Technische Hochschule Ingolstadt

Principles of Autonomy and Decision Making

(AI_PrincAutonomy_2808)

Week 5: Value- and Policy-Iteration

Team:

- Prof. Dr. rer. nat. Lenz Belzner
- Chidvilas Karpenahalli Ramakrishna, M.Eng.
- Adithya Mohan, M.Eng.
- Zahra Zeinaly, Ph.D.

MDPs: Recap

- MDP components:
  - States $s \in S$
  - Actions $a \in A$
  - Reward function $R(s, a, s')$
  - Discount factor γ
  - Transition function $T(s, a, s') = P(s'|s, a)$

- Quantities:
  - Optimal Policy
    - $\pi^*(s)$
    - Maps states to actions
  - Utility
    - Expected sum of discounted rewards
  - Optimal value of a state
    - $V^*(s)$
    - Expected future utility from a state and acting optimally thereafter
  - Optimal Q-value of a state-action pair
    - $Q^*(s, a)$
    - Expected future utility of a state by taking an action a and acting optimally thereafter

Week 5: Value- and Policy-Iteration

Bellman Equation: Recap

- Bellman equation:
  - Links the quantities and shows what is needed to take optimal actions
  - Take correct first action
  - Keep being optimal later

- Derivation:
  - $V^*(s) = max_a Q^*(s,a)$
  - $Q^*(s,a) = \sum_{s'} T(s,a,s') [R(s,a,s') + \gamma V^*(s')]$
  - **Bellman equation**:
    - $V^*(s) = max_a \sum_{s'} P(s'|a,s) [R(s,a,s') + \gamma V^*(s')]$

- How to solve the Bellman equation?
  - Value-iteration
  - Policy-iteration
  - Dynamic programming (both value- and policy-iteration are DP)
  - Reinforcement Learning

