Artificial Intelligence Techniques
CS4375
Lecture 7: Planning under transition uncertainty

Matthijs Spaan

Delft University of Technology

Delft, The Netherlands

September 26, 2023

Decision Making under Uncertainty

- Intelligent distributed systems are becoming ubiquitous
- Devices can sense, compute, act and interact

- How to make the right decisions?
- Key issue: uncertainty

Decision Making under Uncertainty

Markov Models		Do we have control over the state transitions?	
		No	Yes
Are the states completely observable?	Yes	Markov Chain	Markov Decision Process (MDP)
	No	Hidden Markov Model (HMM)	Partially Observable Markov Decision Process (POMDP)

(Littman, POMDP FAQ)

Outline for the remainder of the course

- Planning under transition uncertainty (MDP)
- Planning under sensing uncertainty (POMDP)
- Reinforcement Learning (model-free)
- Reinforcement Learning (model-based)
- Advanced RL topics (inverse RL, safe RL)
- Multiple actors

Pointers to follow-up courses

- CS4400 Deep Reinforcement Learning (Q2)
- CS4345 Seminar Formal Methods for Learned Systems (Q3)
- CS4210-A Algorithms for Intelligent Decision Making (Q3)
- CS4210-B Intelligent Decision Making Project (Q4)

Outline for today

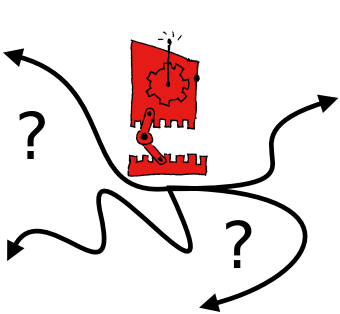
1. Planning under uncertainty

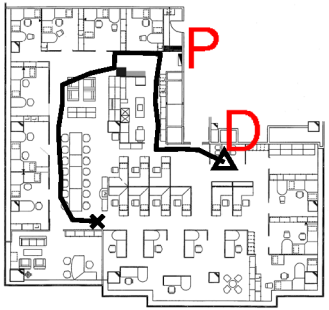
2. Markov Decision Processes
   - Model
   - Example
   - Policies and value functions
   - Algorithms

Planning under uncertainty

Introduction

- Goal in Artificial Intelligence: to build intelligent agents.
- Our definition of “intelligent”: perform an assigned task as well as possible.
- Problem: how to act?
- We will explicitly model uncertainty.

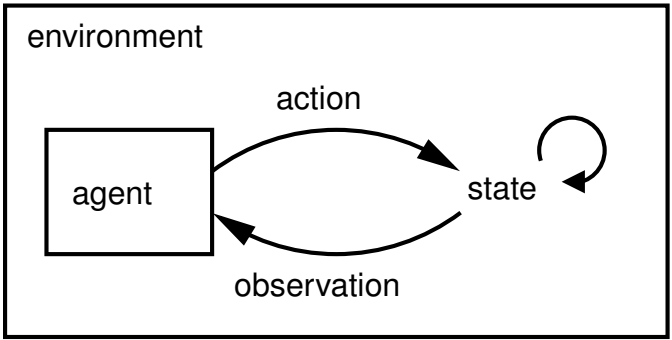




Agents

- An agent is a (rational) decision maker who is able to perceive its external (physical) environment and act autonomously upon it.
- Rationality means reaching the optimum of a performance measure.
- Examples: humans, robots, some software programs.

Agents



- It is useful to think of agents as being involved in a perception-action loop with their environment.
- But how do we make the right decisions?

Planning

Planning:

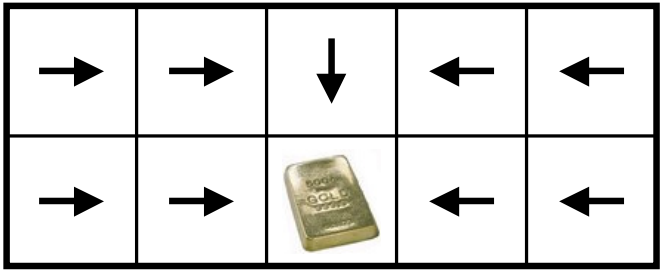
- A plan tells an agent how to act.
- For instance
  - A sequence of actions to reach a goal.
  - What to do in a particular situation.
- We need to model:
  - the agent's actions
  - its environment
  - its task

We will model planning as a sequence of decisions.

Classical planning

- Classical planning: sequence of actions from start to goal.
- Task: robot should get to gold as quickly as possible.
- Actions: $\rightarrow \downarrow \leftarrow \uparrow$
- Limitations:
  - New plan for each start state.
  - Environment is deterministic.
- Three optimal plans: $\rightarrow \rightarrow \downarrow, \rightarrow \downarrow \rightarrow, \downarrow \rightarrow \rightarrow$.

Conditional planning



- Assume our robot has noisy actions (wheel slip, overshoot).
- We need conditional plans.
- Map situations to actions.

