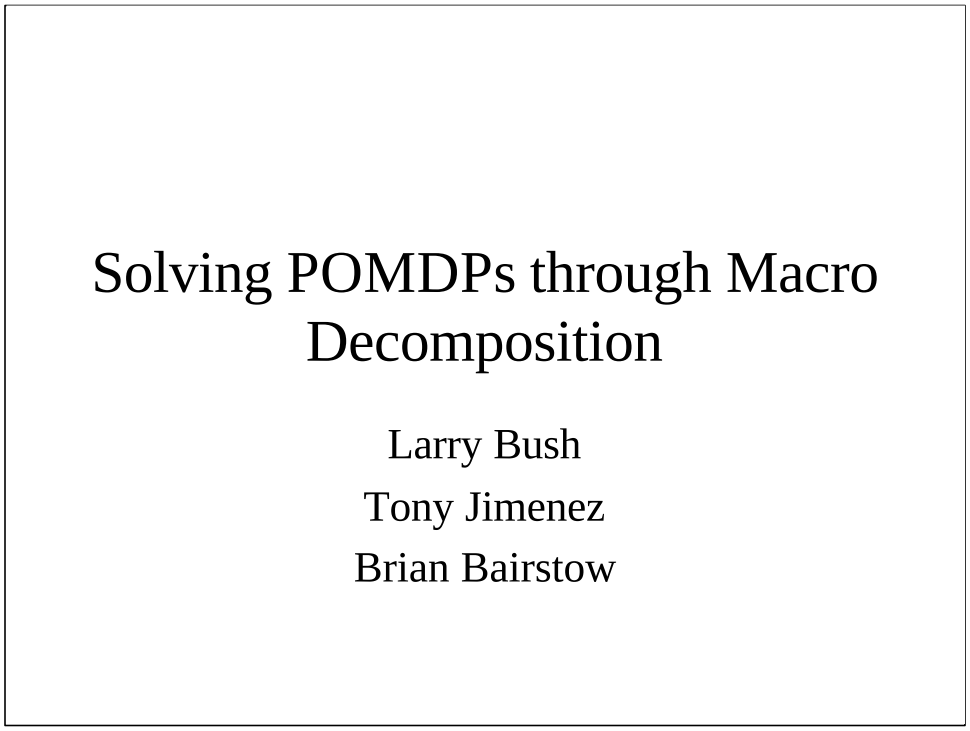
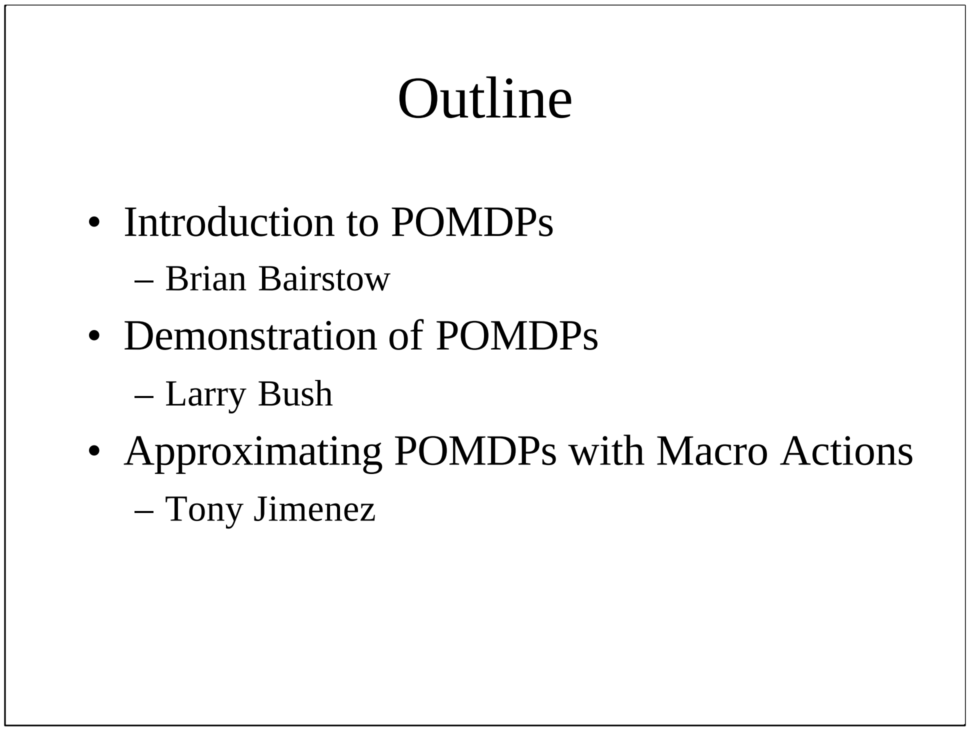


Solving POMDPs through Macro Decomposition

Larry Bush
Tony Jimenez
Brian Bairstow

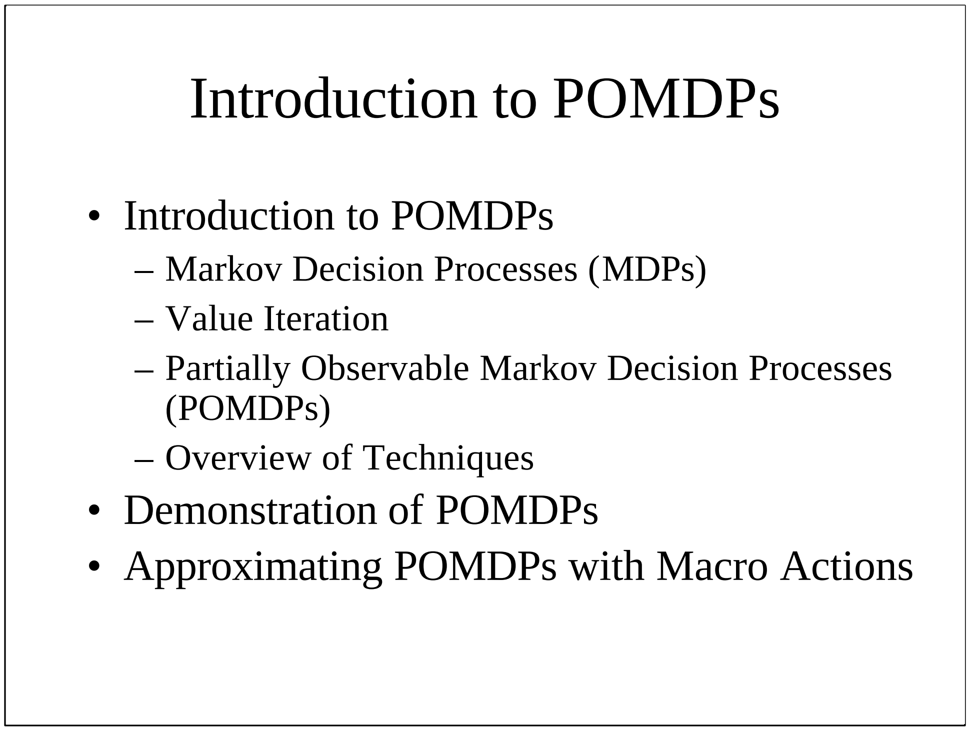
POMDPs are a planning technique that accounts for uncertainty in the world. While they have potential, they are very computationally complex. Macro operators can reduce this complexity.



Outline

- Introduction to POMDPs
  - Brian Bairstow
- Demonstration of POMDPs
  - Larry Bush
- Approximating POMDPs with Macro Actions
  - Tony Jimenez

We will begin with an overview of MDP and POMDPs, and then a visual demonstration of simple MDPs and POMDPs. Finally we will discuss our advanced topic in POMDPS: the approximation of POMDPs using macro actions.

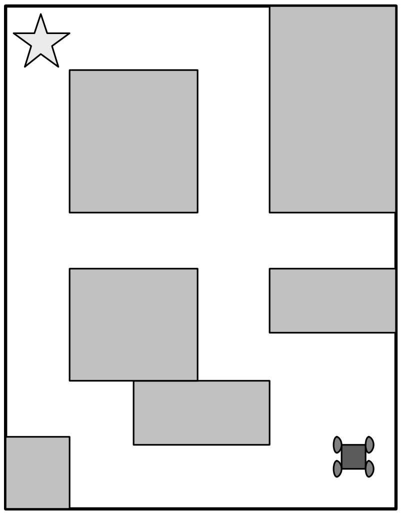


Introduction to POMDPs

- Introduction to POMDPs
  - Markov Decision Processes (MDPs)
  - Value Iteration
  - Partially Observable Markov Decision Processes (POMDPs)
  - Overview of Techniques
- Demonstration of POMDPs
- Approximating POMDPs with Macro Actions

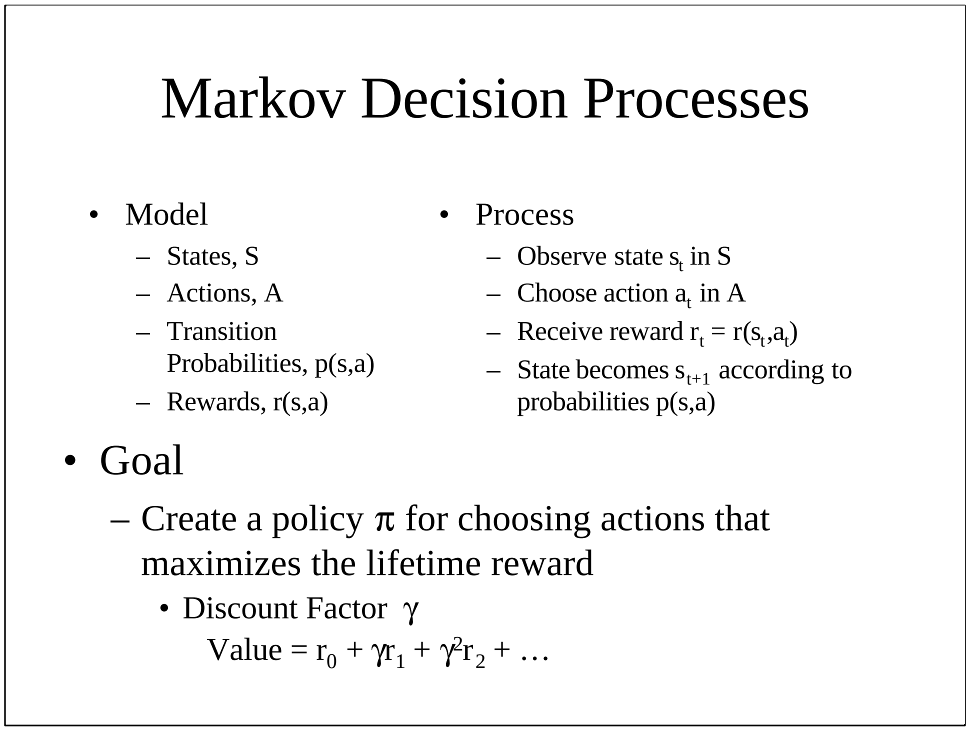
Begin with completely observable Markov Decision Processes, or MDPs. Then discuss value iteration as a method of solving MDPs. This will lay the groundwork for *Partially* Observable Markov Decision Processes, or POMDPs. Finally there will be an overview of methods for solving POMDPs.

Navigation of a Building



- Robot
  - Knows building map
  - Wants to reach goal
  - Uncertainty in actions
  - What is the best way to get there?

Imagine a robot (in the lower right of the graphic) trying to navigate a building. It wants to reach a goal (the star in the graphic). It has difficulties in that there are uncertainties in its actions, for example its wheels could slip or catch. This might cause it to run into a wall, which is undesirable. The problem then is to find the best way to reach the goal while dealing with the problem of uncertainty in actions. How can we solve this problem?



Markov Decision Processes

- Model
  - States, S
  - Actions, A
  - Transition Probabilities, p(s,a)
  - Rewards, r(s,a)
- Process
  - Observe state s_t in S
  - Choose action a_t in A
  - Receive reward $r_t = r(s_t,a_t)$
  - State becomes s_{t+1} according to probabilities p(s,a)
- Goal
  - Create a policy π for choosing actions that maximizes the lifetime reward
    - Discount Factor γ

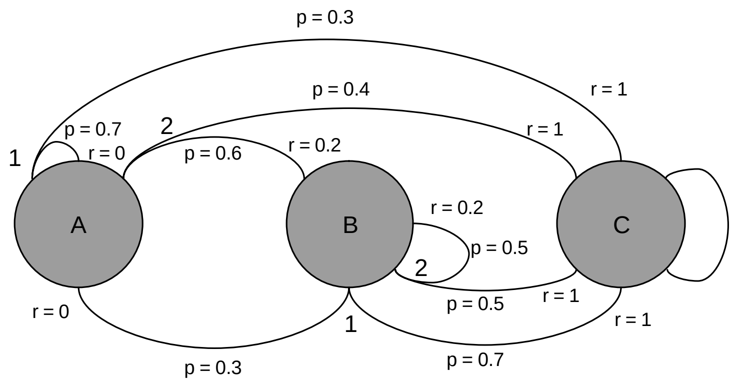
      Value = $r_0 + \gamma r_1 + \gamma^2 r_2 + \ldots$

We can use a MDP to solve the previous problem. An MDP consists of a model with states, actions, transitions, and expected rewards. The states in the previous problem could be positions on the map. States are discrete, so the map would have to be divided into a grid or something similar. The actions then could be to move north, east, south, or west on the map. The transition probabilities tell you the chances that a certain action from a certain state takes you to different other states. For instance, if the robot was commanded to move north, there might be a large probability that it transitions to the next state north, and small probabilities it ends up east, west, or not move at all. The reward function tells you the expected reward received by taking an action from a state. In the previous problem this could be a large reward for reaching the goal, and a large negative reward for hitting a wall.

The process in carrying out an MDP solution is to observe the state in time step t, to choose an appropriate action, to receive the reward corresponding to that state and action, and to change the state according to the transition probabilities. Note that the robot is in exactly one state at a time, that time is discrete, and all actions take one time step.

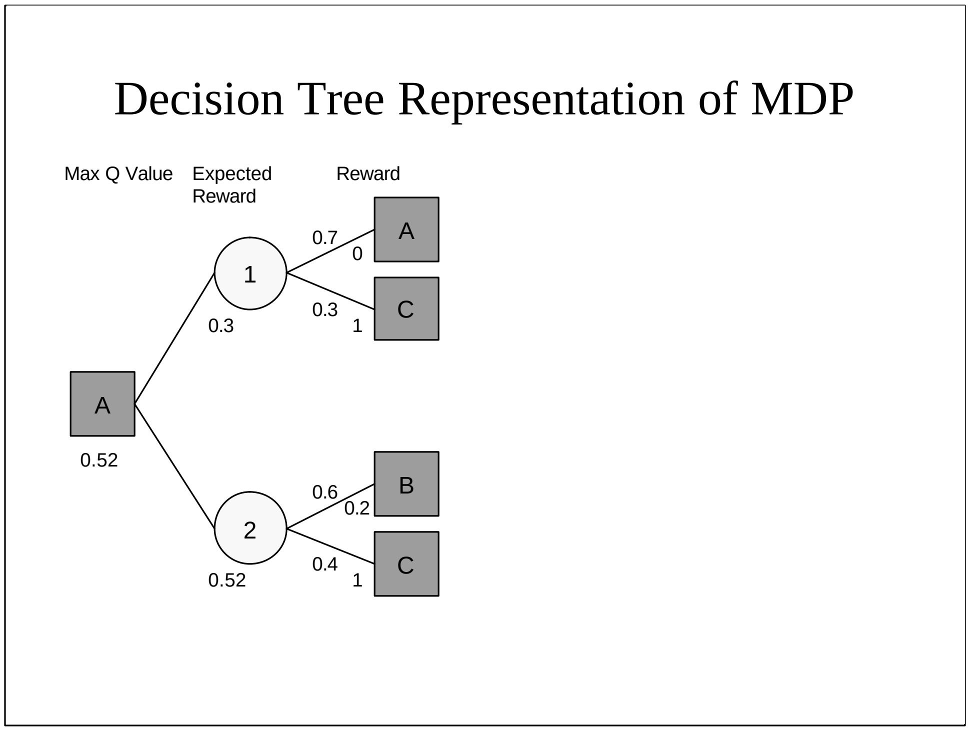
The goal in solving an MDP is to create a policy (a method for choosing actions) that maximizes the expected lifetime reward. The lifetime reward is the sum of all rewards received. Thus a policy should not always maximize immediate reward, but also plan ahead. Future rewards are discounted by a factor for each time step they are in the future. This follows an economic principle of the effect of time on values. Also, this makes the math work in calculating lifetime reward. Without discounting, it could be possible to receive a small reward over and over to get an infinite reward, and this is not useful in choosing a policy. A typical discount factor might be 0.9.

MDP Model Example

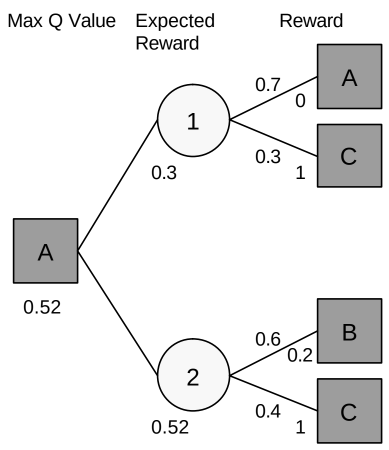


- States A, B, C
- Actions 1, 2
- Transition probabilities p and rewards r in diagram
- C is terminal state

This is a simple example of an MDP model (unrelated to the previous robot example). There are 3 states and two actions. The probabilities and rewards are written in the diagram. For example, from state A, if action 1 is chosen then there is a 70% chance of staying in state A with 0 reward, and a 30% chance of moving to state C with 1 reward. State C is the termi nal state, since there are no actions that move out of state C.



Decision Tree Representation of MDP



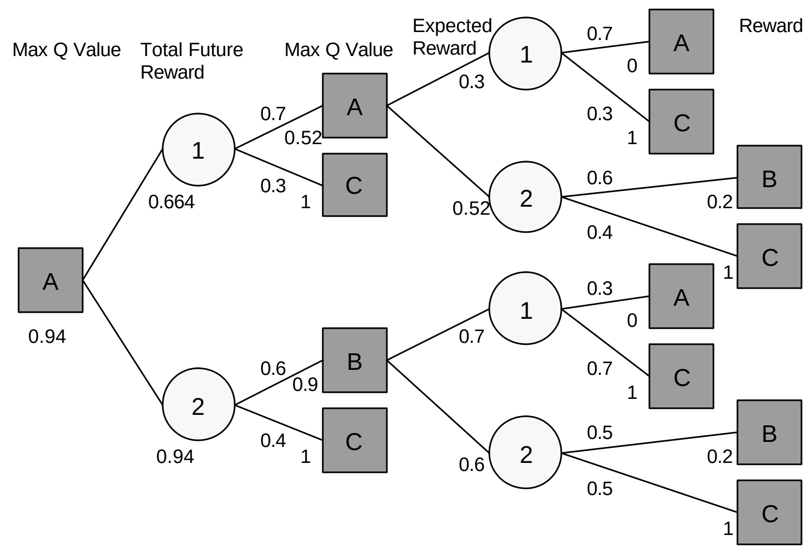
A Decision tree is another form of visualization, and allows you to evaluate the values of states.

Here is a 1 step horizon decision tree for the simple model shown. From the starting point A there are two actions, 1 and 2. The action chosen gives probabilities of moving to different states.

Notice that the rewards for moving to the states are listed in right at the leaf nodes. Expected rewards can then be calculated for taking actions. For instance, for action 1 the expected reward is .7(0) + .3(1) = 0.3. When the expected rewards for the actions are known, then the highest reward path should be followed. This means that in a 1 step problem at state A, action 2 should be taken, and state A has a value of 0.52, equal to the expected reward of taking action 2.

However this is only a one step problem. We have not considered that it is important what state you end up in because of future rewards. For instance, we have just discovered that state A has a value of .52, but in the tree in the upper right we have evaluated it as having 0 value. Thus we need to look at a larger horizon problem.

Decision Tree Representation of MDP



Now we have a decision tree for a 2 step horizon starting at state A. For simplicity a discount value of 1 has been used.

Again we start at the right and calculate the expected rewards, and then the values for the states. Thus after 1 step A has a value of 0.52. Note that B has a value of 0.7 (future reward) + 0.2 (immediate reward) = 0.9. When the values of the states for the one step horizon are known, they can be used to calculate the 2 step horizon values. The expected values for the actions are calculated again, and then the max value (0.94) is assigned to state A for the 2 step problem. In general this process can be iterated out.

