Introduction to Artificial Intelligence

Planning and Markov Decision Process

Andres Mendez-Vazquez

April 20, 2019

Outline

Outline

In Classic Planning

Planning

- Planning is the process of computing several steps of a problem-solving procedure before executing any of them

Something Notable

- This problem can be solved by search

Differences

- The main difference between search and planning is the representation of states

In Classic Planning

Planning

- Planning is the process of computing several steps of a problem-solving procedure before executing any of them

Something Notable

- This problem can be solved by search

Differences

- The main difference between search and planning is the representation of states

In Classic Planning

Planning

- Planning is the process of computing several steps of a problem-solving procedure before executing any of them

Something Notable

- This problem can be solved by search

Differences

- The main difference between search and planning is the representation of states

Outline

Differences

Search

- In search, states are represented as a single entity
  - They may be quite a complex object, but its internal structure is not used by the search algorithm.

Planning

- In planning, states have structured representations which are used by the planning algorithm.

Differences

Search

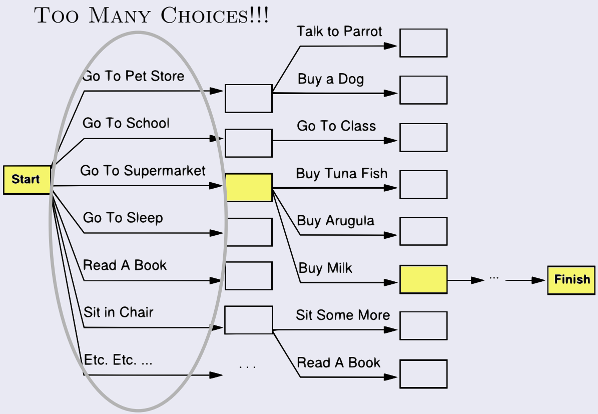
- In search, states are represented as a single entity
  - They may be quite a complex object, but its internal structure is not used by the search algorithm.

Planning

- In planning, states have structured representations which are used by the planning algorithm.

For Example, search search seems to fail

Consider: **get milk, bananas, and a cordless drill**



Search Vs Planning

We have the following steps for planning

1. Open up action and goal representation to allow selection
2. Divide-and-conquer by sub-goaling
3. Relax requirement for sequential construction of solutions

Thus, we have that

	Search	Classic Planning
States	Data Structures	Sentences
Actions	Cost function	Predictions/Outcomes
Goal	Cost function	Sentences
Plan	Sequences from S_0	Constrain on Actions

Search Vs Planning

We have the following steps for planning

1. Open up action and goal representation to allow selection
2. Divide-and-conquer by sub-goaling
3. Relax requirement for sequential construction of solutions

Thus, we have that

	Search	Classic Planning
States	Data Structures	Sentences
Actions	Cost function	Predictions/Outcomes
Goal	Cost function	Sentences
Plan	Sequences from S_0	Constrain on Actions

Draw Backs of Classic Planning

We have the following assumptions

1. Environment is deterministic - Problem how many?
2. Environment is observable - Probability can handle hidden variables
3. Environment is static (it only in response to the agent's actions) Again how many?

Therefore, we need something better

- Classic Planning cannot handle dynamic and noisy environments!!!!

Draw Backs of Classic Planning

We have the following assumptions

1. Environment is deterministic - Problem how many?
2. Environment is observable - Probability can handle hidden variables
3. Environment is static (it only in response to the agent's actions)
   Again how many?

Therefore, we need something better

- Classic Planning cannot handle dynamic and noisy environments!!!!

Outline

Outline

STRIPS

History

- STRIPS planning language (Fikes and Nilsson, 1971)

Properties

- Tidily arranged actions descriptions, restricted language
  - ACTION: $Buy(x)$
  - PRECONDITION: $At(p), Sells(p, x)$
  - EFFECTS: $Have(x)$

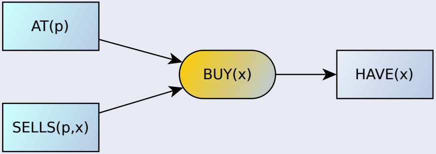
STRIPS

History

- STRIPS planning language (Fikes and Nilsson, 1971)

Properties

- Tidily arranged actions descriptions, restricted language
  - ACTION: $Buy\,(x)$
  - PRECONDITION: $At(p), Sells(p,x)$
  - EFFECTS: $Have\,(x)$



Outline

PDDL

PDDL (the "Planning Domain Definition Language")

- It was an attempt to standardize planning domain and problem description languages.

Something Notable

- It was developed mainly to make the International Planning Competition (IPC) series possible.

PDDL

PDDL (the "Planning Domain Definition Language")

- It was an attempt to standardize planning domain and problem description languages.

Something Notable

- It was developed mainly to make the International Planning Competition (IPC) series possible.

Outline

Components of a PDDL planning task

Components

- **Objects**: Things in the world that interest us.
- **Predicates**: Properties of objects that we are interested in; can be true or false.
- **Initial state**: The state of the world that we start in.
- **Goal specification**: Things that we want to be true.
- **Actions/Operators**: Ways of changing the state of the world.

Components of a PDDL planning task

Components

- **Objects**: Things in the world that interest us.
- **Predicates**: Properties of objects that we are interested in; can be true or false.
- **Initial state**: The state of the world that we start in.
- **Goal specification**: Things that we want to be true.
- **Actions/Operators**: Ways of changing the state of the world.

Components of a PDDL planning task

Components

- **Objects**: Things in the world that interest us.
- **Predicates**: Properties of objects that we are interested in; can be true or false.
- **Initial state**: The state of the world that we start in.
- **Goal specification**: Things that we want to be true.
- **Actions/Operators**: Ways of changing the state of the world.