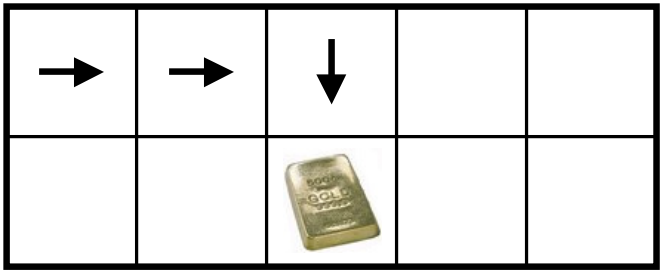
Decision-theoretic planning

−0.1	−0.1	−0.1	−0.1	−0.1
−0.1	−0.1	10	−0.1	−0.1

- Positive reward when reaching goal, small penalty for all other actions.
- Agent's plan maximizes **value**: the sum of future rewards.
- Decision-theoretic planning successfully handles noise in acting and sensing.

Decision-theoretic planning

Plan #1:

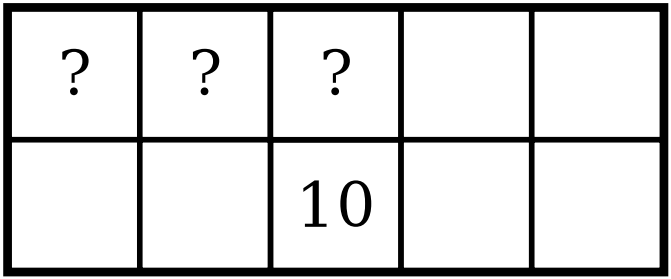


Reward:

−0.1	−0.1	−0.1	−0.1	−0.1
−0.1	−0.1	10	−0.1	−0.1

Decision-theoretic planning

Values of this plan:



Reward:

−0.1	−0.1	−0.1	−0.1	−0.1
−0.1	−0.1	10	−0.1	−0.1

Decision-theoretic planning

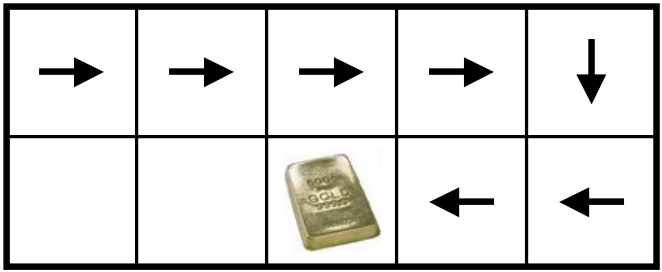
Values of this plan:

Reward:

−0.1	−0.1	−0.1	−0.1	−0.1
−0.1	−0.1	10	−0.1	−0.1

Decision-theoretic planning

Plan #2:



Reward:

−0.1	−0.1	−0.1	−0.1	−0.1
−0.1	−0.1	10	−0.1	−0.1

Decision-theoretic planning

Values of this plan:

?	?	?	?	?
		10	?	?

Reward:

−0.1	−0.1	−0.1	−0.1	−0.1
−0.1	−0.1	10	−0.1	−0.1

Decision-theoretic planning

Values of this plan:

9.3	9.4	9.5	9.6	9.7
		10	9.9	9.8

Reward:

−0.1	−0.1	−0.1	−0.1	−0.1
−0.1	−0.1	10	−0.1	−0.1

Decision-theoretic planning

Optimal values (encode optimal plan):

9.7	9.8	9.9	9.8	9.7
9.8	9.9	10	9.9	9.8

Reward:

−0.1	−0.1	−0.1	−0.1	−0.1
−0.1	−0.1	10	−0.1	−0.1

Markov Decision Processes

Model

Sequential decision making under uncertainty

- Uncertainty is abundant in **real-world planning** domains.
- **Bayesian** approach $\Rightarrow$ probabilistic models.

Main assumptions:

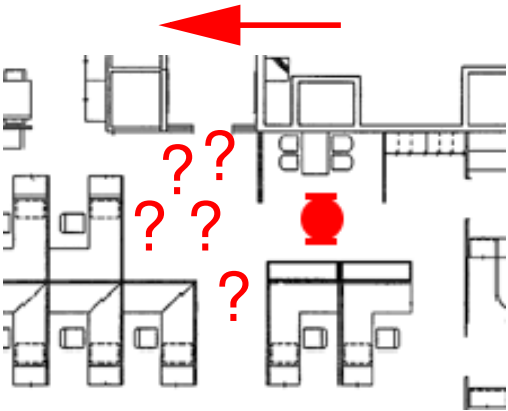
Sequential decisions: problems are formulated as a sequence of "independent" decisions;

Markovian environment: the state at time t depends only on the events at time $t-1$;

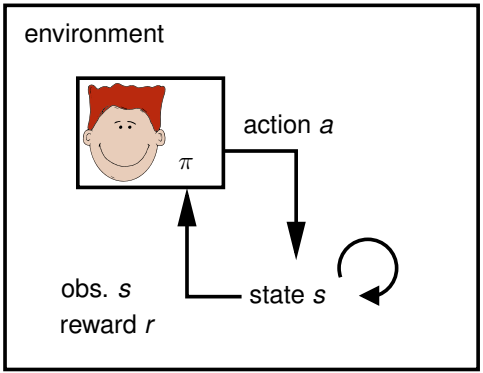
Evaluative feedback: use of a reinforcement signal as performance measure (reinforcement learning);

Transition model

- For instance, robot motion is inaccurate.
- Transitions between states are **stochastic**.
- $p(s'|s, a)$ is the probability to jump from state s to state s' after taking action a.



