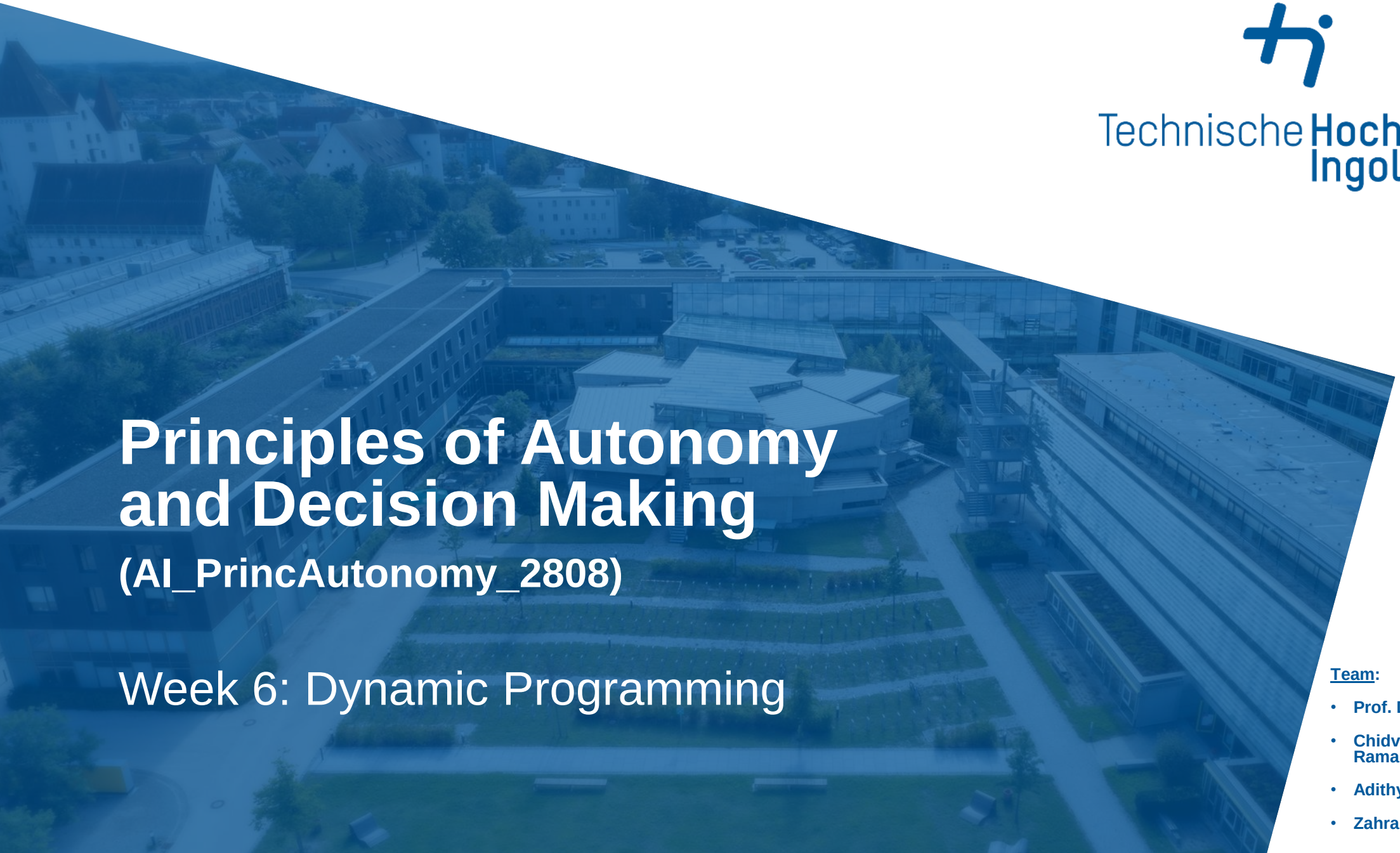

Technische Hochschule Ingolstadt

Principles of Autonomy and Decision Making

(AI_PrincAutonomy_2808)

Week 6: Dynamic Programming

Team:

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

Introduction

- Definition:
  - Dynamic Programming (DP) is a method to solve complex problems by breaking them down into simpler problems and solving them recursively by storing the intermediate results
- History:
  - Introduced by Richard Bellman in 1953
  - Was used to solve multi-stage decision processes
- Application areas: Where problems can be broken down into sub-problems
  - Computer Science
  - Operations Research
  - Economics

Principles of Dynamic Programming

- Overlapping Substructure:
  - The problem can be broken down into subproblems which can be reused several times without generating new sub-problems

- Optimal Substructure:
  - The optimal solution to a problem contains optimal solutions to its sub-problems

Steps to Solve a Problem Using Dynamic Programming

1. Identify if DP is applicable
   - Optimal substructure
   - Overlapping substructure
2. Decide the State Expression
   - The state in DP represents the smallest subproblem whose solution is part of the larger problem
3. Formulate the State Relationship
   - Involves a recurrence relation
4. Initialize the DP table
5. Iterate to Fill in the DP Table
6. Deduce the Final Answer

Types of Dynamic Programming

- Top-down (Memoization):
  - Start solving the problem by breaking it down
  - If the same subproblem appears again, instead of recomputing its solution, the previously stored result is reused
- Bottom-up (Tabulation):
  - Start from the simplest subproblems and iteratively solve more complex problems using previously stored solutions
- Demo:
  - Week 6 lecture code Fibonacci series - *dynamic_programming.py*

