first couple of data points, easily outperforming algorithms that only combine function approximation (generalization) or model-based exploration, not both.

Fitted Value Functions for PuddleWorld

Policy and value function with 250 instances

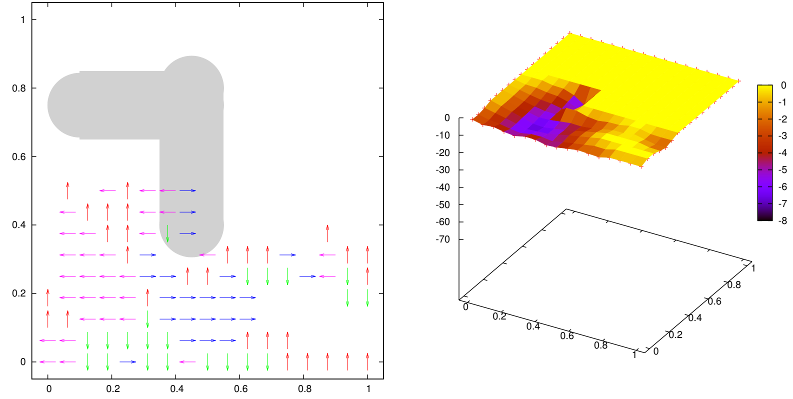


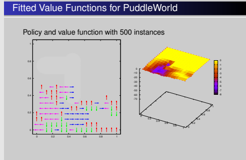


These figure demonstrate the evolution of the policy and value function over time. The left figure shows the policy action at each “known” state in the finite sample X. The first 1000 or so instances all come from the agent’s first episode, as it seeks to reach the unknown frontier, which has optimistic value 0. After finding the goal in the upper right corner, the agent tends to spend episodes exploiting instead of exploring, except when the random start state is near unexplored regions. Note that the agent is only willing to explore in the middle of the costly puddles if it begins an episode within the puddle already.

Fitted Value Functions for PuddleWorld

Policy and value function with 500 instances

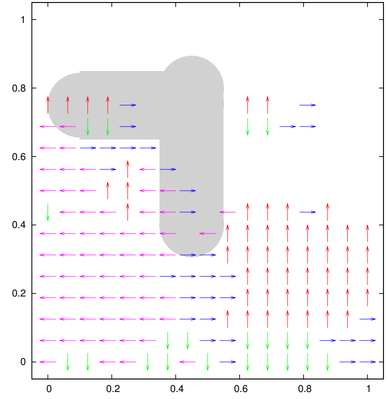


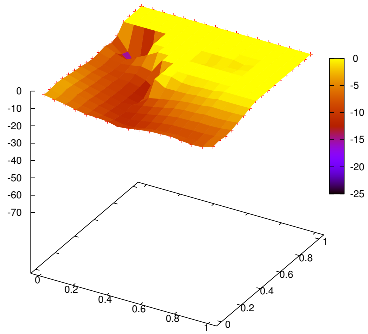


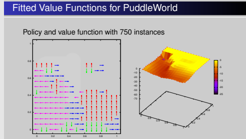
These figure demonstrate the evolution of the policy and value function over time. The left figure shows the policy action at each "known" state in the finite sample X. The first 1000 or so instances all come from the agent's first episode, as it seeks to reach the unknown frontier, which has optimistic value 0. After finding the goal in the upper right corner, the agent tends to spend episodes exploiting instead of exploring, except when the random start state is near unexplored regions. Note that the agent is only willing to explore in the middle of the costly puddles if it begins an episode within the puddle already.

Fitted Value Functions for PuddleWorld

Policy and value function with 750 instances



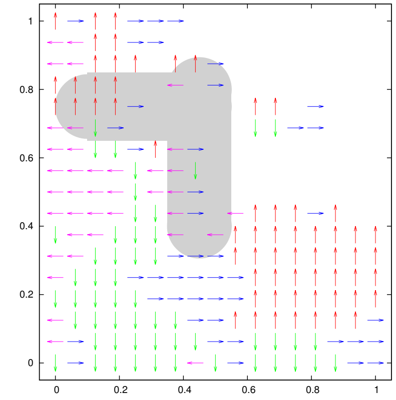


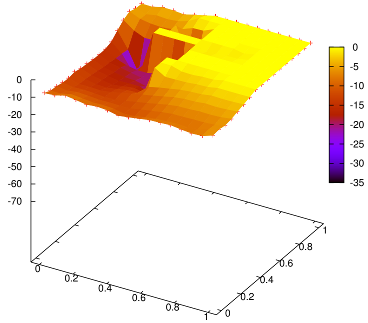


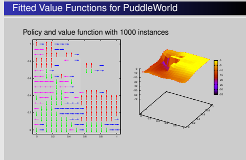
These figure demonstrate the evolution of the policy and value function over time. The left figure shows the policy action at each “known” state in the finite sample X. The first 1000 or so instances all come from the agent’s first episode, as it seeks to reach the unknown frontier, which has optimistic value 0. After finding the goal in the upper right corner, the agent tends to spend episodes exploiting instead of exploring, except when the random start state is near unexplored regions. Note that the agent is only willing to explore in the middle of the costly puddles if it begins an episode within the puddle already.

Fitted Value Functions for PuddleWorld

Policy and value function with 1000 instances



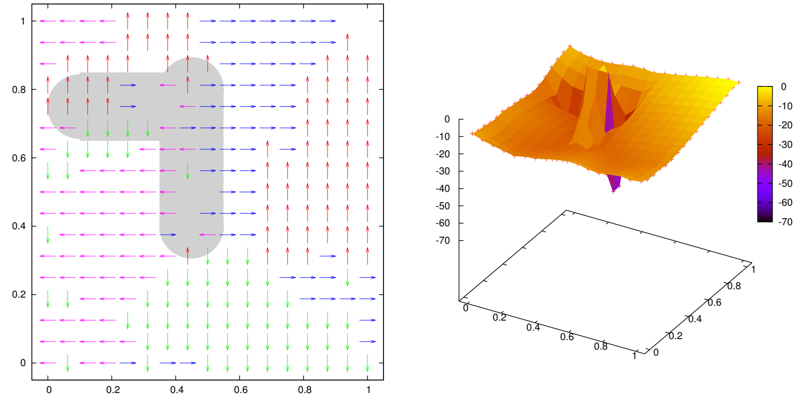


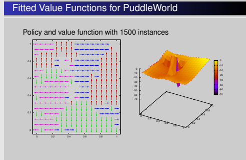


These figure demonstrate the evolution of the policy and value function over time. The left figure shows the policy action at each “known” state in the finite sample X. The first 1000 or so instances all come from the agent’s first episode, as it seeks to reach the unknown frontier, which has optimistic value 0. After finding the goal in the upper right corner, the agent tends to spend episodes exploiting instead of exploring, except when the random start state is near unexplored regions. Note that the agent is only willing to explore in the middle of the costly puddles if it begins an episode within the puddle already.

Fitted Value Functions for PuddleWorld

Policy and value function with 1500 instances

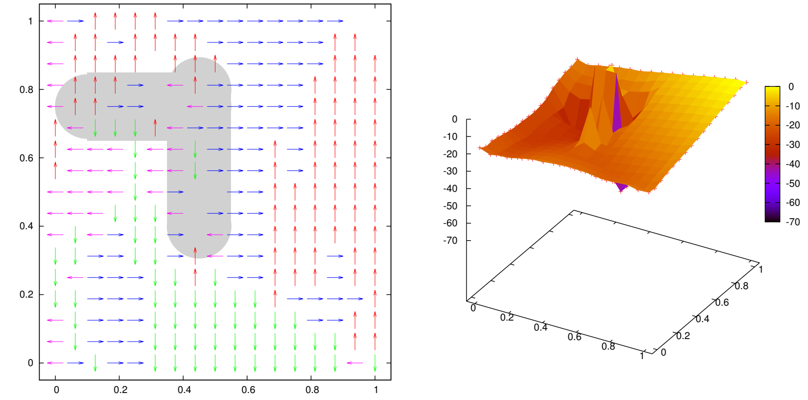


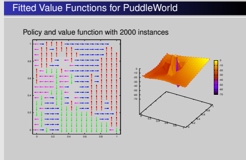


These figure demonstrate the evolution of the policy and value function over time. The left figure shows the policy action at each “known” state in the finite sample X. The first 1000 or so instances all come from the agent’s first episode, as it seeks to reach the unknown frontier, which has optimistic value 0. After finding the goal in the upper right corner, the agent tends to spend episodes exploiting instead of exploring, except when the random start state is near unexplored regions. Note that the agent is only willing to explore in the middle of the costly puddles if it begins an episode within the puddle already.

