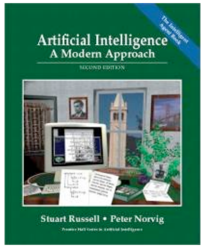
Introduction to Multi-Agent Programming

14. Learning in Multi-Agent Systems (Part A)

SDP, MDPs, Value Iteration, Policy Iteration

Alexander Kleiner, Bernhard Nebel

Lecture Material



Artificial Intelligence – A Modern Approach, 2nd Edition

by Stuart Russell - Peter Norvig

Reinforcement Learning: An Introduction

By Richard S. Sutton and Andrew G. Barto

MIT Press, Cambridge, MA, 1998 A Bradford Book

Online Version: http://www.cs.ualberta.ca/~sutton/book/the-book.html

Some illustrations have been taken from the books above.

Contents

Introduction

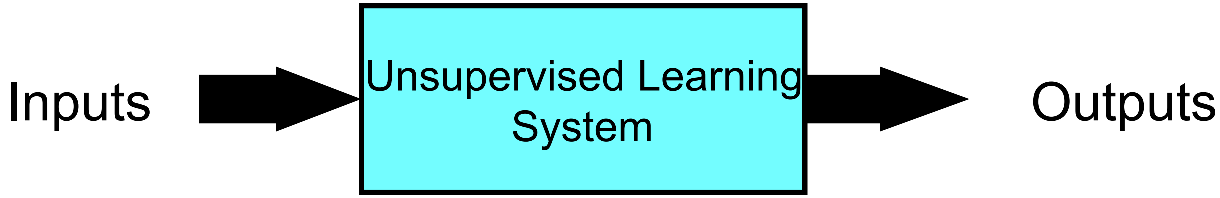
- The importance of learning in MAS:
  - Agents are typically deployed in complex domains, i.e., dynamic domains with large state spaces, and uncertainty of action execution
  - Sometimes impossible to prepare agents for any situation
- Learning methods can be used to
  - enable the agent to do rich decisions based on little experience (generalization)
  - enable the agent to change its behavior online according to changes in the world (adaption)
- However, machine learning suffers from the "curse of dimensionality"
  - Exponential growth of the state space with an increasing number of state variables
  - Exponential growth of action space with an increasing number of actions (In MAS hard when learning joint actions)

Different Types Of Learning feedback

- The learning feedback indicates the performance level achieved so far

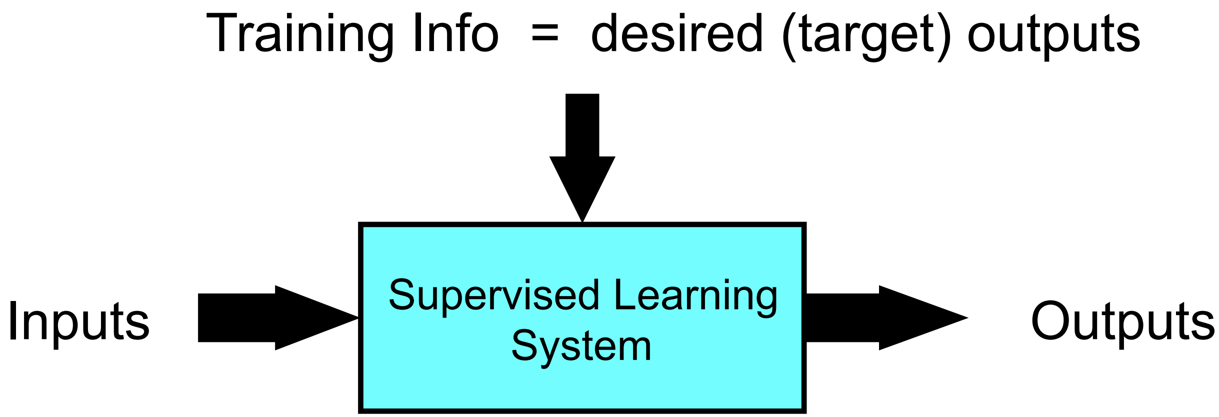
- The following learning feedbacks are distinguished:
  - Supervised learning (teacher)
  - Reinforcement learning (critic)
  - Unsupervised learning (observer)

Unsupervised Learning



Example: clustering of texts on the Internet according to counted word frequencies

