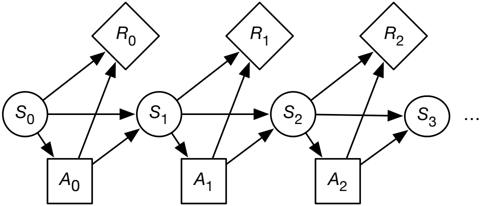
Second half of course. . .

- A belief network is a representation of conditional independence:
  in a total ordering of the variables, each variable is independent of its predecessors given it's parents
- Variable elimination computes the posterior probability of a variable given evidence by summing out the non-observed non-query variables.

Second half of course. . .

- A belief network is a representation of conditional independence:
  in a total ordering of the variables, each variable is independent of its predecessors given it's parents
- Variable elimination computes the posterior probability of a variable given evidence by summing out the non-observed non-query variables.
- A causal model predicts the effect of a intervention.
- A naive bayes model can be used for learning and help systems.
- Utility is a measure of preferences that is defined in terms of lotteries.
- A decision network has random nodes, decision nodes and a utility node. A policy has an expected utility.
- A Markov decision process can model ongoing activity with rewards, but only fully observable case is feasible.

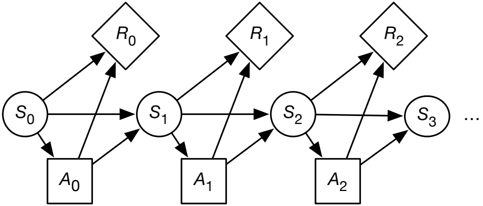
Markov Decision Processes



An MDP consists of:

- set S of states.
- set A of actions.

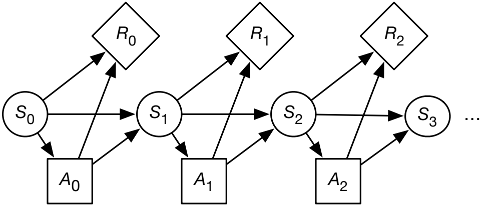
Markov Decision Processes



An MDP consists of:

- set S of states.
- set A of actions.
- $P(S_{t+1} \mid S_t, A_t)$ specifies the dynamics.

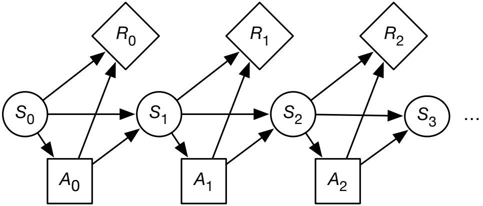
Markov Decision Processes



An MDP consists of:

- set S of states.
- set A of actions.
- $P(S_{t+1} \mid S_t, A_t)$ specifies the dynamics.
- $R(S_t, A_t)$ specifies the expected reward at time t.
  $R(s, a)$ is the expected reward of doing a in state s

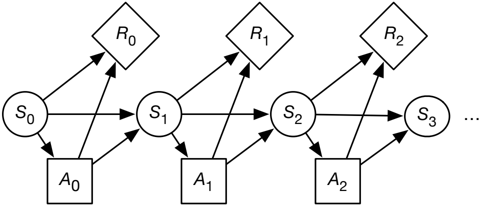
Markov Decision Processes



An MDP consists of:

- set S of states.
- set A of actions.
- $P(S_{t+1} \mid S_t, A_t)$ specifies the dynamics.
- $R(S_t, A_t)$ specifies the expected reward at time t.
  $R(s, a)$ is the expected reward of doing a in state s
- γ is discount factor.

Markov Decision Processes



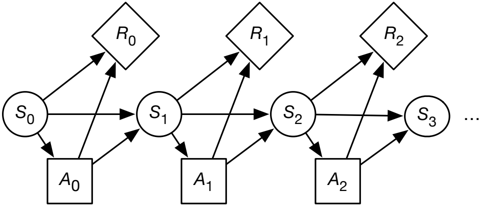
An MDP consists of:

- set S of states.
- set A of actions.
- $P(S_{t+1} \mid S_t, A_t)$ specifies the dynamics.
- $R(S_t, A_t)$ specifies the expected reward at time t.
  $R(s, a)$ is the expected reward of doing a in state s
- γ is discount factor.

Discounted reward: If V_t is the value obtained from time step t

$$V_t =$$

Markov Decision Processes



An MDP consists of:

- set S of states.
- set A of actions.
- $P(S_{t+1} \mid S_t, A_t)$ specifies the dynamics.
- $R(S_t, A_t)$ specifies the expected reward at time t.
  $R(s, a)$ is the expected reward of doing a in state s
- γ is discount factor.

Discounted reward: If V_t is the value obtained from time step t

$$V_t = r_t + \gamma V_{t+1}$$

Value of a Policy

Given a policy π:

- $Q^{\pi}(s, a)$, where a is an action and s is a state, is the expected value of doing a in state s, then following policy π.