Components of a PDDL planning task

Components

- **Objects**: Things in the world that interest us.
- **Predicates**: Properties of objects that we are interested in; can be true or false.
- **Initial state**: The state of the world that we start in.
- **Goal specification**: Things that we want to be true.
- **Actions/Operators**: Ways of changing the state of the world.

Components of a PDDL planning task

Components

- **Objects**: Things in the world that interest us.
- **Predicates**: Properties of objects that we are interested in; can be true or false.
- **Initial state**: The state of the world that we start in.
- **Goal specification**: Things that we want to be true.
- **Actions/Operators**: Ways of changing the state of the world.

Classic Planning

We have

Planning problem = planning domain + initial state

Goal is a conjunction of literals

$$Have\left(Jaguar\right) \wedge \neg At\left(Jail\right)$$

Therefore

- We can use search strategies as **backtracking** to solve the problem

Classic Planning

We have

$$\textbf{Planning problem} = \textbf{planning domain} + \textbf{initial state}$$

Goal is a conjunction of literals

$$Have\,(Jaguar) \wedge \neg At\,(Jail)$$

Therefore

- We can use search strategies as **backtracking** to solve the problem

Classic Planning

We have

Planning problem = planning domain + initial state

Goal is a conjunction of literals

$$Have\,(Jaguar) \wedge \neg At\,(Jail)$$

Therefore

- We can use search strategies as **backtracking** to solve the problem

Outline

Planning Domain

Such Domains Look like

- (define (domain <domain name>)
- <PDDL code for predicates>
- <PDDL code for first action>
- [...]
- <PDDL code for last action>)

For Example

We have

- $a \in Actions\,(s)$ iff $s \vDash Precond\,(a)$
- $Result\,(s, a) = (s - Del\,(a)) \cup Add\,(a)$

Where $Del(a)$ is a list of literals

- They appear negatively in the effect of a

Finally

- $Add(a)$ is the list of positive literals in the effect of a.

For Example

We have

- $a \in Actions(s)$ iff $s \vDash Precond(a)$
- $Result(s, a) = (s - Del(a)) \cup Add(a)$

Where $Del(a)$ is a list of literals

- They appear negatively in the effect of a

Finally

- $Add(a)$ is the list of positive literals in the effect of a.

For Example

We have

- $a \in Actions\,(s)$ iff $s \vDash Precond\,(a)$
- $Result\,(s, a) = (s - Del\,(a)) \cup Add\,(a)$

Where $Del(a)$ is a list of literals

- They appear negatively in the effect of a

Finally

- $Add(a)$ is the list of positive literals in the effect of a.

Example of how they can be used

We have the following structures

- Action: $Buy(x)$
- Precondition: $At(p), Sells(p, x), Have(Money)$
- Effect: $Have(x), \neg Have(Money)$

Then, we have the following

- $Del(Buy(Jaguar)) = \{Have(Money)\}$
- $Add(Buy(Jaguar)) = \{Have(Jaguar)\}$

Example of how they can be used

We have the following structures

- Action: $Buy(x)$
- Precondition: $At(p), Sells(p,x), Have(Money)$
- Effect: $Have(x), \neg Have(Money)$

Then, we have the following

- $Del(Buy(Jaguar)) = \{Have(Money)\}$
- $Add(Buy(Jaguar)) = \{Have(Jaguar)\}$

Then

We have the following

- $s =$
  $\{At(JDealer), Sells(JDealer, Jaguar), Blue(Sky), Have(Money)\}$

Now, we have

- $Buy(Jaguar) \in Actions(s)$

Finally

$$\begin{aligned} Result(s, Buy(Jaguar)) &= \{s - \{Have(Money)\}\} \cup \{Have(Jaguar)\}\} \\ &== \{At(JDealer), Sells(JDealer, Jaguar), \\ &\ldots Blue(Sky), Have(Jaguar)\} \end{aligned}$$

Then

We have the following

- $s =$
  $\{At(JDealer), Sells(JDealer, Jaguar), Blue(Sky), Have(Money)\}$

Now, we have

- $Buy(Jaguar) \in Actions(s)$

Finally

$$Result(s, Buy(Jaguar)) = \{s - \{Have(Money)\}) \cup \{Have(Jaguar)\}\}$$
$$== \{At(JDealer), Sells(JDealer, Jaguar),$$
$$\ldots Blue(Sky), Have(Jaguar)\}$$

Then

We have the following

- $s =$
  $\{At(JDealer), Sells(JDealer, Jaguar), Blue(Sky), Have(Money)\}$

Now, we have

- $Buy(Jaguar) \in Actions(s)$

Finally

$$
\begin{aligned}
Result(s, Buy(Jaguar) = \{s - \{Have(Money)\}) \cup \{Have(Jaguar)\}\} \\
== \{At(JDealer), Sells(JDealer, Jaguar), \\
... Blue(Sky), Have(Jaguar)\}
\end{aligned}
$$

Outline

As in the Logic Part of this course

Solving the problem

- It is possible to use forward and backward procedures to solve classic planning

Backward and Forward Search

- It can use any search method, breadth-first or depth-first or iterative deepening or A^* ...

What with would do?

- If there are several goal states, search backwards from each in turn.

As in the Logic Part of this course

Solving the problem

- It is possible to use forward and backward procedures to solve classic planning

Backward and Forward Search

- It can use any search method, breadth-first or depth-first or iterative deepening or A^* ...

What with would do?

- If there are several goal states, search backwards from each in turn.

As in the Logic Part of this course

Solving the problem

- It is possible to use forward and backward procedures to solve classic planning

Backward and Forward Search

- It can use any search method, breadth-first or depth-first or iterative deepening or A^* ...

What with would do?

- If there are several goal states, search backwards from each in turn.