MDP Agent



Optimality criterion

For instance, agent should maximize the value

$$E\Big[\sum_{t=0}^{h-1}\gamma^t R_t\Big], \tag{1}$$

where

- h is the planning horizon, can be finite or ∞
- γ is a discount rate, $0 \leq \gamma < 1$

Reward hypothesis (Sutton and Barto, 1998):
All goals and purposes can be formulated as the maximization of the cumulative sum of a received scalar signal (reward).

Discrete MDP model

Discrete Markov Decision Process model (Puterman, 1994; Bertsekas, 2000):

- Time t is discrete.
- State space S.
- Set of actions A.
- Reward function $R : S \times A \mapsto \mathbb{R}$.
- Transition model $p(s'|s, a)$, $T_a : S \times A \mapsto \Delta(S)$.
- Initial state s_0 is drawn from $\Delta(S)$.

The Markov property entails that the next state s_{t+1} only depends on the previous state s_t and action a_t:

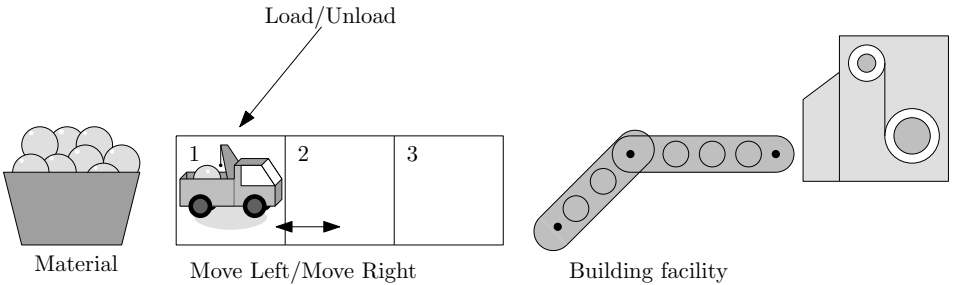
$$p(s_{t+1}|s_t, s_{t-1}, \ldots, s_0, a_t, a_{t-1}, \ldots, a_0) = p(s_{t+1}|s_t, a_t). \tag{2}$$

Example

A simple problem

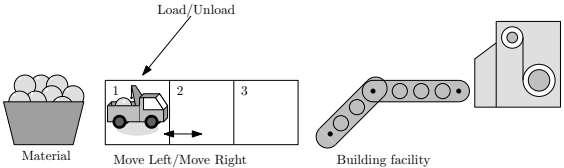
Problem:

An autonomous robot must learn how to transport material from a deposit to a building facility.



(thanks to F. Melo)

Load/Unload as an MDP



- States: $S = \{1_U, 2_U, 3_U, 1_L, 2_L, 3_L\}$;
  - 1_U Robot in position 1 (unloaded);
  - 2_U Robot in position 2 (unloaded);
  - 3_U Robot in position 3 (unloaded);
  - 1_L Robot in position 1 (loaded);
  - 2_L Robot in position 2 (loaded);
  - 3_L Robot in position 3 (loaded)
- Actions: $A = \{\text{Left, Right, Load, Unload}\}$;

Load/Unload as an MDP (1)

- Transition probabilities: “Left”/“Right” move the robot in the corresponding direction; “Load” loads material (only in position 1); “Unload” unloads material (only in position 3).
  Ex:

$$\begin{aligned}(2_L, \text{Right}) &\to 3_L;\\ (3_L, \text{Unload}) &\to 3_U;\\ (1_L, \text{Unload}) &\to 1_L.\end{aligned}$$

- Reward: We assign a reward of $+10$ for every unloaded package (payment for the service).

Load/Unload as an MDP (2)

- For each action $a \in A$, T_a is a matrix.
  Ex:

$$T_{\text{Right}} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

- Recall: $S = \{1_U, 2_U, 3_U, 1_L, 2_L, 3_L\}$.

Load/Unload as an MDP (3)

- The reward $R(s, a)$ can also be represented as a matrix
  Ex:

$$R = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & +10 \end{bmatrix}$$

$S = \{1_U, 2_U, 3_U, 1_L, 2_L, 3_L\}$, $A = \{\text{Left, Right, Load, Unload}\}$

Policies and value functions

Policies and value

- Policy π: tells the agent how to act.
- A deterministic policy $\pi : S \mapsto A$ is a mapping from states to actions.
- Value: how much reward $E[\sum_{t=0}^{h-1} \gamma^t R_t]$ does the agent expect to gather.
- Value denoted as $Q^\pi(s, a)$: start in s, do a and follow π afterwards.

