Autonomous Mobile Systems

The Markov Decision Problem

Value Iteration and Policy Iteration

Cyrill Stachniss Wolfram Burgard

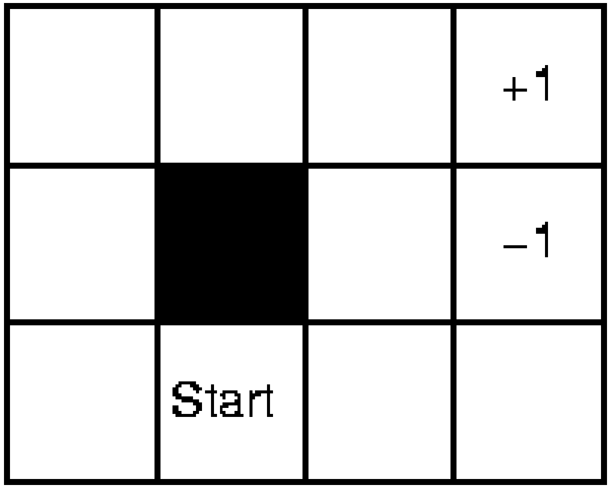
What is the problem?

- Consider a non-perfect system.
- Actions are performed with a probability less then 1.
- What is the best action for an agent under this constraint?

- Example: a mobile robot does not *exactly* perform the desired action.

➡ Uncertainty about performing actions!

Example (1)

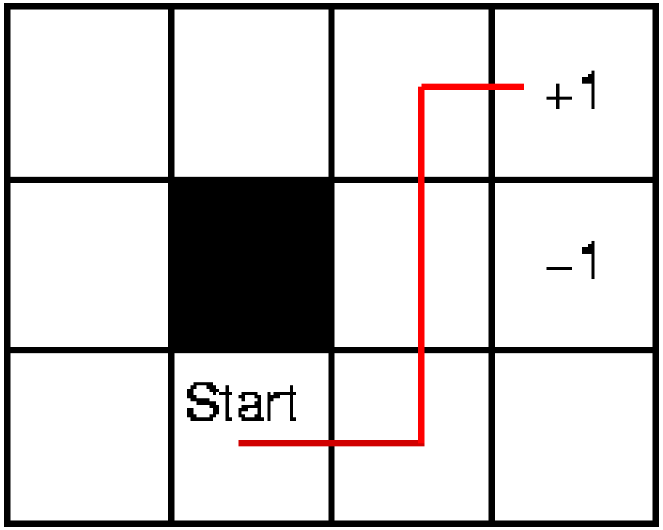


- Bumping to wall “reflects” to robot.
- Reward for free cells -0.04 (travel-cost).

- What is the best way to reach the cell labeled with +1 without moving to –1 ?

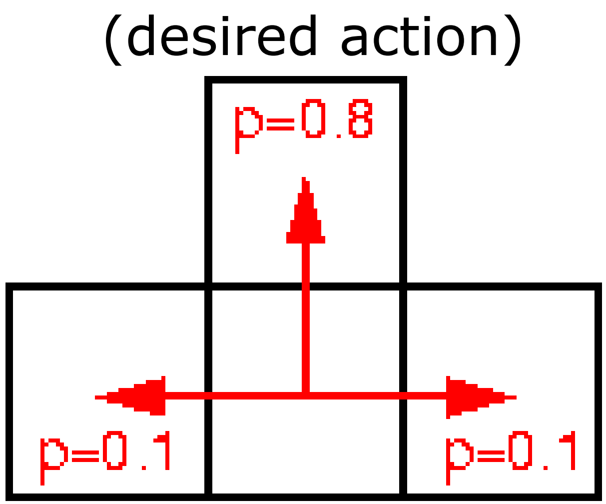
Example (2)

- Deterministic Transition Model: move on the shortest path!