Therefore

Something Notable

- Planning can use both forward and backward search (progression and regression planning)

Outline

We have this

Planning domain

- Predicates: At, Sells, Have
-
- Two action schemas:
- Action: $Buy(x)$
- Precondition: $At(p), Sells(p,x), Have(Money)$
- Effect: $Have(x), \neg Have(Money)$
- Action: $Go(x,y)$
- Precondition: $At(x)$
- Effect: $At(y), \neg At(x)$

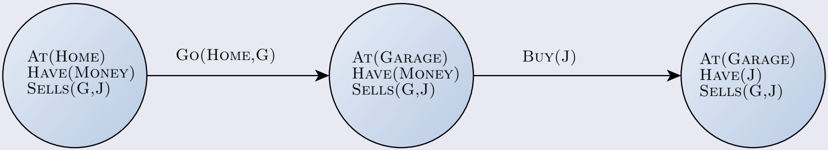
Now

Planning domain

- Planning domain above plus
- Objects: $Money, J$ (for Jaguar), $Home$, G (for Garage)
- Initial state: $At(Home) \wedge Have(Money) \wedge Sells(G, J)$
- Goal state: $Have(J)$

Then, we have

The following plan



Outline

History

Markov Decision Process (MDP) is a discrete time stochastic control process

- It provides a mathematical framework for modeling decision making

They are perfect for

- MDP's are useful for studying optimization problems solved via dynamic programming and reinforcement learning.

MDP's are know as early as 1950's

- A core body of research on Markov decision processes resulted from Howard's 1960 book, **Dynamic Programming and Markov Processes**.

History

Markov Decision Process (MDP) is a discrete time stochastic control process

- It provides a mathematical framework for modeling decision making

They are perfect for

- MDP's are useful for studying optimization problems solved via dynamic programming and reinforcement learning.

MDP's are know as early as 1950's

- A core body of research on Markov decision processes resulted from Howard's 1960 book, **Dynamic Programming and Markov Processes**.

History

Markov Decision Process (MDP) is a discrete time stochastic control process

- It provides a mathematical framework for modeling decision making

They are perfect for

- MDP's are useful for studying optimization problems solved via dynamic programming and reinforcement learning.

MDP's are know as early as 1950's

- A core body of research on Markov decision processes resulted from Howard's 1960 book, **Dynamic Programming and Markov Processes**.

Furthermore

The name of MDPs comes

- From the Russian mathematician Andrey Markov.

Andrey Andreyevich Markov (1856-1822)

- It was a Russian mathematician best known for his work on stochastic processes.

Remark

- A primary subject of his research later became known as Markov chains and Markov processes.

Furthermore

The name of MDPs comes

- From the Russian mathematician Andrey Markov.

Andrey Andreyevich Markov (1856-1822)

- It was a Russian mathematician best known for his work on stochastic processes.

Remark

- A primary subject of his research later became known as Markov chains and Markov processes.

Furthermore

The name of MDPs comes

- From the Russian mathematician Andrey Markov.

Andrey Andreyevich Markov (1856-1822)

- It was a Russian mathematician best known for his work on stochastic processes.

Remark

- A primary subject of his research later became known as Markov chains and Markov processes.

And Here Came the Cavalry

MDP's in AI in the 90's

1. **Reinforcement Learning**
2. **Probabilistic Planning**

Beyond the Static Environments

- But it took almost 30 years...

And Here Came the Cavalry

MDP's in AI in the 90's

1. **Reinforcement Learning**
2. **Probabilistic Planning**

Beyond the Static Environments

- But it took almost 30 years...

Outline

What are MDP's good for?

Expanding the reach of planning

1. Uncertain Domain Dynamics
2. Sequential Decision Making
3. Cyclic Domain Structures
4. Fair Nature
5. Rational Decision Making

What are MDP's good for?

Expanding the reach of planning

1. Uncertain Domain Dynamics
2. Sequential Decision Making
3. Cyclic Domain Structures
4. Fair Nature
5. Rational Decision Making

What are MDP's good for?

Expanding the reach of planning

1. Uncertain Domain Dynamics
2. Sequential Decision Making
3. Cyclic Domain Structures
4. Fair Nature
5. Rational Decision Making

What are MDP's good for?

Expanding the reach of planning

1. Uncertain Domain Dynamics
2. Sequential Decision Making
3. Cyclic Domain Structures
4. Fair Nature
5. Rational Decision Making

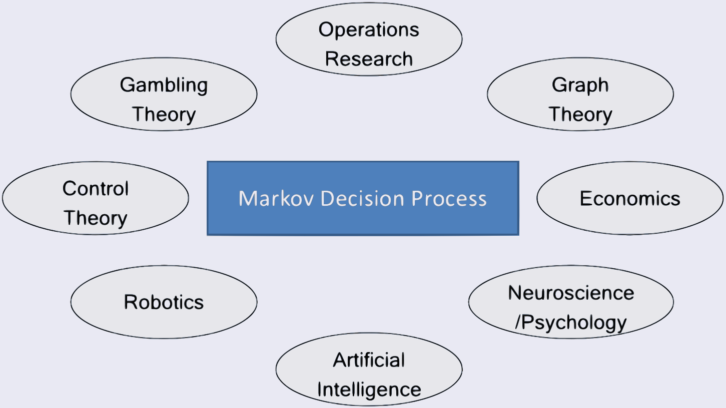
What are MDP's good for?

Expanding the reach of planning

1. Uncertain Domain Dynamics
2. Sequential Decision Making
3. Cyclic Domain Structures
4. Fair Nature
5. Rational Decision Making

They have several applications

Many!!!



