Overview

I Artificial Intelligence

II Problem Solving

III Knowledge, Reasoning, Planning

IV Uncertain Knowledge and Reasoning

- 12. Quantifying Uncertainty
- 13. Probabilistic Reasoning
- 14. Probabilistic Reasoning over Time
- 15. Probabilistic Programming
- 16. Making Simple Decisions
- **17. Making Complex Decisions**
- 18. Multiagent Decision Making

V Machine Learning

VI Communicating, Perceiving, and Acting

VII Conclusions

Making Complex Decisions

- Sequential Decision Problems
- Algorithms for Markov Decision Processes (MDPs)
- Bandit Problems
- Partially Observable MDPs (POMDPs)
- Algorithms for Solving POMDPs

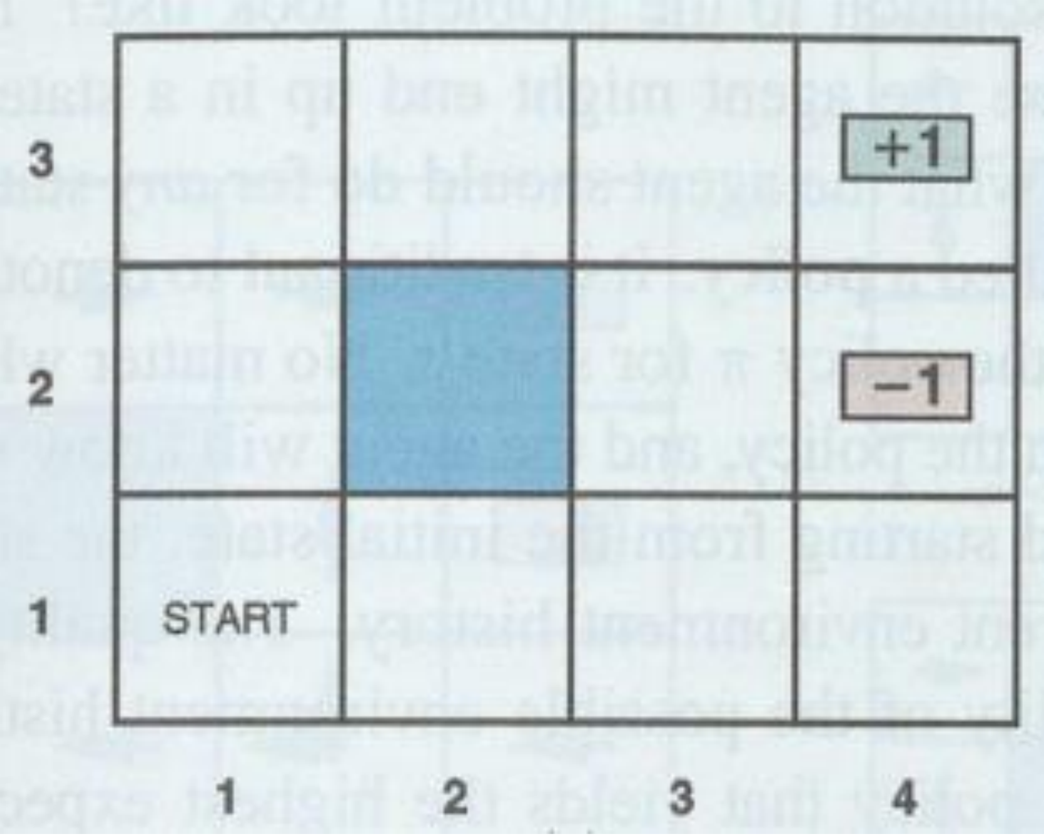
Sequential Decision Problems

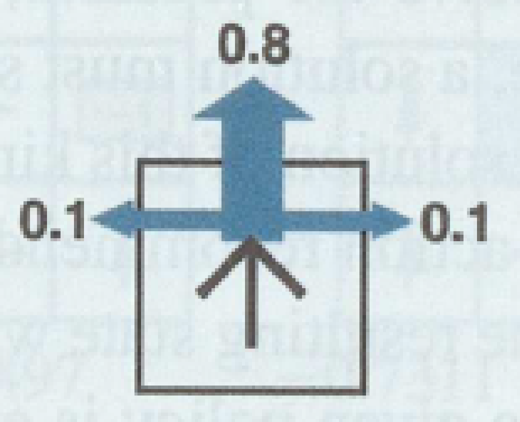
- **Problem:**
  - Often, an agent cannot reach the goal in one step, but needs multiple steps
  - If the outcome of each action (step) is indeterministic, what is the optimal sequence?
  - Indeterminism is represented by a transition model:
    - For each state and action, the probability of successor states is specified
    - Our known techniques (planning and replanning) ignore probabilities and utilities

- **Solution approach:**
  - Specifiy for each state a ruleset (policy) depending on available evidence
    - The ruleset defines implicitely an exponential number of sequential plans
  - The derivation of the rules depends on state utilities and action probabilities

Example for Sequential Decision Problem

- The agent wants to maximize its reward
    - For reaching a terminal state (4,2) and (4,3) the reward is -1 resp. +1.
    - For each movement (action) the reward is -0.04
- In each state, the agent can choose among four actions (up, down, left, right) and has a probability of 80% to reach the intended state, and 20% to move at right angles to the intended direction
    - Collision with a wall results in no movement
- If the agent reaches the state (4,3) e.g. in 10 steps, its total utility is 9 * -0.04 + 1= +0.64