Fitted Value Functions for PuddleWorld

Policy and value function with 2000 instances

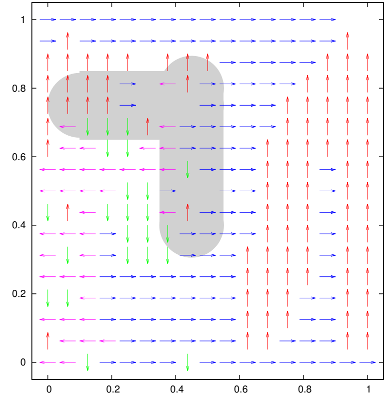


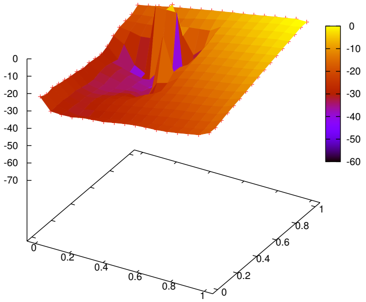


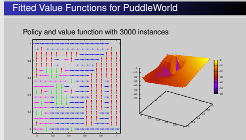
These figure demonstrate the evolution of the policy and value function over time. The left figure shows the policy action at each “known” state in the finite sample X. The first 1000 or so instances all come from the agent’s first episode, as it seeks to reach the unknown frontier, which has optimistic value 0. After finding the goal in the upper right corner, the agent tends to spend episodes exploiting instead of exploring, except when the random start state is near unexplored regions. Note that the agent is only willing to explore in the middle of the costly puddles if it begins an episode within the puddle already.

Fitted Value Functions for PuddleWorld

Policy and value function with 3000 instances



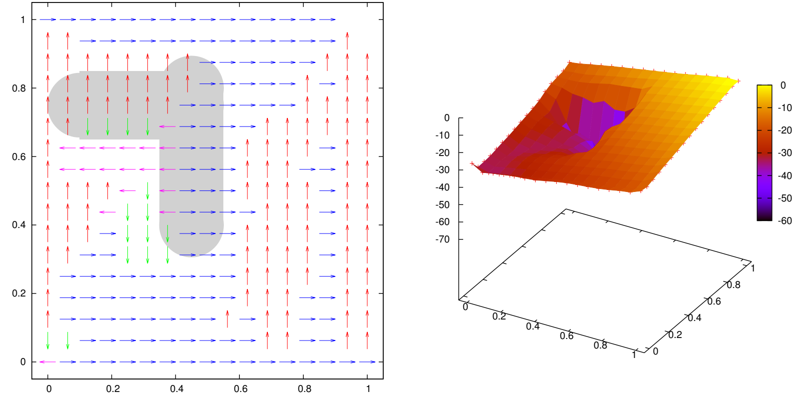


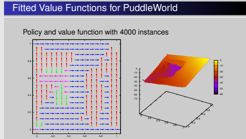


These figure demonstrate the evolution of the policy and value function over time. The left figure shows the policy action at each “known” state in the finite sample X. The first 1000 or so instances all come from the agent’s first episode, as it seeks to reach the unknown frontier, which has optimistic value 0. After finding the goal in the upper right corner, the agent tends to spend episodes exploiting instead of exploring, except when the random start state is near unexplored regions. Note that the agent is only willing to explore in the middle of the costly puddles if it begins an episode within the puddle already.

Fitted Value Functions for PuddleWorld

Policy and value function with 4000 instances

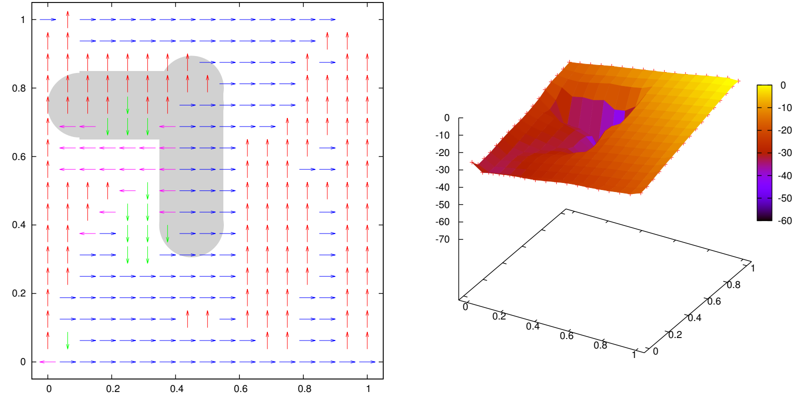


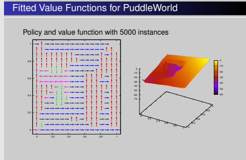


These figure demonstrate the evolution of the policy and value function over time. The left figure shows the policy action at each “known” state in the finite sample X. The first 1000 or so instances all come from the agent’s first episode, as it seeks to reach the unknown frontier, which has optimistic value 0. After finding the goal in the upper right corner, the agent tends to spend episodes exploiting instead of exploring, except when the random start state is near unexplored regions. Note that the agent is only willing to explore in the middle of the costly puddles if it begins an episode within the puddle already.

Fitted Value Functions for PuddleWorld

Policy and value function with 5000 instances





These figure demonstrate the evolution of the policy and value function over time. The left figure shows the policy action at each “known” state in the finite sample X. The first 1000 or so instances all come from the agent’s first episode, as it seeks to reach the unknown frontier, which has optimistic value 0. After finding the goal in the upper right corner, the agent tends to spend episodes exploiting instead of exploring, except when the random start state is near unexplored regions. Note that the agent is only willing to explore in the middle of the costly puddles if it begins an episode within the puddle already.

Generalization and Exploration

Inductive Bias

Model-Free Similar states have similar values

Model-Based Similar states have similar dynamics

Model Generalization

- Is the effect of *sa* known or unknown?
- Less generalization leads to more exploration

Value Generalization

- How good is my policy π?
- Less generalization leads to more computation

Generalization and Exploration
Inductive Bias
Model-Free Similar states have similar values
Model-Based Similar states have similar dynamics
Model Generalization
Is the effect of sa known or unknown?
Less generalization leads to more exploration
Value Generalization
How good is my policy π?
Less generalization leads to more computation

Fitted R-MAX can learn efficiently in part because it separates generalization of state values from generalization of state dynamics, in contrast to model-free methods which tend to conflate the two. Experiments with Fitted R-MAX show that model generalization more directly controls the amount of exploration the agent performs: smaller neighborhoods of generalization entail more neighborhoods to explore. In contrast, decreasing the amount of generalization in the value function increases the accuracy of policy evaluation, which mostly affects the equality of the policy obtained after exploration. After a certain point, finer value approximations only increase the computational cost of planning without affecting the actual rewards earned by the agent.

Generalization and Exploration

Inductive Bias

Model-Free Similar states have similar values

Model-Based Similar states have similar dynamics

Model Generalization

- Is the effect of *sa* known or unknown?
- Less generalization leads to more exploration

Value Generalization

- How good is my policy π?
- Less generalization leads to more computation

Fitted R-MAX can learn efficiently in part because it separates generalization of state values from generalization of state dynamics, in contrast to model-free methods which tend to conflate the two. Experiments with Fitted R-MAX show that model generalization more directly controls the amount of exploration the agent performs: smaller neighborhoods of generalization entail more neighborhoods to explore. In contrast, decreasing the amount of generalization in the value function increases the accuracy of policy evaluation, which mostly affects the equality of the policy obtained after exploration. After a certain point, finer value approximations only increase the computational cost of planning without affecting the actual rewards earned by the agent.

Generalization and Exploration

Inductive Bias

Model-Free Similar states have similar values

Model-Based Similar states have similar dynamics

Model Generalization

- Is the effect of *sa* known or unknown?
- Less generalization leads to more exploration

Value Generalization

- How good is my policy π?
- Less generalization leads to more computation

Generalization and Exploration
Inductive Bias
Model-Free Similar states have similar values
Model-Based Similar states have similar dynamics
Model Generalization
Is the effect of sa known or unknown?
Less generalization leads to more exploration
Value Generalization
How good is my policy π?
Less generalization leads to more computation

Fitted R-MAX can learn efficiently in part because it separates generalization of state values from generalization of state dynamics, in contrast to model-free methods which tend to conflate the two. Experiments with Fitted R-MAX show that model generalization more directly controls the amount of exploration the agent performs: smaller neighborhoods of generalization entail more neighborhoods to explore. In contrast, decreasing the amount of generalization in the value function increases the accuracy of policy evaluation, which mostly affects the equality of the policy obtained after exploration. After a certain point, finer value approximations only increase the computational cost of planning without affecting the actual rewards earned by the agent.

Outline

Outline
Introduction
Exploration and Approximation
Exploration and Hierarchy
Hierarchical Decomposition
The R-MAXQ and Fitted R-MAXQ Algorithms
The Utility of Hierarchy
Conclusion

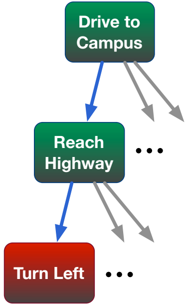
The Appeal of Hierarchy

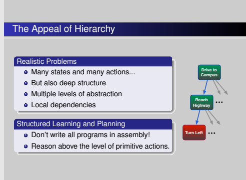
Realistic Problems

- Many states and many actions...
- But also deep structure
- Multiple levels of abstraction
- Local dependencies

Structured Learning and Planning

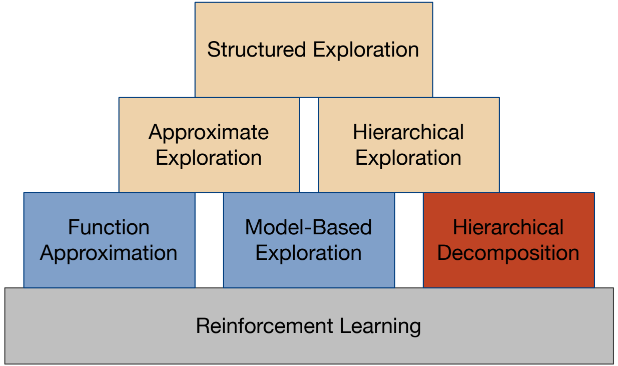
- Don't write all programs in assembly!
- Reason above the level of primitive actions.