Value of a Policy

Given a policy π:

- $Q^\pi(s, a)$, where a is an action and s is a state, is the expected value of doing a in state s, then following policy π.
- $V^\pi(s)$, where s is a state, is the expected value of following policy π in state s.

Value of a Policy

Given a policy π:

- $Q^{\pi}(s, a)$, where a is an action and s is a state, is the expected value of doing a in state s, then following policy π.
- $V^{\pi}(s)$, where s is a state, is the expected value of following policy π in state s.
- Q^{π} and V^{π} can be defined mutually recursively:

$$
\begin{aligned}
V^{\pi}(s) &= \\
Q^{\pi}(s, a) &=
\end{aligned}
$$

Value of a Policy

Given a policy π:

- $Q^{\pi}(s,a)$, where a is an action and s is a state, is the expected value of doing a in state s, then following policy π.
- $V^{\pi}(s)$, where s is a state, is the expected value of following policy π in state s.
- Q^{π} and V^{π} can be defined mutually recursively:

$$
\begin{aligned}
V^{\pi}(s) &= Q^{\pi}(s,\pi(s)) \\
Q^{\pi}(s,a) &=
\end{aligned}
$$

Value of a Policy

Given a policy π:

- $Q^\pi(s, a)$, where a is an action and s is a state, is the expected value of doing a in state s, then following policy π.
- $V^\pi(s)$, where s is a state, is the expected value of following policy π in state s.
- Q^π and V^π can be defined mutually recursively:

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

Value of the Optimal Policy

- $Q^*(s,a)$, where a is an action and s is a state, is the expected value of doing a in state s, then following the optimal policy.

Value of the Optimal Policy

- $Q^*(s, a)$, where a is an action and s is a state, is the expected value of doing a in state s, then following the optimal policy.
- $V^*(s)$, where s is a state, is the expected value of following the optimal policy in state s.

Value of the Optimal Policy

- $Q^*(s, a)$, where a is an action and s is a state, is the expected value of doing a in state s, then following the optimal policy.
- $V^*(s)$, where s is a state, is the expected value of following the optimal policy in state s.
- Q^* and V^* can be defined mutually recursively:

$$
\begin{aligned}
Q^*(s, a) &= \\
V^*(s) &= \\
\pi^*(s) &=
\end{aligned}
$$

Value of the Optimal Policy

- $Q^*(s, a)$, where a is an action and s is a state, is the expected value of doing a in state s, then following the optimal policy.
- $V^*(s)$, where s is a state, is the expected value of following the optimal policy in state s.
- Q^* and V^* can be defined mutually recursively:

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

Value of the Optimal Policy

- $Q^*(s, a)$, where a is an action and s is a state, is the expected value of doing a in state s, then following the optimal policy.
- $V^*(s)$, where s is a state, is the expected value of following the optimal policy in state s.
- Q^* and V^* can be defined mutually recursively:

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

Value of the Optimal Policy

- $Q^*(s,a)$, where a is an action and s is a state, is the expected value of doing a in state s, then following the optimal policy.
- $V^*(s)$, where s is a state, is the expected value of following the optimal policy in state s.
- Q^* and V^* can be defined mutually recursively:

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

Asynchronous VI: storing $Q[s, a]$

- Repeat forever:
  - Select state s, action a
  - $Q[s, a] \leftarrow$

Asynchronous VI: storing $Q[s, a]$

- Repeat forever:
  - Select state s, action a
  - $Q[s,a] \leftarrow R(s,a) + \gamma \sum_{s'} P(s' \mid s,a) \left(\max_{a'} Q[s',a'] \right)$

Example: Go (the game)

- Go is a board game on 19×19 grid. Each position can have black stone or white stone or no stone.
- Approximately $2 * 10^{170}$ states (legal board positions)

AlphaGo / AlphaZero:

- "Store" $V(s)$ and $\pi(s)$.

Example: Go (the game)

- Go is a board game on 19×19 grid. Each position can have black stone or white stone or no stone.
- Approximately $2 * 10^{170}$ states (legal board positions)

AlphaGo / AlphaZero:

- "Store" $V(s)$ and $\pi(s)$.
- Assume opponent is also playing $\pi(s)$ + some randomness (for exploration)

Example: Go (the game)

- Go is a board game on 19×19 grid. Each position can have black stone or white stone or no stone.
- Approximately $2 * 10^{170}$ states (legal board positions)

AlphaGo / AlphaZero:

- "Store" $V(s)$ and $\pi(s)$.
- Assume opponent is also playing $\pi(s)$ + some randomness (for exploration)
- Compute expected values using forward sampling (Monte-Carlo tree search); play the game to end, multiple times and average.
- Try a few other actions to see if one is better (exploration).