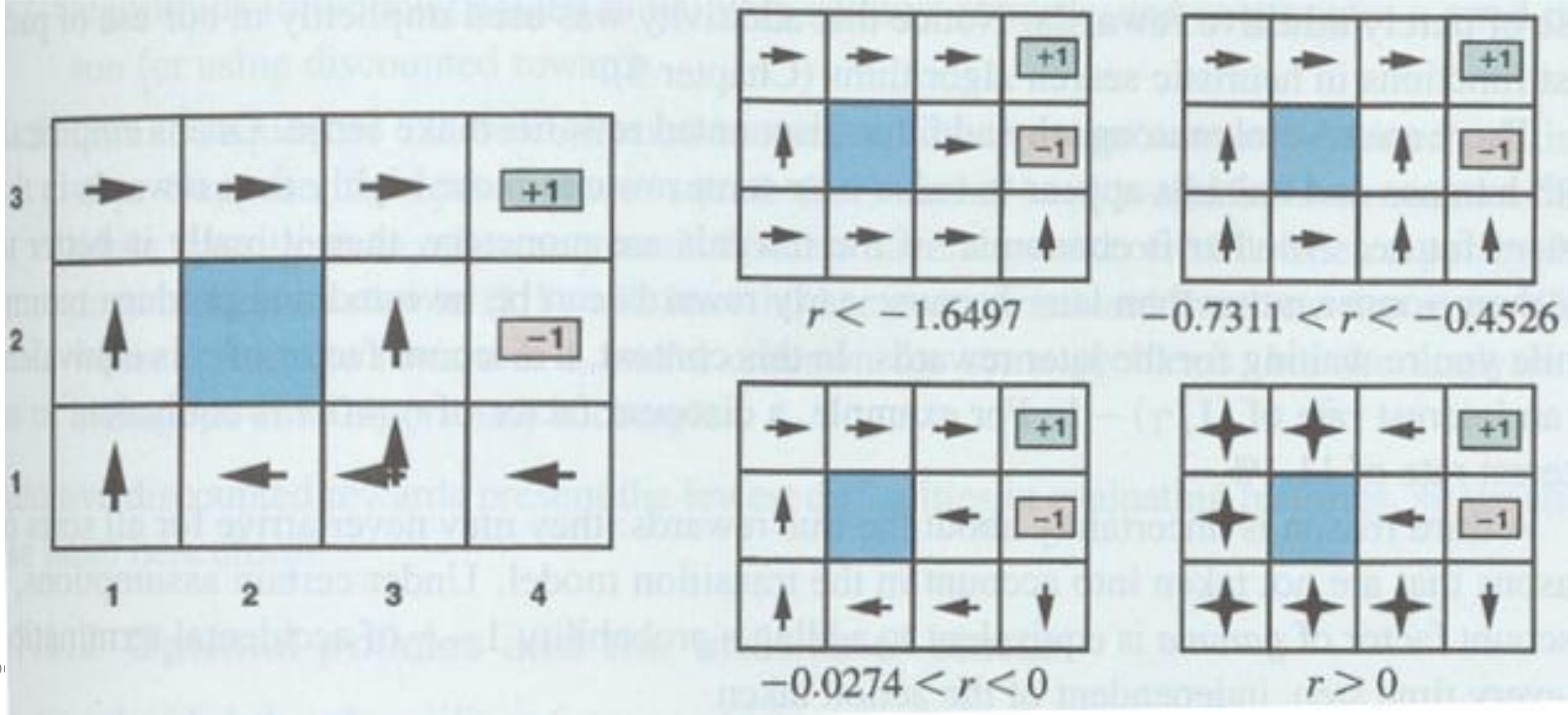
Markov Decison Process (MDP)

- Specification:
  - Set of states (with an initial state s_0)
  - Set of actions in each state
  - Transition model P(s'|s,a)
  - Reward function R(s, a, s')

- Total utility depends on sequence of states and actions (environment history) and is the sum of the reward functions for reaching each state
- Solution is not a fixed action sequence (because of indeterminism), but a policy π
  - The policy recommends for each state s an action $\pi(s)$
- Optimal policy π^* yields the highest expected utility

Examples for Optimal Policies with Different Reward Functions

- Left: Optimal policy with reward of -0.04 between nonterminal states
  - In state (3,1) there are two policies, because both „Up“ and „Left“ are optimal
- Right: Different policies for different rewards between nonterminal states



Utilities over Time

- The optimal policy also depends on the time horizon:
  - **Finite horizon**: Fixed time N, after which nothing matters (game over)
    - Optimal policy depends on how much time is left: **Nonstationary policy**
  - **Infinite horizon** without fixed deadline
    - **Stationary policy** not depending on time left (simpler)
    - But may have terminal states
    - Often with additive discounted rewards
      - Future rewards are less valuable than current rewards
      - Utility of a history: $U_h([s_0,a_0,s_2,a_1,s_2, \ldots]) = R(S_0,a_0,s_1) + \gamma R(S_1,a_1,s_2) + \gamma^2 R(S_2,a_2,s_3) + ..$
      - **Discount factor** γ: number between 0 and 1
        - Justification of discount factor: empirical (intuitive for humans and animals), economic (rewards can be re-invested), unsure future, …
      - With discounted rewards, the utility of an inifinite sequence is finite
      - **Proper policy**: Guaranteed to reach a terminal state (so we can use γ = 1)
      - Infinite sequences can be compared in terms of the **average reward** per time