Value Iteration

- Finite Horizon, 1 step

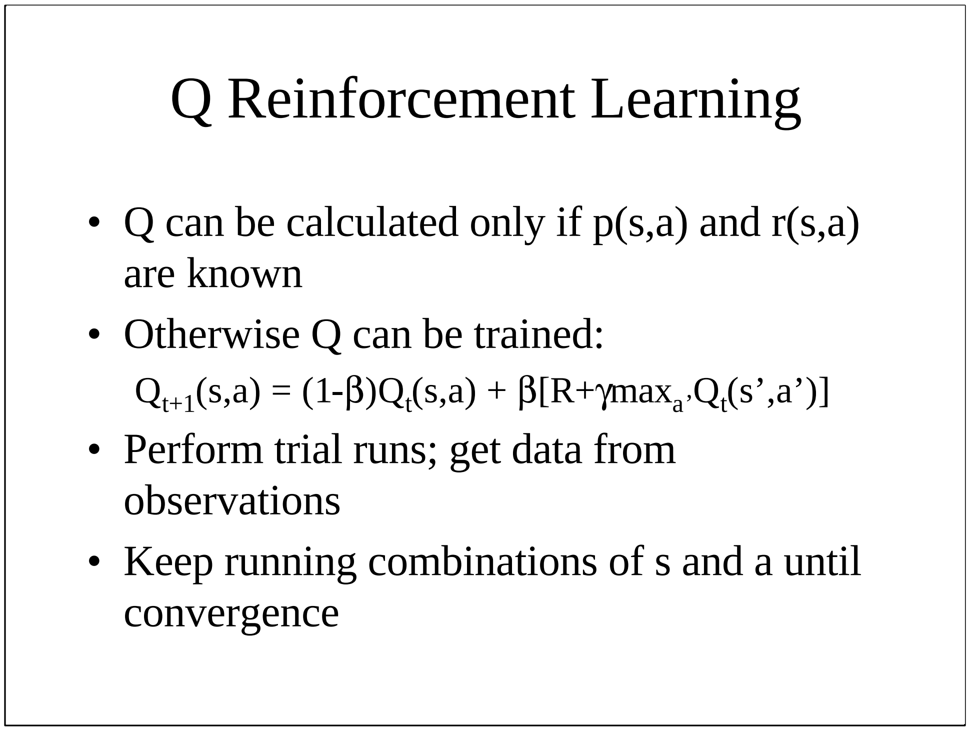
  $Q_1(s,a) = r(s,a)$
- Finite Horizon, n steps

  $Q_n(s,a) = r(s,a) + \gamma\Sigma[p(s'|s,a) \max_{a'} Q_{n-1}(s',a')]$
- Infinite Horizon

  $Q(s,a) = r(s,a) + \gamma\Sigma[p(s'|s,a) \max_{a'} Q(s',a')]$
- Policy

  $\pi(s) = \text{argmax}_a Q(s,a)$

This brings us to the concept of value iteration. Value iteration is the process of assigning values to all states, which then solves the MDP. As shown, in a 1 step horizon the value is merely the expected reward. In a larger horizon, the value is the expected reward plus the expected future reward discounted by the discount factor. After iterating to larger and larger horizons, the values change less and less. Eventually convergence criteria is met, and the problem is considered solved for an infinite horizon. At this point the policy is simply to take the action from each state that gives the largest reward.



Q Reinforcement Learning

- Q can be calculated only if p(s,a) and r(s,a) are known
- Otherwise Q can be trained:

  $Q_{t+1}(s,a) = (1-\beta)Q_t(s,a) + \beta[R+\gamma \max_{a'} Q_t(s',a')]$
- Perform trial runs; get data from observations
- Keep running combinations of s and a until convergence

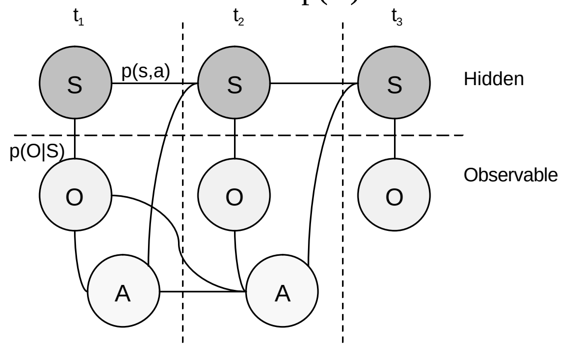
Note that value iteration can only be performed if the p and r functions are known. An alternate technique becomes very useful if you don't have a model. Instead of calculating p and r, they can be observed. The actions are carried out many times from each state, and the resultant state and reward are observed. The Q values are calculated from the observations according to the equation above, and eventually the data converges ($Q_{t+1} \sim= Q_t$).

Q learning is used in our topic paper.

Beta is a weighting factor, R is the observed reward

Partially Observable MDP’s

- Can no longer directly observe the state of the system
- Instead at each step make an observation O, and know the probability of each observation for each state p(O|S)
- Also have initial state distribution $p(s^0)$

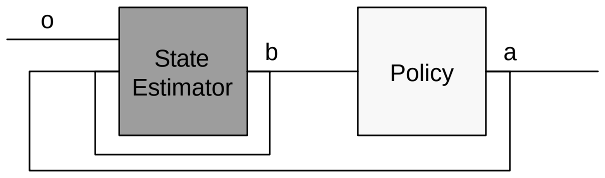


Now we move to POMDPs. Before actions were uncertain, but we always knew what state we were in. However, the real world has uncertainty, and POMDPs model that. For example, sensors can give false or imprecise data. Thus instead of knowing the state, at each time step we receive an observation that gives us some data about the state. In this graphic we see that the states influence the observations, which influence the actions, which in turn influence the state.

State Estimation

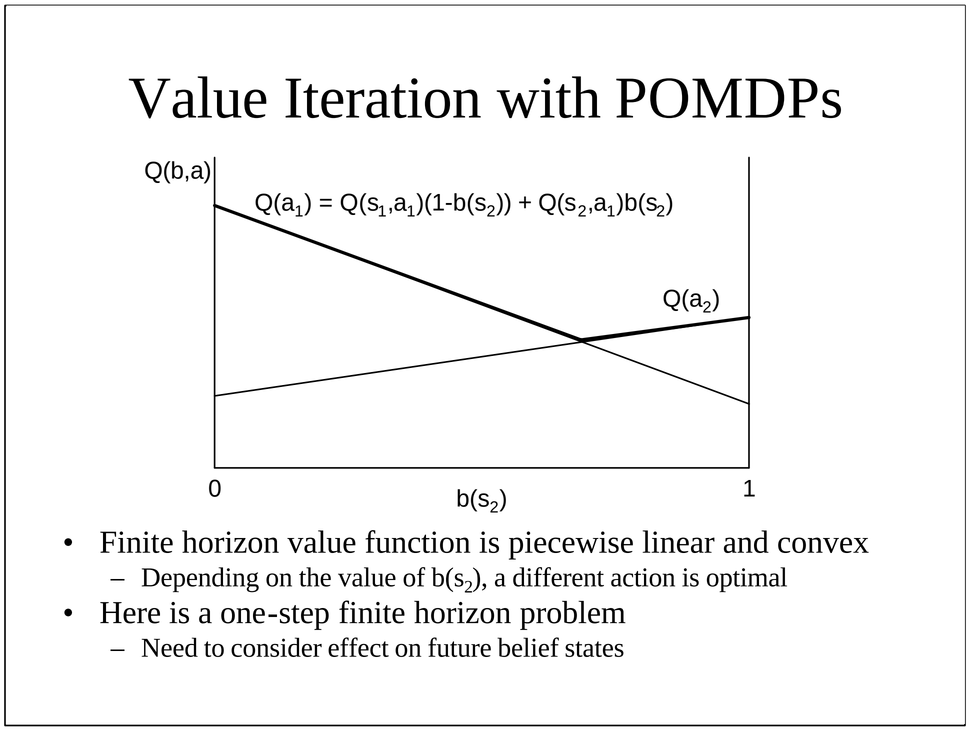
- States and Observations not independent of past observations
- Maintain a distribution of state probabilities called a belief b

$b^t = p(s^t|o^t,a^t,o^{t-1},a^{t-1},\ldots,o^0,a^0,s^0)$

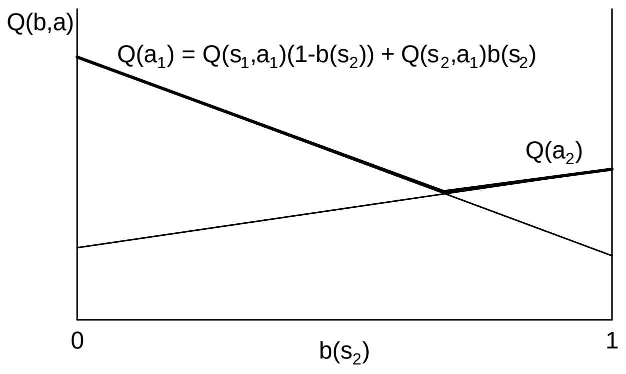


Since we don’t know exactly what state we are in, we instead have a belief state. A belief state is a mass distribution of the probabilities of being in each state. For instance, a belief could be b = < .5 .2 .3> which would be 50% chance of state 1, 20% chance of state 2, and 30% chance of state 3 (the probabilities sum to 100%). Thus instead of being in a state at a time step, we are in a belief state. These belief states a continuous distribution of discrete states. Instead of being in a finite number of states, we are in a infinite number of belief states over those finite states.

There is an additional step in the process of POMDPs compared to MDPs. Instead of observing a state and performing an action, we must make an observation, calculate the belief state, and then choose an action.



Value Iteration with POMDPs

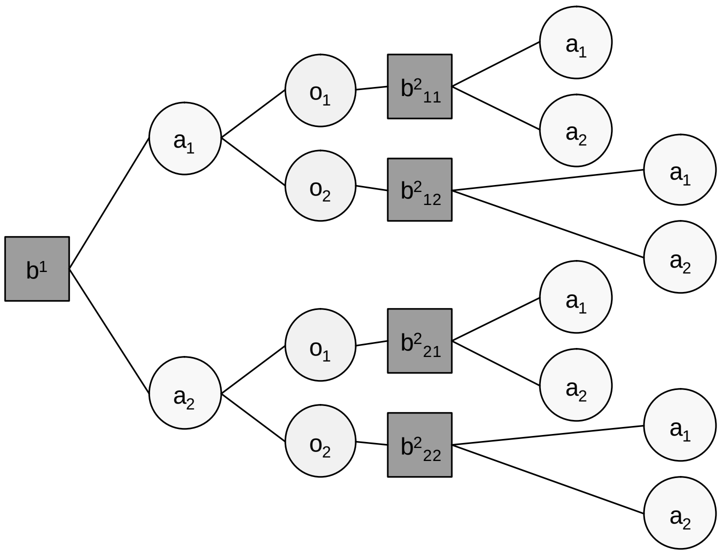


- Finite horizon value function is piecewise linear and convex
  - Depending on the value of $b(s_2)$, a different action is optimal
- Here is a one-step finite horizon problem
  - Need to consider effect on future belief states

Value iteration is more complicated with POMDPs because we have an infinite number of belief states. Above is a representation of a 2 state problem with probability of being at state 2 in the x axis. Plotted are the values of taking the two actions. Thus on the left there is certainty of being at state 1, and at the right there is certainty of being at state 2, which a linear change between them. Thus the values of actions are also linear as described in the equation on the slide. At any point in the belief space we choose the highest valued action to carry out. Thus in the left side we choose action 1, and on the right we choose action 2. In this manner our value function ends up piecewise linear and convex. In general there are more than two states, so we have hyperplanes instead of lines, but the principle of partitioning the belief space remains the same.

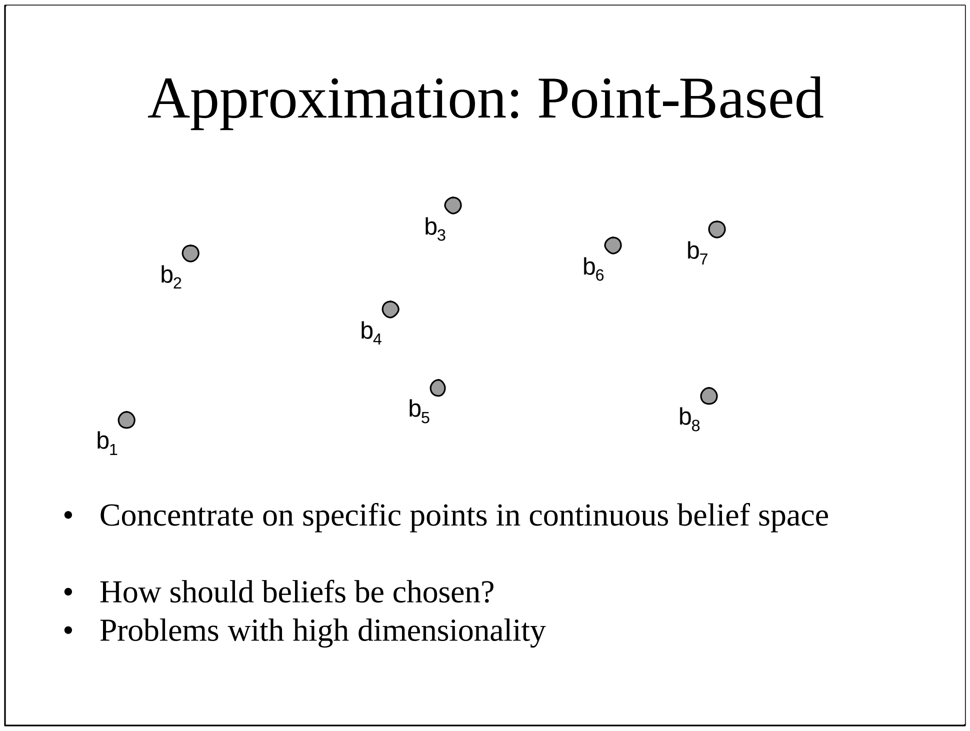
This is merely a one step problem, and it gets much more complicated over more time steps.

Decision Tree Representation of POMDP

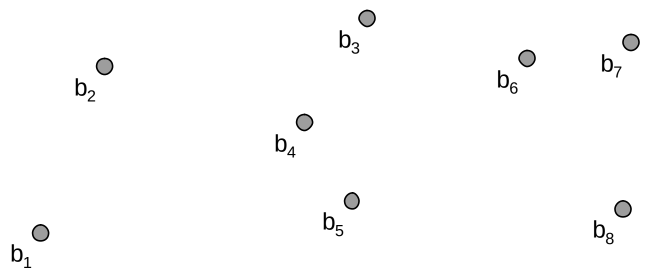


To tackle the problem of additional time steps, we will return to decision trees. This decision tree is similar to the one for MDPs, with a couple differences. The states have become belief states. The actions used to have uncertainty shown in the tree, but now that is encapsulated in the probabilities of the belief states. The observations have been added here as an additional step, because getting different observations brings the problem to different belief states.

Thus the process for value iteration is to first calculate the probabilities of observations given actions, and thus the distributions of the resultant belief states. Then the values of the belief states on the right in can be calculated based on expected rewards as one-step problems. Then since we know the probabilities of observations given actions, we can assign expected rewards to the actions a1 and a2. Then the value of belief b1 will be known. In general this can be extended to larger problems of n steps. However, since the number of belief states visited becomes larger exponentially by the number of actions, observations, and steps, this quickly becomes complex. In order to get good results, advanced techniques need to be used.



Approximation: Point-Based



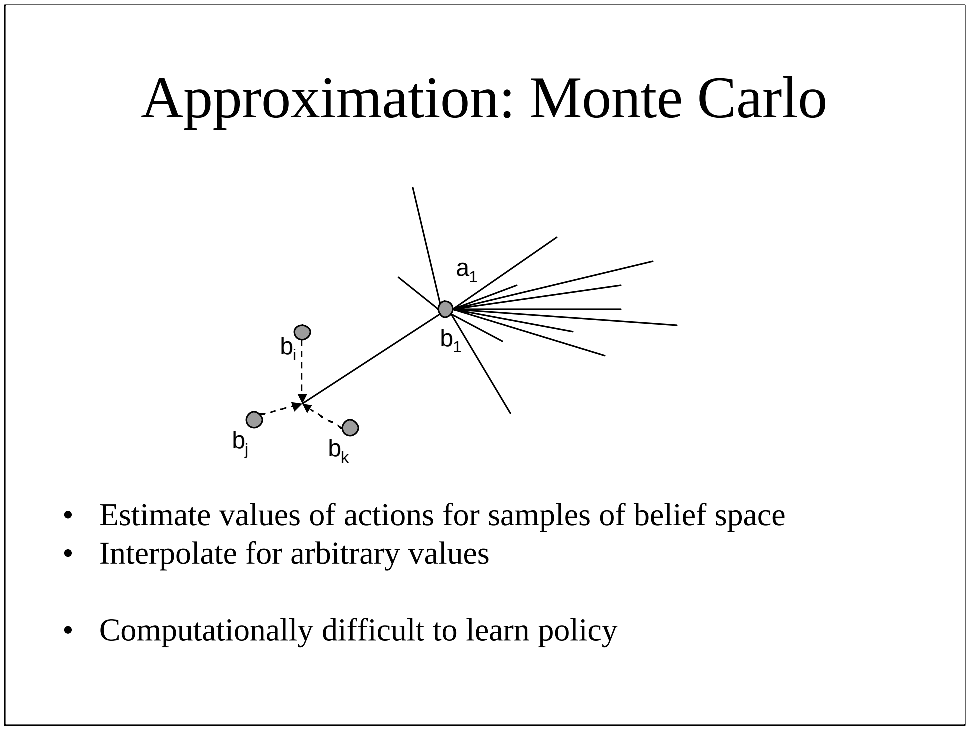
- Concentrate on specific points in continuous belief space

- How should beliefs be chosen?
- Problems with high dimensionality

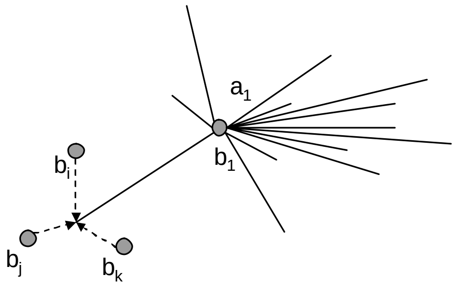
Approximation techniques allow simplification of POMDPs to make them tractable.

The point-based approximation technique follows the idea of optimizing the value function only for certain points in the belief space; not all points. This changes the problem from calculating a policy for a belief space to calculating a policy for a finite number of points.

This raises a question of how to choose belief points. One possibility is to do so dynamically. Also, when there is high dimensionality many points must be chosen in order to cover the space in a meaningful way.



Approximation: Monte Carlo



- Estimate values of actions for samples of belief space
- Interpolate for arbitrary values

- Computationally difficult to learn policy

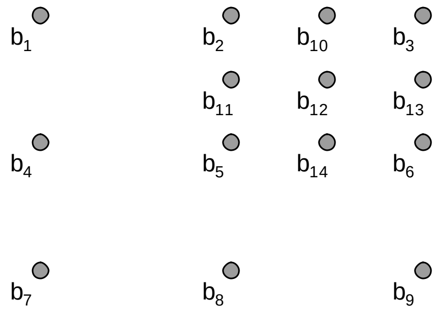
The Monte Carlo approach runs repeatedly on samples of belief space to learn their values. It is similar to reinforcement learning.

Arbitrary values are interpolated from nearby points.

This has the advantage of being a non-parametric representation, however it is still computationally difficult to learn a policy.

This technique is used in our advanced topic.

Approximation: Grid-Based



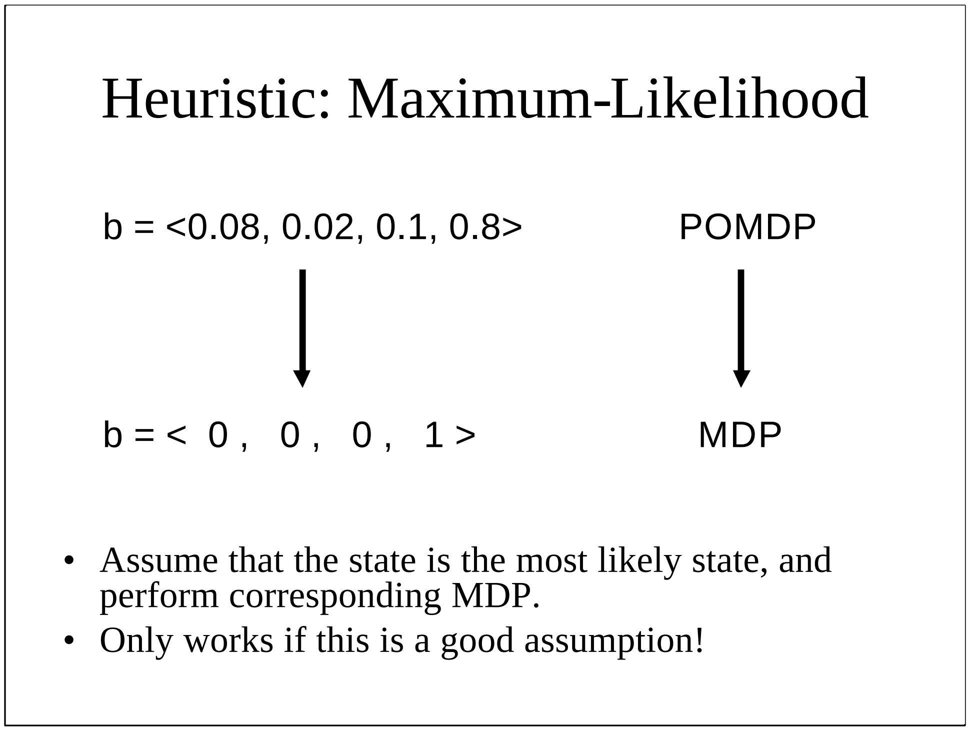
- Like point-based approximation, but with regular spacing
- Can use variable resolution

- Size increases with dimensionality and resolution

Grid-based approximations are point-based approximations with a regular grid spacing. This solves the problem of choosing points, however number of points still increases with dimensionality and with grid resolution.