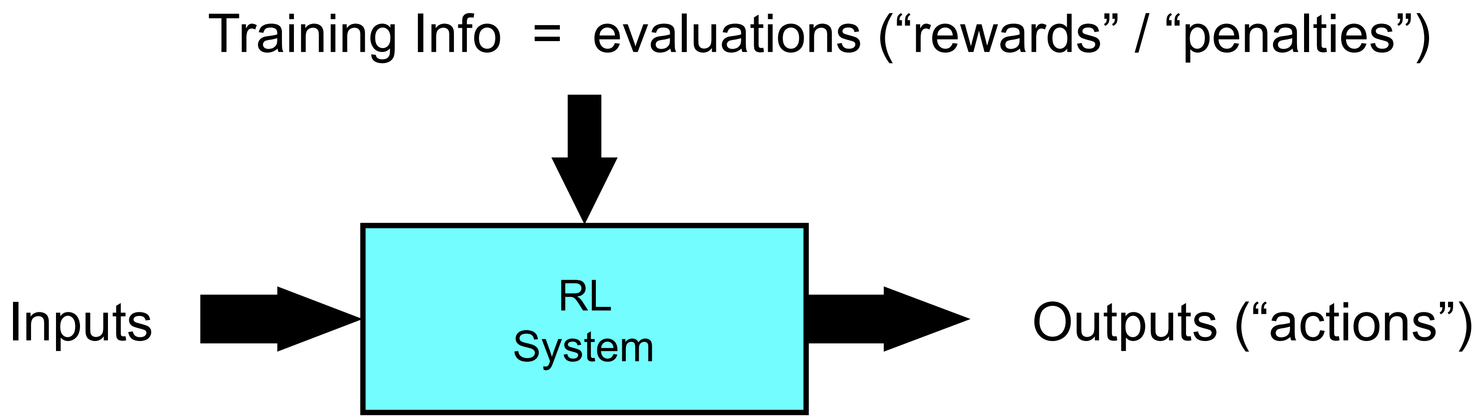
Supervised Learning



Error = (target output – actual output)

Example: detecting faces in images

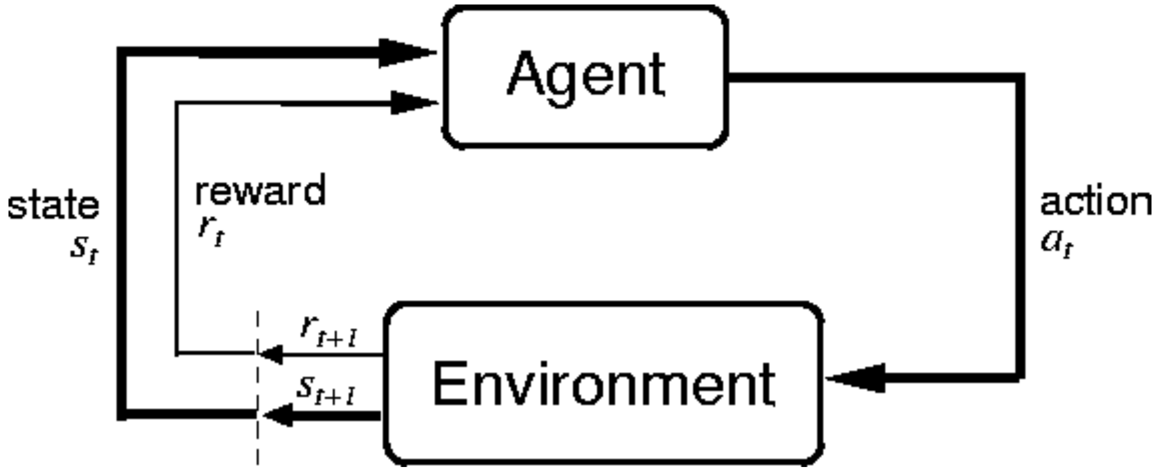
Reinforcement Learning



Objective: get as much reward as possible

Example: robot driving collision free

The Agent-Environment Interface



$$\textit{Agent and environment interact at discrete time steps}: \; t = 0,1,2,\ldots$$

$$\textit{Agent observes state at step } t: \quad s_t \in S$$

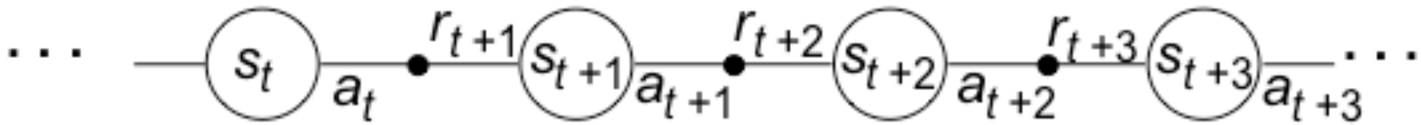
$$\textit{produces action at step } t: \; a_t \in A(s_t)$$