Policies and value (1)

- Extracting a policy π from a value function Q is easy:

$$\pi(s) = \arg\max_{a \in A} Q(s, a). \tag{3}$$

- Optimal policy π^*: one that maximizes $E[\sum_{t=0}^{h-1} \gamma^t R_t]$ (for every state).
- In an infinite-horizon MDP there is always an optimal deterministic stationary (time-independent) policy π^*.
- There can be many optimal policies π^*, but they all share the same optimal value function Q^*.

Dynamic programming

Since S and A are finite, $Q^*(s, a)$ is a matrix.
Iterations of dynamic programming ($\gamma = 0.95$):

$$Q_0 = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \qquad Q_1 = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 10 \end{bmatrix}$$

$S = \{1_U, 2_U, 3_U, 1_L, 2_L, 3_L\}$, $A = \{\text{Left, Right, Load, Unload}\}$

Dynamic programming

Since S and A are finite, $Q^*(s, a)$ is a matrix.
Iterations of dynamic programming ($\gamma = 0.95$):

$$Q_1 = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 10 \end{bmatrix} \qquad Q_2 = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 9.5 & 0 & 0 \\ 0 & 9.5 & 9.5 & 10 \end{bmatrix}$$

$S = \{1_U, 2_U, 3_U, 1_L, 2_L, 3_L\}$, $A = \{\text{Left, Right, Load, Unload}\}$

Dynamic programming

Iterations of dynamic programming ($\gamma = 0.95$):

$$Q_3 = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 9.03 & 0 & 0 \\ 0 & 9.5 & 9.03 & 9.03 \\ 9.03 & 9.5 & 9.5 & 10 \end{bmatrix}$$

$S = \{1_U, 2_U, 3_U, 1_L, 2_L, 3_L\}$, $A = \{\text{Left, Right, Load, Unload}\}$

Dynamic programming

Iterations of dynamic programming ($\gamma = 0.95$):

$$Q_4 = \begin{bmatrix} 0 & 0 & 8.57 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 8.57 & 9.03 & 8.57 & 8.57 \\ 8.57 & 9.5 & 9.03 & 9.03 \\ 9.03 & 9.5 & 9.5 & 10 \end{bmatrix}$$

$S = \{1_U, 2_U, 3_U, 1_L, 2_L, 3_L\}$, $A = \{\text{Left, Right, Load, Unload}\}$

Dynamic programming

Iterations of DP:

$$Q_{10} = \begin{bmatrix} 8.15 & 7.74 & 14.88 & 8.15 \\ 8.15 & 7.35 & 7.74 & 7.74 \\ 7.74 & 7.35 & 7.35 & 7.35 \\ 14.88 & 15.66 & 14.88 & 14.88 \\ 14.88 & 16.48 & 15.66 & 15.66 \\ 15.66 & 16.48 & 16.48 & 17.35 \end{bmatrix}$$

$S = \{1_U, 2_U, 3_U, 1_L, 2_L, 3_L\}$, $A = \{\text{Left, Right, Load, Unload}\}$

Dynamic programming

Final Q^* and policy:

$$Q^* = \begin{bmatrix} 30.75 & 29.21 & 32.36 & 30.75 \\ 30.75 & 27.75 & 29.21 & 29.21 \\ 29.21 & 27.75 & 27.75 & 27.75 \\ 32.36 & 34.07 & 32.36 & 32.37 \\ 32.36 & 35.86 & 34.07 & 34.07 \\ 34.07 & 35.86 & 35.86 & 37.75 \end{bmatrix}$$

$$\pi^* = \begin{bmatrix} \text{Load} \\ \text{Left} \\ \text{Left} \\ \text{Right} \\ \text{Right} \\ \text{Unload} \end{bmatrix}$$

Algorithms

Value iteration

- Value iteration: successive approximation technique.
- Start with all values set to 0.
- In order to consider one step deeper into the future, i.e., to compute Q^*_{n+1} from Q^*_n:

$$Q^*_{n+1}(s,a) := R(s,a) + \gamma \sum_{s' \in S} p(s'|s,a) \max_{a' \in A} Q^*_n(s',a'), \tag{4}$$

  which is known as the dynamic programming update or Bellman backup.
- Bellman (1957) equation:

$$Q^*(s,a) = R(s,a) + \gamma \sum_{s' \in S} p(s'|s,a) \max_{a' \in A} Q^*(s',a'). \tag{5}$$

Value iteration (1)

Initialize Q_0 arbitrarily, e.g., $Q_0(s,a) = 0, \forall s \in S, a \in A$
$n \leftarrow 0$
repeat
  $\delta \leftarrow 0$
  $n \leftarrow n + 1$
  for all $s \in S, a \in A$ **do**
    $Q_n(s,a) \leftarrow R(s,a) + \gamma \sum_{s' \in S} p(s'|s,a) \max_{a' \in A} Q_{n-1}(s',a')$
    $\delta \leftarrow \max(\delta, |Q_{n-1}(s,a) - Q_n(s,a)|)$
  end for
until $\delta < \epsilon$
Return Q_n

Value iteration

Value iteration discussion:

- As $n \to \infty$, value iteration converges.
- Value iteration has converged when the largest update δ in an iteration is below a certain threshold ϵ.
- Exhaustive sweeps are not required for convergence, provided that in the limit all states are visited infinitely often.
- This can be exploited by backing up the most promising states first, known as prioritized sweeping.