One solution is to use variable resolution. This is the process of starting with a coarse grid resolution, and increasing the resolution only for interesting areas, essentially allowing zooming in on certain parts of the belief space.

Our advanced topic uses grid based approximations.



Heuristic: Maximum-Likelihood

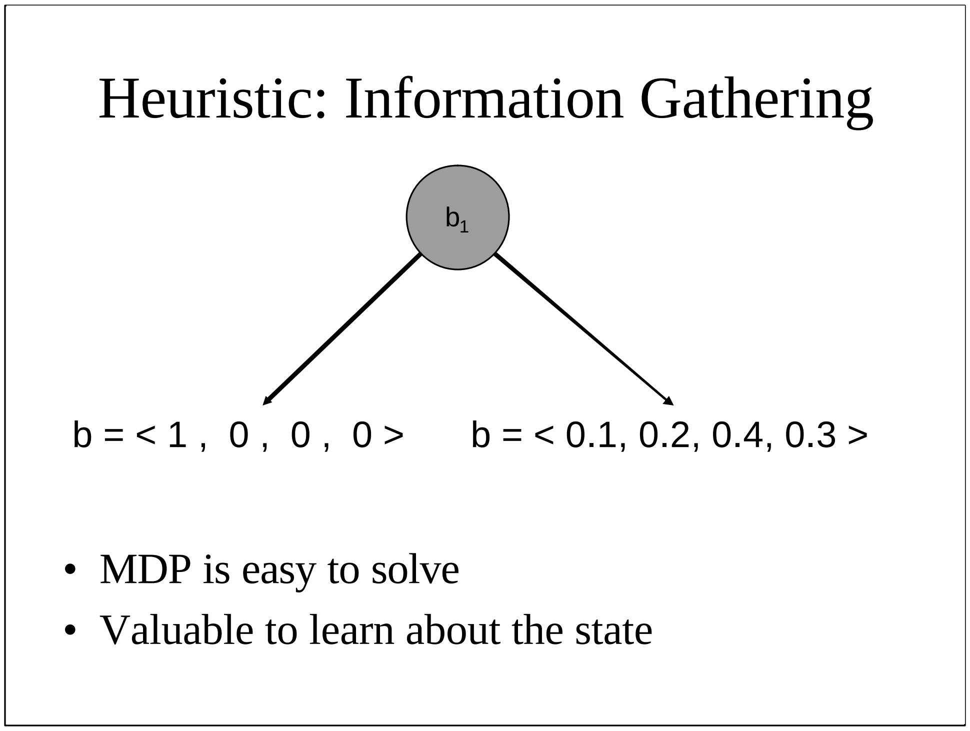
$$\begin{tikzcd} b = \langle 0.08, 0.02, 0.1, 0.8 \rangle \arrow[d] & \text{POMDP} \arrow[d] \\ b = \langle 0, 0, 0, 1 \rangle & \text{MDP} \end{tikzcd}$$

- Assume that the state is the most likely state, and perform corresponding MDP.
- Only works if this is a good assumption!

Now we move into different policy heuristics.

The Maximum likelihood heuristic is simply assuming that the state is the most likely state. This reduces the POMDP into an MDP, which can then be easily solved.

Of course, this only works if this is a good assumption. It works well when the uncertainty is due to bounded noise rather than uncertain state features.

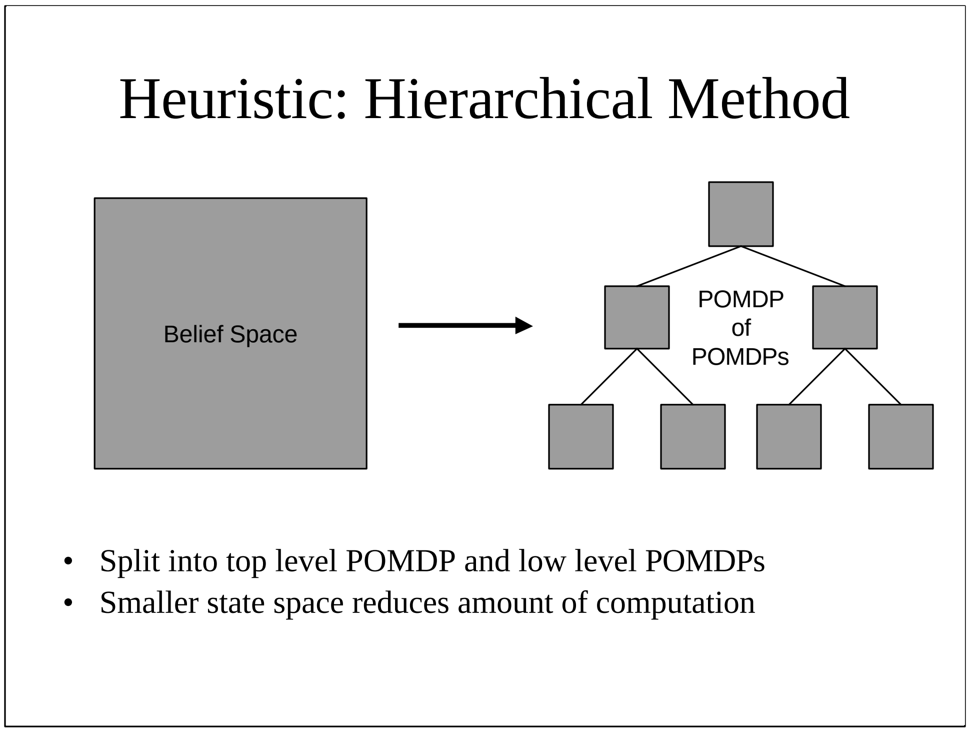


Heuristic: Information Gathering

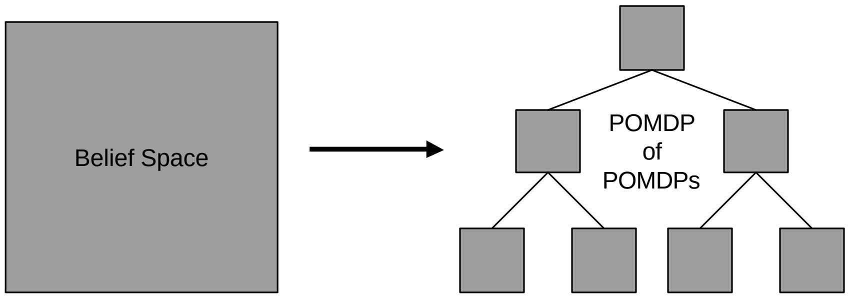
- MDP is easy to solve
- Valuable to learn about the state

The Information Gathering heuristic works on the idea that it is valuable to learn about the state, because then it is easy to maximize the reward.

Thus if two candidate actions have similar rewards, but one reduces the uncertainty in the system, then that one should be favored.



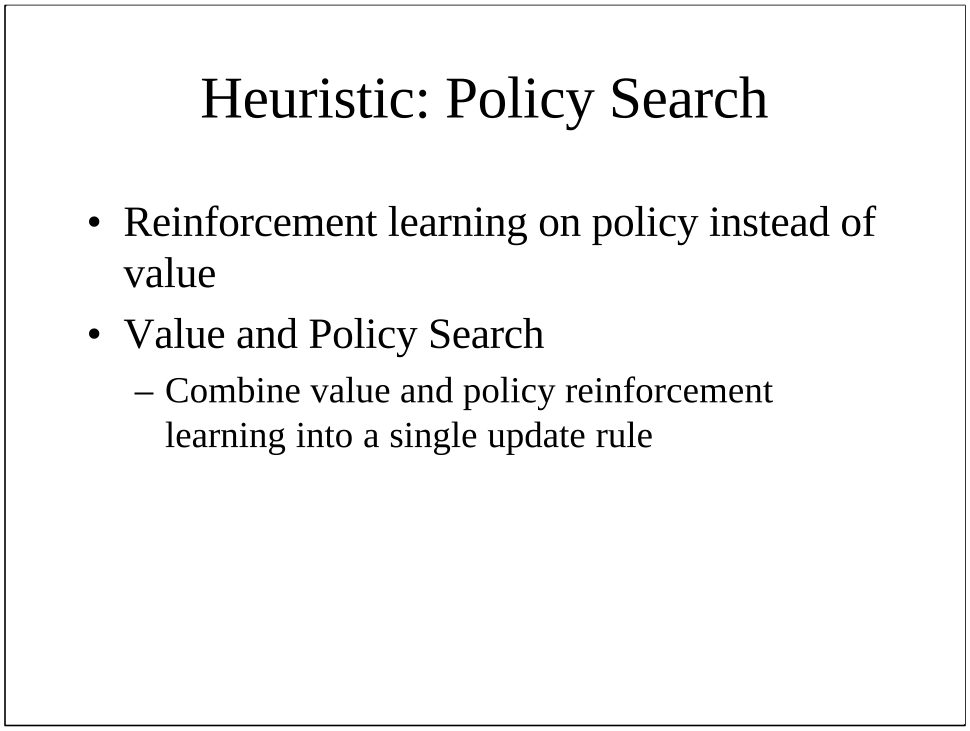
Heuristic: Hierarchical Method



- Split into top level POMDP and low level POMDPs
- Smaller state space reduces amount of computation

Hierarchical Methods involve splitting a POMDP into a top level POMDP and low level POMDPs. Each smaller POMDP has a smaller belief space, and thus the complexity is exponentially smaller. This is offset by the fact that there are more POMDPs to solve, but the number of POMDPs is polynomial, while the savings are exponential.

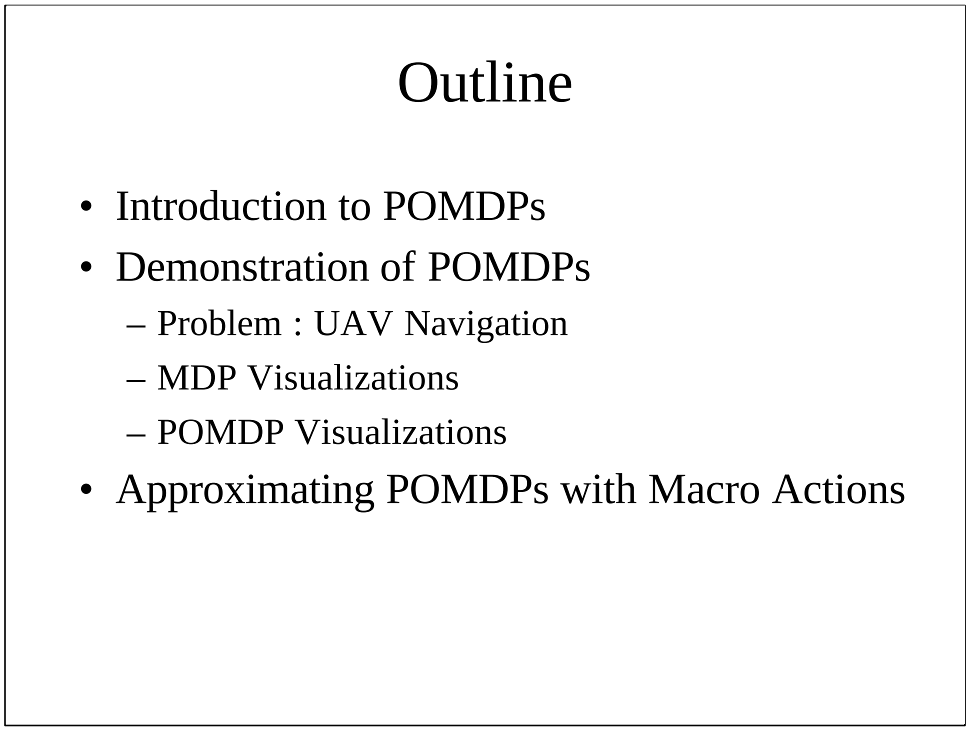
An example would be macro actions, which are described and used in our advanced topic.



Heuristic: Policy Search

- Reinforcement learning on policy instead of value
- Value and Policy Search
  - Combine value and policy reinforcement learning into a single update rule

The Policy Search heuristic is the idea of learning about the policy, not the value. It is very similar to Q reinforcement learning but is on the policy. Essentially the policy is optimized directly by training it to produce best value.



Outline

- Introduction to POMDPs
- Demonstration of POMDPs
  - Problem : UAV Navigation
  - MDP Visualizations
  - POMDP Visualizations
- Approximating POMDPs with Macro Actions

This is the demonstration portion of our talk. The objective is : understanding POMDPs through visualization.

I will explain to you a UAV Navigation problem that I modeled as an MDP and a POMDP.

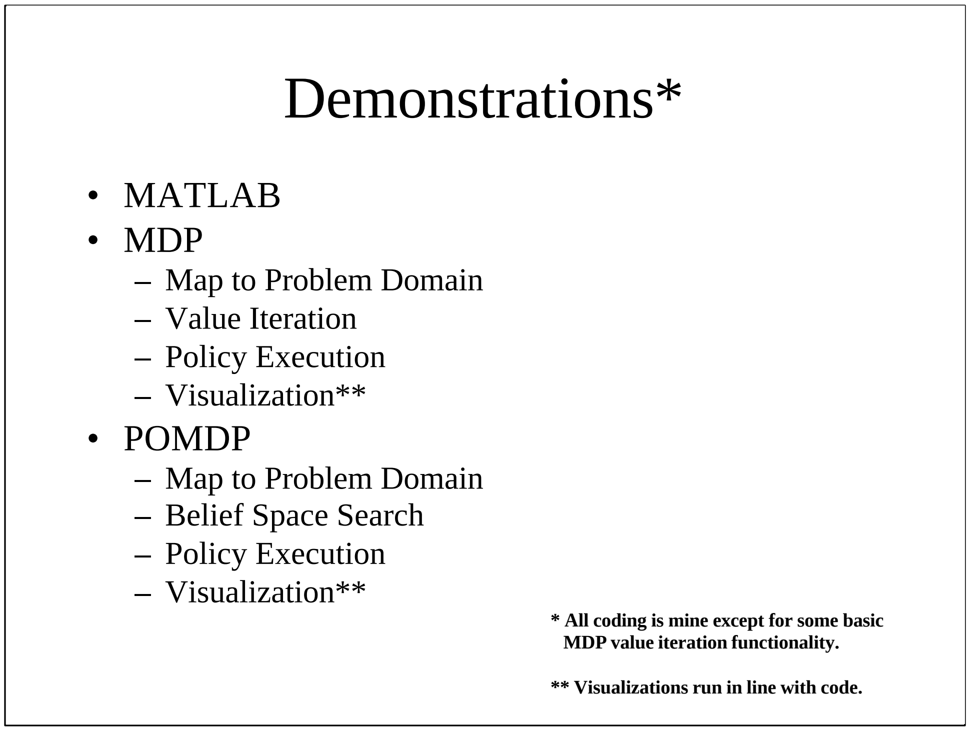
The visualizations will show the solution process and the policy execution process for an MDP, as a primer, and then a POMDP.

2 key points that I want you to take away from this demonstration are the essence of how POMDPs work, and the complexity of solving them.

For example:

The POMDP has a Continuous Belief State Space.

The POMDP's Policy Tree is polynomial in the number of actions and observations and exponential in the length of the planning horizon.



Demonstrations*

- MATLAB
- MDP
  - Map to Problem Domain
  - Value Iteration
  - Policy Execution
  - Visualization**
- POMDP
  - Map to Problem Domain
  - Belief Space Search
  - Policy Execution
  - Visualization**

*** All coding is mine except for some basic MDP value iteration functionality.**

**** Visualizations run in line with code.**

I implemented and solved the UAV Navigation problem, using MDPs and a POMDPs, in MATLAB.

This includes the policy execution and visualization code.

The visualizations that I created run inline with the code.

They were not generated after the fact.

They were generated in Matlab, as the code was running.

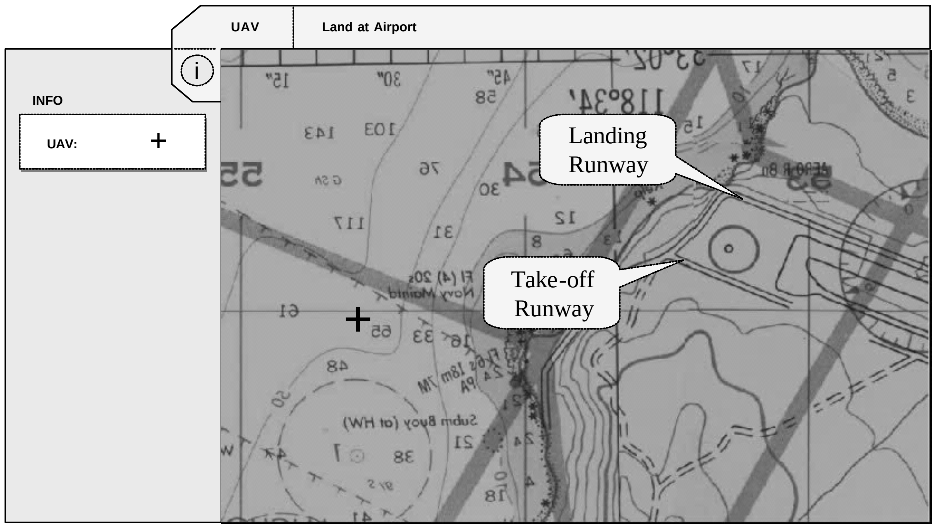
So, they allow you to peer inside and see what is happening.

I implemented everything that you will see with the exception of some basic MDP value iteration functionality.

Note:

I ran the code in various conditions and situations, some of which I will show you here.

Problem : Introduction

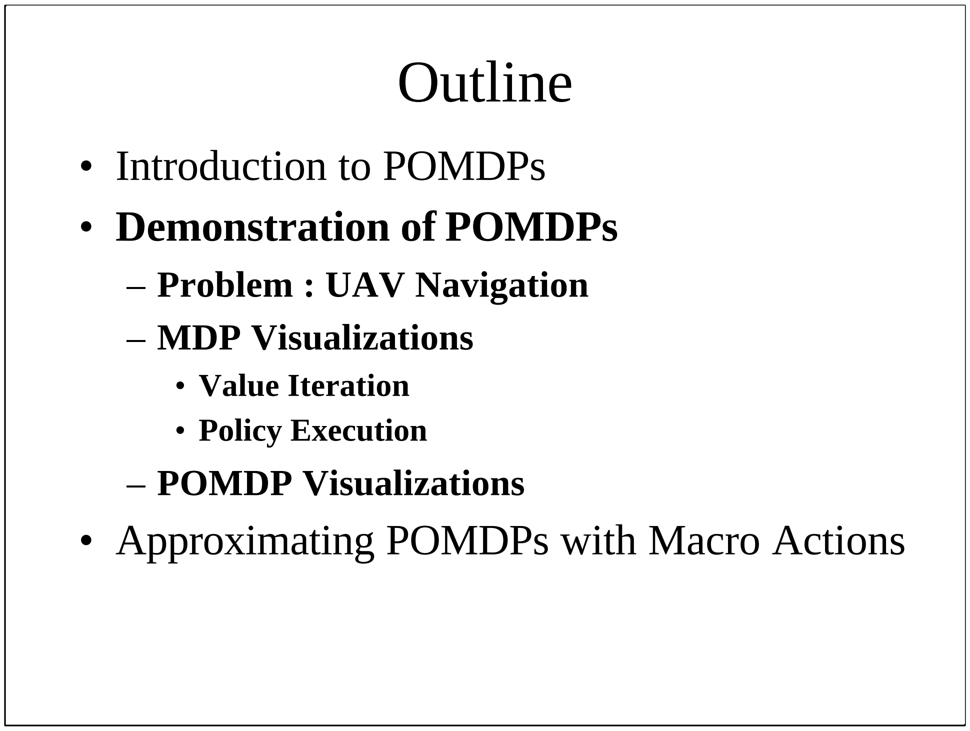


The point of this slide is to motivate the MDP and POMDP models / algorithms.

I mapped a small UAV navigation problem to the MDP and POMDP models.

In this navigation problem, we have a UAV (represented by the red +) who's objective is to land on the landing runway at an airport.

The map above shows an airport on an island. The airport has a landing runway and a take-off runway.



Outline

- Introduction to POMDPs
- **Demonstration of POMDPs**
  - **Problem : UAV Navigation**
  - **MDP Visualizations**
    - **Value Iteration**
    - **Policy Execution**
  - **POMDP Visualizations**
- Approximating POMDPs with Macro Actions

This is the MDP portion of the demonstration.

You will see how the UAV navigation problem is mapped to the MDP model.

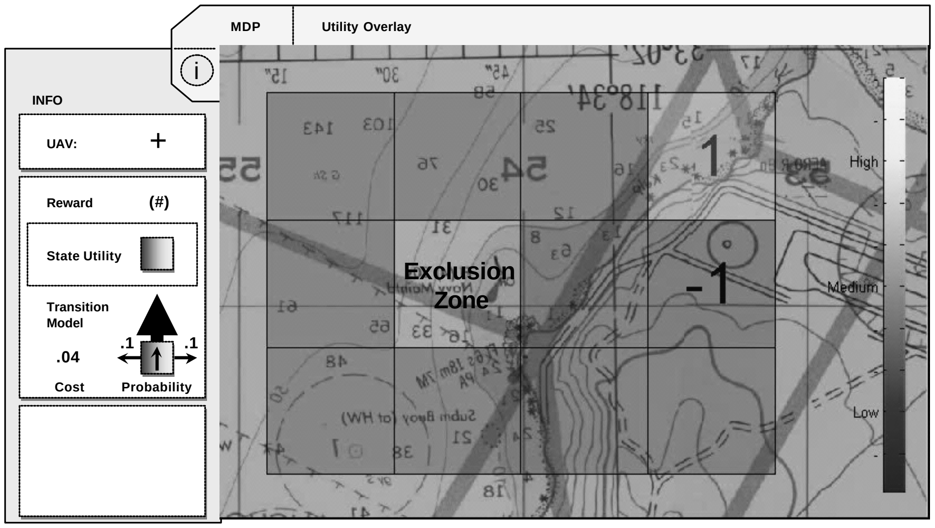
You will then see a visualization of MDP value iteration.