Example: Go (the game)

- Go is a board game on 19×19 grid. Each position can have black stone or white stone or no stone.
- Approximately $2 * 10^{170}$ states (legal board positions)

AlphaGo / AlphaZero:

- "Store" $V(s)$ and $\pi(s)$.
- Assume opponent is also playing $\pi(s)$ + some randomness (for exploration)
- Compute expected values using forward sampling (Monte-Carlo tree search); play the game to end, multiple times and average.
- Try a few other actions to see if one is better (exploration).
- Use neural networks to represent $V(s)$ and $\pi(s)$ (see CPSC 340)

Difficulty in solving Go

Why is Go difficult?

- Enormous state space $2 * 10^{170}$ states (number of atoms in observable universe $\approx 10^{80}$).
- Many moves (up to 361)

Difficulty in solving Go

Why is Go difficult?

- Enormous state space $2 * 10^{170}$ states (number of atoms in observable universe $\approx 10^{80}$).
- Many moves (up to 361)

Why easy?

Difficulty in solving Go

Why is Go difficult?

- Enormous state space $2 * 10^{170}$ states (number of atoms in observable universe $\approx 10^{80}$).
- Many moves (up to 361)

Why easy?

- Well defined rules (we know the transition rules and goal)

Difficulty in solving Go

Why is Go difficult?

- Enormous state space $2 * 10^{170}$ states (number of atoms in observable universe $\approx 10^{80}$).
- Many moves (up to 361)

Why easy?

- Well defined rules (we know the transition rules and goal)
- Simple utility $(0/1)$.
  So expected value $=$ probability of winning

Difficulty in solving Go

Why is Go difficult?

- Enormous state space $2 * 10^{170}$ states (number of atoms in observable universe $\approx 10^{80}$).
- Many moves (up to 361)

Why easy?

- Well defined rules (we know the transition rules and goal)
- Simple utility (0/1).
  So expected value = probability of winning
- Fully observable, with a well define state (board position)

Difficulty in solving Go

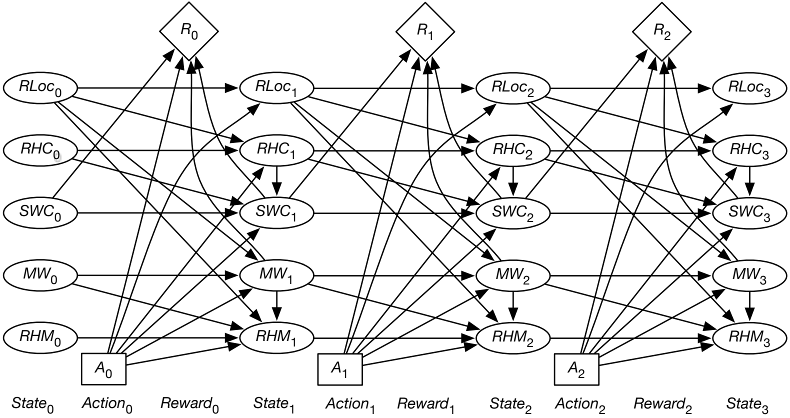
Why is Go difficult?

- Enormous state space $2 * 10^{170}$ states (number of atoms in observable universe $\approx 10^{80}$).
- Many moves (up to 361)

Why easy?

- Well defined rules (we know the transition rules and goal)
- Simple utility (0/1).
  So expected value = probability of winning
- Fully observable, with a well define state (board position)
- Two players that are adversaries. (Multiple-agent reasoning is *much* more difficult with partial observability (eg. simple interaction such as poker).

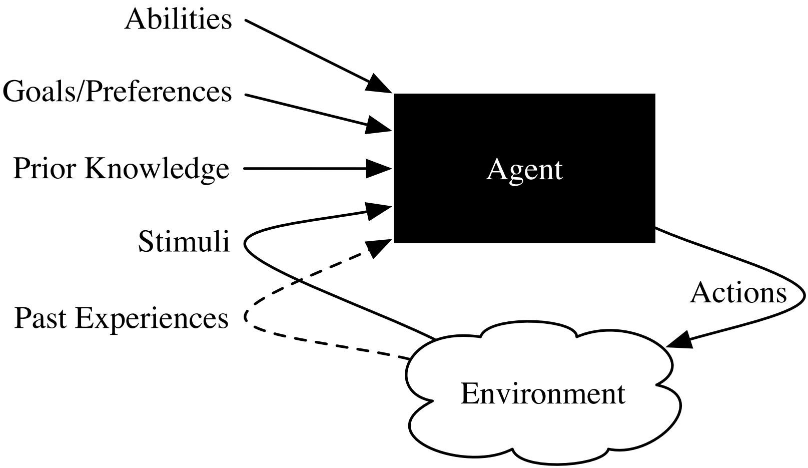
Dynamic Decision Network



Parents of A_i are all of the $State_i$ variables.