Outline

A Broad Definition

MDP is a tuple $\langle S, D, A, T, R \rangle$

1. S is a finite state space
2. D is a sequence of discrete time steps/decision epochs $(1, 2, 3, ..., L)$, L may be ∞
3. A is a finite action set
4. $T : S \times A \times S \times D \rightarrow [0, 1]$ is a transition function
5. $R : S \times A \times S \times D \rightarrow \mathbb{R}$ is a reward function

Basically, we have for such functions

In the case of T

$$T\left(s_t, a_t, s_{t+1}, t\right) \in [0, 1]$$

- Basically you can think as probability

In the case of R

$$R\left(s_t, a_t, s_{t+1}, t\right)$$

- It could be a positive or negative quantity

Basically, we have for such functions

In the case of T

$$T\left(s_t, a_t, s_{t+1}, t\right) \in [0, 1]$$

- Basically you can think as probability

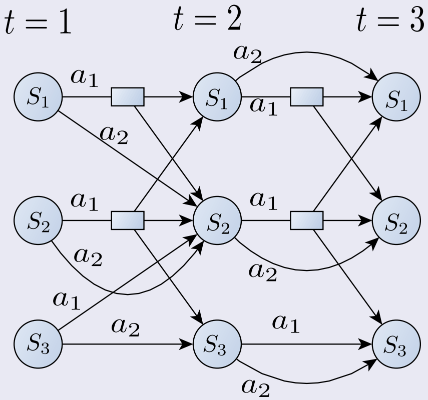
In the case of R

$$R\left(s_t, a_t, s_{t+1}, t\right)$$

- It could be a positive or negative quantity

Graphically

We have a structure over time



$T(s_3, a_1, s_2, 1) = 1.0$
$R(s_3, a_1, s_2, 1) = -7.2$

Outline

What do we want?

We want a way to choose an action in a state

- We want a policy π

What does a policy look like?

- We can pick actions based on
  - **States-Visited + actions used**
    - Basically an execution history

$$h = (s_1, a_1) \longrightarrow (s_2, a_2) \longrightarrow s_3 \ldots$$

  - **Random actions...**

What do we want?

We want a way to choose an action in a state

- We want a policy π

What does a policy look like?

- We can pick actions based on
  - **States-Visited + actions used**
    - Basically an execution history

$$h = (s_1, a_1) \longrightarrow (s_2, a_2) \longrightarrow s_3 \ldots$$

  - **Random actions...**

First than anything

An MDP solution is a probabilistic history-dependent

$$\pi : S \times H \longrightarrow A$$

- with a set of states S, H a set of execution stories and A a set of actions

Additionally, executing a policy yields a sequence of random variable rewards

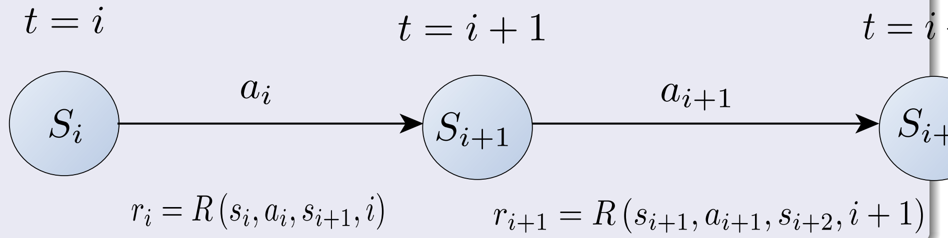
First than anything

An MDP solution is a probabilistic history-dependent

$$\pi : S \times H \longrightarrow A$$

- with a set of states S, H a set of execution stories and A a set of actions

Additionally, executing a policy yields a sequence of random variable rewards



Then, we can define

Define a utility function as a "quality measure"

$$u\left(R_1, R_2, \ldots\right)$$

Thus, we can define a value function

$$V : H \longrightarrow [-\infty, \infty]$$

Then, we can define

Define a utility function as a "quality measure"

$$u\left(R_1, R_2, \ldots\right)$$

Thus, we can define a value function

$$V : H \longrightarrow [-\infty, \infty]$$

Then

We can use the utility function to define the value function after history h

$$V^{\pi}(h) = u_h^{\pi}(R_1, R_2, ...)$$

Thus, we want the optimal policy π^*

$$V^*(h) \geq V^{\pi}(h) \text{ for all } \pi \text{ and for all } h$$

Properties

- Intuitively, a policy is optimal if its utility vector dominates.
- h^* not necessarily unique.

Then

We can use the utility function to define the value function after history h

$$V^{\pi}(h) = u_h^{\pi}(R_1, R_2, \ldots)$$

Thus, we want the optimal policy π^*

$$V^*(h) \geq V^{\pi}(h) \text{ for all } \pi \text{ and for all } h$$

Properties

- Intuitively, a policy is optimal if its utility vector dominates.
- h^* not necessarily unique.

Then

We can use the utility function to define the value function after history h

$$V^{\pi}(h) = u_h^{\pi}(R_1, R_2, ...)$$

Thus, we want the optimal policy π^*

$$V^{*}(h) \geq V^{\pi}(h) \text{ for all } \pi \text{ and for all } h$$

Properties

- Intuitively, a policy is optimal if its utility vector dominates.
- h^* not necessarily unique.

Outline

We have a Grid World

Where walls block the way and reward is fixed as

-3	-3	-3	+100
-3	W	-3	-10
Start → 0	-3	-3	-3

We have the following situation

1. 80% of the time, the action North takes the agent North (if there is no wall there)
2. 10% of the time, North takes the agent West; 10% East
3. If there is a wall in the direction the agent would have been taken, the agent stays put

We have a Grid World

Where walls block the way and reward is fixed as

-3	-3	-3	+100
-3	W	-3	-10
Start $\rightarrow$ 0	-3	-3	-3

We have the following situation

1. 80% of the time, the action North takes the agent North (if there is no wall there)
2. 10% of the time, North takes the agent West; 10% East
3. If there is a wall in the direction the agent would have been taken, the agent stays put

In classic Planning

You will need to create a search tree

- Look at the board - The Search Tree

Therefore Problems

- Branching Factor
- Tree to deep
- Many states visited more than once!!!