Optimal Policies and Utility of States

- **Utility of an action sequence**: Sum of discounted rewards of the actions in each state
- **Expected utility of a policy:** Evidence (E) weighted sum of discounted rewards (R) of actions of policy $\pi(S_t)$

  $U^{\pi}(s) = E\ [\sum_{t=0}^{\infty} \gamma^{t}\ \ R(S_t, \pi(S_t), S_{t+1})]$
- **Optimal policy π_s*:** Policy with highest expected utility from all policies:

  $\pi_s{}^* = \text{argmax}_{\pi}\ U^{\pi}(s)$
  - For infinite horizons the optimal policy π^* is independant from the start state, because it states for each state the same policy

Computing the Utilities of States

Computing the optimal policy for a state is choosing the action leading to the neighbor states with highest utility:

$$\pi^*(s) = \text{argmax}_{a \in A(s)} [\sum_{s'} P(S'|s,a) [R(s,a,s') + \gamma U(s')]]$$

Computing the utility of a state depends on the utilities of the neighboring states:

$$U(s) = \max_{a \in A(s)} [\sum_{s'} P(s'|s,a) [R(s,a,s') + \gamma U(s')]]$$

Thus we get n equations with n unknowns, the **Bellman equations**

3	0.8516	0.9078	0.9578	+1
2	0.8016		0.7003	–1
1	0.7453	0.6953	0.6514	0.4279
	1	2	3	4

example utilities (with γ = 1; R = -0.04)

Bellman equation for U(1,1)

For simplicity with discount factor $\gamma = 1$

U(1,1) =

max { [0.8(-0.04 + U(1,2)) + 0.1 (-0.04 + U(2,1)) + 0.1 (-0.04 + U(1,1))],

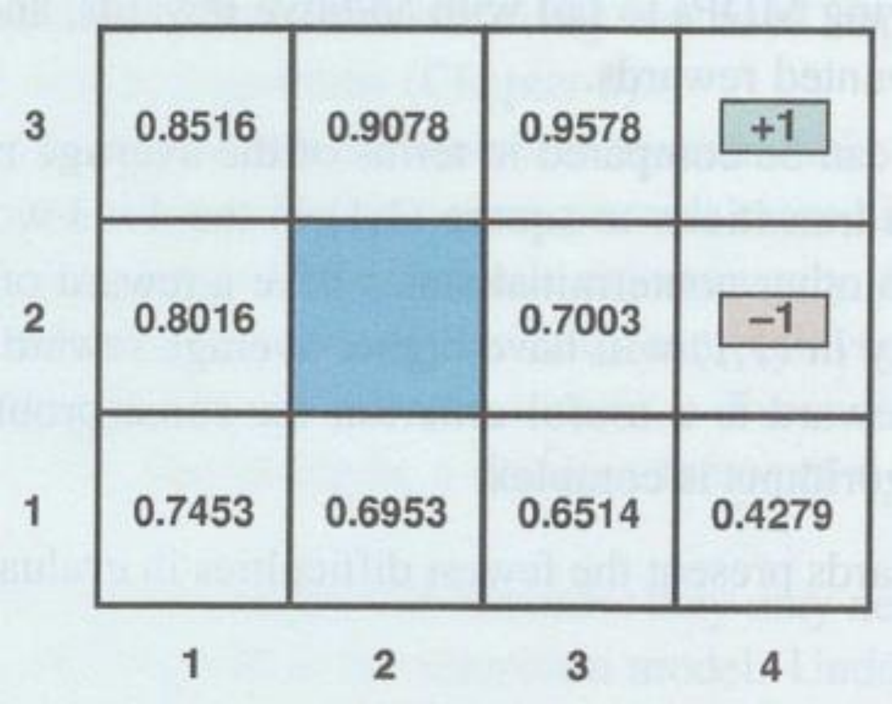
[0.9(-0.04 + U(1,1)) + 0.1 (-0.04 + U(1,2))

[0.9(-0.04 + U(1,1)) + 0.1 (-0.04 + U(2,1))

[0.8(-0.04 + U(2,1)) + 0.1 (-0.04 + U(1,2)) + 0.1 (-0.04 + U(1,1))] }

= 0.7453

where the 4 expressions correspond to UP, Left, Down, Right



Q-Function (Action-Utility Function)

The Q-function Q(s,a) is the expected utility for an action in a state. Thus:

$U(s) = \max_a Q(s,a)$

$\pi^*(s) = \text{argmax}_a Q(s,a)$

The Bellman equation can be rewritten with Q-functions:

$$\mathbf{Q(s,a) = \sum_{s'} P(s'|s,a)\,[R(s,a,s') + \gamma U(s')] = \sum_{s'} P(s'|s,a)\,[R(s,a,s') + \gamma \max_{a'} Q(s',a')]}$$

General function Q-value (mdp, s, a, U) **returns** a utility value

return $\sum_{s'} P(s'|s,a)\,[R(s,a,s') + \gamma\, U[s']]$

Algorithms for MDPs

- Solving n equations with n unknowns, but the equations are nonlinear because of the max operator
- Iterative approach: Start with arbitrary initial values for each state, update the states and repeat this until reaching an equilibrium
  - Value iteration: Iterate, until the utilities of the states converge
  - Policy iteration: Iterate, until the policy stops changing
  - (Linear programming)
  - (Online algorithms for MDPs)