Overview

Agents acting in an environment



Dimensions of Complexity

Dimension	Values
Modularity	flat, modular, hierarchical

Dimensions of Complexity

Dimension	Values
Modularity	flat, modular, hierarchical
Planning horizon	non-planning, finite stage, indefinite stage, infinite stage

Dimensions of Complexity

Dimension	Values
Modularity	flat, modular, hierarchical
Planning horizon	non-planning, finite stage, indefinite stage, infinite stage
Representation	states, features, relations

Dimensions of Complexity

Dimension	Values
Modularity	flat, modular, hierarchical
Planning horizon	non-planning, finite stage, indefinite stage, infinite stage
Representation	states, features, relations
Computational limits	perfect rationality, bounded rationality

Dimensions of Complexity

Dimension	Values
Modularity	flat, modular, hierarchical
Planning horizon	non-planning, finite stage, indefinite stage, infinite stage
Representation	states, features, relations
Computational limits	perfect rationality, bounded rationality
Learning	knowledge is given, knowledge is learned

Dimensions of Complexity

Dimension	Values
Modularity	flat, modular, hierarchical
Planning horizon	non-planning, finite stage, indefinite stage, infinite stage
Representation	states, features, relations
Computational limits	perfect rationality, bounded rationality
Learning	knowledge is given, knowledge is learned
Sensing uncertainty	fully observable, partially observable

Dimensions of Complexity

Dimension	Values
Modularity	flat, modular, hierarchical
Planning horizon	non-planning, finite stage, indefinite stage, infinite stage
Representation	states, features, relations
Computational limits	perfect rationality, bounded rationality
Learning	knowledge is given, knowledge is learned
Sensing uncertainty	fully observable, partially observable
Effect uncertainty	deterministic, stochastic

Dimensions of Complexity

Dimension	Values
Modularity	flat, modular, hierarchical
Planning horizon	non-planning, finite stage, indefinite stage, infinite stage
Representation	states, features, relations
Computational limits	perfect rationality, bounded rationality
Learning	knowledge is given, knowledge is learned
Sensing uncertainty	fully observable, partially observable
Effect uncertainty	deterministic, stochastic
Preference	goals, complex preferences

Dimensions of Complexity

Dimension	Values
Modularity	flat, modular, hierarchical
Planning horizon	non-planning, finite stage, indefinite stage, infinite stage
Representation	states, features, relations
Computational limits	perfect rationality, bounded rationality
Learning	knowledge is given, knowledge is learned
Sensing uncertainty	fully observable, partially observable
Effect uncertainty	deterministic, stochastic
Preference	goals, complex preferences
Number of agents	single agent, multiple agents

Dimensions of Complexity

Dimension	Values
Modularity	flat, modular, hierarchical
Planning horizon	non-planning, finite stage, indefinite stage, infinite stage
Representation	states, features, relations
Computational limits	perfect rationality, bounded rationality
Learning	knowledge is given, knowledge is learned
Sensing uncertainty	fully observable, partially observable
Effect uncertainty	deterministic, stochastic
Preference	goals, complex preferences
Number of agents	single agent, multiple agents
Interaction	offline, online

State-space Search

Dimension	Values
Modularity	flat, modular, hierarchical
Planning horizon	non-planning, finite stage, indefinite stage, infinite stage
Representation	states, features, relations
Computational limits	perfect rationality, bounded rationality
Learning	knowledge is given, knowledge is learned
Sensing uncertainty	fully observable, partially observable
Effect uncertainty	deterministic, stochastic
Preference	goals, complex preferences
Number of agents	single agent, multiple agents
Interaction	offline, online

Classical Forward/Regression Planning

Dimension	Values
Modularity	flat, modular, hierarchical
Planning horizon	non-planning, finite stage, indefinite stage, infinite stage
Representation	states, features,relations
Computational limits	perfect rationality, bounded rationality
Learning	knowledge is given, knowledge is learned
Sensing uncertainty	fully observable, partially observable
Effect uncertainty	deterministic, stochastic
Preference	goals, complex preferences
Number of agents	single agent, multiple agents
Interaction	offline, online

CSP Planning

Dimension	Values
Modularity	flat, modular, hierarchical
Planning horizon	non-planning, finite stage, indefinite stage, infinite stage
Representation	states, features, relations
Computational limits	perfect rationality, bounded rationality
Learning	knowledge is given, knowledge is learned
Sensing uncertainty	fully observable, partially observable
Effect uncertainty	deterministic, stochastic
Preference	goals, complex preferences
Number of agents	single agent, multiple agents
Interaction	offline, online

Decision Networks

Dimension	Values
Modularity	flat, modular, hierarchical
Planning horizon	non-planning, finite stage, indefinite stage, infinite stage
Representation	states, features, relations
Computational limits	perfect rationality, bounded rationality
Learning	knowledge is given, knowledge is learned
Sensing uncertainty	fully observable, partially observable
Effect uncertainty	deterministic, stochastic
Preference	goals, complex preferences
Number of agents	single agent, multiple agents
Interaction	offline, online

Markov Decision Processes (MDPs)