$$\textit{gets resulting reward}: \quad r_{t+1} \in \Re$$

$$\textit{and resulting next state}: \; s_{t+1}$$

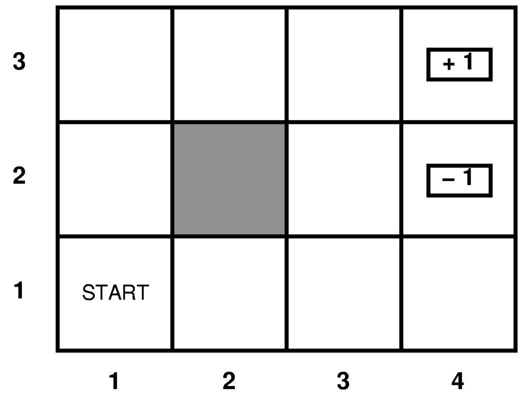
The Credit-Assignment Problem

- The problem of properly assigning feedback for an overall performance change to each of the system activities that contributed to that change



- Which actions were invariant, which were important?
- Can be decomposed into two sub-problems:
  - The inter-agent CAP
    - Assignment of credit for an overall performance change to the external actions of the agents
  - The intra-agent CAP
    - Assignment of credit for a particular external action of an agent to its internal modules

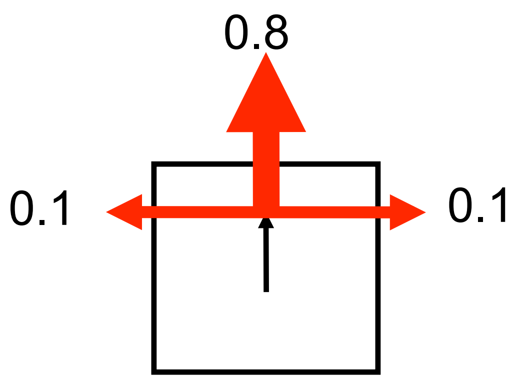
Sequential Decision Problems (1)



- Beginning in the start state the agent must choose an action at each time step
- The interaction with the environment terminates if the agent reaches one of the goal states (4, 3) (reward of +1) or (4,2) (reward –1). Each other location has a reward of -.04
- In each location the available actions are Up, Down, Left, Right

Sequential Decision Problems (2)

- Deterministic version: All actions always lead to the next square in the selected direction, except that moving into a wall results in no change in position.
- Stochastic version: Each action achieves the intended effect with probability 0.8, but the rest of the time, the agent moves at right angles to the intended direction.



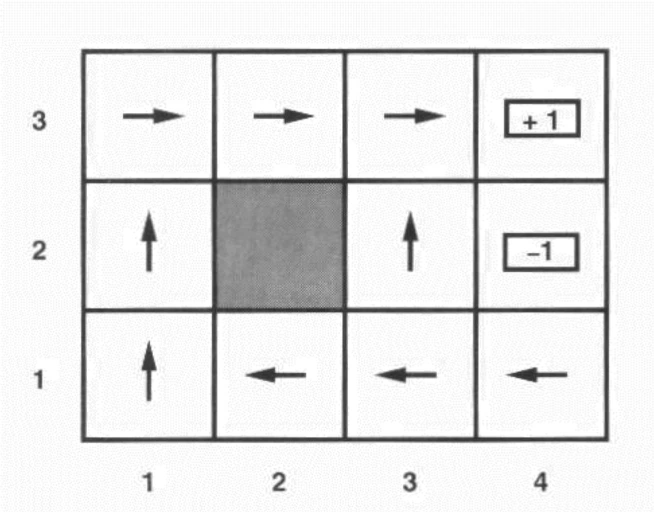
Markov Decision Problem (MDP)

- Given a set of actions *A*, a set of states *S* in an accessible, stochastic environment, an MDP is defined by:
  - Initial state S_0
  - Transition Model T(s,a,s')
  - Reward function R(s)
- Transition model: T(s,a,s') is the probability that state s' is reached, if action a is executed in state s
- Policy: Complete mapping π that specifies for each state s which action π(s) to take
- Wanted: The ***optimal policy*** π^* is the policy that maximizes the expected utility