Value Iteration

- Compute utilities of states iteratively until the biggest change (δ) is very small ($\leq \varepsilon(1-\gamma)/\gamma$)
- Bellman update: $U_{i+1}(s) \leftarrow \max_{a \in A(s)} \sum_{s'} P(s'|s,a)\,[R(s,a,s') + \gamma U_i(s')]$ or
  $U_{i+1}(s) \leftarrow \max_{a \in A(s)}$ Q-value (mdp, s, a, U)

function VALUE-ITERATION(*mdp*, ϵ) **returns** a utility function
  inputs: *mdp*, an MDP with states S, actions $A(s)$, transition model $P(s' \mid s, a)$,
    rewards $R(s, a, s')$, 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 relative change in the utility of any state

  repeat
    $U \leftarrow U'$; $\delta \leftarrow 0$
    for each state s **in** S **do**
      $U'[s] \leftarrow \max_{a \in A(s)}$ Q-VALUE(*mdp*, s, a, U)
      if $|U'[s] - U[s]| > \delta$ **then** $\delta \leftarrow |U'[s] - U[s]|$
  until $\delta \leq \epsilon(1-\gamma)/\gamma$
  return U

Example for Value Iteration

U(3,3) = max { [0.8 *1 + 0.1 (-0.04 + U(3,2)) + 0.1 (-0.04 + U(3,3))], //Right
[0.8(-0.04 + U(3,3)) + 0.1 *1 + 0.1 (-0.04 + U(2,3))] //Up
[0.8(-0.04 + U(2,3)) + 0.1 (-0.04 + U(3,3)) + 0.1 (-0.04 + U(3,2))] //Left
[0.8(-0.04 + U(3,2)) + 0.1 * 1 + 0.1 (-0.04 + U(2,3))] } //Down

3	0	0	0	1
2	0		0	-1
1	0	0	0	0
	1	2	3	4

Computations 1. Iteration:

U(3,3): Right: 0.8*1 + 0.1*-0.04 + 0.1*-0.04 = 0.792

U(3,2): Left: 0.8*-0.04 + 0.1*-0.04 + 0.1*-0.04 = -0.04

… (other states = -0.04)

3			→	1
2			←	-1
1				↓
	1	2	3	4

3	–0.04	–0.04	0.792	1
2	–0.04		–0.04	-1
1	–0.04	–0.04	–0.04	–0.04
	1	2	3	4

Computations 2. Iteration:

U(3,3): Right: 0.8*1 + 0.1*(-0.04 -0.04) + 0.1*(0.792-0.04) = 0.8672

U(3,3): Left: **0.8*(-0.04-0.04) + 0.1*(0.792-0.04) + 0.1*(-0.04-0.04) = 0.0032**