Dimension	Values
Modularity	flat, modular, hierarchical
Planning horizon	non-planning, finite stage, indefinite stage, infinite stage
Representation	states, features, relations
Computational limits	perfect rationality, bounded rationality
Learning	knowledge is given, knowledge is learned
Sensing uncertainty	fully observable, partially observable
Effect uncertainty	deterministic, stochastic
Preference	goals, complex preferences
Number of agents	single agent, multiple agents
Interaction	offline, online

Humans

Dimension	Values
Modularity	flat, modular, hierarchical
Planning horizon	non-planning, finite stage, indefinite stage, infinite stage
Representation	states, features, relations
Computational limits	perfect rationality, bounded rationality
Learning	knowledge is given, knowledge is learned
Sensing uncertainty	fully observable, partially observable
Effect uncertainty	deterministic, stochastic
Preference	goals, complex preferences
Number of agents	single agent, multiple agents
Interaction	offline, online

Review: Searching

- Generic search algorithm expands paths in frontier, until it expands a goal

Review: Searching

- Generic search algorithm expands paths in frontier, until it expands a goal
- Frontier is a stack $\longrightarrow$ depth-firt search
- Frontier is a queue $\longrightarrow$ breadth-first search
- Frontier is a priority queue ordered by path cost $\longrightarrow$ least-cost-first search

Review: Searching

- Generic search algorithm expands paths in frontier, until it expands a goal
- Frontier is a stack $\longrightarrow$ depth-firt search
- Frontier is a queue $\longrightarrow$ breadth-first search
- Frontier is a priority queue ordered by path cost $\longrightarrow$ least-cost-first search
- Frontier is a priority queue ordered by $f(p) = cost(p) + h(p)$ $\longrightarrow$ A^* search

Review: Searching

- Generic search algorithm expands paths in frontier, until it expands a goal
- Frontier is a stack $\longrightarrow$ depth-firt search
- Frontier is a queue $\longrightarrow$ breadth-first search
- Frontier is a priority queue ordered by path cost $\longrightarrow$ least-cost-first search
- Frontier is a priority queue ordered by $f(p) = cost(p) + h(p)$ $\longrightarrow$ A^* search
- A^* finds shortest path if h is admissible

Review: Searching

- Generic search algorithm expands paths in frontier, until it expands a goal
- Frontier is a stack $\longrightarrow$ depth-firt search
- Frontier is a queue $\longrightarrow$ breadth-first search
- Frontier is a priority queue ordered by path cost $\longrightarrow$ least-cost-first search
- Frontier is a priority queue ordered by $f(p) = cost(p) + h(p)$ $\longrightarrow$ A^* search
- A^* finds shortest path if h is admissible
- Cycle pruning prunes paths that loop back on themselves
- Multiple-path pruning prunes paths to nodes that have already been expanded.

Review: Searching

- Generic search algorithm expands paths in frontier, until it expands a goal
- Frontier is a stack $\longrightarrow$ depth-firt search
- Frontier is a queue $\longrightarrow$ breadth-first search
- Frontier is a priority queue ordered by path cost $\longrightarrow$ least-cost-first search
- Frontier is a priority queue ordered by $f(p) = cost(p) + h(p)$ $\longrightarrow$ A^* search
- A^* finds shortest path if h is admissible
- Cycle pruning prunes paths that loop back on themselves
- Multiple-path pruning prunes paths to nodes that have already been expanded.
- Depth-first branch-and-bound combines space saving of depth-first search with the optimality of A^*.

Review: Searching

- Generic search algorithm expands paths in frontier, until it expands a goal
- Frontier is a stack $\longrightarrow$ depth-firt search
- Frontier is a queue $\longrightarrow$ breadth-first search
- Frontier is a priority queue ordered by path cost $\longrightarrow$ least-cost-first search
- Frontier is a priority queue ordered by $f(p) = cost(p) + h(p)$ $\longrightarrow A^*$ search
- A^* finds shortest path if h is admissible
- Cycle pruning prunes paths that loop back on themselves
- Multiple-path pruning prunes paths to nodes that have already been expanded.
- Depth-first branch-and-bound combines space saving of depth-first search with the optimality of A^*.
  (but doesn't have space saving with multiple-path pruning)

Constraint Satisfaction Problems

- Constraint satisfaction problems defined in terms of variables, domains, constraints

Constraint Satisfaction Problems

- Constraint satisfaction problems defined in terms of variables, domains, constraints
- Constraint satisfactions problems can be solved with search

Constraint Satisfaction Problems

- Constraint satisfaction problems defined in terms of variables, domains, constraints
- Constraint satisfactions problems can be solved with search
- An arc $\langle X, c\rangle$ is arc consistent if for every value for X there are values for the other variables that satisfies c.