In classic Planning

You will need to create a search tree

- Look at the board - The Search Tree

Therefore Problems

- Branching Factor
- Tree to deep
- Many states visited more than once!!!

Additionally

We have the following

- $+1$ is a goal state with a reward

And a "Bad State"

- With reward -100

Additionally

We have the following

- $+1$ is a goal state with a reward

And a "Bad State"

- With reward -100

Finally

Big rewards come at the end

- Thus, we are looking for the optimal policy!!!

$$\pi^* : S \times \{0, ..., H\} \to A$$

Thus

- An optimal policy maximizes expected sum of rewards

Finally

Big rewards come at the end

- Thus, we are looking for the optimal policy!!!

$$\pi^* : S \times \{0, ..., H\} \rightarrow A$$

Thus

- An optimal policy maximizes expected sum of rewards

Outline

Expected Linear Additive Utility

Find $\pi : S \times \{0, 1, 2, ..., H\} \rightarrow A$ that maximize expected sum of rewards

$$\pi^* = \arg\max_{\pi} E\left[\sum_{t=0}^{H} R_t\left(s_t, a_t, s_{t+1} | \pi\right)\right]$$

Examples

1. Cleaning Robots
2. Walking Robots
3. Pole Balancing
4. Games: tetris, backgammon
5. Server management
6. etc

Expected Linear Additive Utility

Find $\pi : S \times \{0, 1, 2, ..., H\} \rightarrow A$ that maximize expected sum of rewards

$$\pi^* = \arg\max_{\pi} E\left[\sum_{t=0}^{H} R_t\left(s_t, a_t, s_{t+1} | \pi\right)\right]$$

Examples

1. Cleaning Robots
2. Walking Robots
3. Pole Balancing
4. Games: tetris, backgammon
5. Server management
6. etc

Outline

Expected Linear Additive Utility (ELAU)

Here, we have a discount factor α and an finite horizon

$$V_{\alpha}^{\pi}(s) = E\left[\sum_{i=1}^{H} \alpha^{i} R_i\left(s_t, a_t, s_{t+1}, t\right) | s_0 = s, a_t = \pi\left(s_t\right), s_{t+1}\right]$$

We have different policies

- $\alpha \in [0,1)$ the rewards are more immediate in the history horizon.
- $\alpha > 1$ more distant horizon rewards are preferred.
- $\alpha = 1$ all the rewards are equally valuable.

Expected Linear Additive Utility (ELAU)

Here, we have a discount factor α and an finite horizon

$$V_{\alpha}^{\pi}(s) = E\left[\sum_{i=1}^{H} \alpha^{i} R_{i}(s_t, a_t, s_{t+1}, t) \,|s_0 = s, a_t = \pi(s_t), s_{t+1}\right]$$

We have different policies

- $\alpha \in [0, 1)$ the rewards are more immediate in the history horizon.
- $\alpha > 1$ more distant horizon rewards are preferred.
- $\alpha = 1$ all the rewards are equally valuable.

Something important that we need to mention

Finite Horizon Problem

- A problem has a finite horizon if there is a known upper bound on the number of stages at which one may stop.
  - We saw that in the previous equations

Infinite Horizon Problem

- A problem has an infinite horizon if there is no upper bound on the number of stages.
  - Here $H \longrightarrow \infty$

Something important that we need to mention

Finite Horizon Problem

- A problem has a finite horizon if there is a known upper bound on the number of stages at which one may stop.
  - We saw that in the previous equations

Infinite Horizon Problem

- A problem has an infinite horizon if there is no upper bound on the number of stages.
  - Here $H \longrightarrow \infty$

Outline

The Optimality Principle

Remark

- If the quality of every policy can be measured by its expected linear additive utility, there is a policy that is optimal at every time step.

This was stated by

- Bellman, Denardo, and others

The Optimality Principle

Remark

- If the quality of every policy can be measured by its expected linear additive utility, there is a policy that is optimal at every time step.

This was stated by

- Bellman, Denardo, and others

It is more

Bellman said that

- "An optimal policy has the property that whatever the initial state and initial decision are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision."

In our case, the policy that achieves the highest value

$$\pi^{*} = \arg\max_{\pi} V^{\pi}(s)$$

- Problem!!! The number of policies is exponential... this does not help that much, we need the "Bellman optimality equation"

It is more

Bellman said that

- "An optimal policy has the property that whatever the initial state and initial decision are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision."

In our case, the policy that achieves the highest value

$$\pi^{*} = \arg\max_{\pi} V^{\pi}\left(s\right)$$

- Problem!!! The number of policies is exponential... this does not help that much, we need the "Bellman optimality equation"

Outline

Transitions as probabilities

It is possible to look at transitions as probabilities

$$P\left(s_{t+1}|s_t, a_t = \pi\left(s_t\right)\right) = T\left(s_t, a_t, s_{t+1}, t\right)$$

Then, Bellman proposed the use of an infinite horizon

$$V^{\pi}\left(s\right) = E\left[\sum_{t=0}^{\infty} \alpha^t R\left(s_t, a_t, s_{t+1}, t\right) | s_0 = s, a_t = \pi\left(s_t\right), s_{t+1}|s_t, a_t \sim P\right]$$

- Where $\alpha < 1$ is a discount factor

Then

- It is also possible to use recursion to define such value...

Transitions as probabilities

It is possible to look at transitions as probabilities

$$P\left(s_{t+1}|s_t, a_t = \pi\left(s_t\right)\right) = T\left(s_t, a_t, s_{t+1}, t\right)$$

Then, Bellman proposed the use of an infinite horizon

$$V^{\pi}\left(s\right) = E\left[\sum_{t=0}^{\infty}\alpha^{t}R\left(s_t, a_t, s_{t+1}, t\right)|s_0 = s, a_t = \pi\left(s_t\right), s_{t+1}|s_t, a_t \sim P\right]$$

- Where $\alpha < 1$ is a discount factor

Then

- It is also possible to use recursion to define such value...