U(3,2): Up: 0.8*(0.792-0.04) + 0.1*-1 + 0.1*(-0.04-0.04) = 0.4936

U(2,3): Right: 0.8*(0.792-0.04) + 0.2*(-0.04-0.04) = 0.5856

…

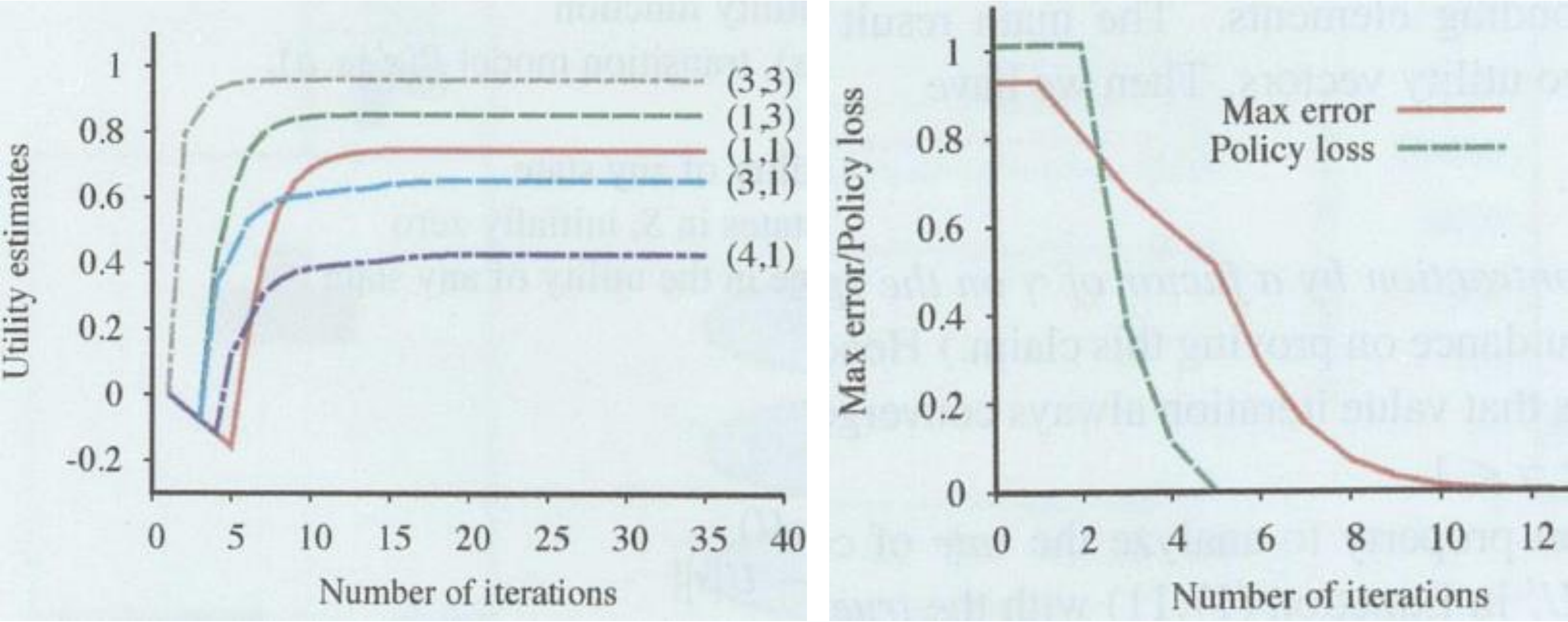
3		→	→	1
2			↑	-1
1				↓
	1	2	3	4

3	–0.08	0.5856	0.8672	1
2	–0.08		0.4936	-1
1	–0.08	–0.08	–0.08	–0.08
	1	2	3	4

Computations 3. Iteration:

…

Value Iteration: Evolution of Utilities and Policy Loss



The value iteration algorithm can be viewed als value propagation; the resulting policy stabelizes quickly

Improvement of Value Iteration

- What is the required exactness of values to find an optimal policy?
  - Sufficient exact to select the best policy.

➢ Iterate until the policy does not change from one iteration to the next

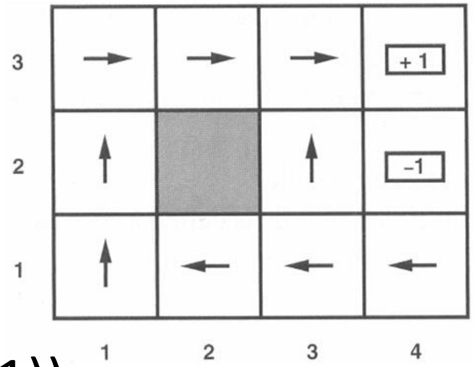
Policy Iteration

- Policy iteration algorithm alternates between two steps, beginning with some initial policy π
  - Policy evaluation: Given a policy π_i, calculate its uility U_l
  - Policy improvement: Calculate a new policy π_{i+1} based on U_i
- Termination, when the policy improvement step yields no change in the utilities

```
function POLICY-ITERATION(mdp) returns a policy
  inputs: mdp, an MDP with states S, actions A(s), transition model P(s' | s, a)
  local variables: U, a vector 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
        a* ← argmax_{a∈A(s)} Q-VALUE(mdp, s, a, U)
        if Q-VALUE(mdp, s, a*, U) > Q-VALUE(mdp, s, π[s], U) then
           π[s] ← a*; unchanged? ← false
  until unchanged?
  return π
```

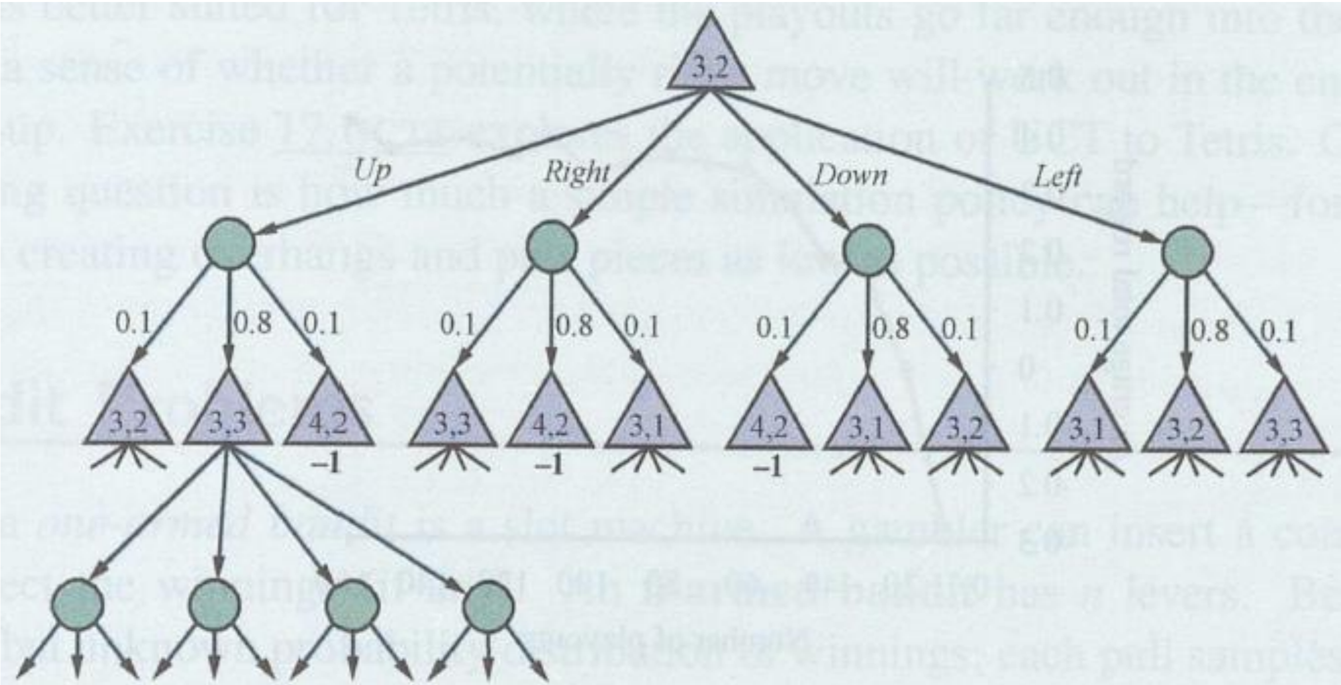
Methods for calculating the Utility in Policy Iteration