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

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

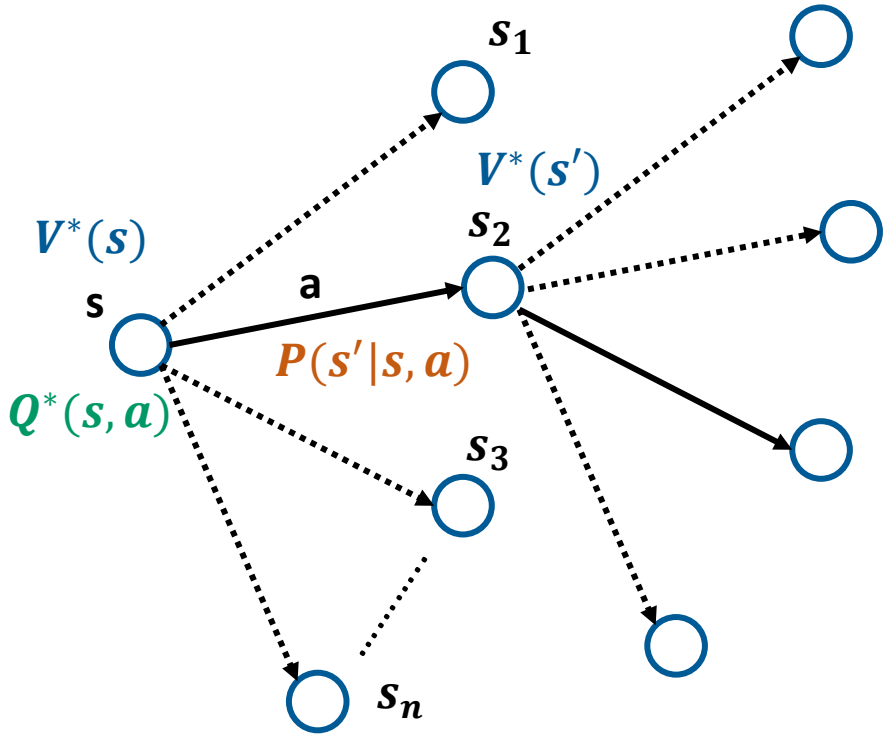

Fig 1. MDP search tree

Recap: Value-Iteration

- Steps:
  - Initialize $V_0(s)$
  - Update values for k iterations
  - $V_k(s) = max_a \sum_{s'} P(s'|a,s) [R(s,a,s') + \gamma V_{k-1}(s')]$
    - Here, $V_k(s)$ is the value of a state in k^{th} iteration
  - Stop when $max_s|V_k(s) - V_{k-1}(s)| < \varepsilon$ i.e. convergence
  - Policy extraction:
    - $\pi^*(s) = argmax_a \sum_{s'} T(s,a,s') [R(s,a,s') + \gamma V^*(s')]$
    - $\pi^*(s) = argmax_a Q^*(s,a)$

Recap: Policy Iteration

- Steps:
  1. Policy evaluation:
     - Fix a policy $\pi_i(s)$
     - $V_k^{\pi_i}(s) = \sum_{s'} T(s, \pi_i(s), s') \left[R(s, \pi_i, s') + \gamma V_{k-1}^{\pi_i}(s') \right]$
     - Continue for a fixed number of iterations m
  2. Policy improvement:
     - Update policy $\pi(s)$ using one-step look-ahead with resulting converged utilities $V^{\pi}(s)$
     - $\pi_{i+1}(s) = argmax_a \sum_{s'} T(s, a, s') \left[R(s, a, s') + \gamma \boldsymbol{V^{\pi_i}}(s') \right]$
     - We updated our old policy $\pi_i(s)$ to $\pi_{i+1}(s)$
     - Check for convergence, if not, repeat step 1
  3. Iteration:
     - Follow this until policy converges i.e. $\pi_{i+n}(s) = \pi_{i+n-1}(s)$

Steps to Solve a Problem Using Dynamic Programming

1. Identify if DP is applicable
   - Overlapping subproblems: The value of each state (subproblem) depends on the values of the states reachable from it
   - Optimal substructure: The optimal value of the current state will rely on the optimal values of the next states

2. Decide the State Expression
   - The state in value iteration is nothing but each possible state in the MDP

3. Formulate the State Relationship (Recurrence)
   - $V_k(s) = max_a \sum_{s'} P(s'|a,s) [R(s,a,s') + \gamma V_{k-1}(s')]$

4. Initialize the DP table
   - Initialize $V_0(s)$

5. Iterate to Fill in the DP Table
   - Iteratively update $V_k(s)$ until convergence i.e. $max_s |V_k(s) - V_{k-1}(s)| < \varepsilon$

6. Deduce the Final Answer
   - Extract the optimal policy
   - $\pi^*(s) = argmax_a \sum_{s'} T(s,a,s') [R(s,a,s') + \gamma V^*(s')]$

Exercise 1: 1D grid-world

- Given:
  - Fig. 1 depicts a 1D grid-world with 5 states. The agent starts in the leftmost state and must reach the goal at the rightmost state. The agent can perform two actions namely moving the right and left. The probability that the agent moves in the intended direction is 0.8 and let the discount factor be 0.9. Consider a living reward of -1 and a reward of +10 for reaching the goal. Consider the reward for each transition from one state to another.

Fig 1. 1D grid-world

- Tasks:
  1. Convert the above statement into mathematical notations and define states, actions, reward function and transition function
  2. Perform value-iteration for 2 iterations and provide the values of all 5 states at the end of the 2nd iteration
     - Consider, the values of all non-goal states to be 0 and the value of goal state to be 10 in the 0th iteration

- Solution:
  - $V_2(s) = [-1.9, -1.9, +9.62, +21.3, +27.1]$