Policy iteration

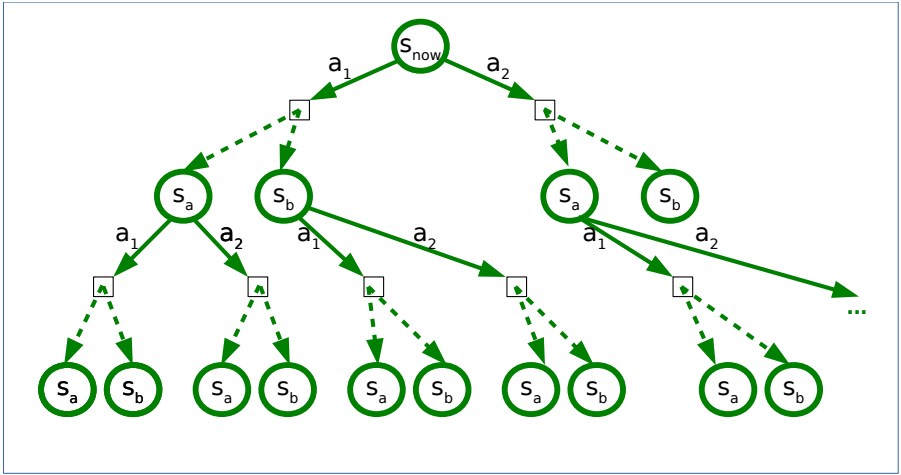
- Instead of iterating over value functions, iterate of policies instead (Howard, 1960).
- Iterate evaluation ($\rightarrow$) and improvement ($\Rightarrow$) of a policy:

$$\pi_0 \rightarrow V^{\pi_0} \Rightarrow \pi_1 \rightarrow V^{\pi_1} \Rightarrow \cdots \Rightarrow \pi^* \rightarrow V^*$$

- Terminates when policy no longer changes, converges as a finite MDP only has a finite number of policies.
- Evaluation is like value iteration but with a fixed policy.