Constraint Satisfaction Problems

- Constraint satisfaction problems defined in terms of variables, domains, constraints
- Constraint satisfactions problems can be solved with search
- An arc $\langle X, c \rangle$ is arc consistent if for every value for X there are values for the other variables that satisfies c.
- Arc consistency can be used to simplify the search space
- Domain splitting can be used to find solutions.

Constraint Satisfaction Problems

- Constraint satisfaction problems defined in terms of variables, domains, constraints
- Constraint satisfactions problems can be solved with search
- An arc $\langle X, c \rangle$ is arc consistent if for every value for X there are values for the other variables that satisfies c.
- Arc consistency can be used to simplify the search space
- Domain splitting can be used to find solutions.
- Local search maintains a complete assignment of a value to each variable, and has a mix of improving and randomized steps.

Constraint Satisfaction Problems

- Constraint satisfaction problems defined in terms of variables, domains, constraints
- Constraint satisfactions problems can be solved with search
- An arc $\langle X, c\rangle$ is arc consistent if for every value for X there are values for the other variables that satisfies c.
- Arc consistency can be used to simplify the search space
- Domain splitting can be used to find solutions.
- Local search maintains a complete assignment of a value to each variable, and has a mix of improving and randomized steps.
- There are many variants of local search to find solutions or minimize an evaluation function

Planning

- STRIPS is used to represent actions in terms of
  - preconditions
  - effects

Planning

- STRIPS is used to represent actions in terms of
  - preconditions
  - effects
- Planning is finding a sequence of actions to achieve a goal from an initial state. Planning is achieved by mapping to a search problem .

Planning

- STRIPS is used to represent actions in terms of
  - preconditions
  - effects
- Planning is finding a sequence of actions to achieve a goal from an initial state. Planning is achieved by mapping to a search problem .
- A forward planner searches from the initial state to a goal state in a state-space graph.

Planning

- STRIPS is used to represent actions in terms of
  - preconditions
  - effects
- Planning is finding a sequence of actions to achieve a goal from an initial state. Planning is achieved by mapping to a search problem .
- A forward planner searches from the initial state to a goal state in a state-space graph.
- A regression planner searchers in a graph where nodes are propositions (representing subgoals), and the start node is the goal to be solved.

Planning

- STRIPS is used to represent actions in terms of
  - preconditions
  - effects
- Planning is finding a sequence of actions to achieve a goal from an initial state. Planning is achieved by mapping to a search problem .
- A forward planner searches from the initial state to a goal state in a state-space graph.
- A regression planner searchers in a graph where nodes are propositions (representing subgoals), and the start node is the goal to be solved.
- A CSP planner converts a planning problem into a CSP for a fixed planning horizon.

Probability and Belief Networks

- Probability is defined in terms of measures over possible worlds
- The probability of a proposition is the measure of the set of worlds in which the proposition is true.
- Conditioning on evidence: make the worlds incompatible with the evidence have measure 0, and multiply the others by a constant to get a measure.

Probability and Belief Networks

- Probability is defined in terms of measures over possible worlds
- The probability of a proposition is the measure of the set of worlds in which the proposition is true.
- Conditioning on evidence: make the worlds incompatible with the evidence have measure 0, and multiply the others by a constant to get a measure.
- A belief network is a representation of conditional independence:
  in a total ordering of the variables, each variable is independent of its predecessors given it's parents

Probability and Belief Networks

- Probability is defined in terms of measures over possible worlds
- The probability of a proposition is the measure of the set of worlds in which the proposition is true.
- Conditioning on evidence: make the worlds incompatible with the evidence have measure 0, and multiply the others by a constant to get a measure.
- A belief network is a representation of conditional independence:
  in a total ordering of the variables, each variable is independent of its predecessors given it's parents
- Variable elimination computes the posterior probability of a variable given evidence by summing out the non-observed non-query variables

Probability and Belief Networks

- Probability is defined in terms of measures over possible worlds
- The probability of a proposition is the measure of the set of worlds in which the proposition is true.
- Conditioning on evidence: make the worlds incompatible with the evidence have measure 0, and multiply the others by a constant to get a measure.
- A belief network is a representation of conditional independence:
  in a total ordering of the variables, each variable is independent of its predecessors given it's parents
- Variable elimination computes the posterior probability of a variable given evidence by summing out the non-observed non-query variables
- A causal model predicts the effect of a intervention.

Probability and Belief Networks

- Probability is defined in terms of measures over possible worlds
- The probability of a proposition is the measure of the set of worlds in which the proposition is true.
- Conditioning on evidence: make the worlds incompatible with the evidence have measure 0, and multiply the others by a constant to get a measure.
- A belief network is a representation of conditional independence:
  in a total ordering of the variables, each variable is independent of its predecessors given it's parents
- Variable elimination computes the posterior probability of a variable given evidence by summing out the non-observed non-query variables
- A causal model predicts the effect of a intervention.
- A naive Bayes model can be used for learning and help systems.