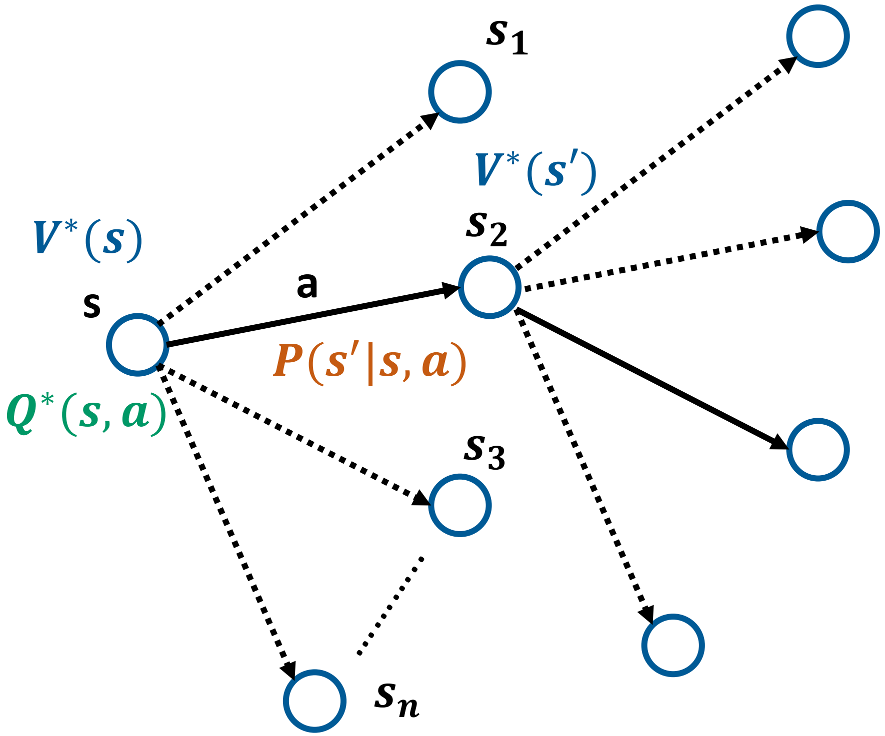

Fig 1. MDP search tree

Value-Iteration: Introduction

- Aim:
  - We wish to find optimal values $V^*(s)$ of each state using which an optimal policy $\pi^*(s)$ can be extracted.
- $V^*(s) = max_a \sum_{s'} P(s'|a,s) [\ R(s,a,s') + \gamma\ V^*(s')\ ]$
  - Here, $V^*(s)$ is the value of being in a state and acting optimally
  - How to obtain $V^*(s)$?
    - Value-iteration
- Value-iteration:
  - Update values for k iterations
  - We keep $V_0(s)$=0
  - We define $V_k(s)$ is the value of a state in k^{th} iteration
  - $V_k(s) = max_a \sum_{s'} P(s'|a,s) [\ R(s,a,s') + \gamma\ V_{k-1}(s')\ ]$
  - Stop when $max_s|V_k(s) - V_{k-1}(s)| < \varepsilon$ i.e. convergence

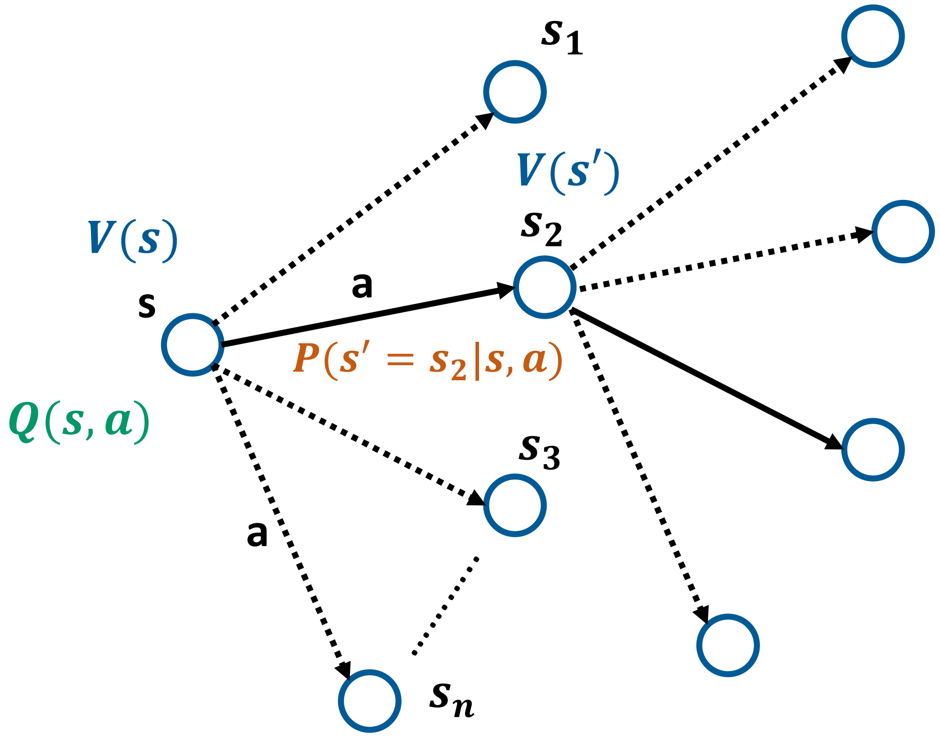

Fig 1. Value-iteration search tree

Value-Iteration: Policy Extraction

- Extracting optimal policy $\pi^*(s)$ from $V^*(s)$ of each state
  - $\pi^*(s) = \ argmax_a \sum_{s'} T(s,a,s') \, [\ R(s,a,s') + \ \gamma \, V^*(s') \,]$
  - $\pi^*(s) = \ argmax_a Q^*(s,a)$
- Demo:
  - Week 5 lecture code – *value_iteration.py*
  - *NOTE: See how policy converged much earlier than values*

Value-Iteration: Problems

- Value-iteration is slow:
  - Why?
    - $V_k(s) = \ max_a \sum_{s'} P(s'|a,s) [\ R(s,a,s') + \ \gamma \ V_{k-1}(s') \ ]$
    - max_a is very slow

- The values of each state do not change significantly after certain iterations

- The policy often converges before the values
  - Wasted computation

- Solution:
  - Occassionally check if the policy has converged and stop computation if convergence is reached
    - Policy-iteration