- **Analytic version:** Solving n equations with n unknowns with linear algebra methods in $O(n^3)$, since the policy is known
  - Example with π_i (1,1) = Up, π_i (1,2) = Up, etc.
    - $U_i(1,1) = 0.8(-0.04 + U_i(1,2)) + 0.1\,(-0.04 + U_i(2,1)) + 0.1\,(-0.04 + U_i(1,1))$
    - $U_i(1,2) = 0.8(-0.04 + U_i(1,3)) + 0.2\,(-0.04 + U_i(1,2))$
    - …
- **Iterative version:** Simplified version of Belman update in value iteration with given policy
  - $U_{i+1}(s) \leftarrow \sum_{s'} P(s'|s, \pi_i(s))\,[R(s, \pi_i(s), s') + \gamma U_i\,(s')]$ *// i.e. Q-value (mdp, s, $\pi_i(s)$, U)*
  - Must be repeated several times to produce next utility estimate
- **Asynchronous policy iteration:**
  - It is not necessary to update all states
  - Choosing any subset and perform either kind of updating of utilities is possible

Online Algorithms for MDPs

- Value iteration and policy iteration are offline algorithms
  - Generate on optimal solution and then execute it
  - Not successful for large problems (like e.g. Tetris)
  - Approximate solutions are based on reinforcement learning
- Online algorithms are similar to game playing algorithms
  - Significant amount of computation at each decision point
  - Adaptation of ExpectiMiniMax algorithm for probabilistic games