Decision Networks

Decision network:

- DAG with three sorts of nodes: decision (rectangle), random (ellipse), utility (diamond)
- Domain for each decision and random node (no domain for utility)
- Factor for each random node and the utility (no initial factor for decision nodes)

Decision Networks

Decision network:

- DAG with three sorts of nodes: decision (rectangle), random (ellipse), utility (diamond)
- Domain for each decision and random node (no domain for utility)
- Factor for each random node and the utility (no initial factor for decision nodes)
- A decision function maps assignments to the parents of a decision node to a value in the domain of the decision node.
- A policy assigns a decision function to each decision node.

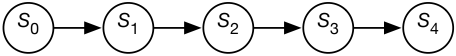
Decision Networks

Decision network:

- DAG with three sorts of nodes: decision (rectangle), random (ellipse), utility (diamond)
- Domain for each decision and random node (no domain for utility)
- Factor for each random node and the utility (no initial factor for decision nodes)
- A decision function maps assignments to the parents of a decision node to a value in the domain of the decision node.
- A policy assigns a decision function to each decision node.
- Find optimal policy: sum out random variables until a decision variable D is in a factor F where all of the other variables in F are parents of D; then maximize D.

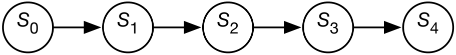
Markov Models

- A Markov chain is a particular sort of belief network that can extend indefinitely:

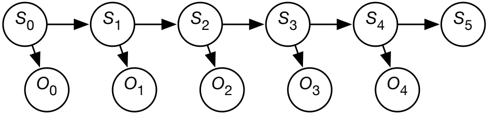


Markov Models

- A Markov chain is a particular sort of belief network that can extend indefinitely:

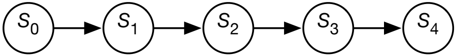


- A hidden Markov model (HMM) is a particular sort of belief network that can extend indefinitely:

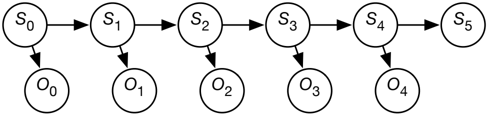


Markov Models

- A Markov chain is a particular sort of belief network that can extend indefinitely:



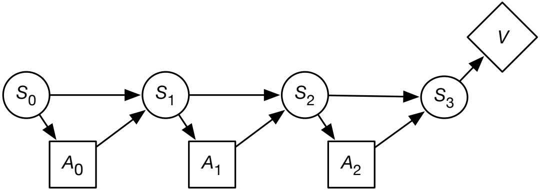
- A hidden Markov model (HMM) is a particular sort of belief network that can extend indefinitely:



- Hidden Markov models can be used for localization and simple language models

Markov Decision Process (MDP)

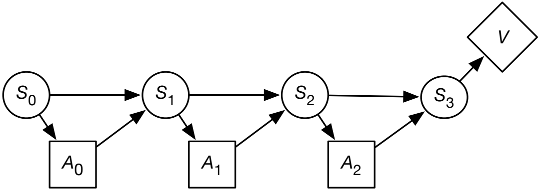
An MDP combines decision network and Markov chain:



- Initial Factors:

Markov Decision Process (MDP)

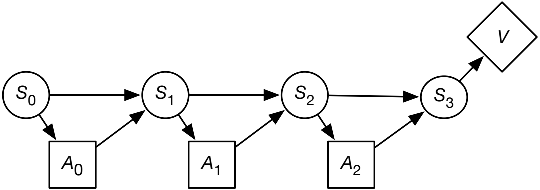
An MDP combines decision network and Markov chain:



- Initial Factors: $P(S_0)$, $P(S_{i+1} \mid S_i, A_i)$ and $V(S_3)$.

Markov Decision Process (MDP)

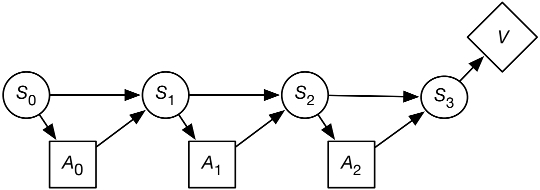
An MDP combines decision network and Markov chain:



- Initial Factors: $P(S_0)$, $P(S_{i+1} \mid S_i, A_i)$ and $V(S_3)$.
- Sum out

Markov Decision Process (MDP)

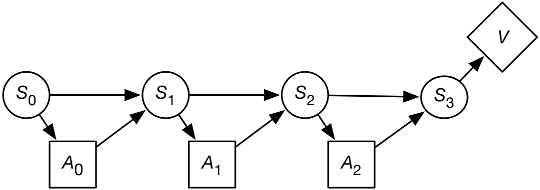
An MDP combines decision network and Markov chain:



- Initial Factors: $P(S_0)$, $P(S_{i+1} \mid S_i, A_i)$ and $V(S_3)$.
- Sum out S_3, create a factor

Markov Decision Process (MDP)