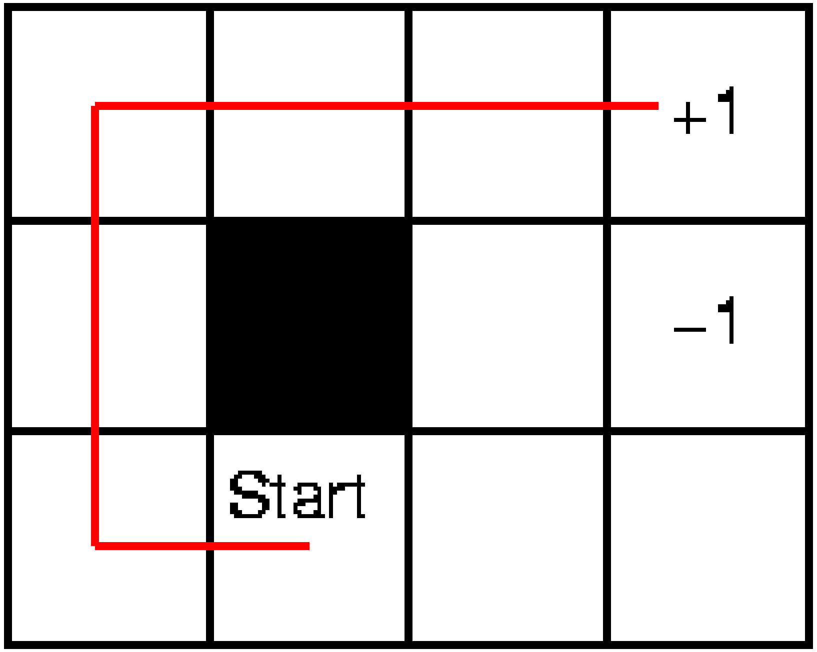
Example (3)

- But now consider the non-deterministic transition model (N / E / S / W):



- What is now the best way?

Example (4)



- Use a longer path with lower probability to move to the cell labeled with –1.
- This path has the **highest overall utility**!

Deterministic Transition Model

- In case of a deterministic transition model use the shortest path in a graph structure.
- Utility = 1 / distance to goal state.
- Simple and fast algorithms exists (e.g. A*-Algorithm, Dijsktra).
- Deterministic models assume a perfect world (which is often unrealistic).
- New techniques need for realistic, non-deterministic situations.

Utility and Policy

- Compute for every state a **utility**:
  "What is the usage (utility) of this state for the overall task?"

- A **Policy** is a complete mapping form states to actions ("In which state should I perform which action?").

$$policy : States \mapsto Actions$$

Markov Decision Problem (MDP)

- Compute the optimal policy in an accessible, stochastic environment with known transition model.

Markov Property:

- The transition probabilities depend only the current state and not on the history of predecessor states.

→ Not every decision problem is a MDP.

The optimal Policy

$$policy^*(i) = \operatorname*{argmax}_a \sum_j M_{ij}^a \cdot U(j)$$

$M_{ij}^a =$ Probability of reaching state *j* form state *i* with action *a*.

$U(j) =$ Utility of state *j*.

- If we know the utility we can easily compute the optimal policy.
- The problem is to compute the correct utilities for all states.

The Utility (1)

- To compute the utility of a state we have to consider a tree of states.
- The utility of a state depends on the utility of all successor states.

→
  - Not all utility functions can be used.
  - The utility function must have the property of separability.
  - E.g. additive utility functions:

$$U([s_0, s_1, \ldots s_n]) = R(s_0) + U([s_1, \ldots s_n])$$

(R = reward function)

The Utility (2)

- The utility can be expressed similar to the policy function:

$$U(i) = R(i) + \max_a \sum_j M_{ij}^a \cdot U(j)$$

- The reward *R(i)* is the “utility” of the state itself (without considering the successors).

Dynamic Programming

- This Utility function is the basis for "dynamic programming".
- Fast solution to compute *n*-step decision problems.
- Naive solution: $O(|A|^n)$.
- Dynamic Programming: $O(n|A||S|)$.
- But what is the correct value of *n*?
- If the graph has loops: $n \longrightarrow \infty$

Iterative Computation

Idea:

- The Utility is computed iteratively:

$$U_{t+1}(i) = R(i) + \max_a \sum_j M_{ij}^a \cdot U_t(j)$$

- Optimal utility: $U^* = \lim_{t \to \infty} U_t$
- Abort, if change in the utility is below a threshold.

The Value Iteration Algorithm

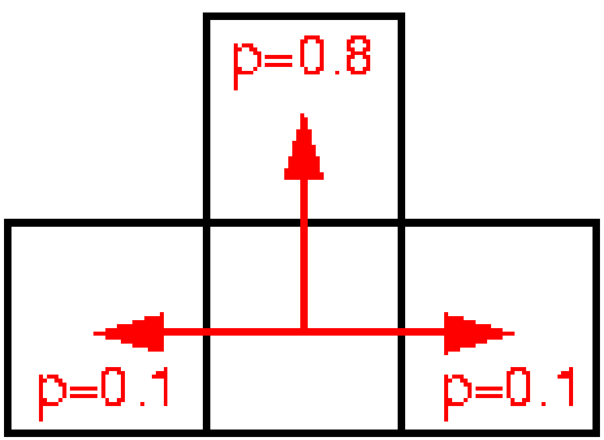
```
function VALUE-ITERATION(M, R) returns a utility function
  inputs: M, a transition model
          R, a reward function on states
  local variables: U, utility function, initially identical to R
                   U', utility function, initially identical to R

  repeat
      U ← U'
      for each state i do
          U'[i] ← R[i] + max_a Σ_j M^a_ij U[j]
      end
  until CLOSE-ENOUGH(U, U')
  return U
```