Policy Methods

- Motivation:
  - The output of solving an MDP is a policy $\pi^*(s)$
  - We do not use the value of states $V^*(s)$ or $Q^*(s,a)$ directly
  - We extract $\pi^*(s)$ from $V^*(s)$ in value-iteration
  - We have to wait till the values converge to extract an optimal policy in value-iteration
  - What if we don't have to wait?
    - Policy-iteration

- Policy methods:
  - Given a policy $\pi(s)$, we wish to know how good it is i.e. what is the outcome if I follow the policy in each state
  - We no longer have to deal with max_a as $\pi(s): s \rightarrow a$ dictates what actions to take
  - In policy methods, we don't act optimally we just act as policy tells
  - The tress now becomes simple as shown in Fig 1.
  - We fix a policy $\pi(s)$ and calculate $V^\pi(s)$
    - $V^\pi(s) = \sum_{s'} T(s,\pi(s),s') \, [\, R(s,\pi(s),s') + \gamma \, V^\pi(s') \,]$
    - Here, $V^\pi(s)$ = expected total discounted rewards starting in s and following $\pi(s)$ (*generally not optimal*)

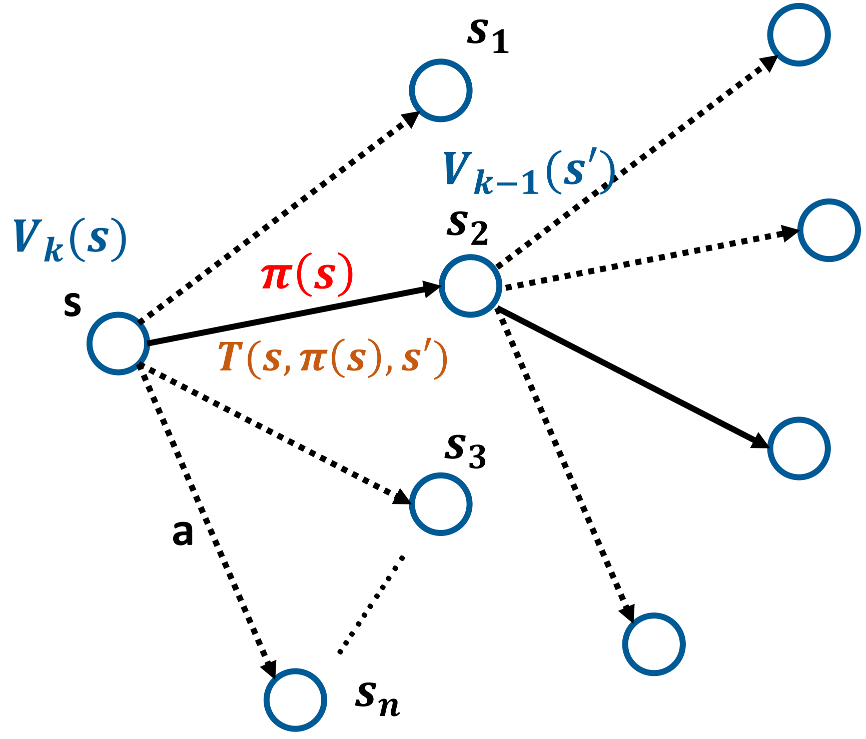

Fig 1. Policy-iteration search tree

Policy-Iteration

- Steps:
  1. Policy evaluation:
     - Fix a policy $\pi(s)$
     - Get utilities i.e. $V^{\pi}(s)$ for $\pi(s)$ until convergence (not $V^{*}(s)$)
  2. Policy improvement:
     - Update policy $\pi(s)$ using one-step look-ahead with resulting converged utilities $V^{\pi}(s)$
  3. Iteration:
     - Repeat steps 1 and 2 until policy converges

- Advantage:
  - We check $\pi(s)$ for convergence before continuing the iteration
  - This might result in faster convergence of $\pi(s)$