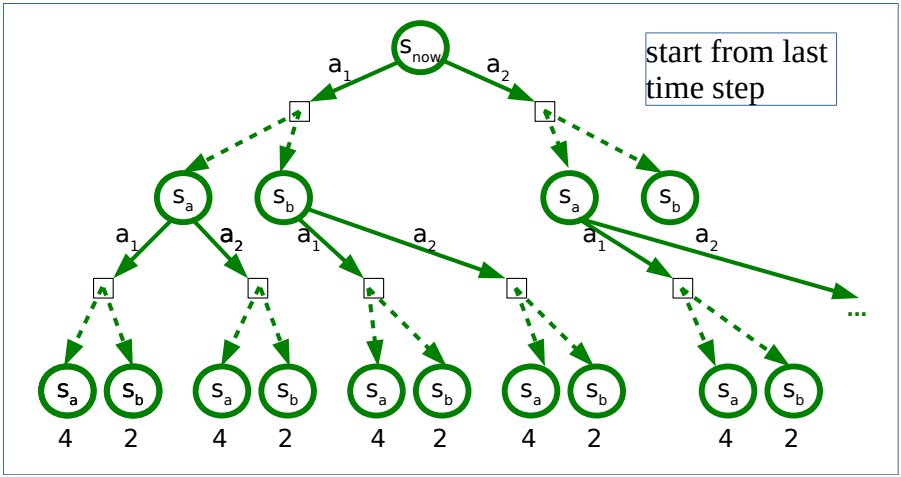
Dynamic Programming in trees

- Construct a plan for *h* time steps into the future



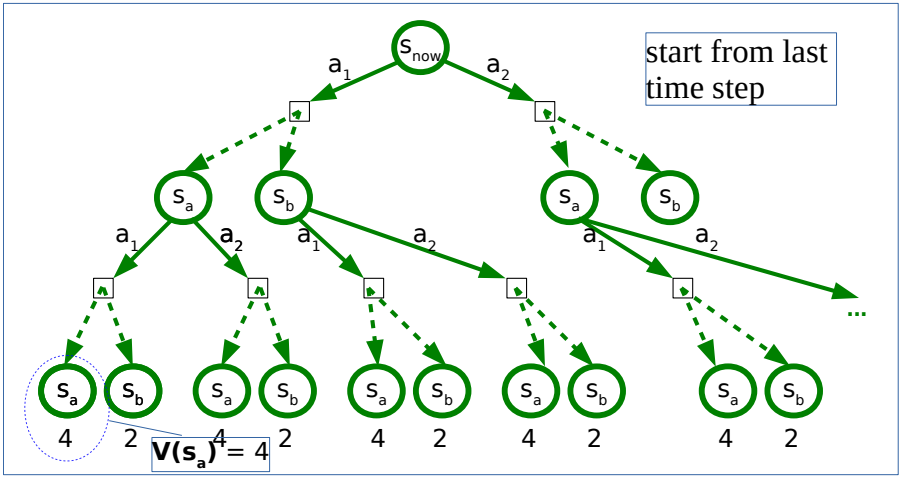
Dynamic Programming in trees

- Construct a plan for *h* time steps into the future



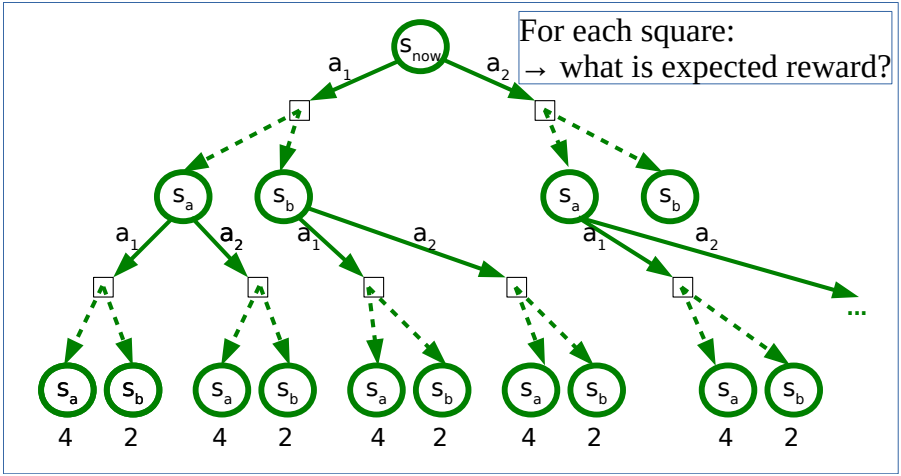
Dynamic Programming in trees

- Construct a plan for *h* time steps into the future



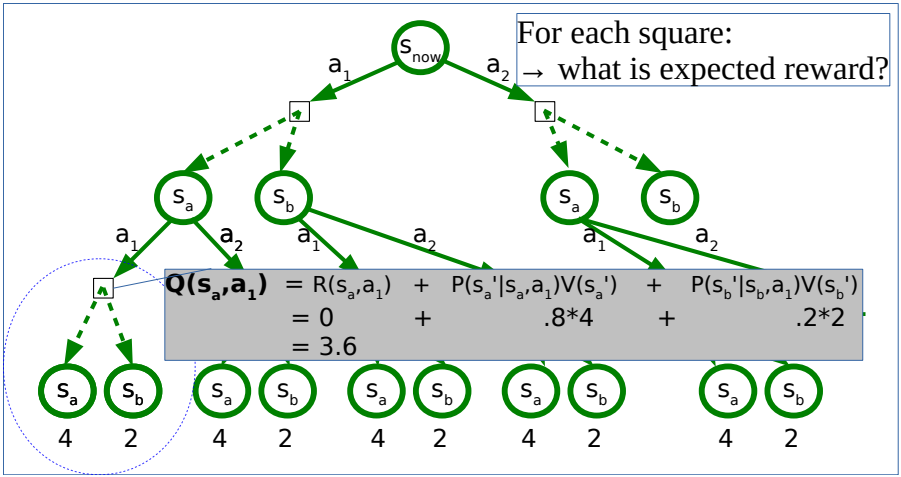
Dynamic Programming in trees

- Construct a plan for *h* time steps into the future



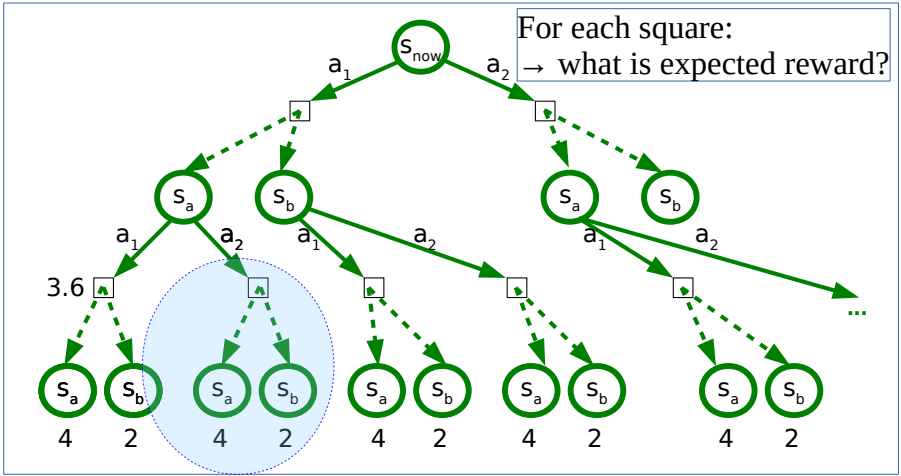
Dynamic Programming in trees

- Construct a plan for *h* time steps into the future



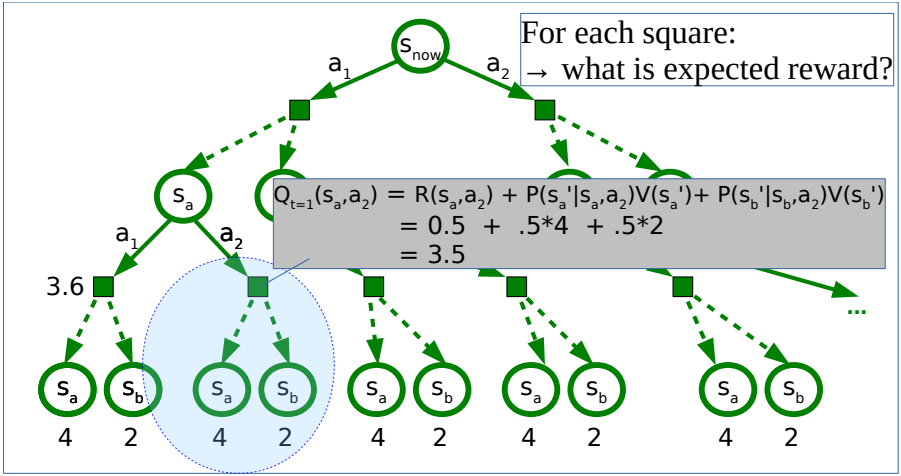
Dynamic Programming in trees

- Construct a plan for *h* time steps into the future



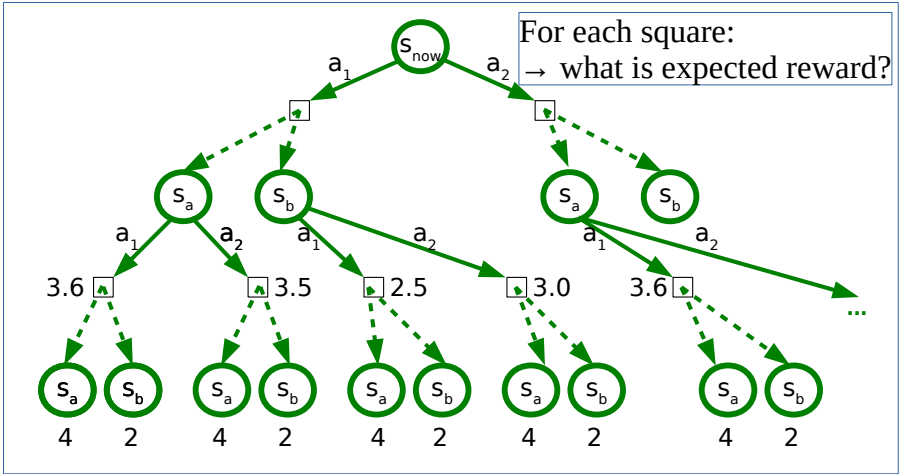
Dynamic Programming in trees

- Construct a plan for *h* time steps into the future



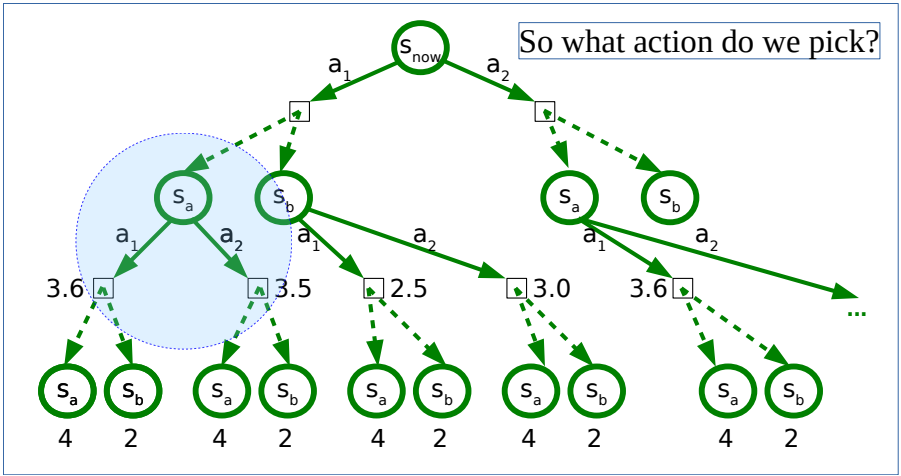
Dynamic Programming in trees

- Construct a plan for *h* time steps into the future