Value Iteration Example

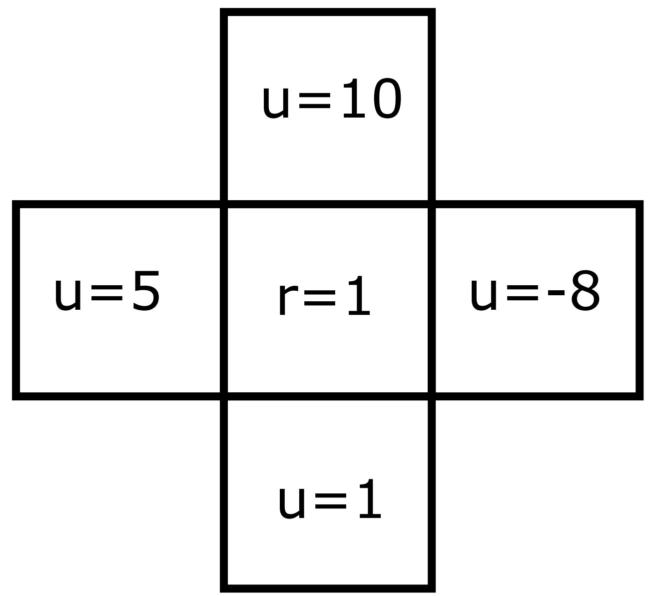
- Calculate utility of the center cell

$$U_{t+1}(i) = R(i) + \max_a \sum_j M_{ij}^a \cdot U_t(j)$$

(desired action=North)



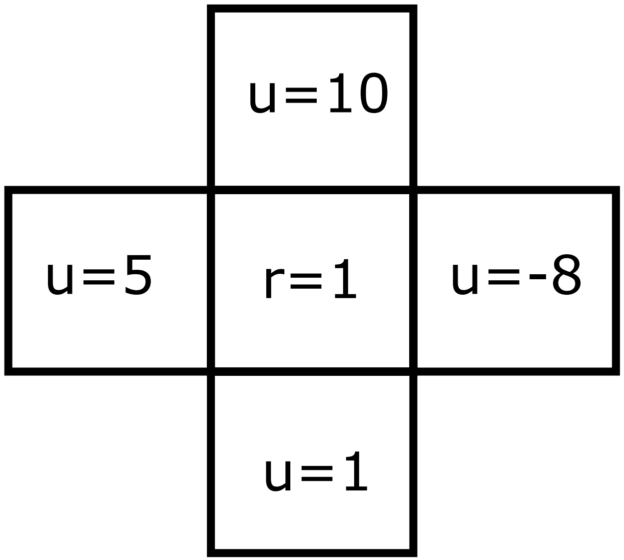
Transition Model



State Space
(u=utility, r=reward)

Value Iteration Example

$$U_{t+1}(i) = R(i) + \max_a \sum_j M_{ij}^a \cdot U_t(j)$$



$$\begin{aligned}
&= reward + \max\{ \\
&\quad 0.1 \cdot 1 + 0.8 \cdot 5 + 0.1 \cdot 10 \quad (\leftarrow), \\
&\quad 0.1 \cdot 5 + 0.8 \cdot 10 + 0.1 \cdot -8 \quad (\uparrow), \\
&\quad 0.1 \cdot 10 + 0.8 \cdot -8 + 0.1 \cdot 1 \quad (\rightarrow), \\
&\quad 0.1 \cdot -8 + 0.8 \cdot 1 + 0.1 \cdot 5 \quad (\downarrow)\} \\
&= 1 + \max\{5.1\,(\leftarrow), 7.7\,(\uparrow), \\
&\quad -5.3\,(\rightarrow), 0.5\,(\downarrow)\} \\
&= 1 + 7.7 \\
&= 8.7
\end{aligned}$$

From Utilities to Policies

- Computes the optimal utility function.
- Optimal Policy can easily be computed using the optimal utility values:

$$policy^*(i) = \underset{a}{\text{argmax}} \sum_j M_{ij}^a \cdot U^*(j)$$

- Value Iteration is an optimal solution to the Markov Decision Problem!