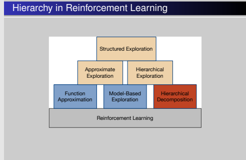




Fitted R-MAX extends model-based exploration to continuous state spaces, but it wants to explore every neighborhood of the state space. In practical applications, agents can't afford to be so exhaustive. These applications often have inherent hierarchical structure from which the agent should be able to benefit. The intuitive appeal of hierarchy is that humans don't only learn or plan at the lowest possible level. The skills I employ and the factors I consider depend on whether I'm driving through an intersection (at the low level) or plotting a route from home to campus (at the high level).

Hierarchy in Reinforcement Learning





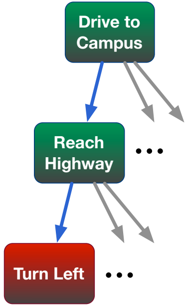
Hierarchy in Reinforcement Learning

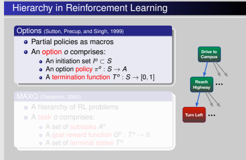
Options (Sutton, Precup, and Singh, 1999)

- Partial policies as macros
- An option o comprises:
  - An initiation set $I^o \subset S$
  - An option policy $\pi^o : S \to A$
  - A termination function $T^o : S \to [0, 1]$

MAXQ (Dietterich, 2000)

- A hierarchy of RL problems
- A task o comprises:
  - A set of subtasks A^o
  - A goal reward function $G^o : T^o \to \mathbb{R}$
  - A set of terminal states T^o





Hierarchical RL has become a popular branch of research, but a variety of formalisms underscore the lack of consensus in how precisely hierarchy can benefit RL algorithms. The two most popular formalisms, options and MAXQ, both define abstract actions as a sequence of lower-level actions executed until reaching a specific termination condition or subgoal. Options assume that a given policy specifies the lower-level actions taken, but MAXQ frames each abstract action as a recursive instance of an RL problem. This thesis adopts the MAXQ approach.

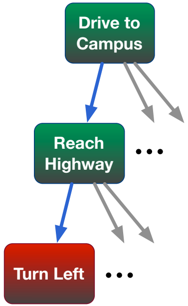
Hierarchy in Reinforcement Learning

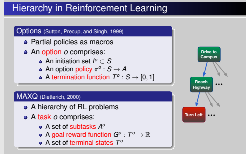
Options (Sutton, Precup, and Singh, 1999)

- Partial policies as macros
- An option o comprises:
  - An initiation set $I^o \subset S$
  - An option policy $\pi^o : S \to A$
  - A termination function $T^o : S \to [0, 1]$

MAXQ (Dietterich, 2000)

- A hierarchy of RL problems
- A task o comprises:
  - A set of subtasks A^o
  - A goal reward function $G^o : T^o \to \mathbb{R}$
  - A set of terminal states T^o



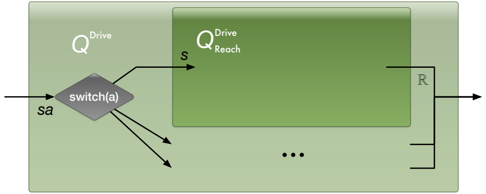


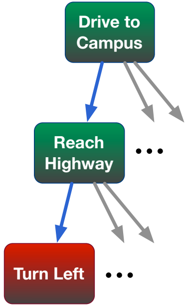
Hierarchical RL has become a popular branch of research, but a variety of formalisms underscore the lack of consensus in how precisely hierarchy can benefit RL algorithms. The two most popular formalisms, options and MAXQ, both define abstract actions as a sequence of lower-level actions executed until reaching a specific termination condition or subgoal. Options assume that a given policy specifies the lower-level actions taken, but MAXQ frames each abstract action as a recursive instance of an RL problem. This thesis adopts the MAXQ approach.

MAXQ Value Function Decomposition

High-Level Rewards are Low-Level Values

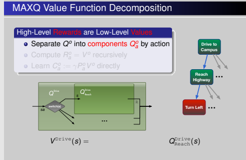
- Separate Q^o into components Q^o_a by action
- Compute $R^o_a = V^o$ recursively
- Learn $C^o_a := \gamma P^o_a V^o$ directly





$V^{\texttt{Drive}}(s) =$

$Q^{\texttt{Drive}}_{\texttt{Reach}}(s)$

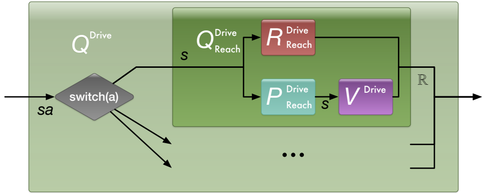


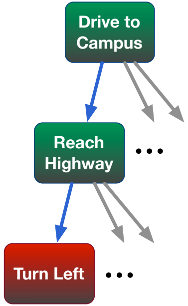
A key insight of MAXQ is that solutions to lower-level instances of RL in a task hierarchy explicitly define part of the higher-level instances. It separates each task's state-action value function $Q^o : S \times A \to \mathbb{R}$ into action-specific pieces $Q^o_a : S \to \mathbb{R}$, each of which directly involve only that portion of the reward and transition functions specific to one action. But the action-specific reward function R^o_a is exactly equal to the value function V^a for the subtask a! MAXQ can therefore treat part of the value function Q^o_a as known, but it must still apply model-free learning to estimate the remainder, which it defines as the "completion" function $C^o_a = P^o_a V^o$. MAXQ learns a value function of sorts for each primitive task in the hierarchy and for each link between tasks. The value of a state for a given hierarchical policy is the sum of the components along the path from root to leaf.

MAXQ Value Function Decomposition

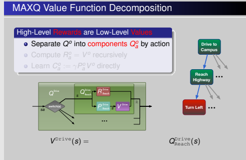
High-Level Rewards are Low-Level Values

- Separate Q^o into components Q^o_a by action
- Compute $R^o_a = V^o$ recursively
- Learn $C^o_a := \gamma P^o_a V^o$ directly





$V^{\texttt{Drive}}(s) =$ $\qquad$ $Q^{\texttt{Drive}}_{\texttt{Reach}}(s)$

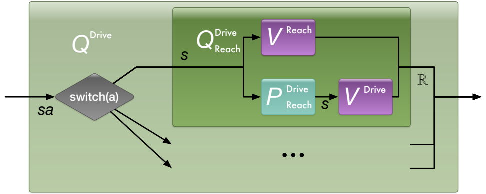


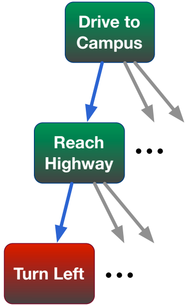
A key insight of MAXQ is that solutions to lower-level instances of RL in a task hierarchy explicitly define part of the higher-level instances. It separates each task's state-action value function $Q^o : S \times A \to \mathbb{R}$ into action-specific pieces $Q^o_a : S \to \mathbb{R}$, each of which directly involve only that portion of the reward and transition functions specific to one action. But the action-specific reward function R^o_a is exactly equal to the value function V^a for the subtask a! MAXQ can therefore treat part of the value function Q^o_a as known, but it must still apply model-free learning to estimate the remainder, which it defines as the "completion" function $C^o_a = P^o_a V^o$. MAXQ learns a value function of sorts for each primitive task in the hierarchy and for each link between tasks. The value of a state for a given hierarchical policy is the sum of the components along the path from root to leaf.

MAXQ Value Function Decomposition

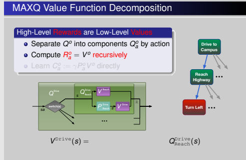
High-Level Rewards are Low-Level Values

- Separate Q^o into components Q^o_a by action
- Compute $R^o_a = V^o$ recursively
- Learn $C^o_a := \gamma P^o_a V^o$ directly





$V^{\texttt{Drive}}(s) =$ $\qquad$ $Q^{\texttt{Drive}}_{\texttt{Reach}}(s)$

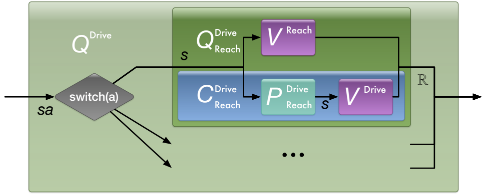


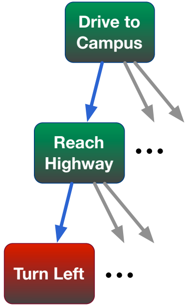
A key insight of MAXQ is that solutions to lower-level instances of RL in a task hierarchy explicitly define part of the higher-level instances. It separates each task's state-action value function $Q^o : S \times A \to \mathbb{R}$ into action-specific pieces $Q^o_a : S \to \mathbb{R}$, each of which directly involve only that portion of the reward and transition functions specific to one action. But the action-specific reward function R^o_a is exactly equal to the value function V^a for the subtask a! MAXQ can therefore treat part of the value function Q^o_a as known, but it must still apply model-free learning to estimate the remainder, which it defines as the "completion" function $C^o_a = P^o_a V^o$. MAXQ learns a value function of sorts for each primitive task in the hierarchy and for each link between tasks. The value of a state for a given hierarchical policy is the sum of the components along the path from root to leaf.