MDP value iteration is the process that MDPs use to find an optimal policy.

You will then see a visualization of the policy being executed.

In our UAV navigation problem, this represents the UAV navigating to the landing runway.

Problem : Utility Overlay



This slide shows the utility overlay for an MDP problem. It shows how to map the UAV problem to an MDP.

As Brian explained, the MDP model assumes that you know what state you are in. For our UAV navigation problem, the state simply represents the UAV's location.

So, each of the red and yellow blocks represent the state that the UAV could be in. There are 11 states that the UAV could be in, each representing a particular location. The color of each state indicates the utility of that state. Yellow indicates high utility and red indicates low utility.

Legend and Chart:

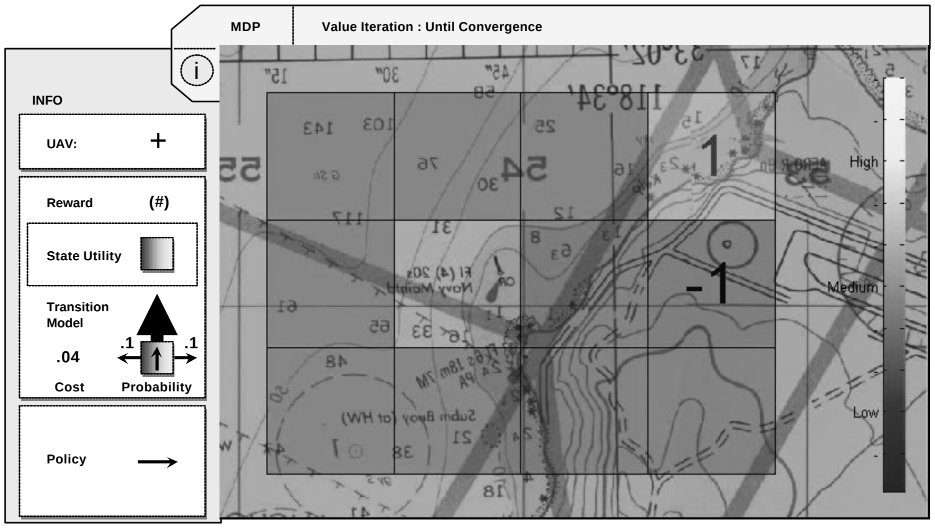
The red + represents the UAV. This will be used in the policy execution phase. The numbers represent the reward. In our model, you receive a reward of +1 for navigating to the landing runway. You receive a reward of -1 (or cost) for ending up on the take-off runway. There is also a transition cost of .04 which is incurred for each action that is taken. This transition cost can also be represented as a negative reward. All of these rewards, collectively, make up the reward matrix.

These rewards, are reflected in the state utility. The chart shows the initial state utilities. The utility of the landing runway, is high (yellow) because you get a reward of 1 if you navigate to it. The take-off runway has a low utility (dark red) because you incur a penalty of 1 if you end up there. The other 9 states have a relatively low utility which reflects the .04 action cost.

Another part of our model is the transition model. Our transition model is probabilistic. In other words, if you take a particular action, you will end up in another state (location, for our problem) probabilistically. For example, if you try to go up, there is a .8 probability that you will end up in the state or block just above you. There is also a small probability (.1) that you will end up in an adjacent state.

One last point is that the "center" block represents an exclusion zone. This simply means that it is not possible to go there.

MDP Value Iteration



This chart shows the value iteration process in action.

Value iteration is the process that an MDP goes through to generate a policy.
It involves looking ahead one step to see which move would generate the highest utility.

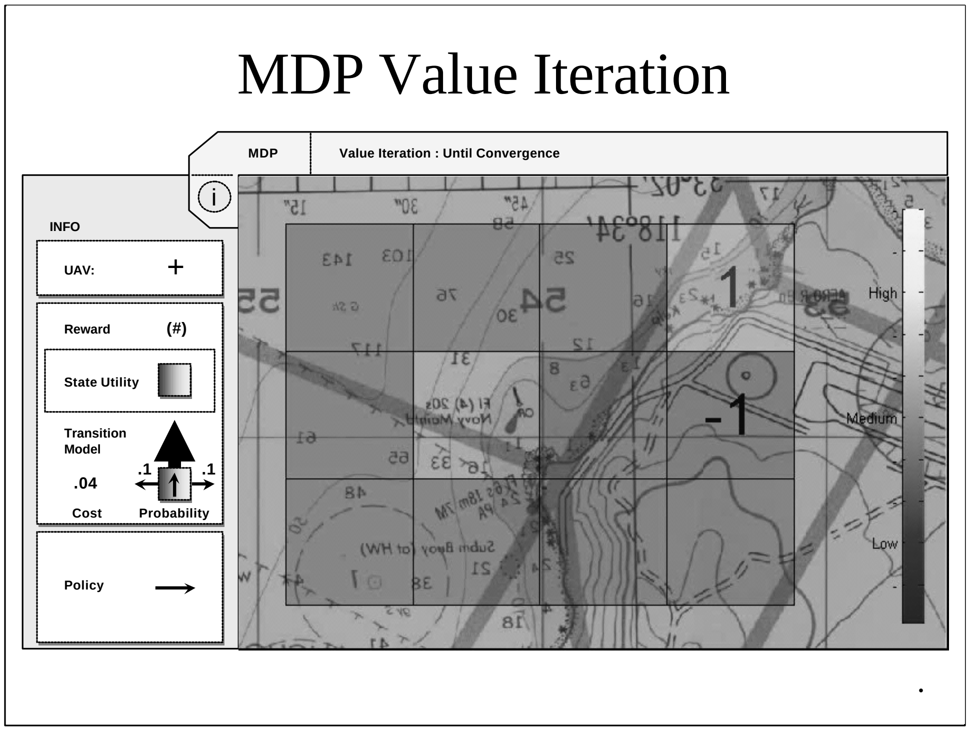
For example, let's say that you (the UAV) are in the location just to the left of the landing runway. The MDP model of the world is able to look ahead one step, and see what reward we get for each of the 4 possible actions. Once it knows what the rewards are, it knows what action is the best. This information is used to set the policy for that state. The policy is set to be that (best) action. In addition to setting the policy, the state also resets the value of its state to reflect the future value of that action.

This process is then repeated. It does this iteratively until the process converges (the utilities stop changing by more than a very small value). An MDP is able to use this iterative process because it assumes that it knows what state it is in. Consequently, an action will generate a probabilistically predicable outcome based on the transition model.

The point is the visualization that you will see is to show you the process in action. When you see it you can try to imagine each state simultaneously looking ahead one step. The visualization should also give you a feel for how the effect of a high utility location (the landing runway) is spread (or conveyed) about the entire state utility model.

You will see the yellow (high utility) state utility in the upper right hand corner be spread toward the lower left-hand corner. As this happens, the policy of each state will be set appropriately. Essentially, the arrows (which indicate the policy) will all point in the general direction of the landing runway.

After the yellow reaches the lower left-hand corner, the MDP has not yet converged. You will also see some of the states at the bottom of the model change their policy.



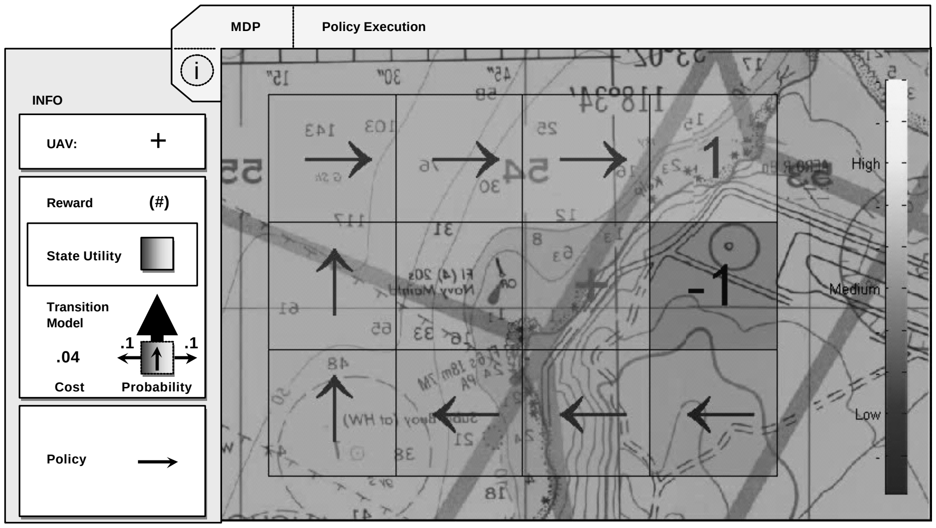
MDP Value Iteration

Animation Slide:

This slide is used to activate the movie.

It looks the same as the previous slide, but it has an embedded movie that runs when the slide is shown.

MDP Policy Execution

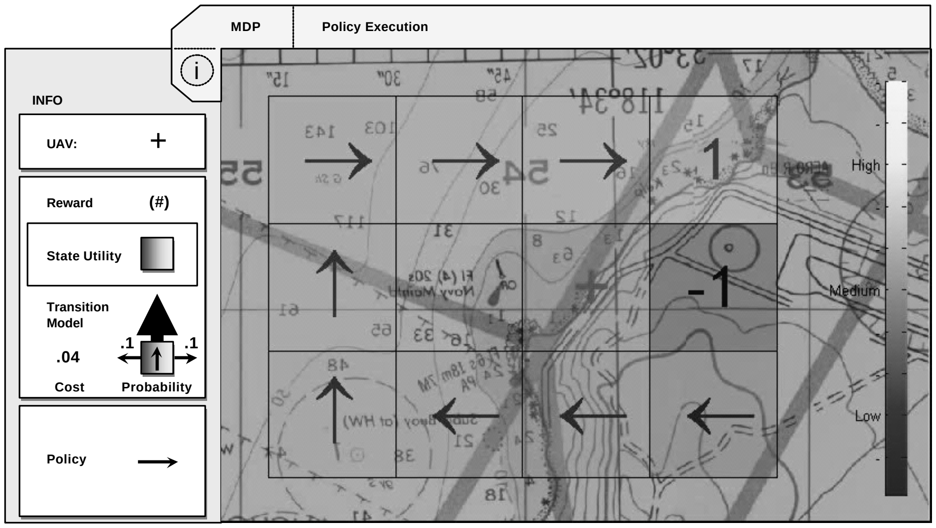


This chart shows the execution of the policy that was discovered using MDP value iteration.

The point of this slide is to show the policy in action and make it clear what is going on / what we have accomplished.

You will see the UAV, represented by the red +, navigate from an initial starting position to the landing runway.

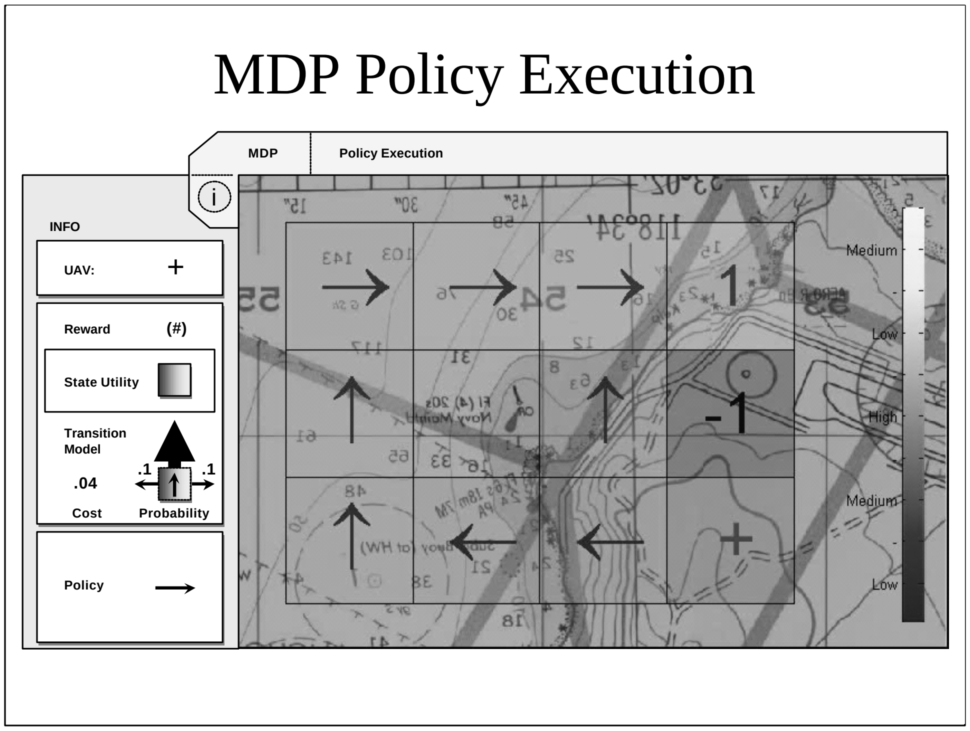
MDP Policy Execution



Animation Slide:

This slide is used to activate the movie.

It looks the same as the previous slide, but it has an embedded movie that runs when the slide is shown.



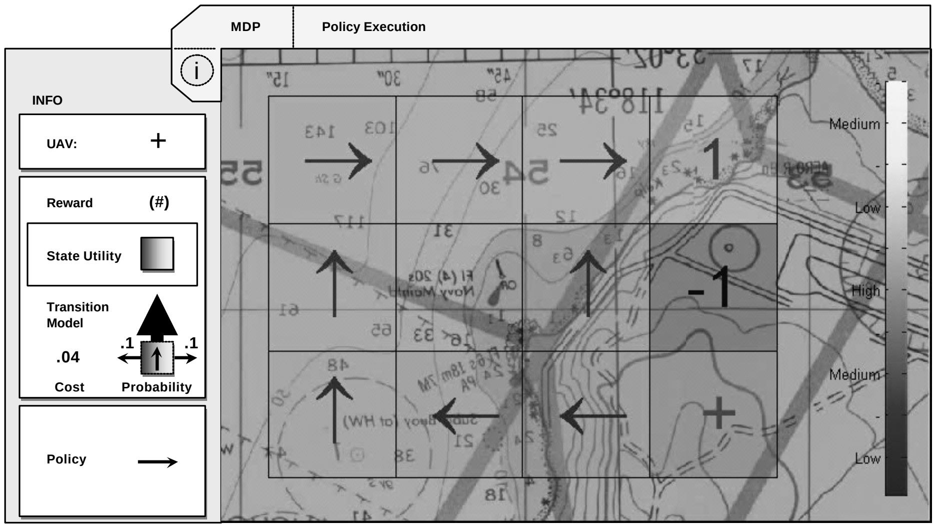
MDP Policy Execution

This chart shows another policy execution. This time, the UAV is starting in a different position.

You will see the UAV, navigate from a different initial starting position to the landing runway. This chart shows the execution of the same policy as before. In this example, the UAV will take a long route to the landing runway, as dictated by the policy.

This brings up an interesting point. You may recall from the value iteration that the last state to change its policy was the state on the bottom, 2nd from the right. It initially was pointing up. That was a shorter route. Eventually it pointed left. This is a longer route. However, it results in a higher expected reward. This is due to the probabilistic transition model. If the UAV went up / then up again, there is a probability that it would end up at the take-off runway, which is bad. Consequently, the UAV takes an alternate circuitous route.

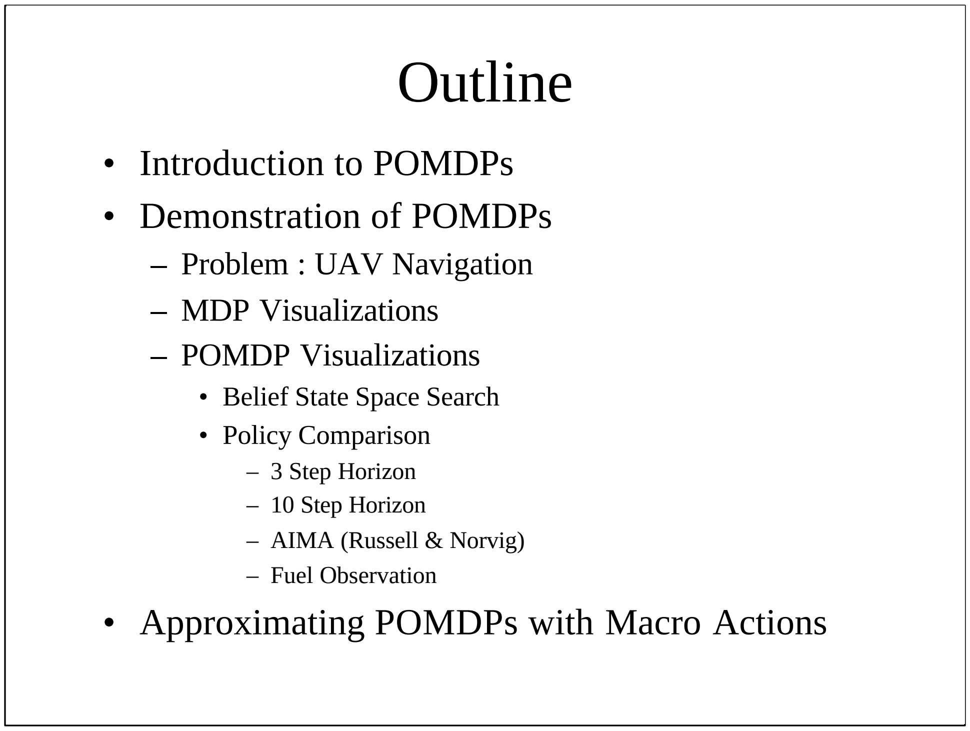
MDP Policy Execution



Animation Slide:

This slide is used to activate the movie.

It looks the same as the previous slide, but it has an embedded movie that runs when the slide is shown.


Outline

- Introduction to POMDPs
- Demonstration of POMDPs
  - Problem : UAV Navigation
  - MDP Visualizations
  - POMDP Visualizations
    - Belief State Space Search
    - Policy Comparison
      - 3 Step Horizon
      - 10 Step Horizon
      - AIMA (Russell & Norvig)
      - Fuel Observation
- Approximating POMDPs with Macro Actions

This is the POMDP portion of the demonstration.

You will see how the UAV navigation problem is mapped to the POMDP model.

You will then see a visualization of the POMDP solution process.

You will then see a visualization of the policy being executed.

The solution process that you will see is a belief state space search.

This is one way to find a policy for the POMDP model.

This method is particularly interesting because it demonstrates directly the complexity of the policy tree, given a finite horizon.

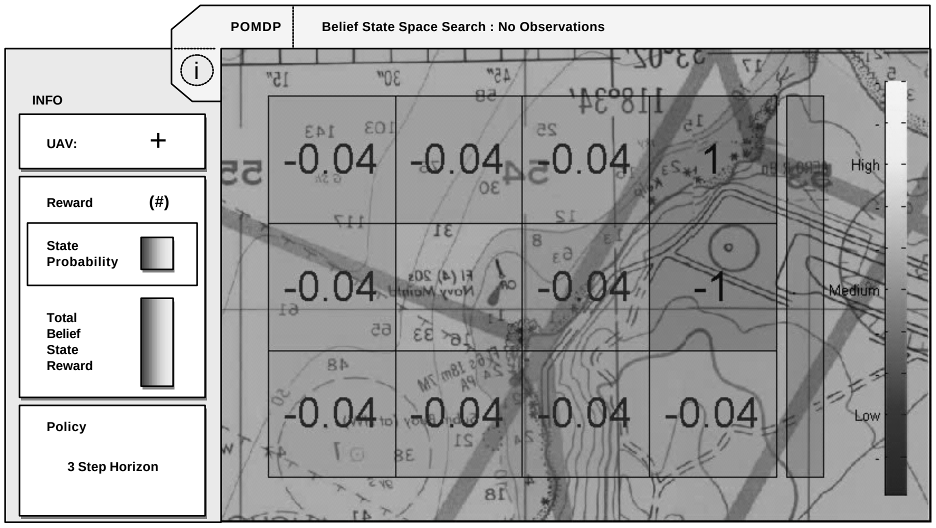
There are also other ways to solve POMDPs, which Brian mentioned.

In addition, this demonstration will provide motivation for Tony’s part, which explains more efficient methods.

In addition to the solution process, you will also see visualizations of the policy execution process.

This visualization is more complex and interesting than the MDP policy execution, due to the fact that POMDPs do not assume that they know where they are in their state space (unlike MDPs).