Convergence “close-enough”

- Different possibilities to detect convergence:
  - RMS error – root mean square error
  - Policy Loss
  - …

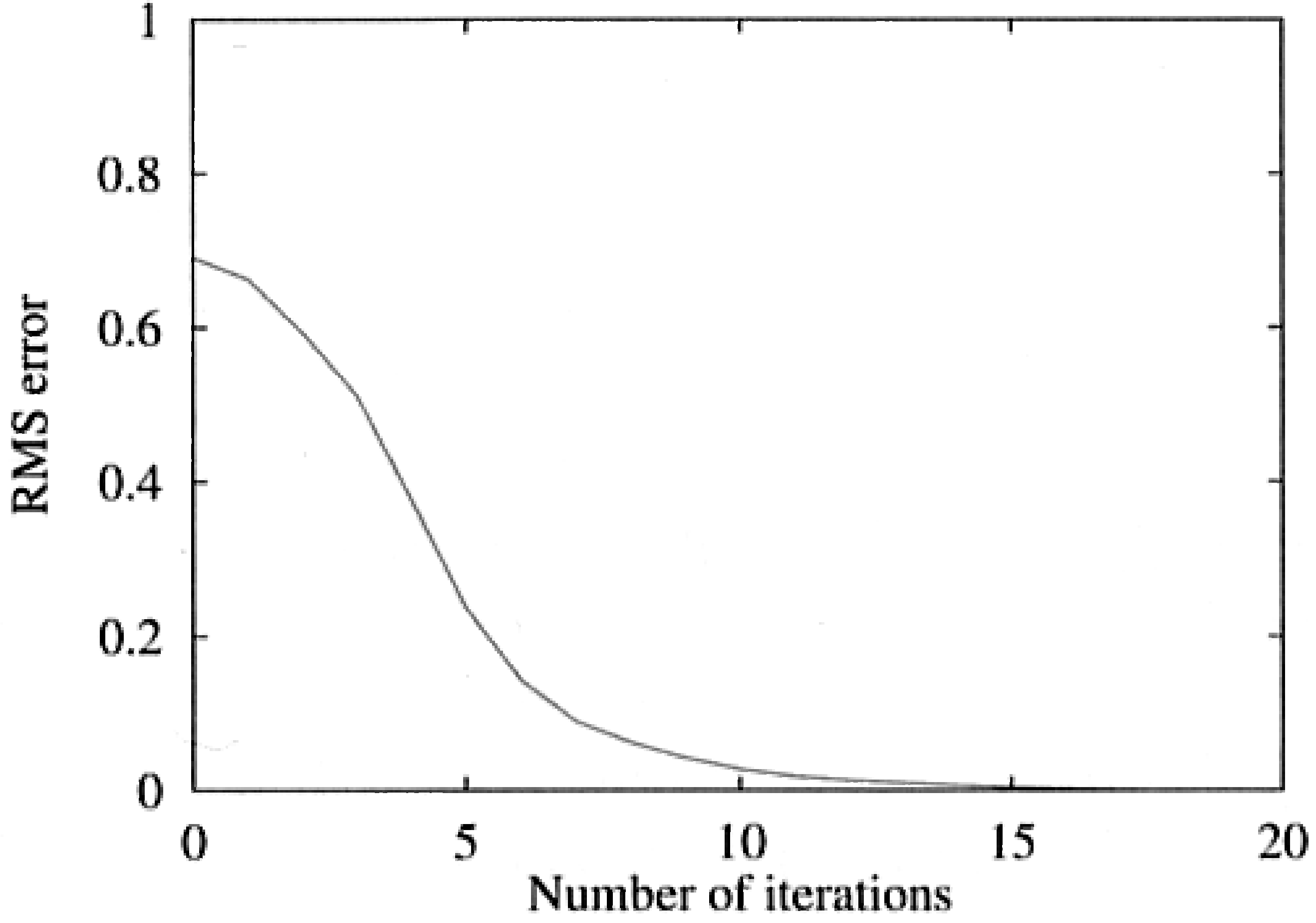
Convergence-Criteria: RMS

$$RMS = \frac{1}{\mid S \mid} \cdot \sqrt{\sum_{i=1}^{|S|} (U(i) - U'(i))^2}$$

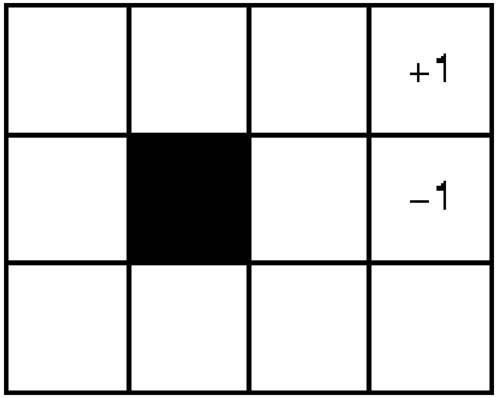
- **`CLOSE-ENOUGH(U,U')`** in the algorithm can be formulated by:

$$RMS(U, U') < \epsilon$$