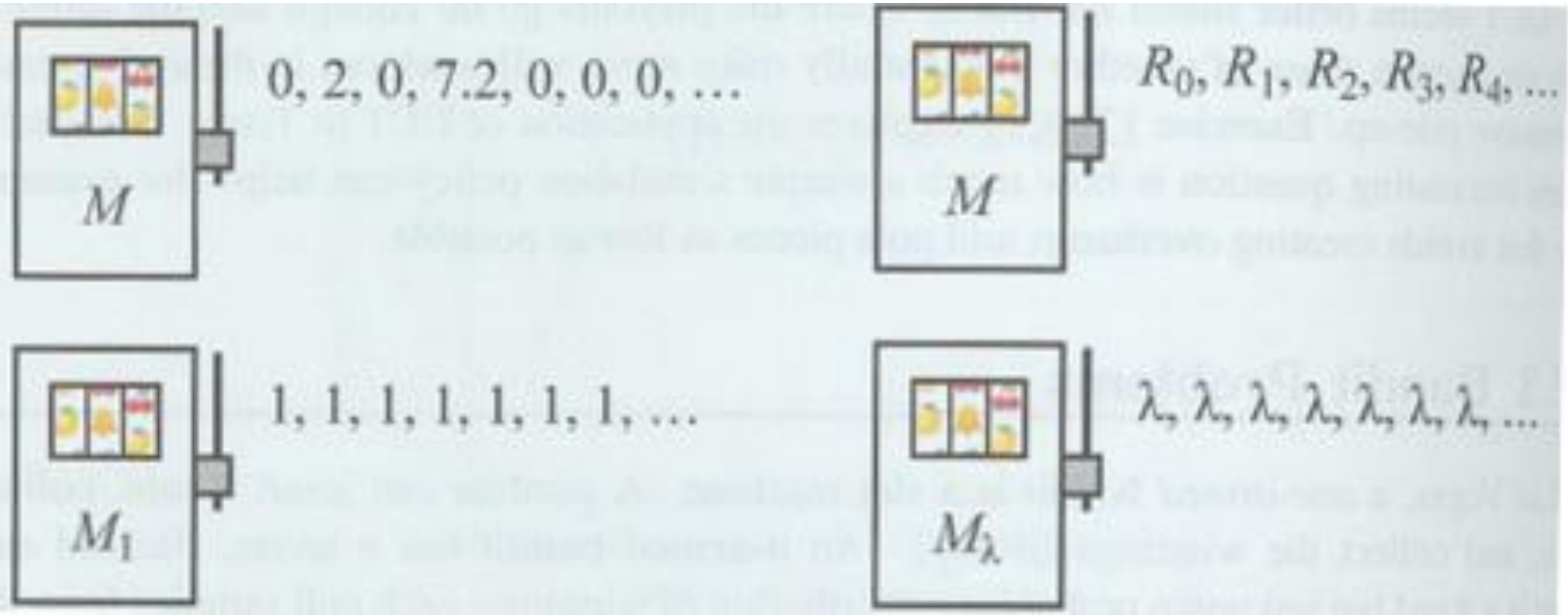
ExpectiMax Algorithm

- Difference to ExpectiMiniMax: Rewards on terminal and non-terminal nodes
- Evaluation function for non-terminal leaves of the tree
- With a low discount factor future rewards are less important allowing to cut the search tree
- Alternate to heuristic evaluation function: Generate N samples from probability distribution P and use the mean: $\sum_x P(x)\, f(x) \approx 1/N \sum_{i=1}^{N} f(x_i)$
- Improvements:
  - The search „tree“ is really a graph, since states appear several times
    - Constrain values of identical states to identical values
    - Adaptation of Bellman equations could be used
    - General approach: Real-time dynamic programming (RTDP)
    - Similar to learning real-time A* (LRTA*) algorithm
    - Useful for moderate sized domains with a high enough chance of repeated states
  - Apply reinforcement learning or an adaption of the Monte Carlo approach for games

Bandit Problems

- In Las Vegas, a **one-armed bandit** is a slot machine. A gambler can insert a coin, pull the lever and collect the winnings (if any). An **n-armed bandit** has n levers.
  - Behind each lever is a fixed, but unknown probability distribution of winnings
- General concept: Choose the best alternative (e.g. medical treatment, possible investment, project funding etc.) based on some samples
- Bandit problems can be defined as Markow reward processes

Example Bandit Problem



- The two bandits can be viewed as one one-armed-bandit with the reward of λ if not pulling an arm
- With just one arm, the only choice is to pull again or to stop (forever)
- Parameter T: Stopping time

With discount factor $\gamma = 0.5$ the bandits yield the followings utilities:

$U(M) = 1.0 * 0 + 0.5 * 2 + 0.5^2 * 0 + 0.5^3 * 7.2 = 1.9$

$U(M_1) = \sum_{t=0}^{\infty} 0.5^t = 2.0$

If switching is allowed once, the best option is switching after the fourth reward:

$U(S) = 1.0 * 0 + 0.5 * 2 + 0.5^2 * 0 + 0.5^3 * 7.2 + \sum_{t=4}^{\infty} 0.5^t = 2.025$

Gittins Index

What is the value of λ of the one-armed bandit, so that an optimal strategy (with the best stopping time) is eqivalent to stop immediately?

$$\max_{T>0} E\left[\left(\sum_{t=0}^{T-1} \gamma^t R_t\right) + \sum_{t=T}^{\infty} \gamma^t \lambda\right] = \sum_{t=0}^{\infty} \gamma^t \lambda$$

which simplifies to

Gittins index:

$$\lambda = \max_{T>0} \frac{E\left(\sum_{t=0}^{T-1} \gamma^t R_t\right)}{E\left(\sum_{t=0}^{T-1} \gamma^t\right)}$$

Computation of Gittins index for example:

T	1	2	3	4	5	6
R_t	0	2	0	7.2	0	0
$\sum \gamma^t R_t$	0.0	1.0	1.0	1.9	1.9	1.9
$\sum \gamma^t$	1.0	1.5	1.75	1.875	1.9375	1.9687
ratio	0.0	0.6667	0.5714	1.0133	0.9806	0.9651