POMDP Belief State Space Search



POMDP | Belief State Space Search : No Observations

INFO

UAV: +

Reward (#)

State Probability

Total Belief State Reward

Policy

3 Step Horizon

-0.04	-0.04	-0.04	1
-0.04		-0.04	-1
-0.04	-0.04	-0.04	-0.04

POMDP, are different than MDPs, in that they do not assume absolute positional (or state) knowledge. MDPs simply assume positional certainty even though it is an incorrect assumption. MDPs may use, for example, our expected location (MLE). This is an acceptable strategy under low uncertainty, however, it results in bad policies when there is high uncertainty.

For a POMDP, the policy creation is different (than an MDP) because we do not assume absolute positional knowledge. Our position is represented in our belief space. Therefore, in order to asses the value of a given action, we have to asses the value of all of the best possible subsequent actions. When we execute an action, we generate a new belief space given our current belief space and the particular action. Which each action, (for example 'up') we have a probability of ending up in the next block up. There is also a probability of ending up in other blocks. (This was also the case with the MDP.)

However, with the POMDP, we do not assume absolute positional (state) knowledge. Therefore, we must propagate forward the chosen action for each possible belief. If we look ahead 1 time-step, we then get a matrix of values for each possible action. We then multiply our current belief distribution by this matrix to find out which action has the best outcome. When we find out which action has the best outcome, we set our policy to the respective reward.

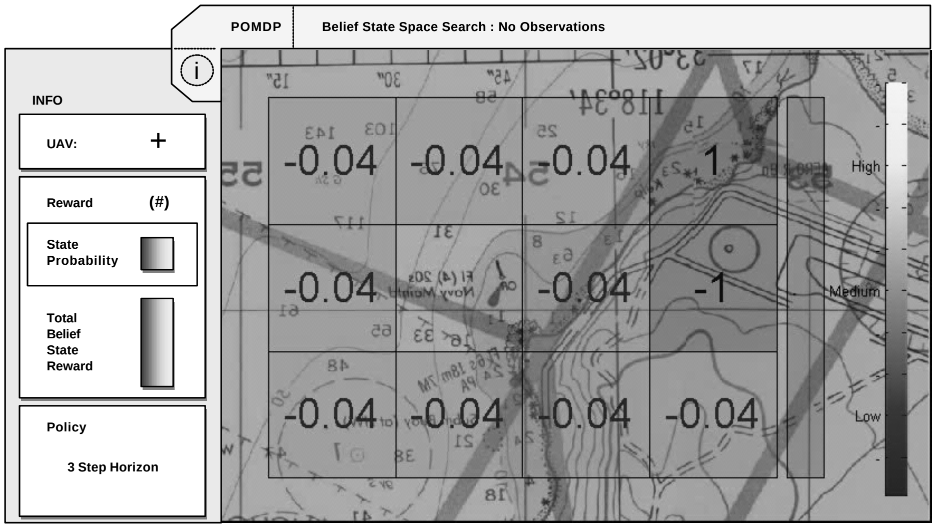
In a 3-step look-ahead / Belief State Space Search, which we will see here, we have to search ahead every combination of action sequences. For the 3 step look-ahead shown here (4 actions), there are 64 combinations which result in 85 possible futures (or beliefs) each with ~11 states. For a 10 step look-ahead, there are 1.4 million possible futures which have to be assessed.

Chart and Legend:

The main difference between the MDP and POMDP representation is that the POMDP does not assume that you know exactly where you are. Therefore, the POMDP represents your location (more generally, your state) probabilistically. For example, there is a .6 probability that I am in state A, and a .4 probability that I am in state B. In the MDP model, we would make a decision based on our expected state. In this case, state A. However, in a POMDP, we represent this as a belief state which assigns a probability to each possible state.

In this chart, the colors of the squares represent something different than they did for the MDP. Here, each square is an individual state. All of them together represent the belief state. The color of the individual states indicates the probability that we are in that individual state. For example, there is an equal probability (of .11) that we are in each of the 9 orange states. Likewise, there is a zero probability, initially, that we are at the landing or take-off runway. Collectively, these represent our belief state.

POMDP Belief State Space Search



(Continued)

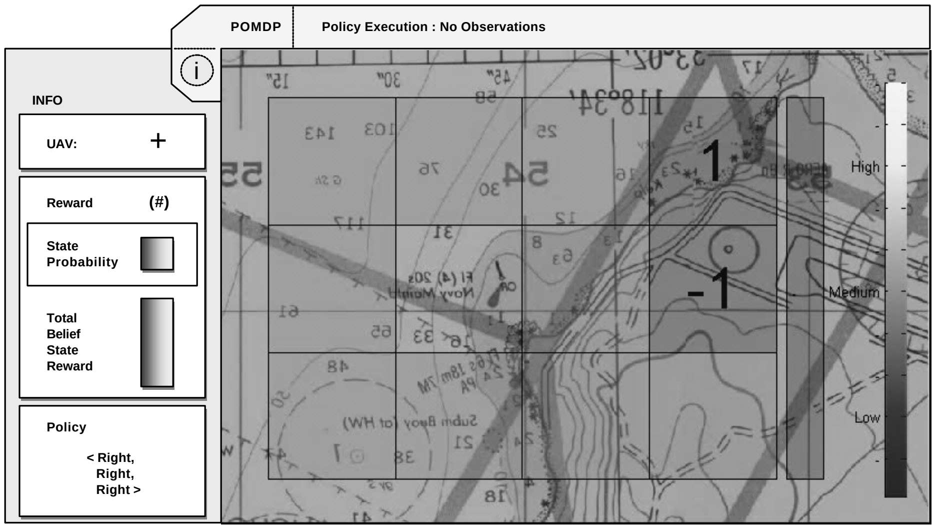
Note that our belief space is continuous, which is a prime reason why POMDPs are difficult to solve. In the MDP model, we created a policy for the 11 possible states. Here we have innumerable belief states. Generating a universal plan (or policy) requires partitioning the belief state into a state-action mapping. In our visualization, you will see a belief state search starting at one particular point in the belief space. That is just one point our of a continuous belief state space (with an infinate number of points). The belief state space search essential explores the policy tree that Brian explained in the previous part of this talk. The size of the policy is exponential in the number of actions taken (depth) and polynomial in the number of possible actions and observations.

The numbers represent the reward matrix.

The color of the rectangle just to the right of the belief state indicates the total current reward for that belief state, which depends on the reward matrix and the current belief state. *The value of a given action depends on that reward and the subsequent actions. However, since we don’t know where we are, the value of a given action depends on the combined value of that action for each individual state in our belief state. If we propagate forward a chosen action we get a matrix of values for that action-belief combination. That is calculated using the new belief state and the reward matrix. *Doing this for each possible action, will tell us which action is best, given our belief state – it is the action that generates the highest future value, summed over the whole belief state. Recursive Process - Propagated forward 3 steps -- at each step, you would generate 4 new belief states, 1 for each action. Each belief state has 11 individual states, with an associated probability. At the 3rd step you would have 64 different belief states, 1 for each 3-step action sequence. For a 10 step look-ahead, there are 1.4 million different action sequences. This visualization shows this process for 3 steps.

Animation Slide: This slide is used to activate the movie. It looks the same as the previous slide, but it has an embedded movie that runs when the slide is shown.

POMDP Policy (3 Iterations)



This chart shows the execution of the policy that was discovered using the 3 step POMDP belief state space search. Specifically, this slide demonstrates the 3 step policy: <Right, Right, Right>. The point of this slide is to show this policy in action.

Since the POMDP represents the UAV location as a belief state, the location of the UAV is captured by the color of the individual states shown in the movie.

Again, for the POMDP visualizations, the colors of the squares represent the probability that we are in that individual state. Initially, there is an equal probability (of .11) that we are in each of the 9 orange states.

Collectively, these represent our belief state.

In the movie, we will execute an action, and generate a new belief state. With each action, ('right') we have a .8 probability of moving to that block as well as a small probability of ending up in another block. (This was also the case with the MDP.) However, with the POMDP, we do not assume absolute positional knowledge, thus, our action generates a new probabilistic belief state.

So, what you will see in this movie, is the higher probabilities move toward the right. The higher probabilities are indicated by yellow, and the lower probabilities are indicated by the red. We initially don't know very accurately where we are. As we execute this policy, we will have a better idea of where we are.

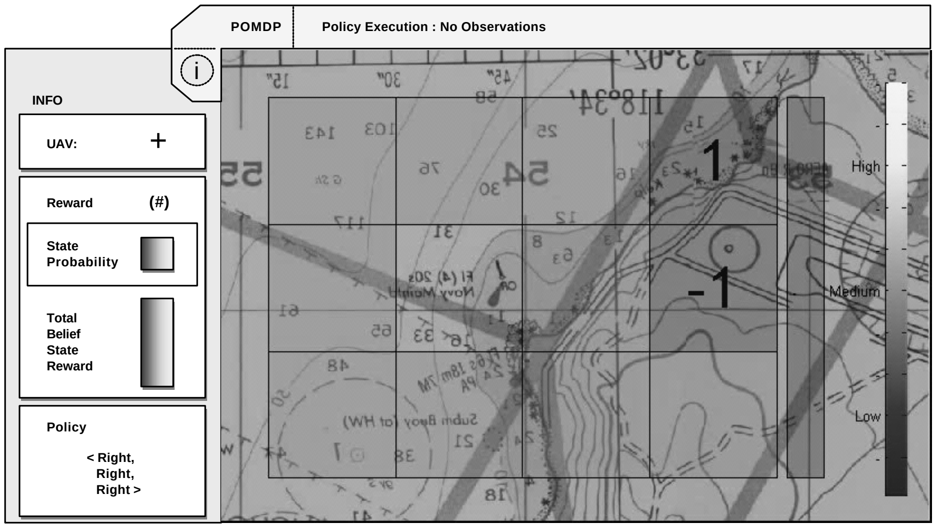
There is one more detail about this representation. The reward incurred by landing on the landing runway should only happen once. The same holds for the take-off runway.

For example, we wouldn't want to reward a plane for sitting on the landing runway for multiple steps. Therefore, the POMDP is modeled such that after the plane lands on either runway, it goes to a sink on the next time-step. A sink is a state that has no reward or penalty. It's transition model dictates that there is a zero probability that you can leave the state.

You may notice the above detail in the visualization. The landing runway will become yellow, indicating a high probability that the UAV just landed there.

This may be followed by the block becoming more red, indicating a low probability of the UAV being there. This is due to the sink model.

POMDP Policy (3 Iterations)



Please watch the video.

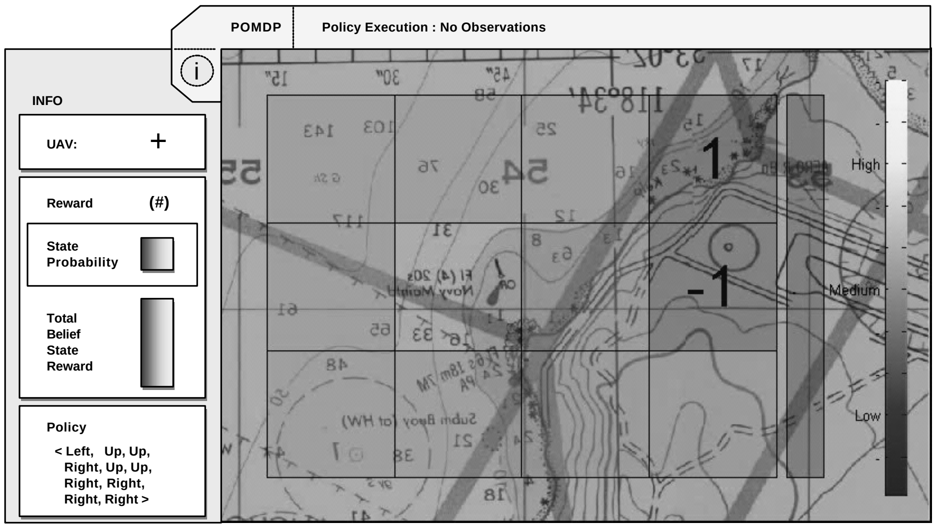
The policy shown here isn’t very good, because it is very likely that we end up in the bottom right corner, or on the take-off runway. You may also notice that a good portion of our belief state got “stuck” behind the exclusion zone. This is because our look-ahead was too short.

In the next slide, we will see the policy that is generated from a 10-step look-ahead.

Animation Slide:

This slide is used to activate the movie.
It looks the same as the previous slide, but it has an embedded movie that runs when the slide is shown.

POMDP Policy (10 Iterations)



Policy

< Left, Up, Up, Right, Up, Up, Right, Right, Right, Right >

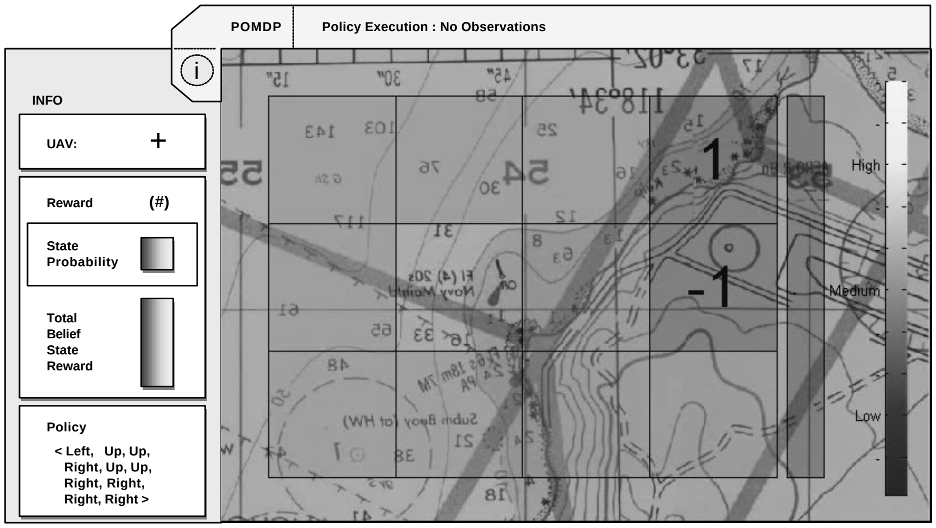
This slide shows the execution of a policy generated from a 10-step look-ahead. Remember that we haven't discussed observations yet. Therefore, this policy assumes an initial position distribution. Our belief space currently only represents our position distribution.

Our action cost is relatively low here. Therefore, the policy is to move left first. This gets us out of the lower right hand corner (if we were there, we don't know exactly).

The we go up… < Left, Up, Up, Right, Up, Up, Right, Right, Right, Right >

With each action, we gain positional accuracy. You can see this from the color scheme. The color scheme indicates the probability that we are in a given location. It is initially uniform for the 9 possible starting positions. It then gets more yellow to the left. The Yellow high probability trail then makes it's way up and over to the runway. The policy exploits the bounded-ness of the grid (we can't go too far to the left) to gain positional accuracy, and generate a better policy. This policy causes us to end up at the desired destination with high probability. Note that the representation has a sink. Once we hit the runway, we go to the sink in the next iteration. That is why the yellow disappears from the runway after it arrives. Note that the yellow high probability disappears from the grid. This problem is the exact same problem represented in the Russel and Norvig (AIMA) book. The book gives a different policy for this POMDP with no observations. The policy we generated is better, as I will show you.

POMDP Policy (10 Iterations)



Policy

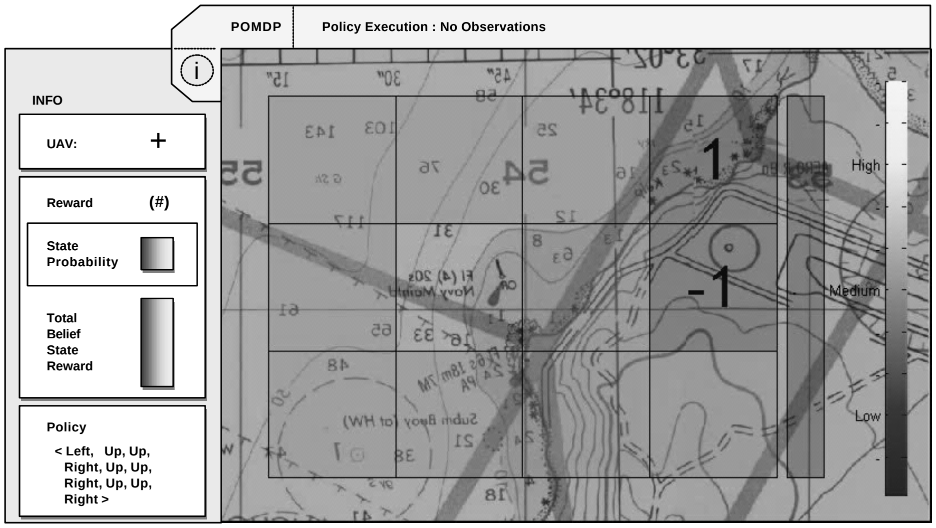
< Left, Up, Up, Right, Up, Up, Right, Right, Right, Right >

Animation Slide:

This slide is used to activate the movie.

It looks the same as the previous slide, but it has an embedded movie that runs when the slide is shown.

POMDP Policy (AIMA)



Policy

< Left, Up, Up, Right, Up, Up, Right, Up, Up, Right >

This visualization shows another interesting policy.

You may recognize this problem for the AIMA / Russell and Norvig book.

In that book, they give an optimal policy that is different (though similar) from the policy that I just demonstrated.

This movie shows that policy being executed for 10 steps.

The policy is as follows:

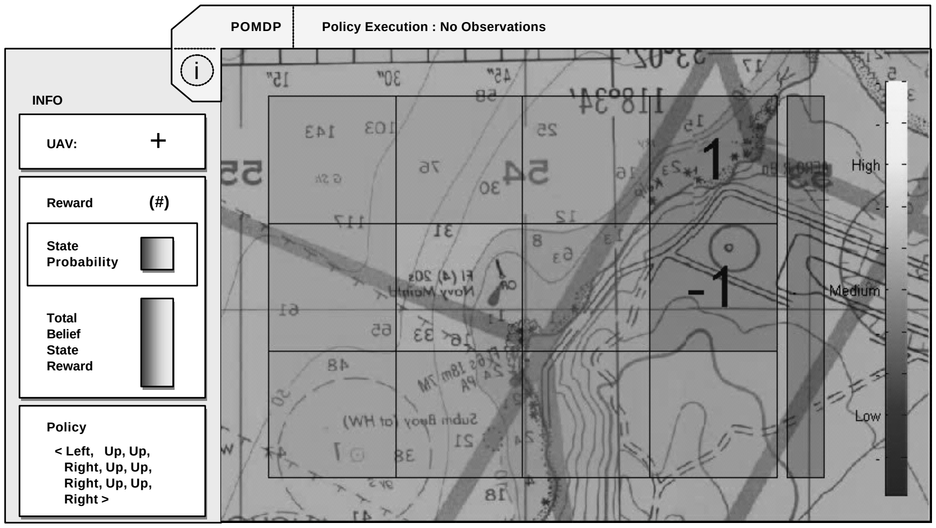
AIMA < Left, Up, Up, Right, Up, Up, Right, Up, Up, Right >

After running this policy for 10 steps, you may notice that it leaves some yellow on the field at the end. This means that there is a high probability that the UAV did not make it to the landing strip, in 10 steps, under this policy.

In other words, this policy is not as effective as the previous policy for 10 steps.

This is because the AIMA policy is oriented toward a longer planning horizon.

POMDP Policy (AIMA)



Policy

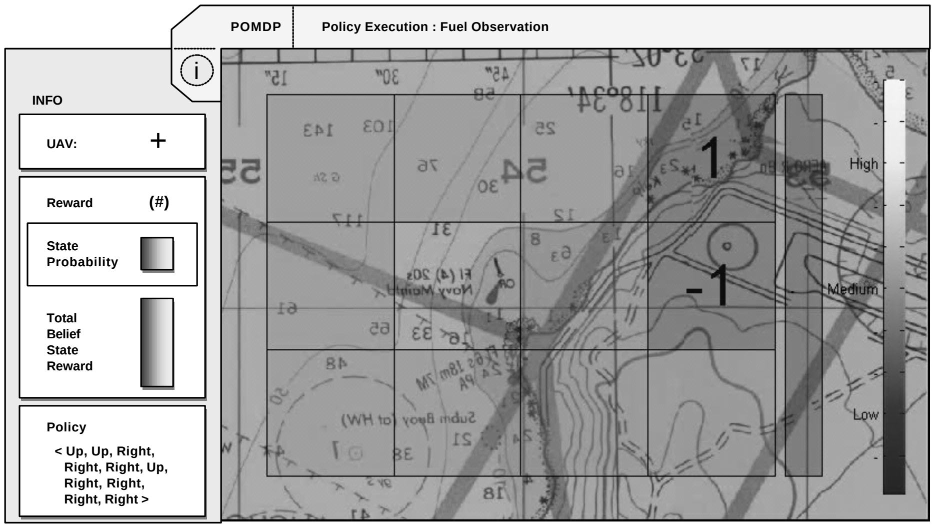
< Left, Up, Up,
Right, Up, Up,
Right, Up, Up,
Right >

Animation Slide:

This slide is used to activate the movie.

It looks the same as the previous slide, but it has an embedded movie that runs when the slide is shown.

POMDP Policy (Fuel Observation)


POMDP
Policy Execution : Fuel Observation
INFO
UAV:
Reward
(#)
State Probability
Total Belief State Reward
High
Medium
Low
1
-1

Policy

< Up, Up, Right, Right, Right, Up, Right, Right, Right, Right >

Next we introduce how observations play into POMDPs.