MAXQ Value Function Decomposition

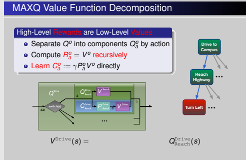
High-Level Rewards are Low-Level Values

- Separate Q^o into components Q^o_a by action
- Compute $R^o_a = V^o$ recursively
- Learn $C^o_a := \gamma P^o_a V^o$ directly





$V^{\texttt{Drive}}(s) =$ $Q^{\texttt{Drive}}_{\texttt{Reach}}(s)$

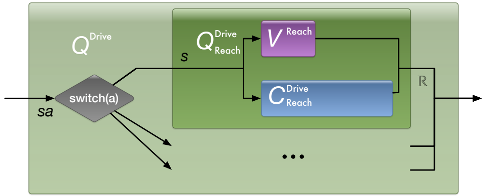


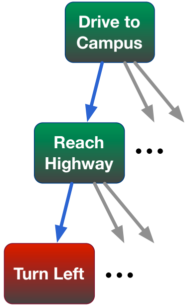
A key insight of MAXQ is that solutions to lower-level instances of RL in a task hierarchy explicitly define part of the higher-level instances. It separates each task's state-action value function $Q^o : S \times A \to \mathbb{R}$ into action-specific pieces $Q^o_a : S \to \mathbb{R}$, each of which directly involve only that portion of the reward and transition functions specific to one action. But the action-specific reward function R^o_a is exactly equal to the value function V^a for the subtask a! MAXQ can therefore treat part of the value function Q^o_a as known, but it must still apply model-free learning to estimate the remainder, which it defines as the "completion" function $C^o_a = P^o_a V^o$. MAXQ learns a value function of sorts for each primitive task in the hierarchy and for each link between tasks. The value of a state for a given hierarchical policy is the sum of the components along the path from root to leaf.

MAXQ Value Function Decomposition

High-Level Rewards are Low-Level Values

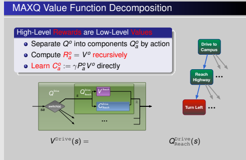
- Separate Q^o into components Q^o_a by action
- Compute $R^o_a = V^o$ recursively
- Learn $C^o_a := \gamma P^o_a V^o$ directly





$V^{\texttt{Drive}}(s) =$

$Q^{\texttt{Drive}}_{\texttt{Reach}}(s)$

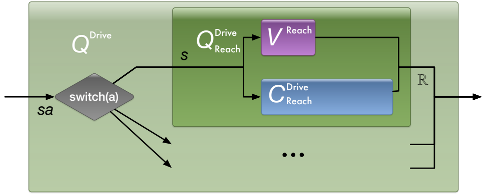


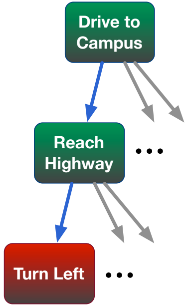
A key insight of MAXQ is that solutions to lower-level instances of RL in a task hierarchy explicitly define part of the higher-level instances. It separates each task's state-action value function $Q^o : S \times A \to \mathbb{R}$ into action-specific pieces $Q^o_a : S \to \mathbb{R}$, each of which directly involve only that portion of the reward and transition functions specific to one action. But the action-specific reward function R^o_a is exactly equal to the value function V^a for the subtask a! MAXQ can therefore treat part of the value function Q^o_a as known, but it must still apply model-free learning to estimate the remainder, which it defines as the "completion" function $C^o_a = P^o_a V^o$. MAXQ learns a value function of sorts for each primitive task in the hierarchy and for each link between tasks. The value of a state for a given hierarchical policy is the sum of the components along the path from root to leaf.

MAXQ Value Function Decomposition

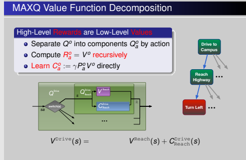
High-Level Rewards are Low-Level Values

- Separate Q^o into components Q^o_a by action
- Compute $R^o_a = V^o$ recursively
- Learn $C^o_a := \gamma P^o_a V^o$ directly





$$V^{\texttt{Drive}}(s) = \quad V^{\texttt{Reach}}(s) + C^{\texttt{Drive}}_{\texttt{Reach}}(s)$$

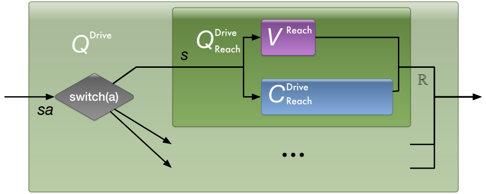


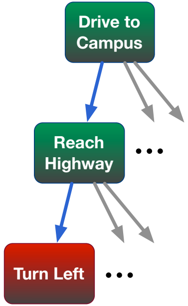
A key insight of MAXQ is that solutions to lower-level instances of RL in a task hierarchy explicitly define part of the higher-level instances. It separates each task's state-action value function $Q^o : S \times A \to \mathbb{R}$ into action-specific pieces $Q^o_a : S \to \mathbb{R}$, each of which directly involve only that portion of the reward and transition functions specific to one action. But the action-specific reward function R^o_a is exactly equal to the value function V^a for the subtask a! MAXQ can therefore treat part of the value function Q^o_a as known, but it must still apply model-free learning to estimate the remainder, which it defines as the "completion" function $C^o_a = P^o_a V^o$. MAXQ learns a value function of sorts for each primitive task in the hierarchy and for each link between tasks. The value of a state for a given hierarchical policy is the sum of the components along the path from root to leaf.

MAXQ Value Function Decomposition

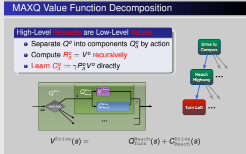
High-Level Rewards are Low-Level Values

- Separate Q^o into components Q^o_a by action
- Compute $R^o_a = V^o$ recursively
- Learn $C^o_a := \gamma P^o_a V^o$ directly





$$V^{\texttt{Drive}}(s) = \qquad Q^{\texttt{Reach}}_{\texttt{Turn}}(s) + C^{\texttt{Drive}}_{\texttt{Reach}}(s)$$

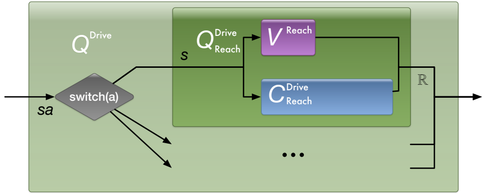


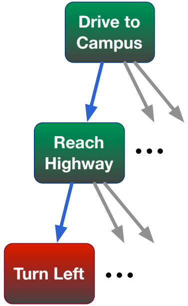
A key insight of MAXQ is that solutions to lower-level instances of RL in a task hierarchy explicitly define part of the higher-level instances. It separates each task's state-action value function $Q^o : S \times A \to \mathbb{R}$ into action-specific pieces $Q^o_a : S \to \mathbb{R}$, each of which directly involve only that portion of the reward and transition functions specific to one action. But the action-specific reward function R^o_a is exactly equal to the value function V^a for the subtask a! MAXQ can therefore treat part of the value function Q^o_a as known, but it must still apply model-free learning to estimate the remainder, which it defines as the "completion" function $C^o_a = P^o_a V^o$. MAXQ learns a value function of sorts for each primitive task in the hierarchy and for each link between tasks. The value of a state for a given hierarchical policy is the sum of the components along the path from root to leaf.

MAXQ Value Function Decomposition

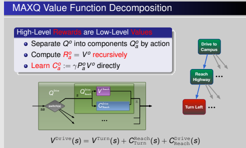
High-Level Rewards are Low-Level Values

- Separate Q^o into components Q^o_a by action
- Compute $R^o_a = V^o$ recursively
- Learn $C^o_a := \gamma P^o_a V^o$ directly



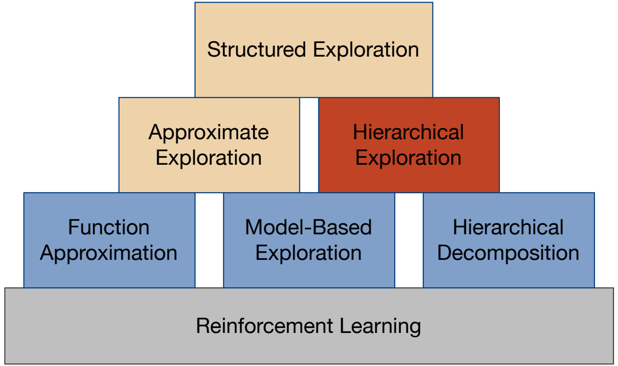


$$V^{\mathtt{Drive}}(s) = V^{\mathtt{Turn}}(s) + C^{\mathtt{Reach}}_{\mathtt{Turn}}(s) + C^{\mathtt{Drive}}_{\mathtt{Reach}}(s)$$

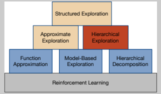


A key insight of MAXQ is that solutions to lower-level instances of RL in a task hierarchy explicitly define part of the higher-level instances. It separates each task's state-action value function $Q^o : S \times A \to \mathbb{R}$ into action-specific pieces $Q^o_a : S \to \mathbb{R}$, each of which directly involve only that portion of the reward and transition functions specific to one action. But the action-specific reward function R^o_a is exactly equal to the value function V^a for the subtask a! MAXQ can therefore treat part of the value function Q^o_a as known, but it must still apply model-free learning to estimate the remainder, which it defines as the "completion" function $C^o_a = P^o_a V^o$. MAXQ learns a value function of sorts for each primitive task in the hierarchy and for each link between tasks. The value of a state for a given hierarchical policy is the sum of the components along the path from root to leaf.