Transitions as probabilities

It is possible to look at transitions as probabilities

$$P\left(s_{t+1}|s_t, a_t = \pi\left(s_t\right)\right) = T\left(s_t, a_t, s_{t+1}, t\right)$$

Then, Bellman proposed the use of an infinite horizon

$$V^{\pi}\left(s\right) = E\left[\sum_{t=0}^{\infty} \alpha^t R\left(s_t, a_t, s_{t+1}, t\right) |s_0 = s, a_t = \pi\left(s_t\right), s_{t+1}|s_t, a_t \sim P\right]$$

- Where $\alpha < 1$ is a discount factor

Then

- It is also possible to use recursion to define such value...

We have the following

The Bellman Equation using probabilities

$$V^{\pi}(s) = R(s) + \alpha \sum_{s' \in S} P(s'|s, \pi(s)) V^{\pi}(s')$$

In this case, we have

- $v^{\pi} \in \mathbb{R}^{|S|}$ be a vector of values for each state,
- $r \in \mathbb{R}^{|S|}$ be a vector of rewards for each state

We have the following

The Bellman Equation using probabilities

$$V^{\pi}(s) = R(s) + \alpha \sum_{s' \in S} P(s'|s, \pi(s)) V^{\pi}(s')$$

In this case, we have

- $v^{\pi} \in \mathbb{R}^{|S|}$ be a vector of values for each state,
- $r \in \mathbb{R}^{|S|}$ be a vector of rewards for each state

Additionally

$P \in \mathbb{R}^{|S| \times |S|}$ be a matrix containing probabilities for each transition under policy π

$$P_{ij}^{\pi} = P\left(s_{t+1} = i | s_t = j, a_t = \pi\left(s_t\right)\right)$$

Then, using our old Linear Algebra, we can see this as solving a linear system

$$\begin{aligned} v^{\pi} &= r + \alpha P^{\pi} v^{\pi} \\ \Rightarrow \left(I - \alpha P^{\pi}\right) v^{\pi} &= r \\ \Rightarrow v^{\pi} &= \left(I - \alpha P^{\pi}\right)^{-1} r \end{aligned}$$

Additionally

$P \in \mathbb{R}^{|S| \times |S|}$ be a matrix containing probabilities for each transition under policy π

$$P_{ij}^{\pi} = P\left(s_{t+1} = i | s_t = j, a_t = \pi\left(s_t\right)\right)$$

Then, using our old Linear Algebra, we can see this as solving a linear system

$$\begin{aligned} v^{\pi} &= r + \alpha P^{\pi} v^{\pi} \\ \Rightarrow \left(I - \alpha P^{\pi}\right) v^{\pi} &= r \\ \Rightarrow v^{\pi} &= \left(I - \alpha P^{\pi}\right)^{-1} r \end{aligned}$$

In the case of Optimality

The optimal value function using the Bellman optimality equation

$$V^*(s) = R(s) + \alpha \max_{a \in A} \sum_{s' \in S} P(s'|s,a) V^*(s')$$

Thus, Optimal policy is simply the action that attains this max

$$\pi^* = \arg\max_{a} \sum_{s' \in S} P(s'|s,a) V^*(s')$$

In the case of Optimality

The optimal value function using the Bellman optimality equation

$$V^*(s) = R(s) + \alpha \max_{a \in A} \sum_{s' \in S} P(s'|s,a) V^*(s')$$

Thus, Optimal policy is simply the action that attains this max

$$\pi^* = \arg\max_{a} \sum_{s' \in S} P(s'|s,a) V^*(s')$$

Outline

Thus

Optimal Control

- Given an MDP (S, A, T, R, α, H), we need to find the optimal policy π^*.

What methods do we have?

- Value Iteration
- Policy Iteration
- Linear Programming

Thus

Optimal Control

- Given an MDP (S, A, T, R, α, H), we need to find the optimal policy π^*.

What methods do we have?

- Value Iteration
- Policy Iteration
- Linear Programming

We have some initial assumptions

Discrete state-action spaces

- They are simpler to get the main concepts across.

Outline

Value Iteration

We have the following idea

- Repeatedly update an estimate of the optimal value function according to Bellman optimality equation

Step 1. Initialize an estimate for the value function arbitrarily

$$\hat{V}(s) \leftarrow 0 \ \forall s \in S$$

Step 2. Repeat and update using the Bellman Equation

$$\hat{V}(s) \leftarrow R(s) + \alpha \max_{a \in A} \sum_{s' \in S} P(s'|s,a)\hat{V}(s'), \ \forall s \in S$$

Value Iteration

We have the following idea

- Repeatedly update an estimate of the optimal value function according to Bellman optimality equation

Step 1. Initialize an estimate for the value function arbitrarily

$$\hat{V}(s) \leftarrow 0 \ \forall s \in S$$

Step 2. Repeat and update using the Bellman Equation

$$\hat{V}(s) \leftarrow R(s) + \alpha \max_{a \in A} \sum_{s' \in S} P(s'|s,a)\, \hat{V}(s'), \ \forall s \in S$$

Value Iteration

We have the following idea

- Repeatedly update an estimate of the optimal value function according to Bellman optimality equation

Step 1. Initialize an estimate for the value function arbitrarily

$$\hat{V}(s) \leftarrow 0 \ \forall s \in S$$

Step 2. Repeat and update using the Bellman Equation

$$\hat{V}(s) \leftarrow R(s) + \alpha \max_{a \in A} \sum_{s' \in S} P(s'|s,a)\, \hat{V}(s'), \ \forall s \in S$$