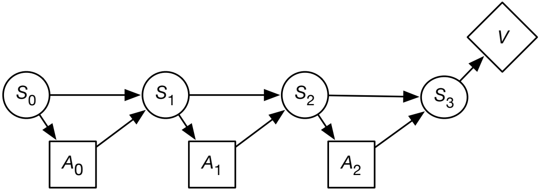
An MDP combines decision network and Markov chain:



- Initial Factors: $P(S_0)$, $P(S_{i+1} \mid S_i, A_i)$ and $V(S_3)$.
- Sum out S_3, create a factor $Q(S_2, A_2)$
- Maximize

Markov Decision Process (MDP)

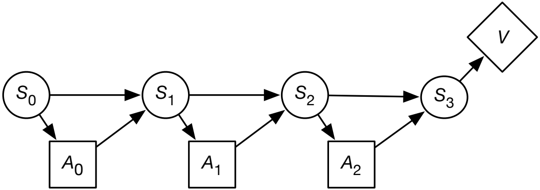
An MDP combines decision network and Markov chain:



- Initial Factors: $P(S_0)$, $P(S_{i+1} \mid S_i, A_i)$ and $V(S_3)$.
- Sum out S_3, create a factor $Q(S_2, A_2)$
- Maximize A_2, create factor

Markov Decision Process (MDP)

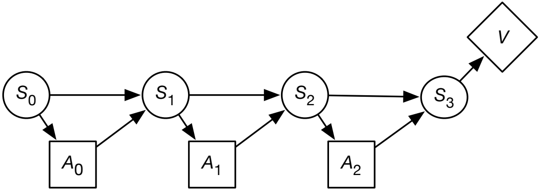
An MDP combines decision network and Markov chain:



- Initial Factors: $P(S_0)$, $P(S_{i+1} \mid S_i, A_i)$ and $V(S_3)$.
- Sum out S_3, create a factor $Q(S_2, A_2)$
- Maximize A_2, create factor $V(S_2)$ & decision function $\pi(S_2)$.
- This is the same form as the original problem, with one less time step

Markov Decision Process (MDP)

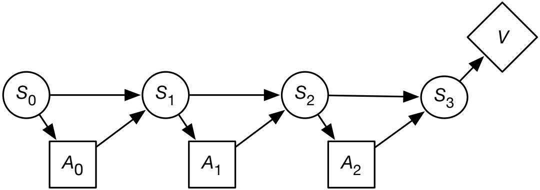
An MDP combines decision network and Markov chain:



- Initial Factors: $P(S_0)$, $P(S_{i+1} \mid S_i, A_i)$ and $V(S_3)$.
- Sum out S_3, create a factor $Q(S_2, A_2)$
- Maximize A_2, create factor $V(S_2)$ & decision function $\pi(S_2)$.
- This is the same form as the original problem, with one less time step $\rightarrow$ solve recursively.

Markov Decision Process (MDP)

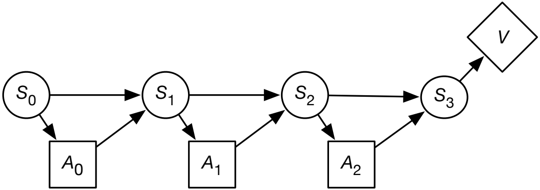
An MDP combines decision network and Markov chain:



- Initial Factors: $P(S_0)$, $P(S_{i+1} \mid S_i, A_i)$ and $V(S_3)$.
- Sum out S_3, create a factor $Q(S_2, A_2)$
- Maximize A_2, create factor $V(S_2)$ & decision function $\pi(S_2)$.
- This is the same form as the original problem, with one less time step $\rightarrow$ solve recursively.
- This works for arbitrary stages (number of times)

Markov Decision Process (MDP)

An MDP combines decision network and Markov chain:



- Initial Factors: $P(S_0)$, $P(S_{i+1} \mid S_i, A_i)$ and $V(S_3)$.
- Sum out S_3, create a factor $Q(S_2, A_2)$
- Maximize A_2, create factor $V(S_2)$ & decision function $\pi(S_2)$.
- This is the same form as the original problem, with one less time step $\rightarrow$ solve recursively.
- This works for arbitrary stages (number of times)
- An MDP extends indefinitely, and often includes rewards at each time. Reinforcement learning typically works by estimating $Q(S, A)$. Assumes fully observable environment.

Overview of Course

	dynamics	observable	repr	stage	preference	rationality
search	det	fully	states	indef	goals	perfect
CSPs	det	fully	feats	static	—	perfect
SLS	det	fully	feats	static	—	bounded
planning	det	fully	feats	indef	goals	perfect
belief nets	stoch	partial	feats	static	—	perfect
stoch siml	stoch	partial	feats	static	—	bounded
decision nets	stoch	partial	feats	finite	utility	perfect
Markov models	stoch	partial	states	infinite	—	perfect
MDP	stoch	fully	states	infinite	utility	perfect