Hierarchical Model Decomposition



Hierarchical Model Decomposition

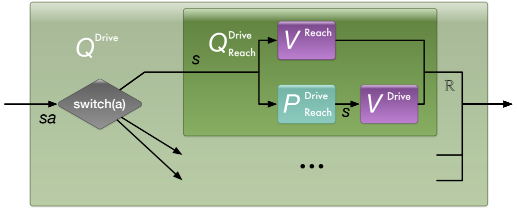


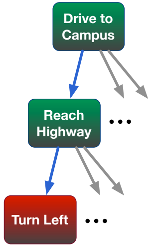
Hierarchical Model Decomposition

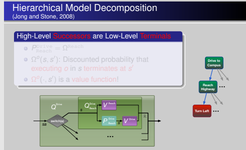
(Jong and Stone, 2008)

High-Level Successors are Low-Level Terminals

- $P^{\text{Drive}}_{\text{Reach}} = \Omega^{\text{Reach}}$
- $\Omega^o(s, s')$: Discounted probability that executing o in s terminates at s'
- $\Omega^o(\cdot, s')$ is a value function!





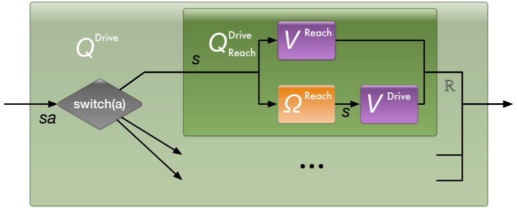


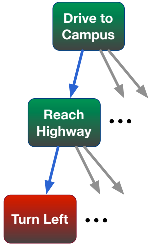
A key contribution of this thesis is to take the MAXQ observation one step further. Instead of obtaining the abstract reward function recursively, we can also define the abstract transitions recursively. The thesis defines $\Omega : S \times S \to [0, 1]$ as the terminal state distribution for a policy in a task: given a start state, the expected terminal states. This distribution can be computed in exactly the same way as a value function: consider defining for each terminal state the value function for the task in which the agent receives reward 1 for reaching that state and 0 for reaching any other terminal state.

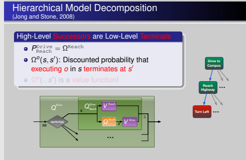
Hierarchical Model Decomposition
(Jong and Stone, 2008)

High-Level Successors are Low-Level Terminals

- $P^{\mathtt{Drive}}_{\mathtt{Reach}} = \Omega^{\mathtt{Reach}}$
- $\Omega^o(s, s')$: Discounted probability that executing o in s terminates at s'
- $\Omega^o(\cdot, s')$ is a value function!





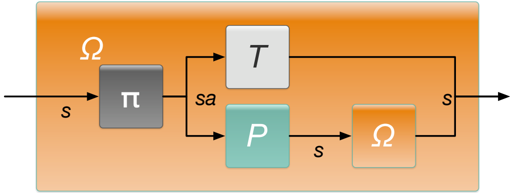


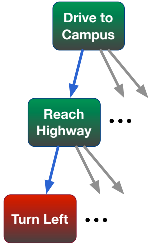
A key contribution of this thesis is to take the MAXQ observation one step further. Instead of obtaining the abstract reward function recursively, we can also define the abstract transitions recursively. The thesis defines $\Omega : S \times S \rightarrow [0, 1]$ as the terminal state distribution for a policy in a task: given a start state, the expected terminal states. This distribution can be computed in exactly the same way as a value function: consider defining for each terminal state the value function for the task in which the agent receives reward 1 for reaching that state and 0 for reaching any other terminal state.

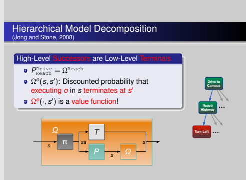
Hierarchical Model Decomposition
(Jong and Stone, 2008)

High-Level Successors are Low-Level Terminals

- $P^{\mathtt{Drive}}_{\mathtt{Reach}} = \Omega^{\mathtt{Reach}}$
- $\Omega^o(s, s')$: Discounted probability that executing o in s terminates at s'
- $\Omega^o(\cdot, s')$ is a value function!

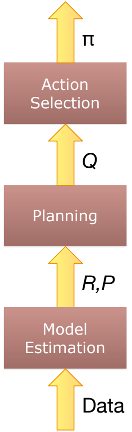






A key contribution of this thesis is to take the MAXQ observation one step further. Instead of obtaining the abstract reward function recursively, we can also define the abstract transitions recursively. The thesis defines $\Omega : S \times S \to [0, 1]$ as the terminal state distribution for a policy in a task: given a start state, the expected terminal states. This distribution can be computed in exactly the same way as a value function: consider defining for each terminal state the value function for the task in which the agent receives reward 1 for reaching that state and 0 for reaching any other terminal state.

The R-MAXQ Algorithm
(Jong and Stone, 2008)

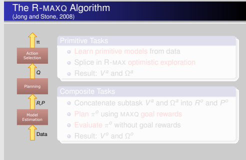


Primitive Tasks

- Learn primitive models from data
- Splice in R-MAX optimistic exploration
- Result: V^a and Ω^a

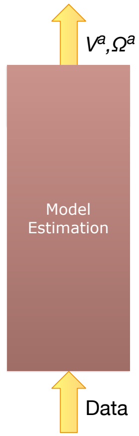
Composite Tasks

- Concatenate subtask V^a and Ω^a into R^o and P^o
- Plan π^o using MAXQ goal rewards
- Evaluate π^o without goal rewards
- Result: V^o and Ω^o



The hierarchical model decomposition on the last slide underlies the R-MAXQ algorithm, which replaces the model estimation of R-MAX with a bottom-up modeling process, given a task hierarchy. It learns primitive action models in the same way as R-MAX, using maximum-likeihood estimation spliced with optimism. For higher-level tasks, it assembles the reward and transition functions for that task using the lower-level action models. It computes the task policy by planning that incorporates the task goal function, as described in more detail in the thesis. Finally, given the policy and task reward and transition functions, policy evaluation computes the value function and terminal states that model the task for even higher-level tasks.

The R-MAXQ Algorithm
(Jong and Stone, 2008)

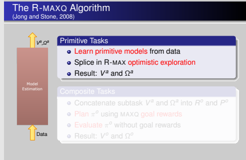


Primitive Tasks

- Learn primitive models from data
- Splice in R-MAX optimistic exploration
- Result: V^a and Ω^a

Composite Tasks

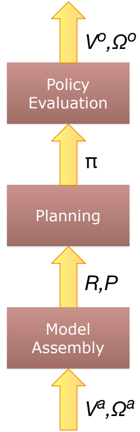
- Concatenate subtask V^a and Ω^a into R^o and P^o
- Plan π^o using MAXQ goal rewards
- Evaluate π^o without goal rewards
- Result: V^o and Ω^o



The hierarchical model decomposition on the last slide underlies the R-MAXQ algorithm, which replaces the model estimation of R-MAX with a bottom-up modeling process, given a task hierarchy. It learns primitive action models in the same way as R-MAX, using maximum-likeihood estimation spliced with optimism. For higher-level tasks, it assembles the reward and transition functions for that task using the lower-level action models. It computes the task policy by planning that incorporates the task goal function, as described in more detail in the thesis. Finally, given the policy and task reward and transition functions, policy evaluation computes the value function and terminal states that model the task for even higher-level tasks.

The R-MAXQ Algorithm

(Jong and Stone, 2008)

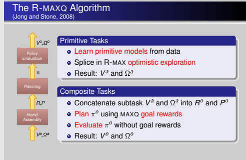


Primitive Tasks

- Learn primitive models from data
- Splice in R-MAX optimistic exploration
- Result: V^a and Ω^a

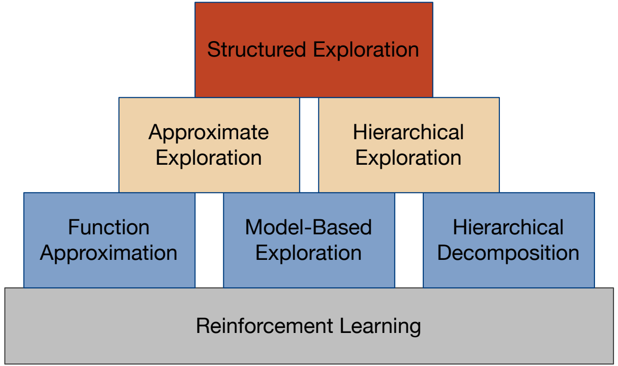
Composite Tasks

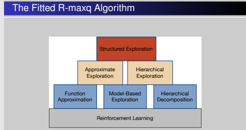
- Concatenate subtask V^a and Ω^a into R^o and P^o
- Plan π^o using MAXQ goal rewards
- Evaluate π^o without goal rewards
- Result: V^o and Ω^o



The hierarchical model decomposition on the last slide underlies the R-MAXQ algorithm, which replaces the model estimation of R-MAX with a bottom-up modeling process, given a task hierarchy. It learns primitive action models in the same way as R-MAX, using maximum-likeihood estimation spliced with optimism. For higher-level tasks, it assembles the reward and transition functions for that task using the lower-level action models. It computes the task policy by planning that incorporates the task goal function, as described in more detail in the thesis. Finally, given the policy and task reward and transition functions, policy evaluation computes the value function and terminal states that model the task for even higher-level tasks.