Let's say you have a fuel gauge and it is telling us that we are low on fuel. This will cause us to take a shorter route home, even if it has additional risk.

This causes the policy to be more risky, we may land on the take-off runway. However, our overall risk is minimized.

The policy generated here assumes we are low on fuel, which makes each action (relatively) more costly. This requires a more complex belief state space which captures our belief about how much fuel we have. This also requires (for this problem) an augmented reward matrix which covers all of these individual belief states.

This slide shows that policy in action: < Up, Up, Right, Right, Right, Up, Right, Right, Right, Right >

You can see that under this policy, it is more likely that the UAV lands on the take-off runway. However, this risky policy optimizes the risk-reward trade-off.

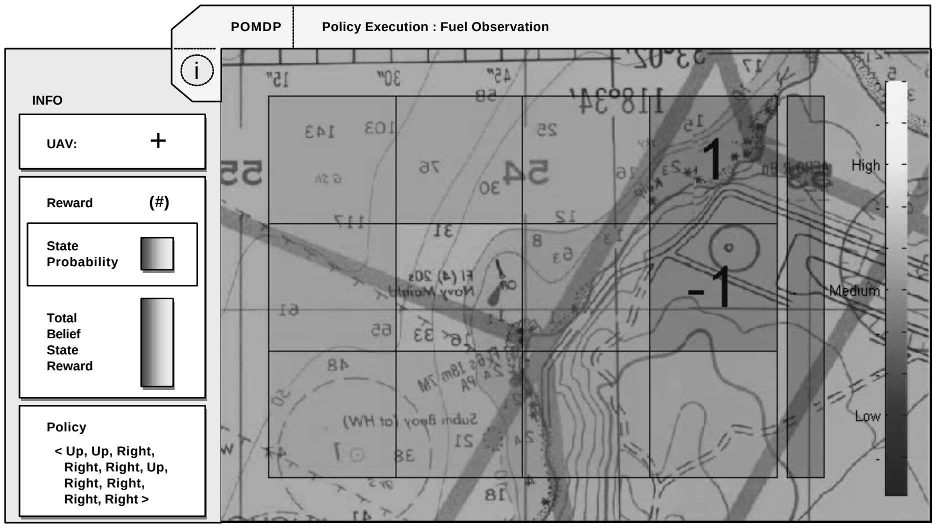
Summary of the complexity of POMDPs in the context of this example:

POMDPs are difficult to solve due to the combination of the belief state space size, and the number of possible action-observation sequences.

Imagine if our fuel gauge blinks on and off (sound familiar?). Then combine this observation with several other possible types of observations, for example, using landmarks to reduce your position uncertainty. This increases the complexity of the policy tree, and thus the solution process.

This is why POMDPs are so difficult to solve. Consequently, efficient methods for solving POMDPs is an area of active research.

POMDP Policy (Fuel Observation)



POMDP

Policy Execution : Fuel Observation

INFO

UAV: +

Reward (#)

State Probability

Total Belief State Reward

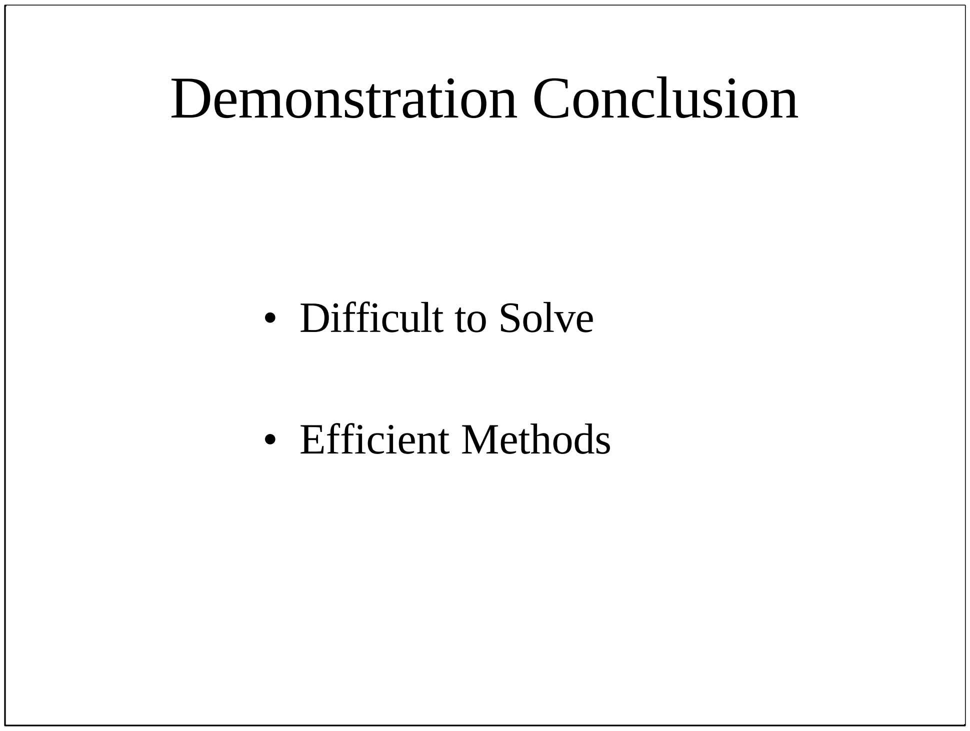
Policy

< Up, Up, Right, Right, Right, Up, Right, Right, Right, Right >

Animation Slide:

This slide is used to activate the movie.

It looks the same as the previous slide, but it has an embedded movie that runs when the slide is shown.



Demonstration Conclusion

- Difficult to Solve
- Efficient Methods

2 key points that I want you to take away from this demonstration is the essence of how POMDPs work, and the complexity of solving them.

The visualizations that you have seen demonstrate the nature of the policy creation process. In particular, they show what it is that you want to explore, and how complex this space is.

In particular, due to the continuous belief state space, POMDPs are difficult to solve. Furthermore, the complexity of the policy tree, explored in the belief state space search adds to the problem.

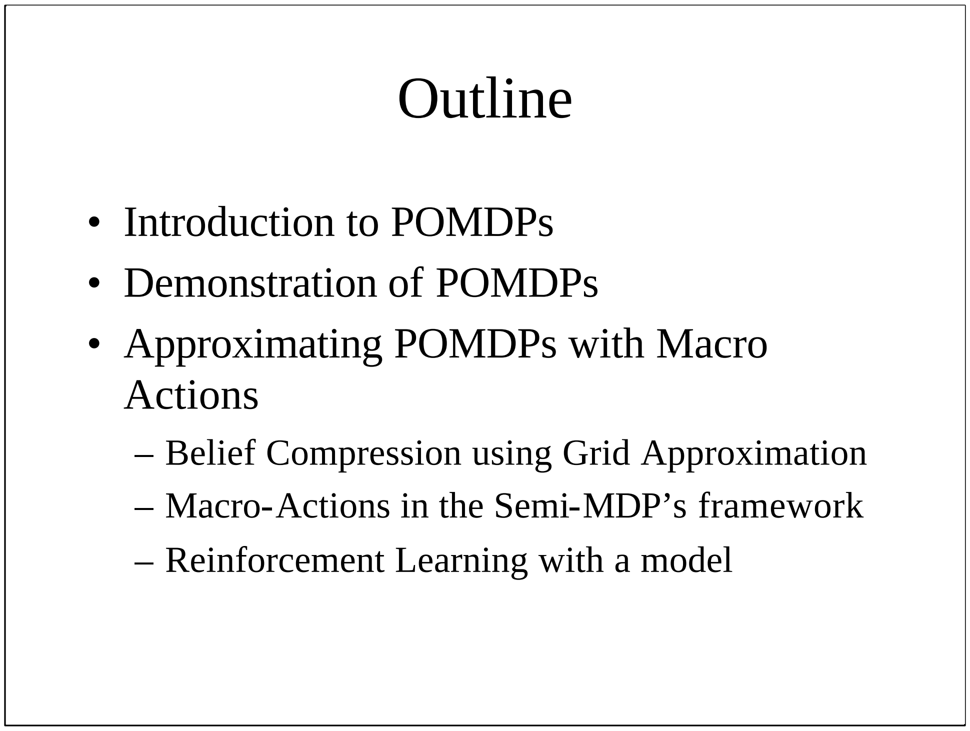
Specifically:

•The POMDP has a Continuous Belief State Space.

•The POMDP's Policy Tree is polynomial in the number of actions and observations and exponential in the length of the planning horizon.

•For every level of the policy tree, there are m^j*n^j possible action/observation sequences.

It is important to realize that just exploring the policy tree, given a particular belief, on a small problem, is somewhat complex. Creating a universal plan for a large problem is incredibly complex. These issues lead to the need for efficient methods for computing policies, which is the topic of Tony's section of our talk.

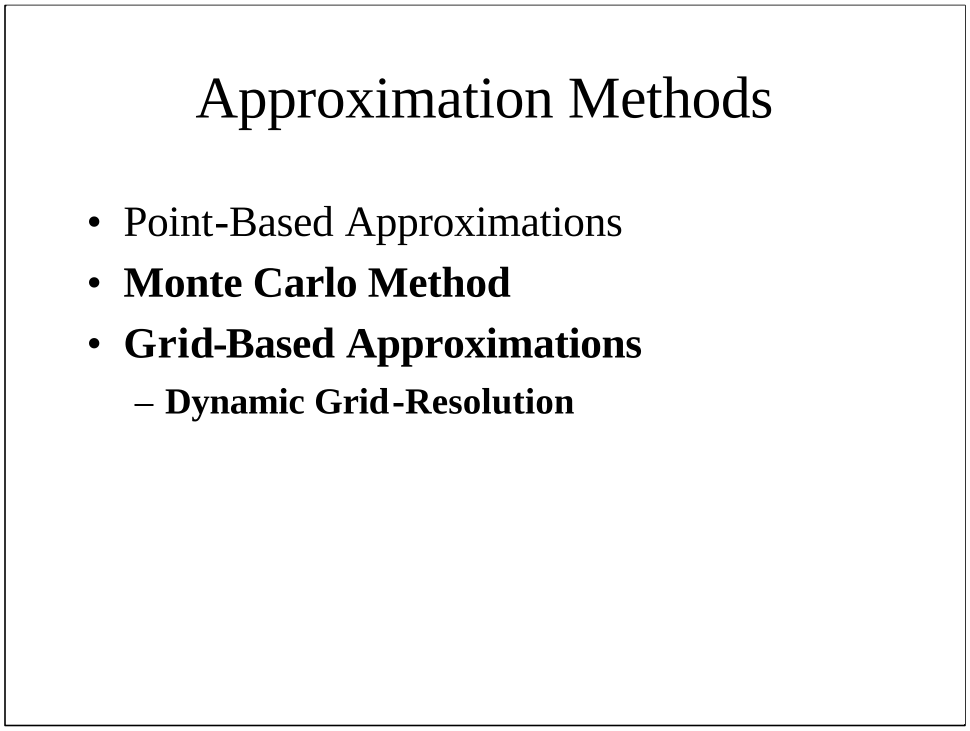


Outline

- Introduction to POMDPs
- Demonstration of POMDPs
- Approximating POMDPs with Macro Actions
  - Belief Compression using Grid Approximation
  - Macro-Actions in the Semi-MDP's framework
  - Reinforcement Learning with a model

The previous demo simplified the problem into 12 states. However, this was a gross simplification of the problem, but trying to solve POMDPs exactly limits us computationally to the number of states on this order. To solve a more realistic navigational problem with over 1000 states will require different methods.

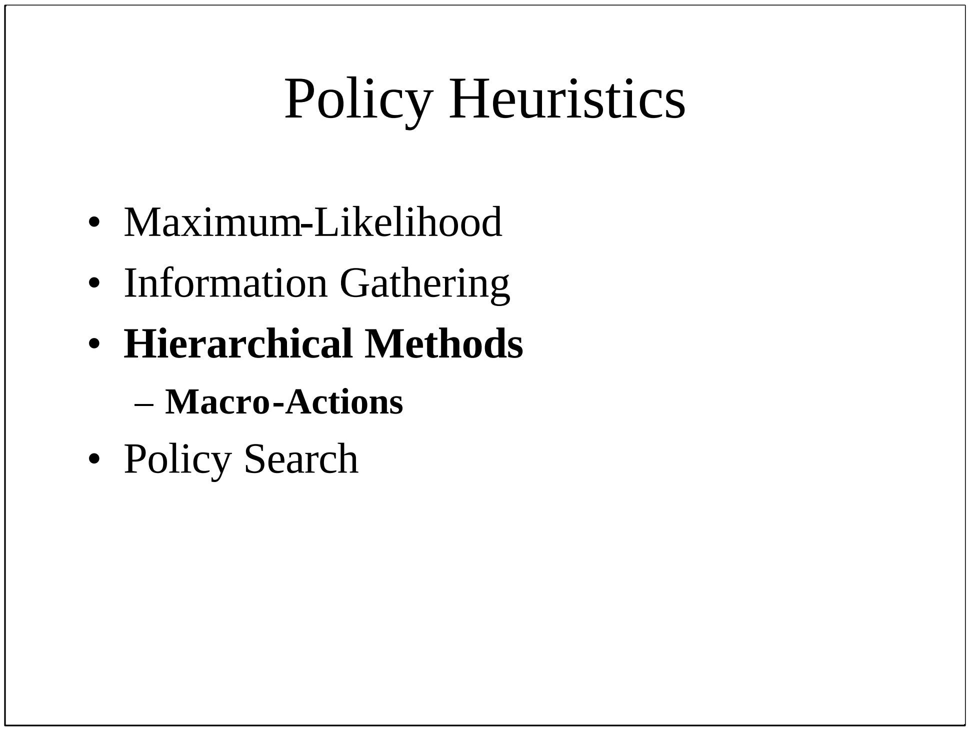
The algorithm from "Approximating POMDPs with Macro Actions" by Georgios Theocharous and Leslie Kaelbling will be presented next. To introduce this, some foundational methods are first presented. What will be covered first is "Belief Compression using Grid Approximation" then "Macro-Actions in the Semi-MDP framework" before finally going into the actual algorithm in "Reinforcement Learning with a model".



Approximation Methods

- Point-Based Approximations
- **Monte Carlo Method**
- **Grid-Based Approximations**
  - **Dynamic Grid-Resolution**

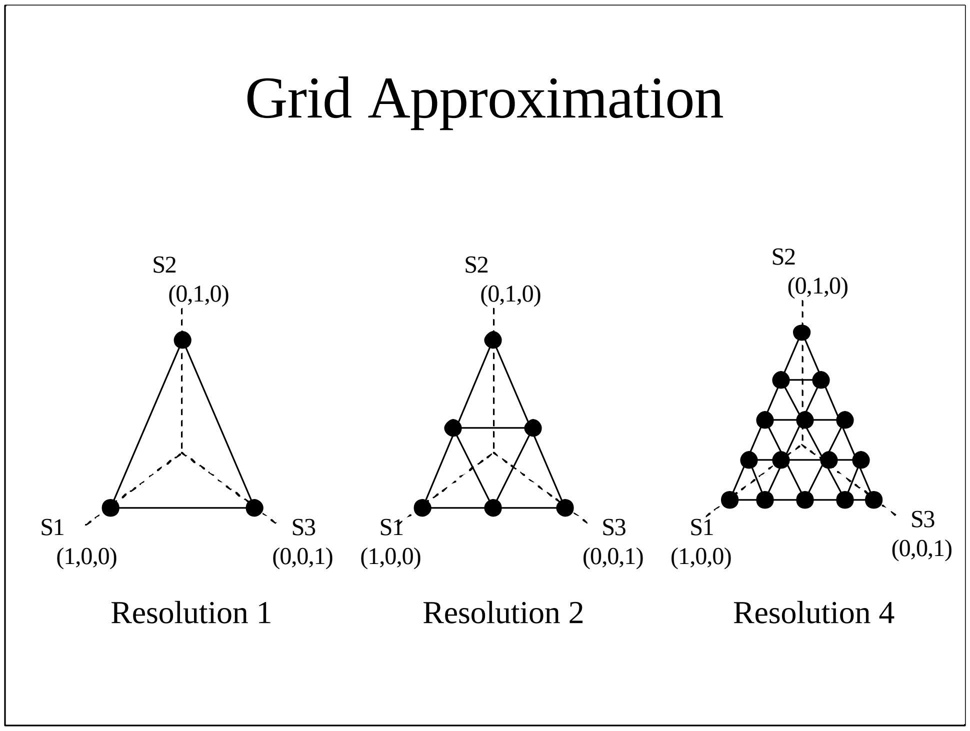
This is a review of the previous approximation methods that were presented in the first part of the lecture. Of these, the algorithm will use a Monte Carlo method and a Grid-based approximation. More specifically, it will use "Dynamic Grid-Resolution". This will be covered in more detail in a few slides.



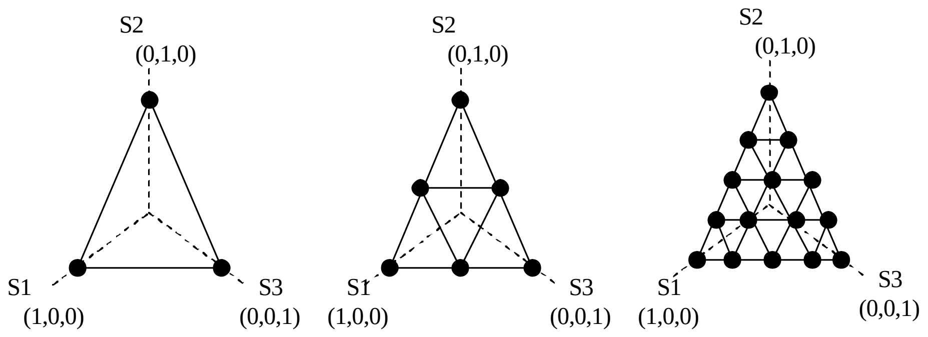
Policy Heuristics

- Maximum-Likelihood
- Information Gathering
- **Hierarchical Methods**
  - **Macro-Actions**
- Policy Search

Of the policy heuristics mentioned, Macro-Actions will be used in addition to the methods from the previous slide.



Grid Approximation



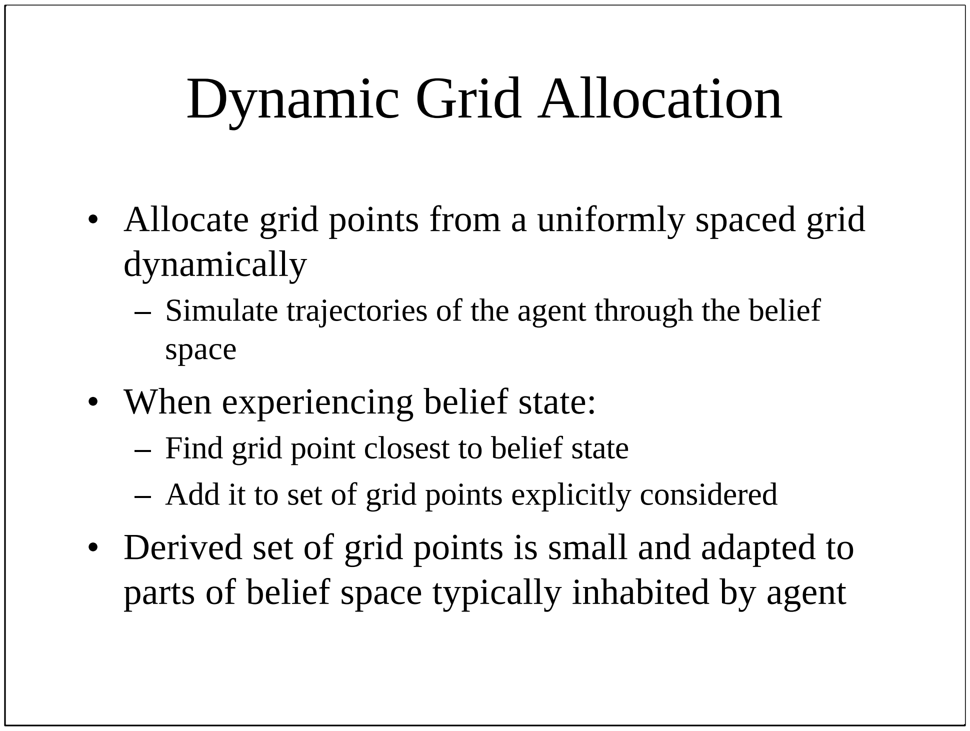
Resolution 1 Resolution 2 Resolution 4

This slide demonstrates the discretization of a 3-dimensional belief simplex. Each point on the graph represents a belief state, which is a probability distribution across the states. In this example, we have 3 states, S1, S2, and S3. So a 100% certain belief that we are in S1 would be represented in this example as (1,0,0), whereas a certain belief that we are in S2 would be (0,1,0). Certain belief for S3 would be (0,0,1). You can see these points on the left-most graph. If we had an equal belief that we are in either S1 or S2, that would be represented as (0.5,0.5,0). This point on the graph would be on the line halfway between S1 and S2. As you can see, a probability distribution across the states will sum to 1. Because of this, every point in the belief space will on the plane formed by the lines between each of the states. This plane is trimmed to a triangle.

This surface of the triangle is referred to as a simplex. This is the belief space of the original POMDP.

If we increase the resolution to 2, we divide the lines into 2 parts, and connect the new grid points to create smaller triangles. These smaller triangles are called sub-simplexes. The algorithm was tested up to a resolution of 4.