- NOTE:
  - $\pi(s)$ - Deterministic policy
  - $\pi(a|s)$ - Stochastic policy

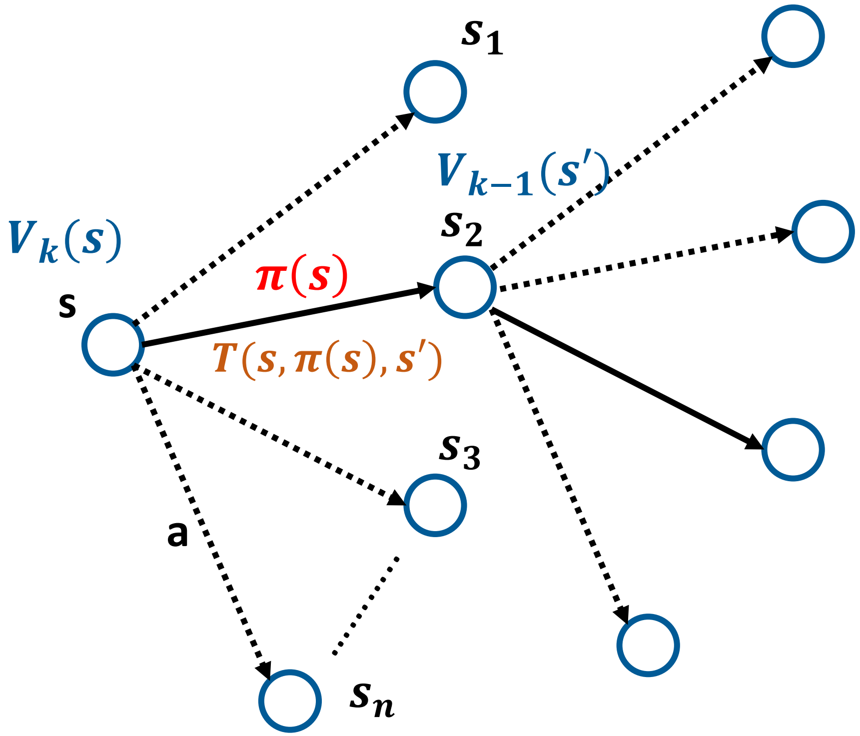

Fig 1. Policy-iteration search tree

Policy-Iteration: Policy Evaluation

- Consider a fixed i^{th} policy $\pi^{i}(s)$

- Get utilities:
  - Update the value of each state using the fixed policy to choose actions
  - $V_k^{\pi_i}(s) = \sum_{s'} T(s, \pi_i(s), s') \, [\, R(s, \pi_i, s') + \gamma \, V_{k-1}^{\pi_i}(s') \,]$
  - Continue for a fixed number of iterations m
  - Upon completing m iterations, update the policy to get $\pi^{i+1}(s)$, check for convergence, if not, repeat
  - NOTE: We do not have max_a in this equation which reduces computation