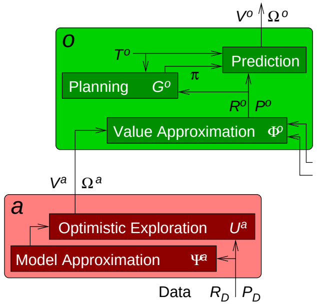
The Fitted R-maxq Algorithm





The Fitted R-maxq Algorithm

(Jong and Stone, 2009)



Prediction

Solve $V^o = \pi^o(R^o + \gamma P^o(I - T^o)V^o)$

Solve $\Omega^o = \pi^o(P^o T^o + \gamma P^o(I - T^o)\Omega^o)$

Planning

Optimize $\tilde{V}^o = T^o G^o +$
$(I - T^o)\pi^o(R^o + \gamma P^o \tilde{V}^o)$

Primitive Action Models

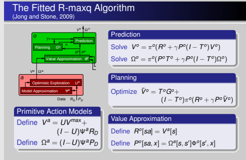
Define $V^a = UV^{\max} +$
$(I - U)\Psi^a R_D$

Define $\Omega^a = (I - U)\Psi^a P_D$

Value Approximation

Define $R^o[sa] = V^a[s]$

Define $P^o[sa, x] = \Omega^a[s, s']\Phi^o[s', x]$



Fitted R-MAX and R-MAXQ both extend R-MAX by modifying the Bellman equation, and these modifications may be composed to obtain Fitted R-MAXQ, an algorithm that extends model-based exploration to both continuous state spaces and hierarchical decomposition. This slide summaries the resulting model-estimation process, using the matrix notation developed in the thesis. Note that all the explicit learning and exploration is confined to the models of the primitive actions, in red, while the upper levels of the hierarchy only perform planning and policy evaluation. The two kinds of tasks interact by propagating low-level optimism up the hierarchy, encouraging exploration, constrained by the goal-reward functions and subtask sets at each task.

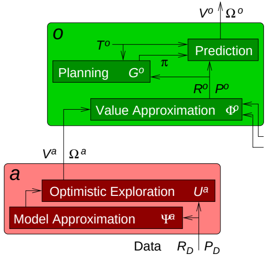
The Software Architecture

Algorithm

1. Execute π^{Root} hierarchically
2. Update data: R_D and P_D
3. Propagate changes to π^{Root}
4. Repeat

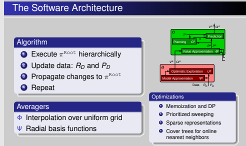
Averagers

- Φ Interpolation over uniform grid
- Ψ Radial basis functions



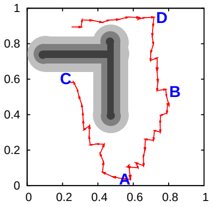
Optimizations

- Memoization and DP
- Prioritized sweeping
- Sparse representations
- Cover trees for online nearest neighbors



In concept, Fitted R-MAXQ simply estimates the model of every task at each time step, while executing the resulting hierarchical policy. In practice, recomputing all the models from scratch would be prohibitively expensive, especially given the cost of computing the averager weights. This thesis contributes an implementation of Fitted R-MAXQ that includes several optimizations, designed to cache as much information as possible between time steps and propagate the changes due to each new instance as efficiently as possible. The code is available at `http://library.rl-community.org/wiki/Fitted_R-MAXQ`.

Example: A Resource Gathering Simulation



Simulated Robot's Task

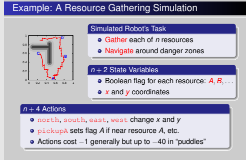
- Gather each of n resources
- Navigate around danger zones

$n + 2$ State Variables

- Boolean flag for each resource: $A, B, \ldots$
- x and y coordinates

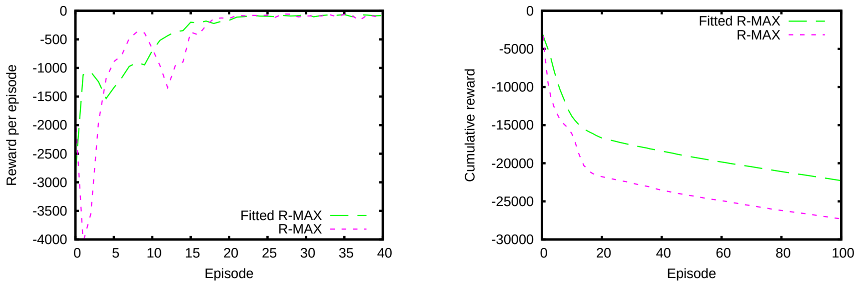
$n + 4$ Actions

- `north`, `south`, `east`, `west` change x and y
- `pickupA` sets flag A if near resource A, etc.
- Actions cost -1 generally but up to -40 in "puddles"



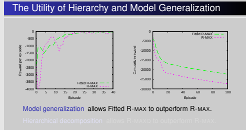
The next few slides describe experimental results in the resource-gathering domain, recapitulated here.

The Utility of Hierarchy and Model Generalization



Model generalization allows Fitted R-MAX to outperform R-MAX.

Hierarchical decomposition allows R-MAXQ to outperform R-MAX.

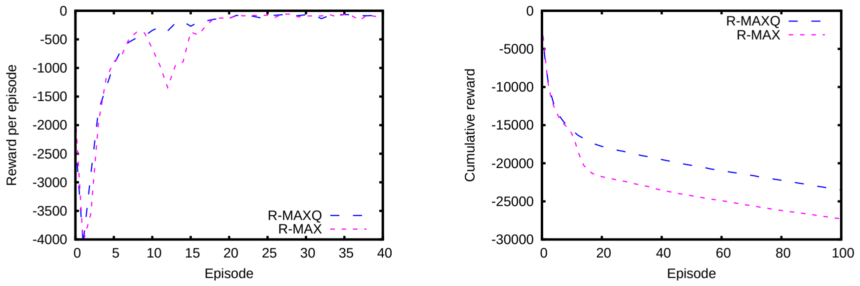


These figures show the learning performance of agents in the resource-gathering domain, with four resources. The left-hand figure shows the quality of the agent's learned policy at each episode, measured as the reward earned in each of the first 40 episodes. The right-hand figure shows the cumulative cost of learning these policies.

Fitted R-MAX outperforms R-MAX by introducing generalization in the model reducing the amount of data the agent attempts to collect in any given neighborhood of the state space. R-MAXQ outperforms R-MAX by introducing hierarchical constraints to the exploration policy.

Fitted R-MAXQ combines both benefits. It still converges to an optimal policy, and its reduction in the cost of learning is greater than the sum of the reductions for either of its component algorithms!

The Utility of Hierarchy and Model Generalization



Model generalization allows Fitted R-MAX to outperform R-MAX.

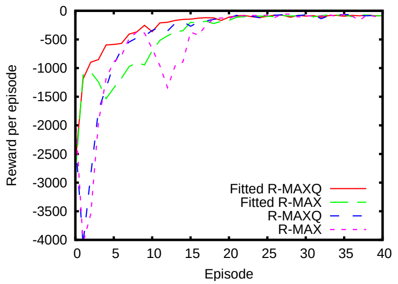
Hierarchical decomposition allows R-MAXQ to outperform R-MAX.

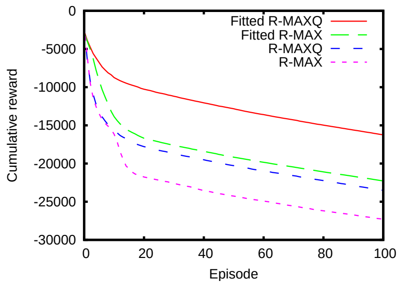
These figures show the learning performance of agents in the resource-gathering domain, with four resources. The left-hand figure shows the quality of the agent's learned policy at each episode, measured as the reward earned in each of the first 40 episodes. The right-hand figure shows the cumulative cost of learning these policies.

Fitted R-MAX outperforms R-MAX by introducing generalization in the model reducing the amount of data the agent attempts to collect in any given neighborhood of the state space. R-MAXQ outperforms R-MAX by introducing hierarchical constraints to the exploration policy.

Fitted R-MAXQ combines both benefits. It still converges to an optimal policy, and its reduction in the cost of learning is greater than the sum of the reductions for either of its component algorithms!

The Utility of Hierarchy and Model Generalization

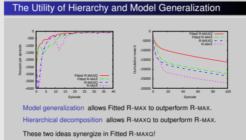




Model generalization allows Fitted R-MAX to outperform R-MAX.