For a given belief state in a POMDP, you can discretize it to its nearest grid point. The new grid points become states in an MDP problem. The key idea here is that it is possible to approximate solutions to a POMDP by solving an MDP that takes the grid points as states.

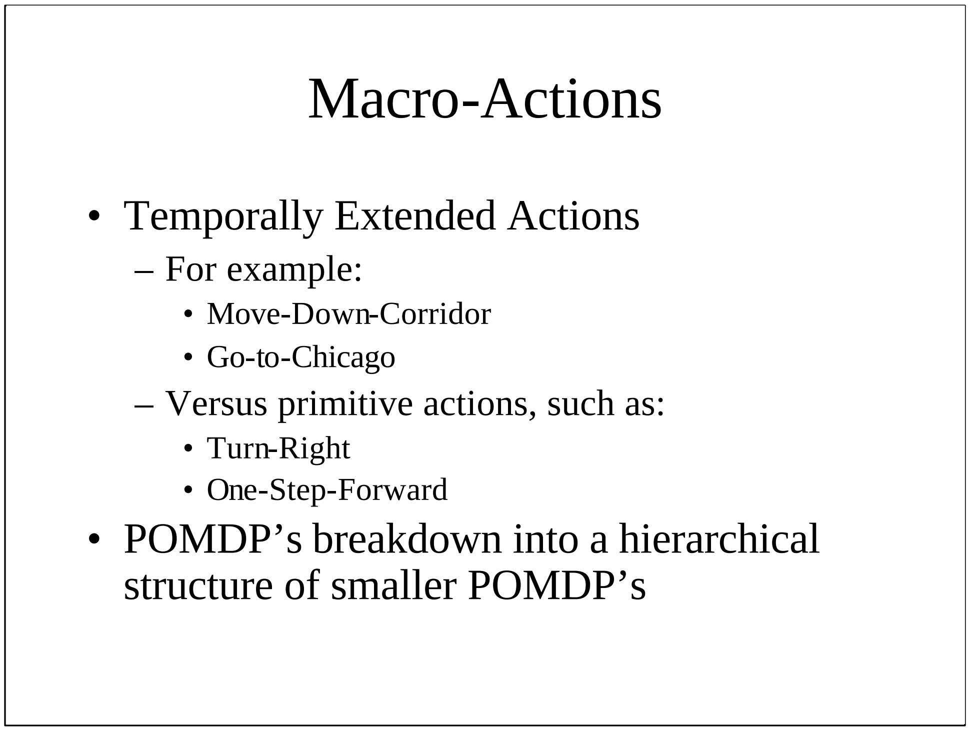


Dynamic Grid Allocation

- Allocate grid points from a uniformly spaced grid dynamically
  - Simulate trajectories of the agent through the belief space
- When experiencing belief state:
  - Find grid point closest to belief state
  - Add it to set of grid points explicitly considered
- Derived set of grid points is small and adapted to parts of belief space typically inhabited by agent

The more states the problem has, the more dimensions there are in the belief simplexes. The number of grid points quickly becomes large in a uniformly spaced grid. Instead of using a variable resolution method, the algorithm will use dynamic allocation. It does this by simulating the trajectories of the agent through the belief space. So when a belief state is experienced, the closest grid point to the belief state is used and added to the set of grid points that are explicitly considered. In other words, instead of allocating the memory for all the grid points initially, it is only allocated when the belief state is experienced. Memory is needed to store the state action value and the state value of the grid-point.

Using this method, the derived set of grid points is small and will cover the belief space that is typically inhabited by the agent.



Macro-Actions

- Temporally Extended Actions
  - For example:
    - Move-Down-Corridor
    - Go-to-Chicago
  - Versus primitive actions, such as:
    - Turn-Right
    - One-Step-Forward
- POMDP’s breakdown into a hierarchical structure of smaller POMDP’s

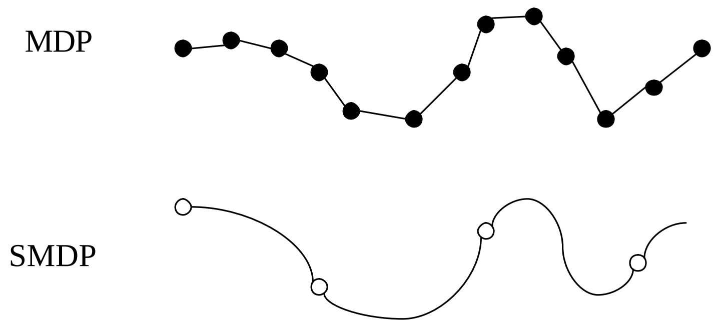
Or “time-extended” actions.

Some examples of macro-actions are given here. These are actions that will take an extended amount of time, versus primitive actions such turn-right or one-step-forward. Macro-actions can take the form of a POMDP policy to move down a corridor. The top-level POMDP can issue the macro-action to move down the corridor and not have to explicitly deal with the smaller POMDP of navigating the corridor and not hitting any walls.

Good macro-actions are ones that will allow us to localize and reduce the uncertainty of our belief state.

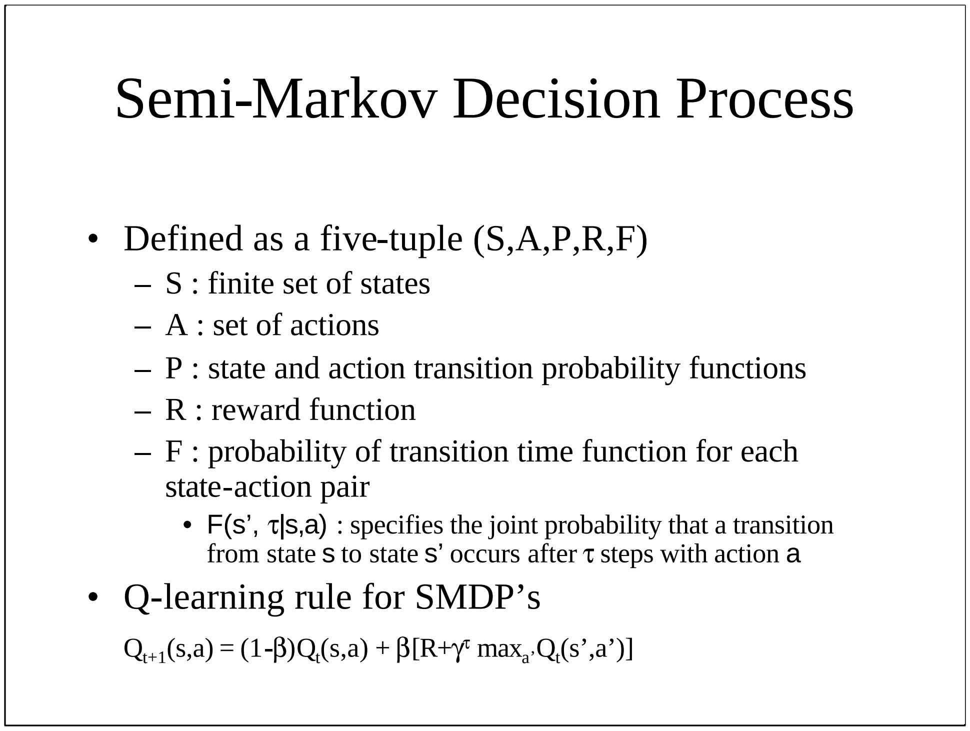
To recap, the hierarchical structure grows polynomially, so we get a polynomially-growing structure of smaller problems that will grow exponentially. This is an improvement over the otherwise exponential growth of the total problem.

Macro-Actions



This is a graphical representation of how primitive actions compare to macro-actions in an MDP. While primitive actions cause the MDP to step through each state, using macro-actions across longer lengths of time, we are able to reduce the complexity of the problem.

To model Macro-actions in a POMDP framework, we use SMDPs, which are covered next.



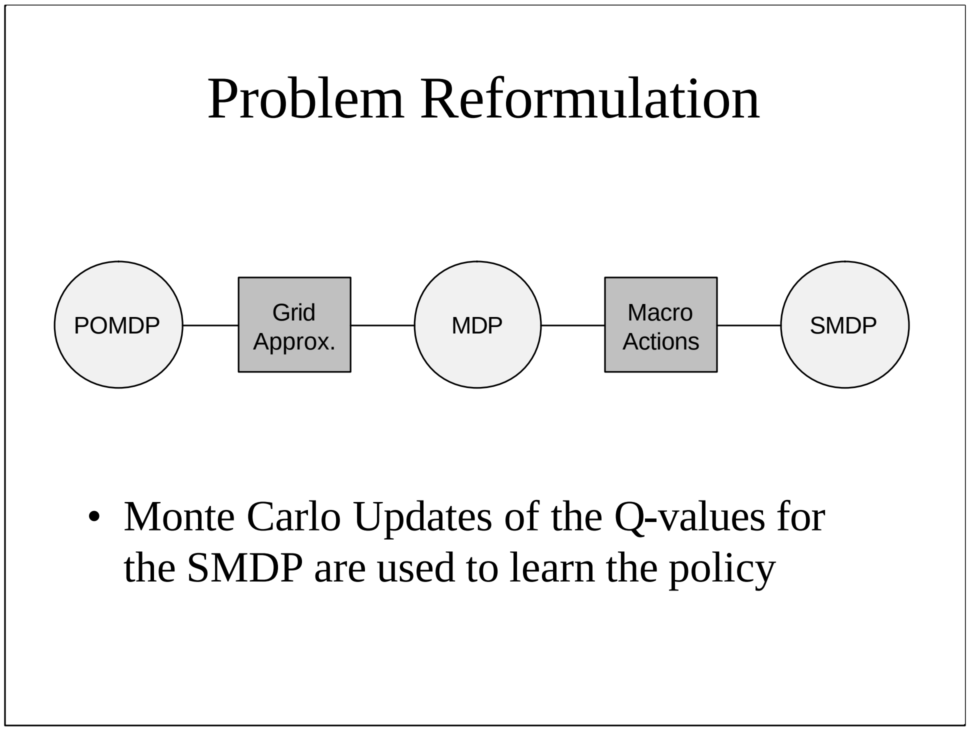
Semi-Markov Decision Process

- Defined as a five-tuple (S,A,P,R,F)
  - S : finite set of states
  - A : set of actions
  - P : state and action transition probability functions
  - R : reward function
  - F : probability of transition time function for each state-action pair
    - $F(s', \tau|s,a)$: specifies the joint probability that a transition from state s to state s' occurs after τ steps with action a
- Q-learning rule for SMDP's

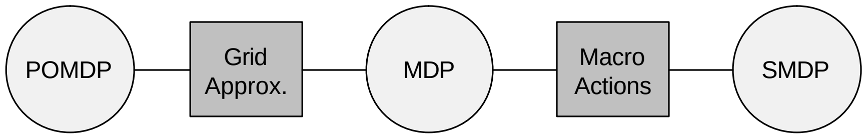
$Q_{t+1}(s,a) = (1-\beta)Q_t(s,a) + \beta[R+\gamma^{\tau} \max_{a'} Q_t(s',a')]$

SMDPs are defined with 5 components. S,A,P, and R are the same type of components found in a regular MDP, but what is new here is F. This is the function that gives the probability of the transition time for each state-action pair. For a discrete time SMDP, this is represented as F(s', tau|s,a).

The Q-learning rule for SMDPs is very similar to the Q-learning rule for regular MDPs that was presented earlier in the lecture. The only difference here is that the discount factor is raised to the tau. As a reminder, this discount factor reduces the rewards of future state-action values. It also causes infinite horizon values to converge.



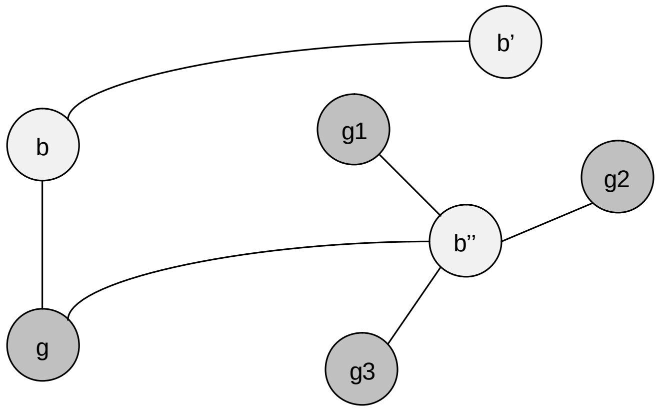
Problem Reformulation



- Monte Carlo Updates of the Q-values for the SMDP are used to learn the policy

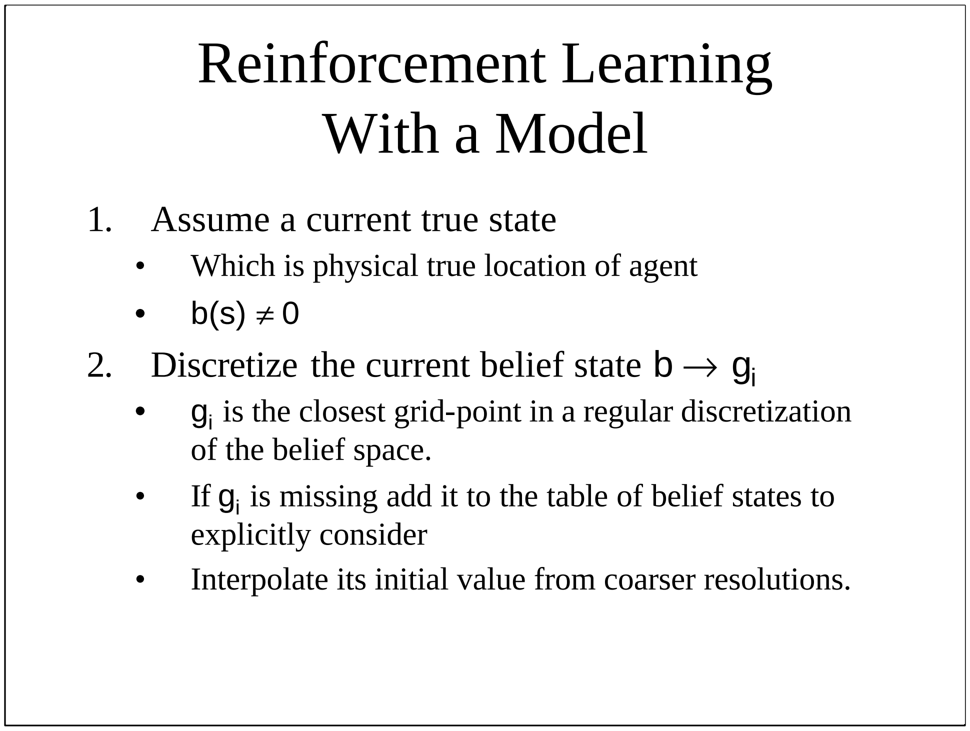
So with these foundational methods presented, here is an overview of how the algorithm will simplify the problem. The original POMDP is converted into an MDP using dynamic grid approximation. This MDP is abstractly converted into an SMDP through the use of macro-actions. Finally, the algorithm will use Monte Carlo updates of the Q-values for the SMDP to quickly learn a policy solution.

Graphical Overview of the Algorithm



This slide shows a graphical overview of the presented algorithm.

b is discretized to g, the nearest grid point to b. Then a macro-action is simulated from g to b’’. At b’’, the Q-value is interpolated from the vertices of the sub-simplex of which b’’ is located. The macro-action is simulated repeatedly to estimate the expected value function of b’’. The state-action value of g is then updated with this expected value function. Finally, the macro-action is executed on state b and b is moved to b’.



Reinforcement Learning With a Model

1. Assume a current true state
   - Which is physical true location of agent
   - $b(s) \neq 0$
2. Discretize the current belief state $b \rightarrow g_i$
   - g_i is the closest grid-point in a regular discretization of the belief space.
   - If g_i is missing add it to the table of belief states to explicitly consider
   - Interpolate its initial value from coarser resolutions.

Reinforcement learning with a model is otherwise called Real Time Dynamic Programming (RTDP)

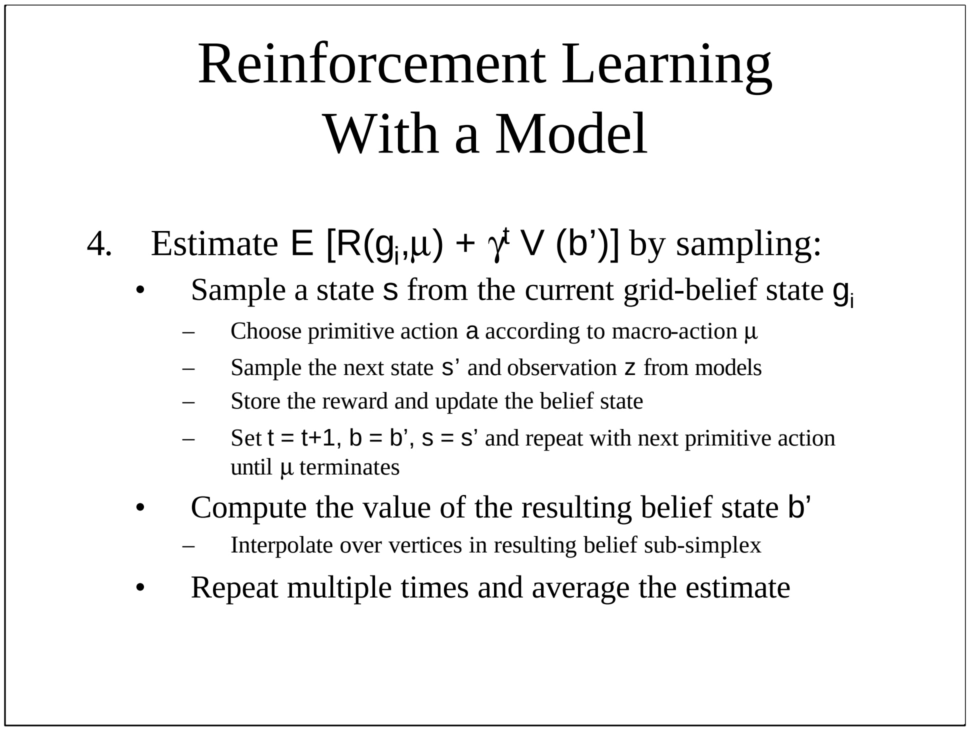
To begin in the algorithm, we start with a current true state, which is the physical true location of the agent in this navigational problem. This means that the belief state cannot equal zero. The model has to believe that we can be located at that state.

In step 2, we discretize the current belief state to its nearest grid point. If that grid point is missing, we add it to the table of belief states that we are explicitly considering. To do this, we allocate memory for the state-action value and the state value for the belief state that this grid point represents. To get its initial value, we interpolate it from coarser resolutions. If the resolution is 1, meaning that there are no coarser resolutions, we initialize gi's value to zero.

Reinforcement Learning With a Model

3. Choose the best macro-action from the current Q values
   - Random action ε% of the time
   - Interpolate over the Q values of the vertices of the sub-simplex that contains b

We choose the best macro-action from the current Q-values of the belief state. However, we need to balance exploitation with exploration, so we will choose a random action a set percentage of the time. To get the Q-values of the belief state, we interpolate over the Q-values of the vertices of the sub-simplex that contains b. In the 3-state example, this is the smaller triangle that b is located in.



Reinforcement Learning With a Model

4. Estimate $E\,[R(g_i,\mu) + \gamma^t\, V\,(b')]$ by sampling:
   - Sample a state s from the current grid-belief state g_i
     - Choose primitive action a according to macro-action μ
     - Sample the next state s' and observation z from models
     - Store the reward and update the belief state
     - Set $t = t+1$, $b = b'$, $s = s'$ and repeat with next primitive action until μ terminates
   - Compute the value of the resulting belief state b'
     - Interpolate over vertices in resulting belief sub-simplex
   - Repeat multiple times and average the estimate

As a note, "E" in this notation means expected value.

The next step is to estimate the expected value function by sampling using a macro-action on the current grid point. We sample the state from the current grid point. As a reminder, the belief state and grid points are probability distributions across the total set of states. Then we choose a primitive action according to our macro-action, which in the test case of a robot navigating corridors, is the policy for how to go down a hall.

Next, we sample the next state and observations from the models. The representation for the models used are: T(s,a,.) for s' and O(a,s',.) for z. We then store the reward and update the belief state along macro-action.

To store the reward, we use these equations:

$$R(g_i,\mu) := R(g_i,\mu) + \gamma^t R(s, a)$$

or with reward-shaping:

$$R(g_i,\mu) := R(g_i,\mu) + \gamma^t V(s') - \gamma^t V(s)$$

Where V(s) is the value function of state s. The test results will show that using reward-shaping is very important in being able to solve the problem.

We move our states resulting from the primitive action and we repeat with the next primitive action until the macro-action terminates. From the resulting belief state, we can compute its value. We do this by interpolating its value from the vertices of the belief sub-simplex that the resulting belief is located in. We repeat this whole step multiple times and average the estimate to get the expected value function.