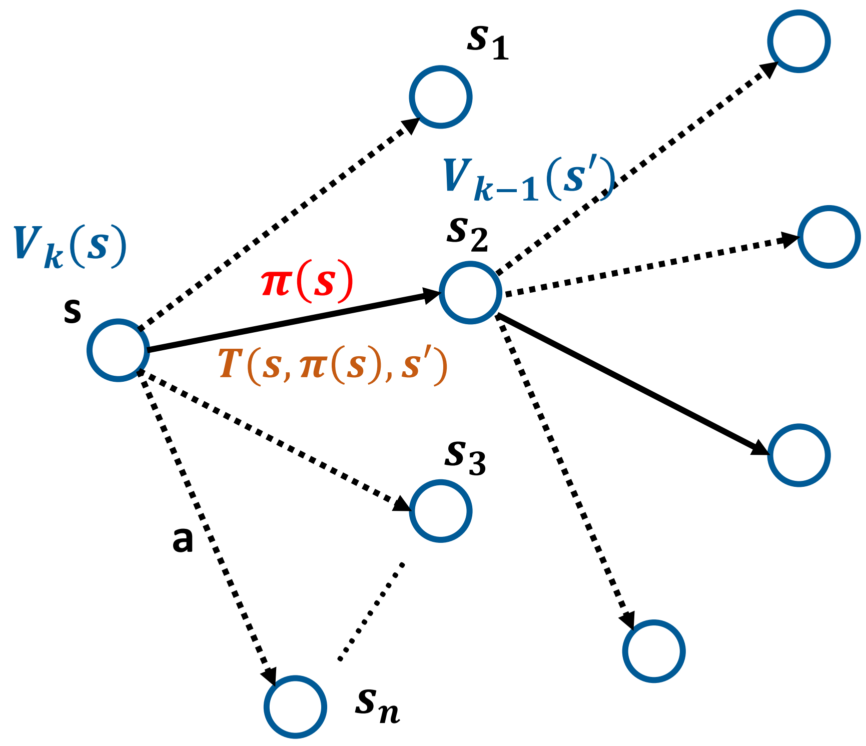

Fig 1. Policy-iteration search tree

Policy-Iteration: Policy Improvement

- Update policy:
  - $\pi_{i+1}(s) = argmax_a \sum_{s'} T(s,a,s') [R(s,a,s') + \gamma V^{\pi_i}(s')]$
  - We updated our old policy $\pi^i(s)$ to $\pi^{i+1}(s)$
  - Upon updating the policy repeat step 1 (Policy evaluation) for another m iterations
  - Follow this until policy converges i.e. $\pi_{i+n}(s) = \pi_{i+n-1}(s)$

- Demo:
  - Week 5 lecture code – *policy_iteration.py*
  - *NOTE: See how policy-iteration ran much faster than value-iteration*

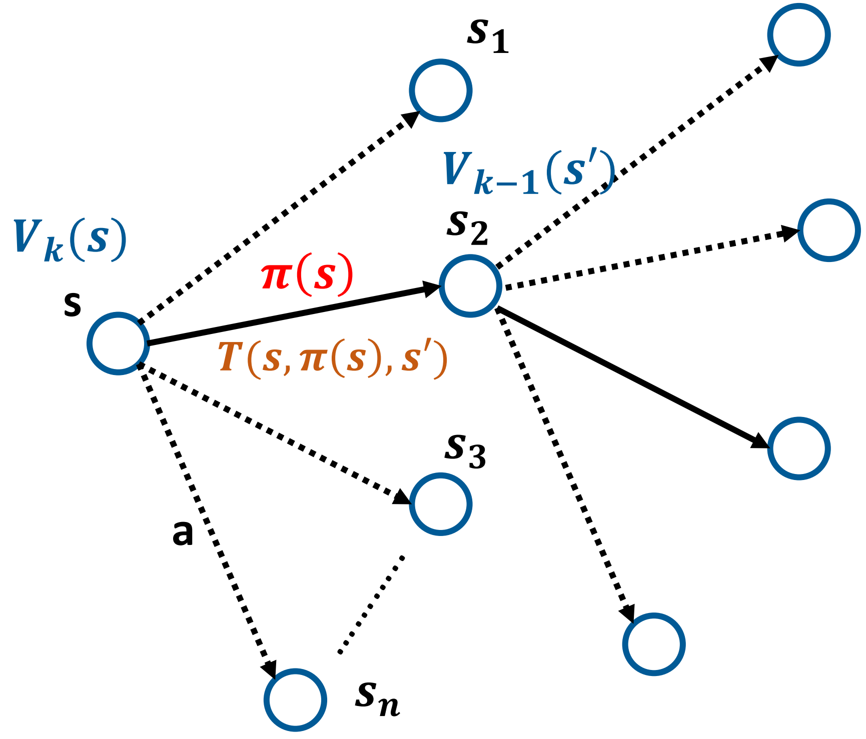

Fig 1. Policy-iteration search tree

Summary

- Value-iteration:
  - We update the utilities till convergence to get the optimal values $V^*(s)$
  - We use $V^*(s)$ to extract an optimal policy $\pi^*(s)$ at the end
  - Slower convergence as policy might converge before values
- Policy-iteration:
  - We update utilities for m number of iterations using a fixed policy $\pi_i(s)$
  - After reaching m iterations, I update my policy to $\pi_{i+1}(s)$
  - I repeat the above steps until policy converges
  - Faster convergence than value-iteration
- Motivation for Dynamic programming (DP):
  - DP is a powerful computational technique that solves problems by breaking them down into simpler subproblems, *solving each of these subproblems just once, and storing their solutions*
  - Both value-iteration and policy-iteration are dynamic programming techniques