Hierarchical decomposition allows R-MAXQ to outperform R-MAX.

These two ideas synergize in Fitted R-MAXQ!

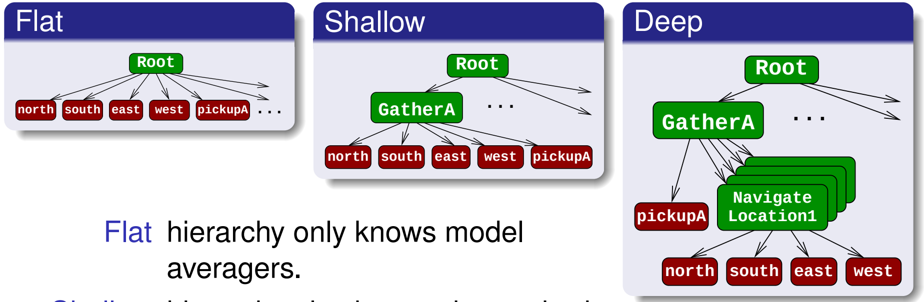


These figures show the learning performance of agents in the resource-gathering domain, with four resources. The left-hand figure shows the quality of the agent's learned policy at each episode, measured as the reward earned in each of the first 40 episodes. The right-hand figure shows the cumulative cost of learning these policies.

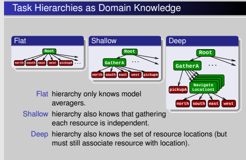
Fitted R-MAX outperforms R-MAX by introducing generalization in the model reducing the amount of data the agent attempts to collect in any given neighborhood of the state space. R-MAXQ outperforms R-MAX by introducing hierarchical constraints to the exploration policy.

Fitted R-MAXQ combines both benefits. It still converges to an optimal policy, and its reduction in the cost of learning is greater than the sum of the reductions for either of its component algorithms!

Task Hierarchies as Domain Knowledge



- **Flat** hierarchy only knows model averagers.
- **Shallow** hierarchy also knows that gathering each resource is independent.
- **Deep** hierarchy also knows the set of resource locations (but must still associate resource with location).

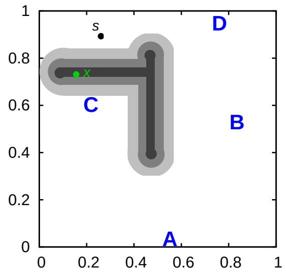


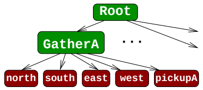
How does hierarchical decomposition benefit Reinforcement Learning? Clearly, a given task hierarchy comprises prior knowledge for a given domain. In these experiments, even a flat hierarchy, which implements R-MAX and Fitted R-MAX, benefits from state abstraction: each primitive action model uses a minimal state representation.

The following slides show experiments in the resource-gathering domain with a shallow hierarchy, which captures the intuition that each resource corresponds to an independent subtask.

The thesis also explores deeper hierarchies in the context of the discrete Taxi domain, where the hierarchy also includes the knowledge that certain coordinates in the environment are important.

Soft Inductive Bias in Hierarchy





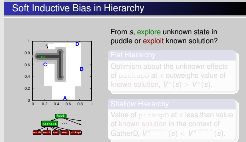
From *s*, explore unknown state in puddle or exploit known solution?

Flat Hierarchy

Optimism about the unknown effects of `pickupD` at *x* outweighs value of known solution, $V^{\pi}(s) > V^{\pi}(s)$.

Shallow Hierarchy

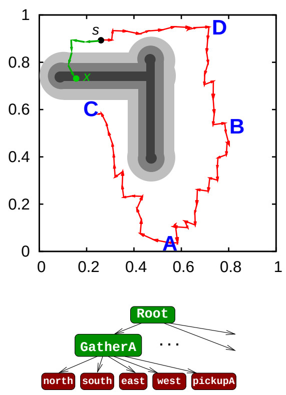
Value of `pickupD` at *x* less than value of known solution in the context of GatherD, $V^{\pi^{\text{GatherD}}}(s) < V^{\pi^{\text{GatherD}}}(s)$.



Hierarchical knowledge can curb the over-enthusiastic exploration of R-MAX. Suppose an agent in state *s* considers using its model to reach state *x*, in the middle of a puddle, where the effect of `pickupD` is unknown. In the absence of hierarchy, the agent compares its optimistic value for this exploration against the estimated value of exploitation: gathering the four resources. In this case, the agent chooses exploration, since the cost of wading through the puddles to reach the optimistic reward seems smaller than the cost of completing the entire task.

Given hierarchical knowledge, the agent only considers exploring `pickupD` at *x* in the context of the `GatherD` subtask, so it only compares the cost of exploration against the cost of completing `GatherD`. Hierarchy therefore allows the agent to apply a higher threshold when considering the value of an exploratory policy, without sacrificing any ability to converge to a (hierarchically) optimal policy.

Soft Inductive Bias in Hierarchy



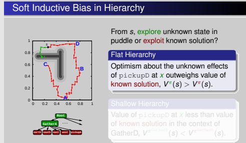
From *s*, explore unknown state in puddle or exploit known solution?

Flat Hierarchy

Optimism about the unknown effects of `pickupD` at *x* outweighs value of known solution, $V^{\pi}(s) > V^{\pi}(s)$.

Shallow Hierarchy

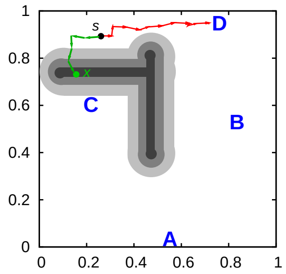
Value of `pickupD` at *x* less than value of known solution in the context of GatherD, $V^{\pi}{}^{\mathrm{GatherD}}(s) < V^{\pi}{}^{\mathrm{GatherD}}(s)$.

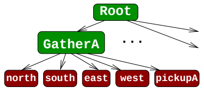


Hierarchical knowledge can curb the over-enthusiastic exploration of R-MAX. Suppose an agent in state *s* considers using its model to reach state *x*, in the middle of a puddle, where the effect of `pickupD` is unknown. In the absence of hierarchy, the agent compares its optimistic value for this exploration against the estimated value of exploitation: gathering the four resources. In this case, the agent chooses exploration, since the cost of wading through the puddles to reach the optimistic reward seems smaller than the cost of completing the entire task.

Given hierarchical knowledge, the agent only considers exploring `pickupD` at *x* in the context of the `GatherD` subtask, so it only compares the cost of exploration against the cost of completing `GatherD`. Hierarchy therefore allows the agent to apply a higher threshold when considering the value of an exploratory policy, without sacrificing any ability to converge to a (hierarchically) optimal policy.

Soft Inductive Bias in Hierarchy





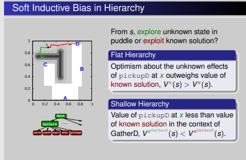
From *s*, explore unknown state in puddle or exploit known solution?

Flat Hierarchy

Optimism about the unknown effects of `pickupD` at x outweighs value of known solution, $V^{\pi}(s) > V^{\pi}(s)$.

Shallow Hierarchy

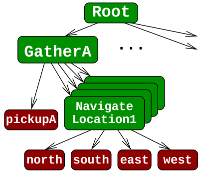
Value of `pickupD` at x less than value of known solution in the context of GatherD, $V^{\pi^{\mathtt{GatherD}}}(s) < V^{\pi^{\mathtt{GatherD}}}(s)$.



Hierarchical knowledge can curb the over-enthusiastic exploration of R-MAX. Suppose an agent in state *s* considers using its model to reach state *x*, in the middle of a puddle, where the effect of `pickupD` is unknown. In the absence of hierarchy, the agent compares its optimistic value for this exploration against the estimated value of exploitation: gathering the four resources. In this case, the agent chooses exploration, since the cost of wading through the puddles to reach the optimistic reward seems smaller than the cost of completing the entire task.

Given hierarchical knowledge, the agent only considers exploring `pickupD` at *x* in the context of the `GatherD` subtask, so it only compares the cost of exploration against the cost of completing `GatherD`. Hierarchy therefore allows the agent to apply a higher threshold when considering the value of an exploratory policy, without sacrificing any ability to converge to a (hierarchically) optimal policy.

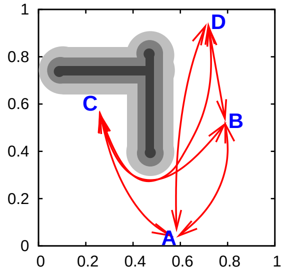
Hierarchical Constraint and Reformulation

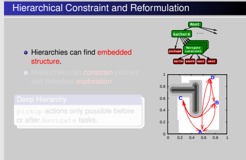


- Hierarchies can find embedded structure.
- Hierarchies can constrain policies and therefore exploration.

Deep Hierarchy

pickup actions only possible before or after Navigate tasks.

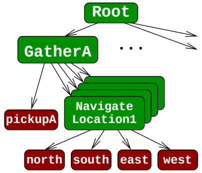




HIerarchy also conveys important computational advantages. Even the shallow hierarchy allows the planning at the `Root` task to focus on higher-level structure. At the root, the agent only considers which resource to gather next. After its models of each subtask converge, the root task essentially solves an embedded instead of the Traveling Salesman Problem, where the four resource locations correspond to the cities that must be visited, and the subtask models define the costs of going from one city to the next. Note that after the first time step, `Root` only selects actions when at one of the four resource locations.