Example, Grid World

At initialization we have the value grid

0	0	0	+100
0	W	0	-10
Start $\rightarrow$ 0	0	0	0

Then at the first iteration

We have the values

$$\hat{V}(1,4) = +100$$
$$\hat{V}(2,4) = -100$$
$$\hat{V}(3,1) = 0$$
$$\hat{V}(i,j) = -3 \text{ for everything else}$$

Calculation

$\alpha = 0.9$ then, we get

$$\hat{V}(1,3) = -3 + \alpha \max_a \sum_{s' \in A} P(s'|s,a)\,\hat{V}(s')$$

Example, in the Board

After the first iteration second step, we have $\alpha = 0.9$

0	0	0	$+100$
0	W	0	-10
Start $\rightarrow$ 0	0	0	0

	P=0.8	
P=0.1	s	P=0.1

Remark

We could have included action down with a probability of 0

- How this will look? Use it as an exercise

Convergence of value iteration

Theorem

- Value iteration converges to optimal value: $\hat{V} \to V^*$

Proof

- For any estimate of the value function $\hat{V}$, we define the Bellman backup operator

$$B\hat{V}(s) = R(s) + \alpha \max_{a \in A} \sum_{s' \in S} P(s'|s,a)\,\hat{V}(s')$$

Convergence of value iteration

Theorem

- Value iteration converges to optimal value: $\hat{V} \to V^*$

Proof

- For any estimate of the value function $\hat{V}$, we define the Bellman backup operator

$$B\hat{V}(s) = R(s) + \alpha \max_{a \in A} \sum_{s' \in S} P(s'|s,a)\, \hat{V}(s')$$

We need to prove the operator is a contraction

Basically given two estimations V_1, V_2

$$\max_{s \in S} \left| BV_1(s) - BV_2(s) \right| \leq \alpha \max_{s \in S} \left| V_1(s) - V_2(s) \right|$$

Remarks

- Since the contraction property also implies the existence and uniqueness of a fixed point

$$BV^* = V^*$$

We need to prove the operator is a contraction

Basically given two estimations V_1, V_2

$$\max_{s \in S} |BV_1(s) - BV_2(s)| \leq \alpha \max_{s \in S} |V_1(s) - V_2(s)|$$

Remarks

- Since the contraction property also implies the existence and uniqueness of a fixed point

$$BV^* = V^*$$

Then, we prove contraction

We do the following

$$
\begin{aligned}
\left|BV_1(s) - BV_2(s)\right| &\leq \alpha \left| \max_{a \in A} \sum_{s' \in S} P(s'|s,a) V_1(s') - \max_{a \in A} \sum_{s' \in S} P(s'|s,a) V_2(s') \right| \\
&\leq \alpha \max_{a \in A} \left| \sum_{s' \in S} P(s'|s,a) V_1(s') - \sum_{s' \in S} P(s'|s,a) V_2(s') \right| \\
&\leq \alpha \max_{a \in A} \sum_{s' \in S} P(s'|s,a) \left|V_1(s') - V_2(s')\right| \\
&\leq \alpha \max_{a \in A} \left|V_1(s') - V_2(s')\right|
\end{aligned}
$$

Then, we prove contraction

We do the following

$$
\begin{aligned}
\left|BV_1(s) - BV_2(s)\right| &\leq \alpha \left| \max_{a \in A} \sum_{s' \in S} P(s'|s,a) V_1(s') - \max_{a \in A} \sum_{s' \in S} P(s'|s,a) V_2(s') \right| \\
&\leq \alpha \max_{a \in A} \left| \sum_{s' \in S} P(s'|s,a) V_1(s') - \sum_{s' \in S} P(s'|s,a) V_2(s') \right| \\
&\leq \alpha \max_{a \in A} \sum_{s' \in S} P(s'|s,a) \left| V_1(s') - V_2(s') \right| \\
&\leq \alpha \max_{a \in A} \left| V_1(s') - V_2(s') \right|
\end{aligned}
$$

Then, we prove contraction

We do the following

$$
\begin{aligned}
\left|BV_1(s) - BV_2(s)\right| &\leq \alpha \left| \max_{a\in A} \sum_{s'\in S} P(s'|s,a) V_1(s') - \max_{a\in A} \sum_{s'\in S} P(s'|s,a) V_2(s') \right| \\
&\leq \alpha \max_{a\in A} \left| \sum_{s'\in S} P(s'|s,a) V_1(s') - \sum_{s'\in S} P(s'|s,a) V_2(s') \right| \\
&\leq \alpha \max_{a\in A} \sum_{s'\in S} P(s'|s,a) \left|V_1(s') - V_2(s')\right| \\
&\leq \alpha \max_{a\in A} \left|V_1(s') - V_2(s')\right|
\end{aligned}
$$

Then, we prove contraction

We do the following

$$
\begin{aligned}
\left|BV_1(s) - BV_2(s)\right| &\leq \alpha \left| \max_{a\in A} \sum_{s'\in S} P(s'|s,a) V_1(s') - \max_{a\in A} \sum_{s'\in S} P(s'|s,a) V_2(s') \right| \\
&\leq \alpha \max_{a\in A} \left| \sum_{s'\in S} P(s'|s,a) V_1(s') - \sum_{s'\in S} P(s'|s,a) V_2(s') \right| \\
&\leq \alpha \max_{a\in A} \sum_{s'\in S} P(s'|s,a) \left|V_1(s') - V_2(s')\right| \\
&\leq \alpha \max_{a\in A} \left|V_1(s') - V_2(s')\right|
\end{aligned}
$$

Why?

First than anything, we have

- $\sum_{s' \in S} P\left(s'|s,a\right) = 1$

and

$$\left|\max_x f\left(x\right) - \max_x g\left(x\right)\right| \leq \max_x \left|f\left(x\right) - g\left(x\right)\right|$$

Why?