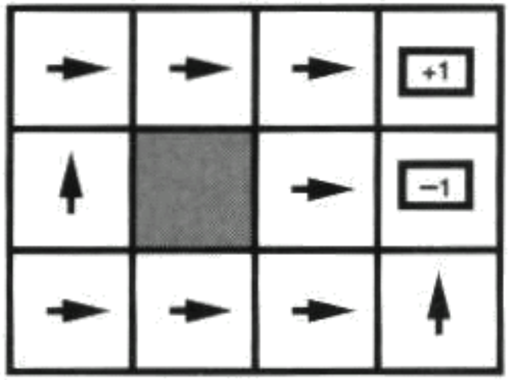
Optimal Policies (1)

- Given the optimal policy, the agent uses its current percept that tells it its current state
- It then executes the action $\pi^*(s)$
- We obtain a simple reflex agent that is computed from the information used for a utility-based agent

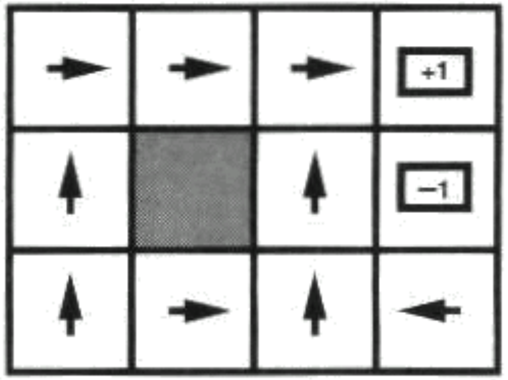
Optimal policy for our MDP when R(s) = -0.4 for non-terminals:



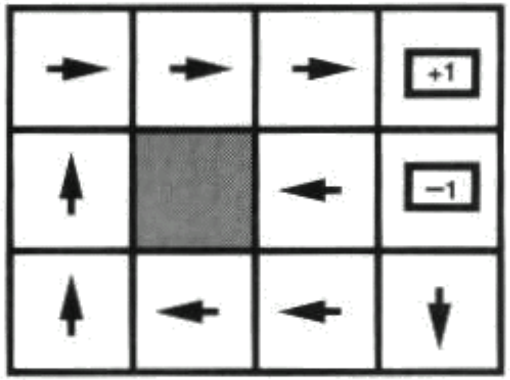
Optimal Policies (2)



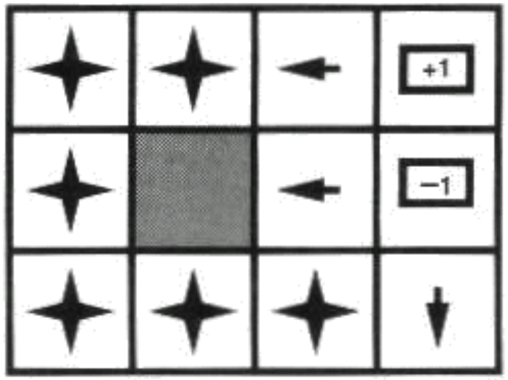
$R(s) \leq -1.6248$



$-0.4278 < R(s) < -0.085$



$-0.0221 < R(s) < 0$



$0 < R(s)$

How to compute optimal policies?

Finite and Infinite Horizon Problems

- Performance of the agent is measured by the sum of rewards for the states visited
- To determine an optimal policy we will first calculate the utility of each state and then use the state utilities to select the optimal action for each state
- The result depends on whether we have a finite or infinite horizon problem
- Utility function for state sequences: $U_h([s_0,s_1,\ldots,s_n])$
- Finite horizon: $U_h([s_0,s_1,\ldots,s_{N+k}]) = U_h([s_0,s_1,\ldots,s_N])$ for all $k > 0$
- For finite horizon problems the optimal policy depends on the horizon N
- In infinite horizon problems the optimal policy only depends on the current state

Assigning Utilities to State Sequences

- For finite horizon problems utilities for each state can be computed by summing-up rewards of each state:
  - $U_h([s_0, s_1\ s_2, \ldots]) = R(s_0) + R(s_1) + R(s_2) + \ldots$
- For infinite horizon problems utilities have to be computed by discounting future rewards:
  - $U_h([s_0, s_1\ s_2, \ldots]) = R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \ldots$
- The term $\gamma \in [0:1[$ is called the discount factor
- With discounted rewards the utility of an infinite state sequence is always finite. The discount factor expresses that future rewards have less value than current rewards