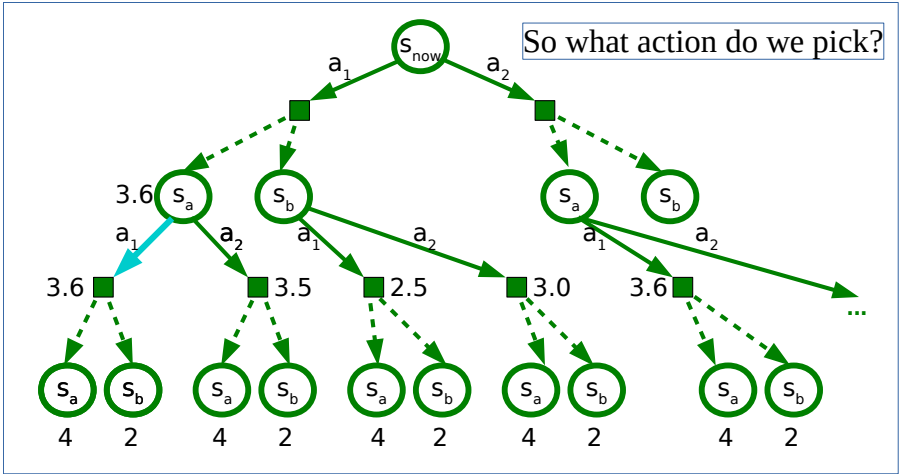
Dynamic Programming in trees

- Construct a plan for *h* time steps into the future



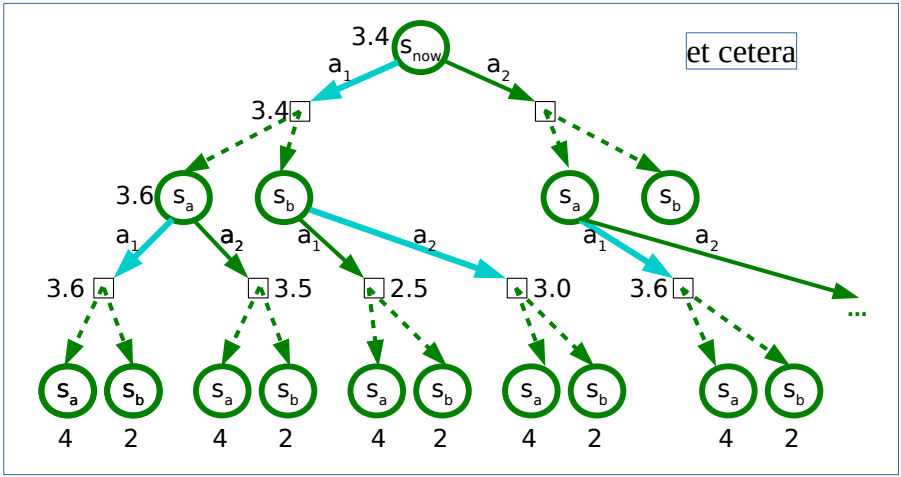
Dynamic Programming in trees

- Construct a plan for *h* time steps into the future



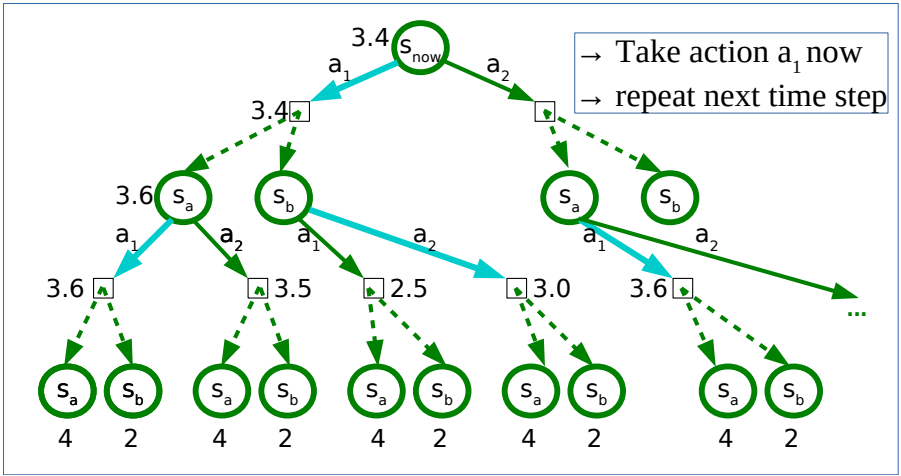
Dynamic Programming in trees

- Construct a plan for *h* time steps into the future



Dynamic Programming in trees

- Construct a plan for *h* time steps into the future



Q-learning

- Reinforcement-learning techniques learn from experience, no knowledge of the model is required.
- Q-learning update:

$$Q(s,a) = (1-\beta)\ Q(s,a) + \beta\Big[r + \gamma \max_{a' \in A} Q(s',a')\Big], \tag{6}$$

where $0 < \beta \leq 1$ is a learning rate.

Q-learning

Q-learning discussion:

- Q-learning is guaranteed to converge to the optimal Q-values if all $Q(s, a)$ values are updated infinitely often.
- In order to make sure all actions will eventually be tried in all states exploration is necessary.
- A common exploration method is to execute a random action with small probability ϵ, which is known as ϵ-greedy exploration.

Solution methods: MDPs

Model based

- Basic: dynamic programming (Bellman, 1957), value iteration, policy iteration.
- Advanced: prioritized sweeping, function approximators.

Model free, reinforcement learning (Sutton and Barto, 1998)

- Basic: Q-learning, TD(λ), SARSA, actor-critic.
- Advanced: generalization in infinite state spaces, exploration/exploitation issues.

References

R. Bellman. *Dynamic programming*. Princeton University Press, 1957.

D. P. Bertsekas. *Dynamic Programming and Optimal Control*. Athena Scientific, Belmont, MA, 2nd edition, 2000.

R. A. Howard. *Dynamic Programming and Markov Processes*. MIT Press and John Wiley & Sons, Inc., 1960.

M. L. Puterman. *Markov Decision Processes—Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc., New York, NY, 1994.

R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 1998.