A deeper task hierarchy can apply the same constraints to `GatherA` and its siblings. After the first time step of each episode, the agent will only attempt `pickupA` at one of the four resource locations, avoiding substantial amounts of unnecessary exploration.

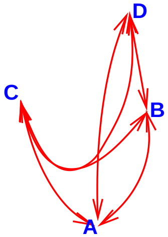
Hierarchical Constraint and Reformulation

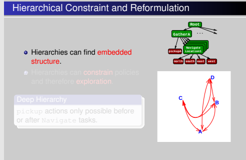


- Hierarchies can find embedded structure.
- Hierarchies can constrain policies and therefore exploration.

Deep Hierarchy

`pickup` actions only possible before or after `Navigate` tasks.





HIerarchy also conveys important computational advantages. Even the shallow hierarchy allows the planning at the `Root` task to focus on higher-level structure. At the root, the agent only considers which resource to gather next. After its models of each subtask converge, the root task essentially solves an embedded instead of the Traveling Salesman Problem, where the four resource locations correspond to the cities that must be visited, and the subtask models define the costs of going from one city to the next. Note that after the first time step, `Root` only selects actions when at one of the four resource locations.

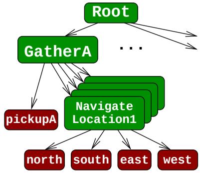
A deeper task hierarchy can apply the same constraints to `GatherA` and its siblings. After the first time step of each episode, the agent will only attempt `pickupA` at one of the four resource locations, avoiding substantial amounts of unnecessary exploration.

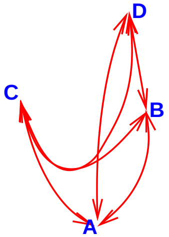
Hierarchical Constraint and Reformulation

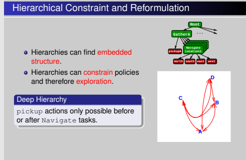
- Hierarchies can find embedded structure.
- Hierarchies can constrain policies and therefore exploration.

Deep Hierarchy

`pickup` actions only possible before or after `Navigate` tasks.







HIerarchy also conveys important computational advantages. Even the shallow hierarchy allows the planning at the `Root` task to focus on higher-level structure. At the root, the agent only considers which resource to gather next. After its models of each subtask converge, the root task essentially solves an embedded instead of the Traveling Salesman Problem, where the four resource locations correspond to the cities that must be visited, and the subtask models define the costs of going from one city to the next. Note that after the first time step, `Root` only selects actions when at one of the four resource locations.

A deeper task hierarchy can apply the same constraints to `GatherA` and its siblings. After the first time step of each episode, the agent will only attempt `pickupA` at one of the four resource locations, avoiding substantial amounts of unnecessary exploration.

Outline

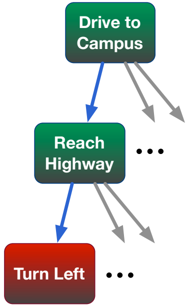
2010-12-15

Structured Exploration for Reinforcement Learning
└Conclusion
└Outline

Outline

Introduction
Exploration and Approximation
Exploration and Hierarchy
Conclusion
- Future Work
- Summary

Discovering Abstractions

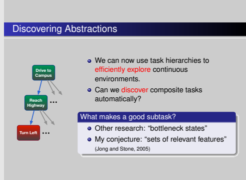


- We can now use task hierarchies to efficiently explore continuous environments.
- Can we discover composite tasks automatically?

What makes a good subtask?

- Other research: “bottleneck states”
- My conjecture: “sets of relevant features”

(Jong and Stone, 2005)



This thesis assumes that task hierarchies are available as prior knowledge, but the open problem of how to discovery such hierarchies automatically directly motivated the synthesis of hierarchy with model-based exploration and function approximation. The thesis directly addresses what makes for a good hierarchy, particularly in the context of model-based methods that already handle exploration, one of the supposed motivations for using hierarchies. In particular, hierarchies should constrain exploration and permit compact models at each level of the hierarchy.

R-MAXQ and Fitted R-MAXQ are designed to serve as foundations for research into hierarchy discovery. In particular, they explicitly confine all the directly learned knowledge to the primitive actions. The composite tasks, which only perform bottom-up planning given primitive action models, can be swapped out on the fly at the cost of replanning.

Prior Distributions Over Inductive Biases

No Free Lunch

No algorithm can learn or discover efficiently in all possible worlds!

Bayesian Reinforcement Learning

- Begin with a prior distribution over environments
- Plan over “belief states”
- Update belief distribution given data

Key question: What is the right prior distribution?

Conjecture: Distributions over task hierarchies

Goal: Efficient appoximation of optimal Bayesian solution

Prior Distributions Over Inductive Biases
No Free Lunch
No algorithm can learn or discover efficiently in all possible worlds!
Bayesian Reinforcement Learning
Begin with a prior distribution over environments
Plan over “belief states”
Update belief distribution given data
Key question What is the right prior distribution?
Conjecture Distributions over task hierarchies
Goal Efficient appoximation of optimal Bayesian solution

More generally, this thesis recognizes that no algorithm can learn efficiently in real-world domains in the absence of any prior knowledge. The MDP formalism is too rich a hypothesis space, and current analyses of sample complexity implicitly assume that any MDP is possible and that all possibilities are equally likely. Function approximation methods necessarily assume some notion of smoothness, but additional structure is necessary to make RL practical.

This thesis employs hierarchy as a natural way for users to communicate domain knowledge to an RL agent. Hierarchy may also provide a reasonable language for expression prior distributions over MDPs.

Natural Knowledge Representations

- Model-free methods learn a monolithic value function.
- Models are a natural form of domain knowledge.
- Models are modular: piecewise independent.

Don't Reinvent the Wheel

- Exploit known reward function
- Exploit known dynamics of some actions
- Exploit known dynamics of some state variables

Natural Knowledge Representations
- Model-free methods learn a monolithic value function.
- Models are a natural form of domain knowledge.
- Models are modular: piecewise independent.
Don't Reinvent the Wheel
- Exploit known reward function
- Exploit known dynamics of some actions
- Exploit known dynamics of some state variables

Models, as well as hierarchies, seem a natural way to communicate domain knowledge to a practical RL agent. A key benefit of the model decomposition developed in this thesis is that each primitive action or composite task can have a model completely independent of its siblings. One consequence is that instead of requiring the agent to learn models for every action, the agent may be given as prior knowledge the model for any task in the hierarchy, removing the need for learning in that subtree. At upper levels of the hierarchy, the correct high-level policy may be known, leaving the agent only to evaluate that policy by learning a model of that task.

Connections to Other Fields of Artificial Intelligence

- Higher-level actions are more deterministic and discrete.
- Abstract actions could help define abstract state variables.
- Example: A Boolean feature predicting that a task will reach a “good” terminal state.
- Possibly define tasks with postconditions that achieve other tasks’ preconditions

Recognize Familiar Problems Emerging from Data

- Classical planning, scheduling, constraint satisfaction
- Object recognition and multi-agent learning

Connections to Other Fields of Artificial Intelligence
- Higher-level actions are more deterministic and discrete.
- Abstract actions could help define abstract state variables.
- Example: A Boolean feature predicting that a task will reach a “good” terminal state.
- Possibly define tasks with postconditions that achieve other tasks’ preconditions
Recognize Familiar Problems Emerging from Data
- Classical planning, scheduling, constraint satisfaction
- Object recognition and multi-agent learning

Finally, this thesis may help bridge the gap between RL methods and the rich literature in classical planning techniques. In Fitted R-MAXQ, the primitive actions may be stochastic and continuous, but its bottom-up planning naturally results in high-level models that are more deterministic and discrete. In the resource-gathering domain, the modeling process essentially recovers an embedded instance of the Travelling Salesman Problem, which could in principle be solved using more sophisticated methods than value iteration. In general, hierarchical models could transform an MDP representation into something closer to classical planning operators. Reasoning about the preconditions and postconditions of existing subtasks could also form the basis for new abstract states and abstract actions.

Summary

- Agents that apply principled exploration to structured environments
- Exploration in continuous domains that require generalization
- Exploration in domains with hierarchical structure
- Publicly available implementation

- The bigger picture
  - Extend the reach of RL closer to the real world
  - Build a foundation for work in structure discovery

Summary
- Agents that apply principled exploration to structured environments
- Exploration in continuous domains that require generalization
- Exploration in domains with hierarchical structure
- Publicly available implementation
- The bigger picture
  - Extend the reach of RL closer to the real world
  - Build a foundation for work in structure discovery

The primary contribution of this thesis is its extension of model-based exploration methods to settings that previously relied on random exploration. In particular, the Fitted R-MAX algorithm brings R-MAX exploration to continuous state spaces by reasoning explicitly about how broadly to generalize data. The R-MAXQ algorithm combines R-MAX exploration with MAXQ decomposition, allowing intuitive domain knowledge to inform the exploration policy.

So that this work might serve as a foundation for ongoing research into exploration and discovery in structured environments, an implementation of the full Fitted R-MAXQ algorithm resides in the RL Library.