First than anything, we have

- $\sum_{s' \in S} P\left(s'|s,a\right) = 1$

and

$$\left|\max_{x} f\left(x\right) - \max_{x} g\left(x\right)\right| \leq \max_{x} \left|f\left(x\right) - g\left(x\right)\right|$$

Finally

By the fixed point remark

$$\max_{s \in S} \left| B\hat{V}(s) - V^*(s) \right| \leq \alpha \max_{s \in S} \left| \hat{V}(s) - V^*(s) \right|$$

Then

$$\hat{V} \longrightarrow V^*$$

Finally

By the fixed point remark

$$\max_{s \in S} \left| B\hat{V}(s) - V^*(s) \right| \leq \alpha \max_{s \in S} \left| \hat{V}(s) - V^*(s) \right|$$

Then

$$\hat{V} \longrightarrow V^*$$

Convergence of Value Iteration

How many iterations will it take to find optimal policy?

- For this we define

$$\|V\| = \max_{s} |U(s)|$$

Assume rewards in $[0, R_{\max})$

$$V^{*}(s) \leq \sum_{t=1}^{\infty} \alpha^{t} R_{max} = \frac{R_{\max}}{1-\alpha}$$

Convergence of Value Iteration

How many iterations will it take to find optimal policy?

- For this we define

$$\|V\| = \max_{s} |U(s)|$$

Assume rewards in $[0, R_{\max}]$

$$V^*(s) \leq \sum_{t=1}^{\infty} \alpha^t R_{max} = \frac{R_{\max}}{1-\alpha}$$

Therefore

Then letting V^k be value after k^{th} iteration

$$\max_{s \in S} \left| V^k(s) - V^*(s) \right| \leq \frac{\alpha^k R_{\max}}{1-\alpha}$$

Therefore

- We have linear convergence to optimal value function.

However

- The time to find an optimal policy depends on the separation value between value of optimal and second suboptimal policy.
  - This is difficult to bound.

Therefore

Then letting V^k be value after k^{th} iteration

$$\max_{s \in S} \left| V^k(s) - V^*(s) \right| \leq \frac{\alpha^k R_{\max}}{1-\alpha}$$

Therefore

- We have linear convergence to optimal value function.

However

- The time to find an optimal policy depends on the separation value between value of optimal and second suboptimal policy.
  - This is difficult to bound.

Therefore

Then letting V^k be value after k^{th} iteration

$$\max_{s \in S} \left| V^k(s) - V^*(s) \right| \leq \frac{\alpha^k R_{\max}}{1-\alpha}$$

Therefore

- We have linear convergence to optimal value function.

However

- The time to find an optimal policy depends on the separation value between value of optimal and second suboptimal policy.
    - This is difficult to bound.

Outline

Policy iteration algorithm

First Step

- Initialize policy $\hat{\pi}$ (For example Randomly)

Second Step

- Compute value of policy, $V^{\hat{\pi}}$

$$V^{\hat{\pi}}(s) \leftarrow R(s) + \alpha \max_{a \in A} \sum_{s' \in S} P(s'|s,a) V^{\hat{\pi}}(s'), \ \forall s \in S$$

Third Step

- Update π to be greedy policy with respect to $V^{\hat{\pi}}$

$$\hat{\pi}(s) \leftarrow \arg\max_{a \in A} \sum_{s' \in S} P(s'|s,a) V^{\hat{\pi}}(s'), \ \forall s \in S$$

Policy iteration algorithm

First Step

- Initialize policy $\hat{\pi}$ (For example Randomly)

Second Step

- Compute value of policy, $V^{\hat{\pi}}$

$$V^{\hat{\pi}}(s) \leftarrow R(s) + \alpha \max_{a \in A} \sum_{s' \in S} P(s'|s,a) V^{\hat{\pi}}(s'),\ \forall s \in S$$

Third Step

- Update π to be greedy policy with respect to $V^{\hat{\pi}}$

$$\hat{\pi}(s) \leftarrow \arg\max_{a \in A} \sum_{s' \in S} P(s'|s,a) V^{\hat{\pi}}(s'),\ \forall s \in S$$

Policy iteration algorithm

First Step

- Initialize policy $\hat{\pi}$ (For example Randomly)

Second Step

- Compute value of policy, $V^{\hat{\pi}}$

$$V^{\hat{\pi}}(s) \leftarrow R(s) + \alpha \max_{a \in A} \sum_{s' \in S} P(s'|s,a) V^{\hat{\pi}}(s'), \ \forall s \in S$$

Third Step

- Update π to be greedy policy with respect to $V^{\hat{\pi}}$

$$\hat{\pi}(s) \leftarrow \arg\max_{a \in A} \sum_{s' \in S} P(s'|s,a) V^{\hat{\pi}}(s'), \ \forall s \in S$$

Furthermore

Fourth Step

- If policy π changed in last iteration, return to the Second Step

Convergence

Convergence property of policy iteration $\pi \longrightarrow \pi^*$

- Proof involves showing that each iteration is also a contraction
  - I left this to you to figure out...

Interesting theoretical note

- Since number of policies is finite (though exponentially large), policy iteration converges to exact optimal policy

Convergence

Convergence property of policy iteration $\pi \longrightarrow \pi^*$

- Proof involves showing that each iteration is also a contraction
  - I left this to you to figure out...

Interesting theoretical note

- Since number of policies is finite (though exponentially large), policy iteration converges to exact optimal policy

Example

$$V^{\hat{\pi}}(s) \leftarrow R(s) + \alpha \max_{a \in A} \sum_{s' \in S} P(s'|s,a) V^{\hat{\pi}}(s'), \ \forall s \in S$$

0	0	0	+10
0	W	0	-1
Start $\rightarrow$ 0	0	0	0

P=1.0

P=0 | s | P=0