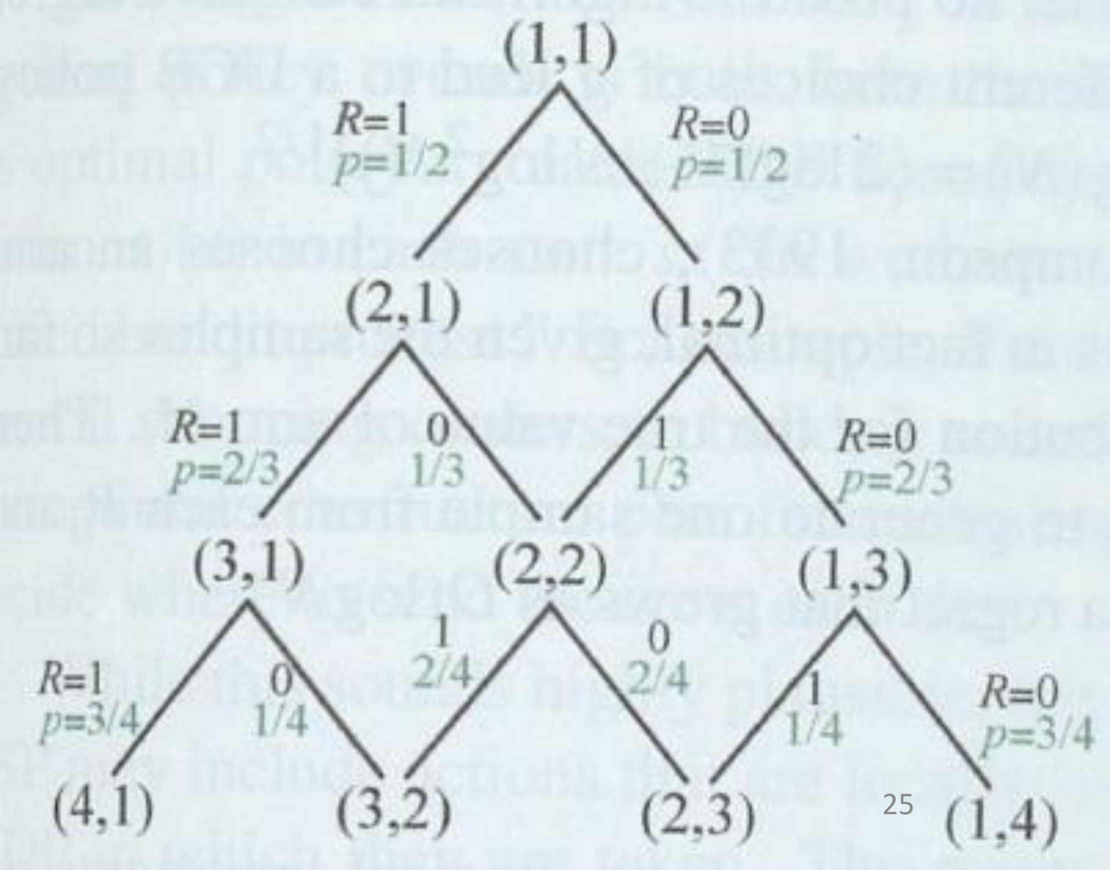
Gittins Index: Consequence

- Optimal policy for any bandit problem:
  - Pull the arm with the highest Gittins index, then update the Gittins index

- Can be expanded into an equivalent MDP problem

Bernoulli Bandit

- **Bernoulli Bandit:** Each arm M_i produces a reward of 0 (failure = f_i) or 1 (success = s_i) with a fixed, but unknown probability μ_i
  - State of an arm is defined by s_i and f_i and the transition probability predicts the next outcome to be 1 by the ratio $s_i / (s_i + f_i)$
  - Counts for s_i and f_i are initialized to 1, so that the initial probability is ½
  - Example with probability p for getting a result R after each sample
  - Should balance selecting the arm with the highest payoff with exploring rarely used arms



Approximately Optimal Bandit Policies

- Calculating Gittins indices for more realistic problems is rarely easy
- Good approximations for combining high pay-off and exploration:
  - Algorithms based on **upper confidence bounds (UCB) heuristic**
    - Compute for each arm a confidence interval
    - Choose the arm with the highest upper bound of the confidence interval
    - UCB(M_i) = $\hat{\mu}_i + g(N)/\sqrt{N_i}$
    - Mit $\hat{\mu}_i$ = geschätzter_Mittelwert $/\sqrt{N_i}$ = proportional zur Standardabweichung; g(N) = geeignet gewählte Funktion, z.B. $g(N) = (2\log(1 + N\log^2 N))^{1/2}$
  - Algorithm based on **Thompson sampling**:
    - Choose an arm randomly according to the probability that the arm is in fact optimal, given the samples so far
      - Simple implementation: Generate for each arm one sample based on its current probability of success and choose the arm that generated the best sample

Further Bandit Related Problems

- Example for bandit problem: Testing new medical treatments on seriously ill patients
  - Goal: Maximizing the total number of successes (saved lives) over time
- Alternate example scenario: Testing different drugs on samples of bacteria
  - Difference: No additional costs for test failures (like lost lives)
    - Implies a tendency to more exploration than bandit problems
  - Goal: Trying to make a good decision as fast as possible („**selection problem**")
  - Candidate solution approach: Monte Carlo tree search algorithm for games with UCB selection heuristic
- Generalisation of bandit process: **Bandit superprocess**, where each arm is a full MDP
  - Several tasks, where one can attend to only one taks at a time (e.g. selecting from multiple projects)
  - Selecting the locally best project not sufficient, but the time for delaying other projects is also important (e.g. spend more money than necessary for a project to be able to do quickly other attractive projects)
    - Computation of „opportunity costs" necessary: How much utility is given up per time step by not devoting that time step to other projects?

Partially Observable MDPs (POMDPs)

- MDP: Environment fully observable
- POMDP: Environment partial observable, i.e. the agent does not know, in which state it is
  - POMDP has same elements as MDP and in addition a sensor model
    - Transition model P(s‘|s,a)
    - Actions A(s)
    - Reward function R(s,a,s‘)
    - Sensor model P(e,s), i.e. probability of perceiving evidence e in state s
  - Example: 4 x 3 world, where the agent does not know its location, but ony has noisy four-bit sensor representing the presence or absence of a wall
  - From sensor model and evidence, agent computes belief state, i.e. a probability distribution of possible states
    - An action changes the belief state
    - POMDPs include the value of information (i.e. actions for improving the belief state)

Algorithms for Solving POMDPs

- The value iteration algorithm for MDPs can be transfered to POMDPs requiring some extensions
  - Is hopelessly inefficient for larger problems – even for the 4 x 3 world.
- Online algorithms for POMDP are more straightforward
  - Start with some prior belief state
  - Choose an action based on some deliberation process on the current belief state
  - Update belief state based on new observation
- Good candidate: Expectimax algorithm for MDPs with belief states instead of physical states
  - Sampling at chance nodes to cut down branching factor
  - For large state space, approximate filtering algorithm like **particle filtering** („sequential importance sampling with resampling“ to focus on high probability regions of state space)
  - For problems with long horizons balance exploration and exploitation similar to MCTS (Monte Carlo Tree Search) with UCT selection policy using UCB (upper confidence bounds)
  - Successful only, if playout in MCTS has a chance of gaining positive rewards

Example for POMDP ExpectiMax algorithm

- (right) Part of ExpectiMax tree with a uniform initial belief state and update after one action and appropriate noisy sensor information
- (below) An example sequence of percepts (error rate ε = 20%), belief states and actions