Reinforcement Learning With a Model

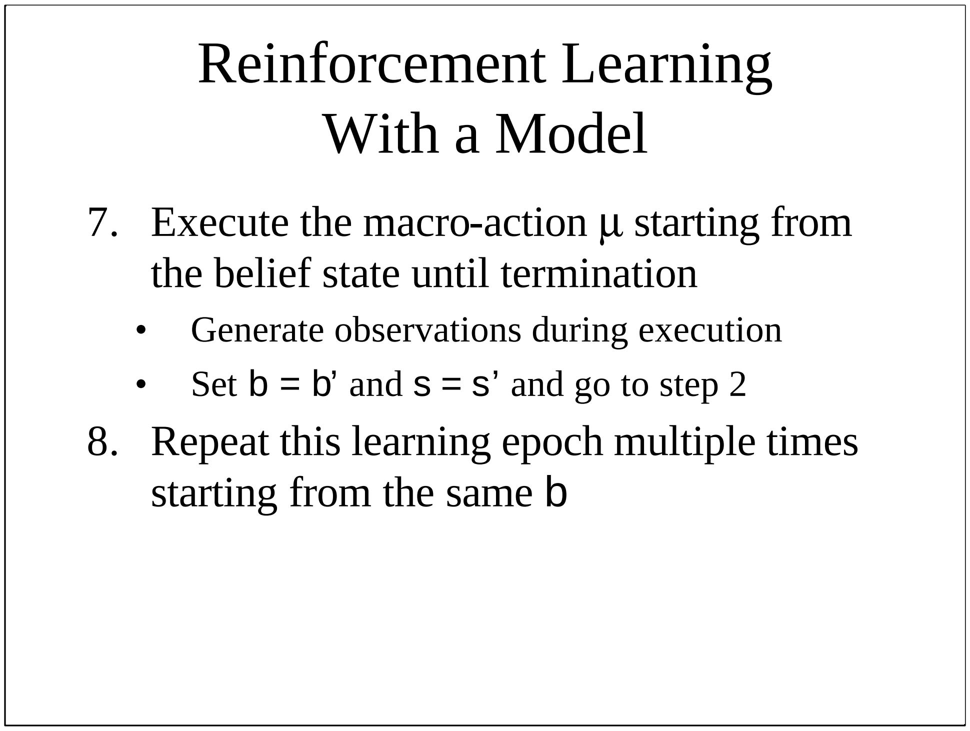
5. Update the state action value:

   $Q(g_i,\mu) = (1-\beta)q(g_i,\mu) + \beta\,[R + \gamma^t\,v(b')]$

6. Update the state value:

   $V(g_i) = \text{argmax}_\mu Q(g_i,\mu)$

Once we have estimated the value function, we must update the Q and V values for the grid point that we discretize to in step 2. This is the belief state for the overall POMDP, which is represented by "b" on the graphical overview of the algorithm.

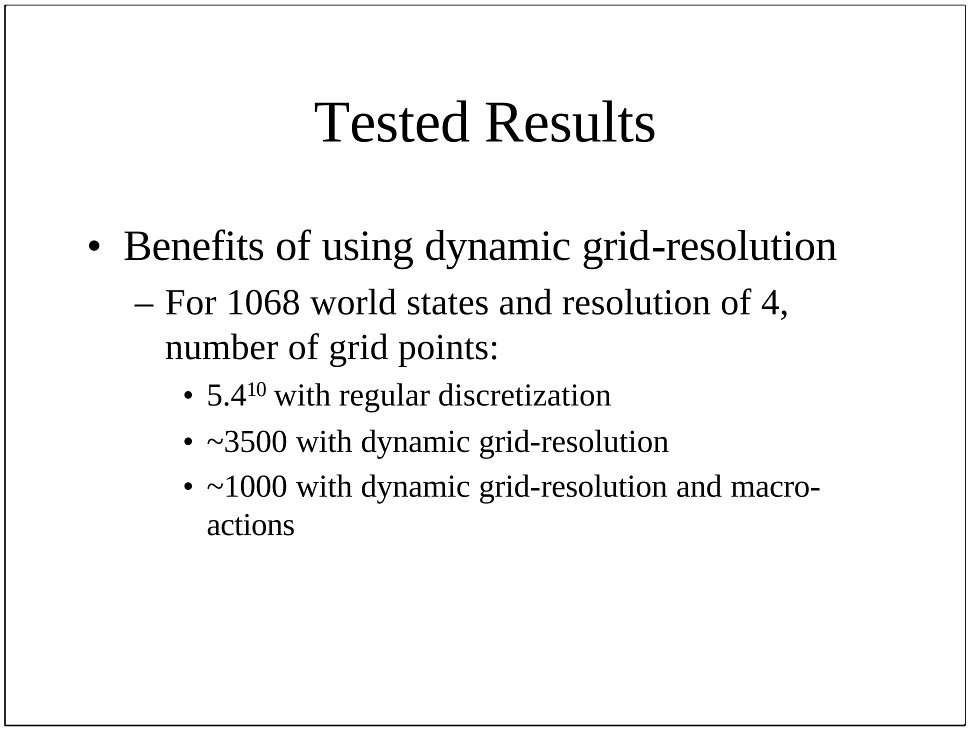


Reinforcement Learning With a Model

7. Execute the macro-action μ starting from the belief state until termination
   - Generate observations during execution
   - Set $b = b'$ and $s = s'$ and go to step 2
8. Repeat this learning epoch multiple times starting from the same b

Once we have updated our state-action values and state values for the grid point, we will execute the macro-action starting from the belief state mentioned on the last slide, until it terminates. We generate observations by sampling the POMDP model during execution. Once the macro-action terminates, we move our state and belief state and repeat from step 2.

For the test results, this learning epoch is repeated multiple times starting from the same b.



Tested Results

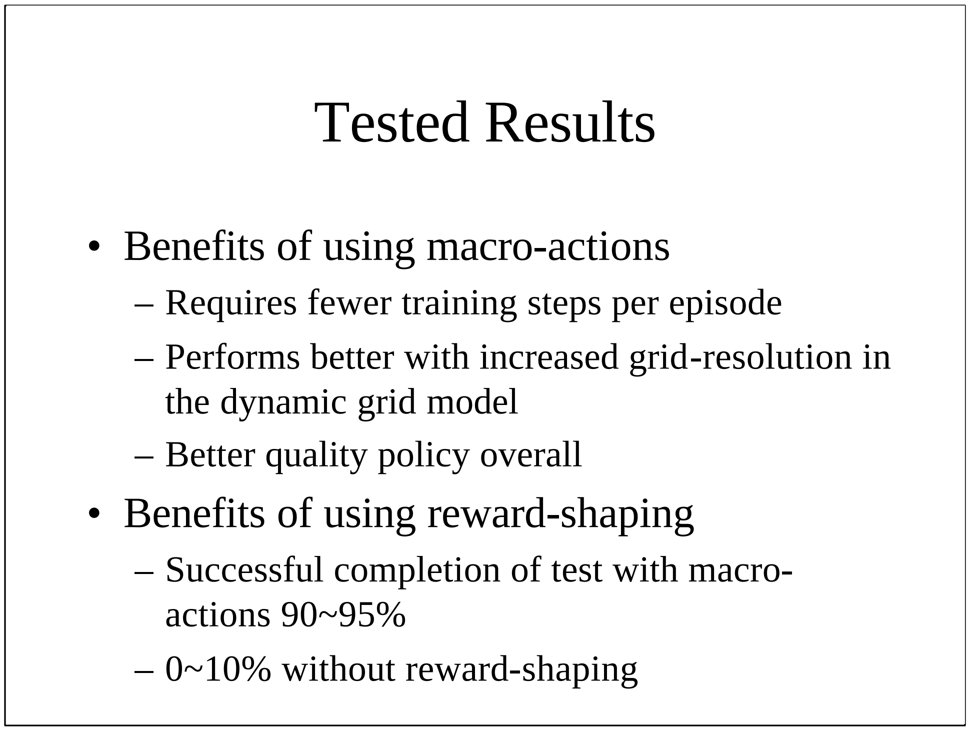
- Benefits of using dynamic grid-resolution
  - For 1068 world states and resolution of 4, number of grid points:
    - 5.4^{10} with regular discretization
    - ~3500 with dynamic grid-resolution
    - ~1000 with dynamic grid-resolution and macro-actions

In the tests run with this algorithm, we can see the benefits of using dynamic grid resolution. The number of grid points for a regular discretization of the space is given by:

$$\frac{(r+|S|-1)!}{r!\,(|S|-1)!}$$

Where |S| is the number of states, and r is the resolution.

This gives 5.4^10 for the test problem with regular discretization. However, adding dynamic grid-resolution will only create ~3500 grid points. And once we add macro-actions, this number reduces to ~1000. This reduction due to macro-actions is because grid points are not created along the simulation of the macro-action, but only at the end of the macro-action. We do not have to discretize to the nearest grid point and create it for every primitive action if we use macro-actions.

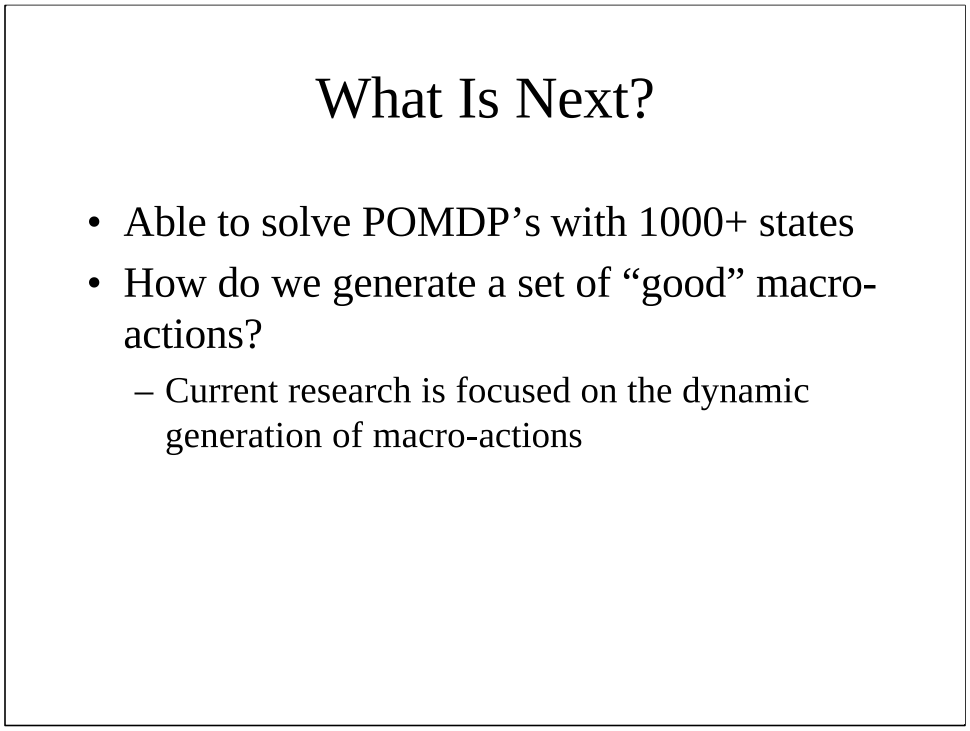


Tested Results

- Benefits of using macro-actions
  - Requires fewer training steps per episode
  - Performs better with increased grid-resolution in the dynamic grid model
  - Better quality policy overall
- Benefits of using reward-shaping
  - Successful completion of test with macro-actions 90~95%
  - 0~10% without reward-shaping

The test results found that fewer training steps per episode were required when using macro-actions. Macro-actions perform better with increased grid-resolution in the dynamic grid model because you don't have to generate so many grid-points when you are not limited to primitive actions. The test results also showed that macro-actions generated a better quality policy overall. These benefits were realized because the test used "good" macro-actions that helped the agent to localize and reduce the uncertainty of its belief state.

One key component that this test showed was that using reward shaping was critical in being able to solve the problem. Without it, the agent was not able to solve the problem in any of the cases except for in 10% of time when it used macro-actions and a resolution of 4. This is in contrast to 90-95% of the time across resolutions 1,2 and 4 when it used macro-actions.



What Is Next?

- Able to solve POMDP’s with 1000+ states
- How do we generate a set of “good” macro-actions?
  - Current research is focused on the dynamic generation of macro-actions

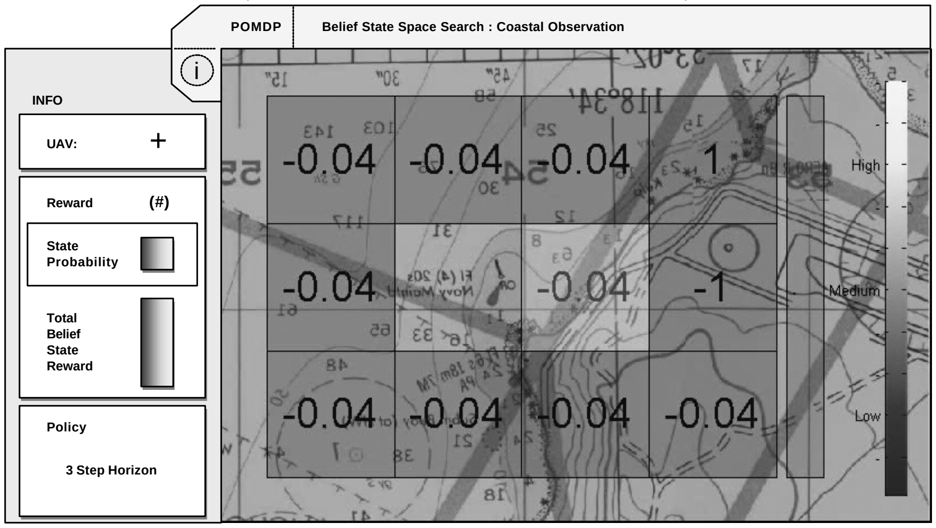
So this lecture showed that we are able to solve relatively large POMDPs by reducing its complexity through the use of spatial and temporal abstractions (grid approximation and macro-actions). We were able to move from solving 10’s of states in a POMDP to solving over 1000 states. However, the question arises, how do we generate “good” macro-actions? Good macro-actions are ones that will allow us to localize and reduce the uncertainty of our belief state. However, the current setup requires that the user create the set of macro-actions. Current research by the author of the paper is focused on the dynamic generation of macro-actions.

End

An Extra Visualization

- This visualization (and others) were cut from the talk, for brevity. However, we thought that it was worthwhile to include this one in the tutorial slides.

POMDP Belief State Space Search (Coastal Observation)



POMDP | Belief State Space Search : Coastal Observation

INFO

UAV: +

Reward (#)

State Probability

Total Belief State Reward

Policy

3 Step Horizon

-0.04	-0.04	-0.04	1
-0.04		-0.04	-1
-0.04	-0.04	-0.04	-0.04

Next we introduce how observations play into POMDPs.

Let’s say that if you are in position 8, you can clearly see the coast line. Consequently, you gain positional accuracy.

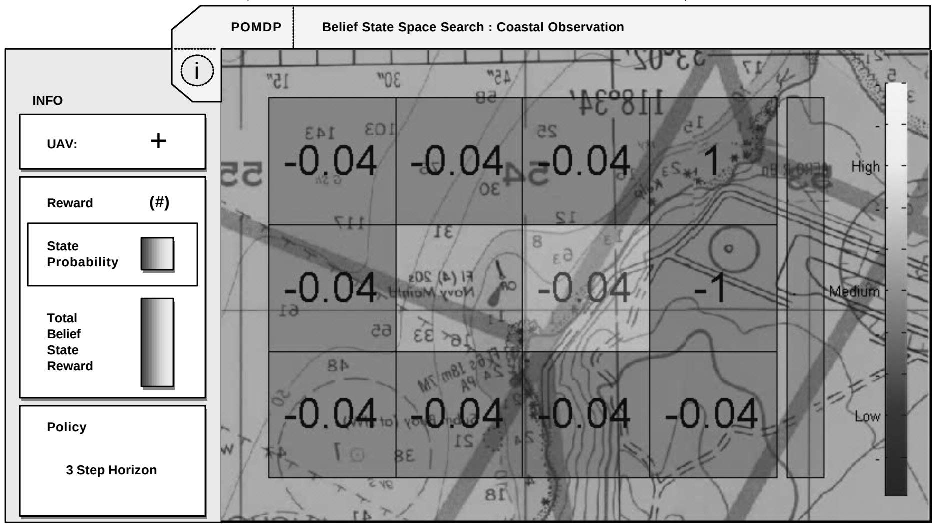
This visualization shows that policy being generated.

The observation model affects (or updates) our belief state. This is reflected above. Our initial belief state assumes that we are in the state shown above (row 2, column 3).

This positional accuracy is due to the positive observation. This level of certainty is simpler, and therefore useful for demonstrating this process.

A nice aspect of this visualization is that the low uncertainty makes it easier to follow (visually) the process.

POMDP Belief State Space Search (Coastal Observation)


POMDP
Belief State Space Search : Coastal Observation
INFO
UAV:
Reward
(#)
State Probability
Total Belief State Reward
Policy
3 Step Horizon
High
Medium
Low

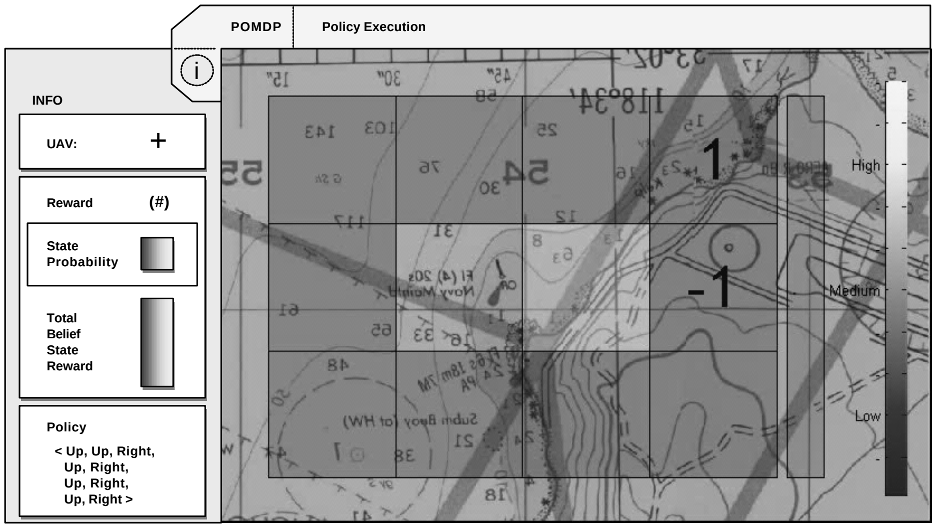
-0.04	-0.04	-0.04	1
-0.04		-0.04	-1
-0.04	-0.04	-0.04	-0.04

Animation Slide:

This slide is used to activate the movie.

It looks the same as the previous slide, but it has an embedded movie that runs when the slide is shown.

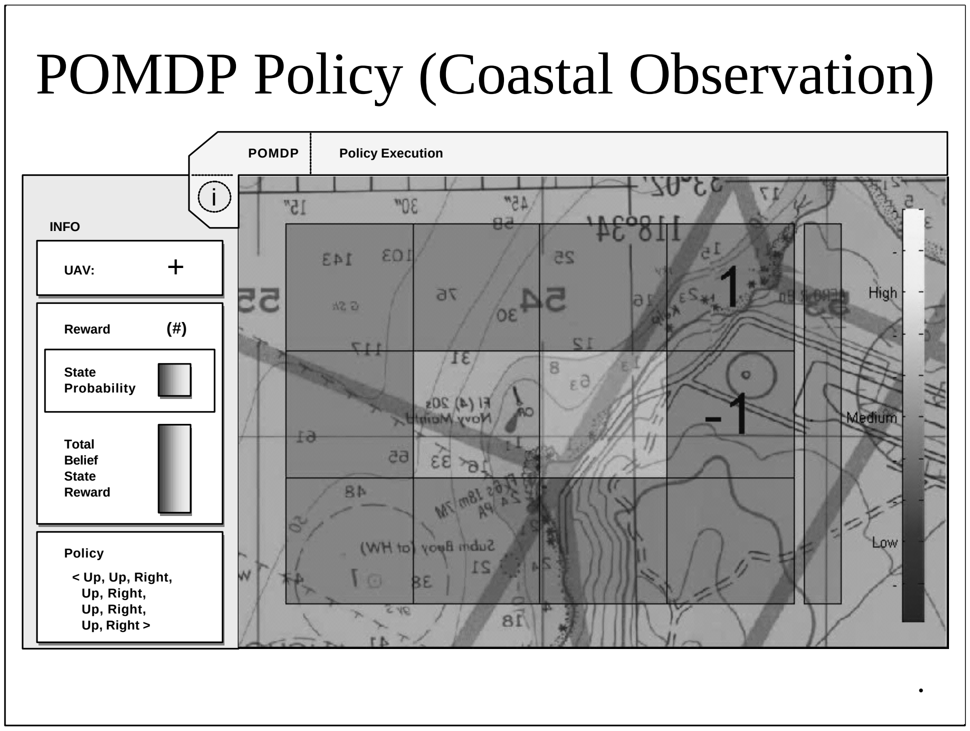
POMDP Policy (Coastal Observation)



Policy
< Up, Up, Right,
Up, Right,
Up, Right,
Up, Right >

This slide shows the execution of a policy generated in the last slide.
It used a 3-step look-ahead and an initial coastal observation.
The policy if as follows:

< Up, Up, Right, Up, Right, Up, Right, Up, Right >



POMDP Policy (Coastal Observation)

Animation Slide:

This slide is used to activate the movie.

It looks the same as the previous slide, but it has an embedded movie that runs when the slide is shown.