Utilities of States

- The utility of a state depends on the utility of the state sequences that follow it
- Let $U^\pi(s)$ be the utility of a state under policy π
- Let s_t be the state of the agent after executing π for t steps. Thus, the utility of s under π is

$$U^\pi(s) = E\left[\sum_{t=0}^{\infty} \gamma^t R(s_t) \mid \pi, s_0 = s\right]$$

- The true utility U(s) of a state is $U^{\pi^*}(s)$
- R(s) is the short-term reward for being in s and U(s) is the long-term total reward from s onwards.

Choosing Actions using the Maximum Expected Utility Principle

The agent simply chooses the action that maximizes the expected utility of the subsequent state:

$$\pi(s) = \operatorname*{argmax}_a \sum_{s'} T(s,a,s')U(s')$$

The utility of a state is the immediate reward for that state plus the expected discounted utility of the next state, assuming that the agent chooses the optimal action:

$$U(s) = R(s) + \gamma \max_a \sum_{s'} T(s,a,s')U(s')$$

Example

The utilities of the states in our 4x3 world with γ=1 and R(s)=-0.04 for non-terminal states:

3	0.812	0.868	0.918	+ 1
2	0.762		0.660	–1
1	0.705	0.655	0.611	0.388
	1	2	3	4

Which action would an optimal agent choose here?

Bellman-Equation

- The equation

$$U(s) = R(s) + \gamma \max_a \sum_{s'} T(s,a,s')U(s')$$

  is also called the Bellman-Equation.
- In our 4x3 world the equation for the state (1,1) is

$$\begin{aligned} U(1,1) = -0.04 + \gamma \max\{ \quad & 0.8\,U(1,2) + 0.1\,U(2,1) + 0.1\,U(1,1), && \textit{(Up)} \\ & 0.9\,U(1,1) + 0.1\,U(1,2), && \textit{(Left)} \\ & 0.9\,U(1,1) + 0.1\,U(2,1), && \textit{(Down)} \\ & 0.8\,U(2,1) + 0.1\,U(1,2) + 0.1\,U(1,1)\ \} && \textit{(Right)} \end{aligned}$$

→ Given the numbers for the optimal policy, Up is the optimal action in (1,1).

Value Iteration (1)

An algorithm to calculate an optimal strategy

Basic Idea: Calculate the utility of each state. Then use the state utilities to select an optimal action for each state

How to calculate the utility of each state?

The bellman equation can be used to build as system of n equations for n states

However, due to the transition model and the therefore required max operator, the system is non-linear

→ Solution can not be computed in closed form (can only be done for deterministic problems)

Value Iteration (2)

Iterative Procedure

Solution:

We can apply an iterative approach in which we replace the equality of the bellman equation by an assignment:

$$U(s) \leftarrow R(s) + \gamma \max_a \sum_{s'} T(s, a, s') U(s')$$

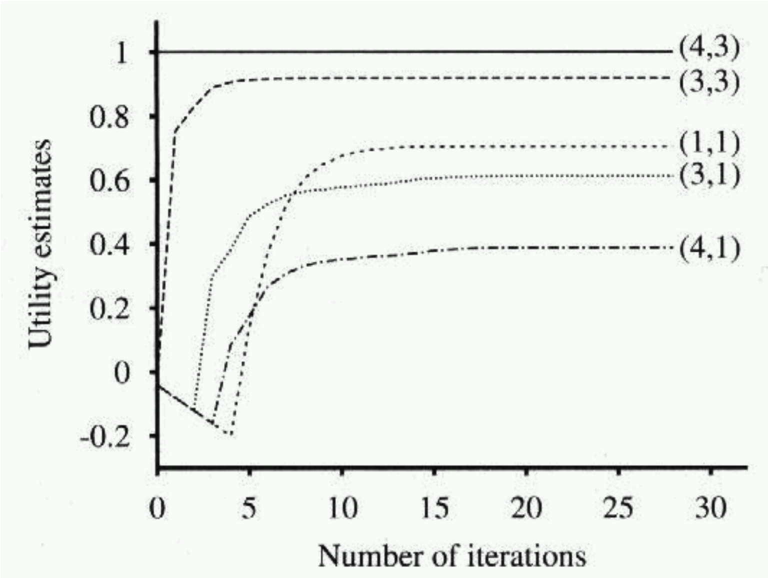
The Value Iteration Algorithm

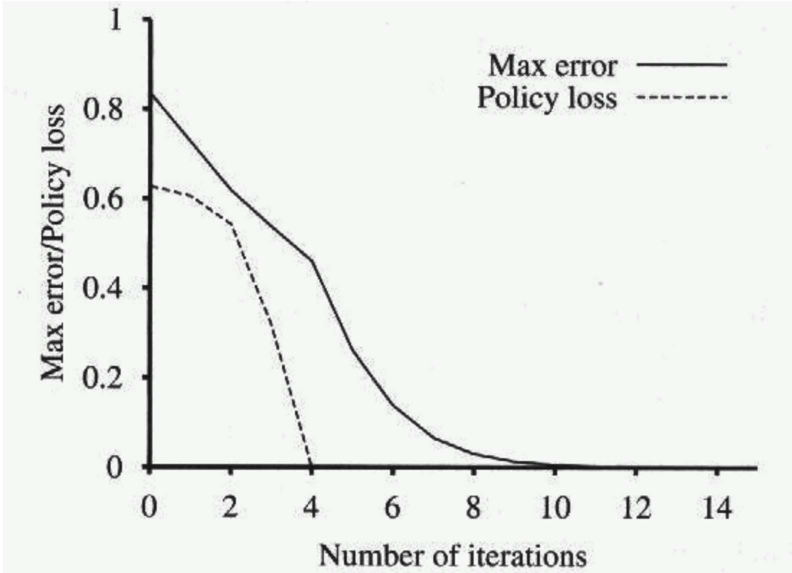
function VALUE-ITERATION(mdp, ϵ) **returns** a utility function
  inputs: mdp, an MDP with states S, transition model T, reward function R, discount γ
    ϵ, the maximum error allowed in the utility of any state
  local variables: U, U', vectors of utilities for states in S, initially zero
    δ, the maximum change in the utility of any state in an iteration

  repeat
    $U \leftarrow U'; \delta \leftarrow 0$
    for each state s **in** S **do**
      $U'[s] \leftarrow R[s] + \gamma \max_a \sum_{s'} T(s,a,s')\, U[s']$
      if $|U'[s] - U[s]| > \delta$ **then** $\delta \leftarrow |U'[s] - U[s]|$
  until $\delta < \epsilon(1-\gamma)/\gamma$
  return U

It can be shown that value iteration converges

Application Example





In practice the policy often becomes optimal before the utility has converged.

Policy Iteration

- Value iteration computes the optimal policy even at a stage when the utility function estimate has not yet converged
- If one action is better than all others, then the exact values of the states involved need not to be known
- Policy iteration alternates the following two steps beginning with an initial policy π_0:
  - Policy evaluation: given a policy π_t, calculate $U_t = U^{\pi_t}$, the utility of each state if π_t were executed.
  - Policy improvement: calculate a new maximum expected utility policy π_{t+1} according to

$$\pi_{t+1}(s) = \operatorname*{argmax}_a \sum_{s'} T(s,a,s')U(s')$$

The Policy Iteration Algorithm

```
function POLICY-ITERATION(mdp) returns a policy
  inputs: mdp, an MDP with states S, transition model T
  local variables: U, U', vectors of utilities for states in S, initially zero
                   π, a policy vector indexed by state, initially random

  repeat
      U ← POLICY-EVALUATION(π, U, mdp)
      unchanged? ← true
      for each state s in S do
          if max_a Σ_{s'} T(s, a, s') U[s'] > Σ_{s'} T(s, π[s], s') U[s'] then
              π[s] ← argmax_a Σ_{s'} T(s, a, s') U[s']
              unchanged? ← false
  until unchanged?
  return P
```

Summary

- Sequential Decision Problems are problems where agents have to execute actions at discrete time steps
  - They can either be deterministic or stochastic
- The framework of Markov Decision Problems is a powerful tool for describing stochastic SDPs
- Value Iteration is a process for calculating the value function and thus the optimal policy
- Policy Iteration is a process for calculating optimal policies by optimizing the relative difference in the value function rather than absolute values leading to faster convergence

Literature

- Reinforcement Learning
  - R. S. Sutton, A. G. Barto, **Reinforcement Learning**, *MIT Press*, 1998, online at: http://www.cs.ualberta.ca/~sutton/book/the-book.html