Example: RMS-Convergence

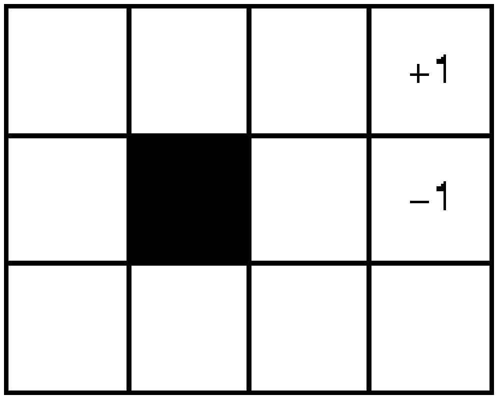


Example: Value Iteration

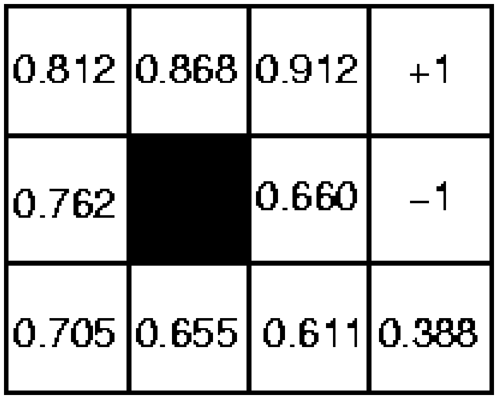


1. The given environment.

Example: Value Iteration

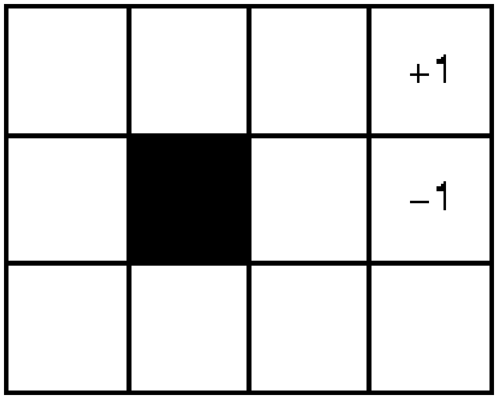


1. The given environment.

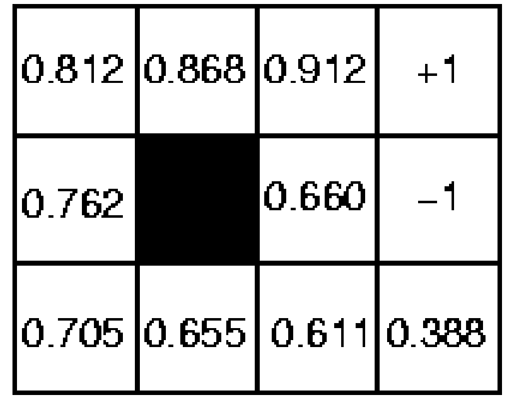


2. Calculate Utilities.

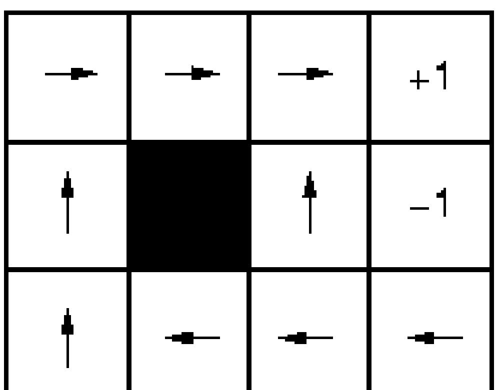
Example: Value Iteration



1. The given environment.

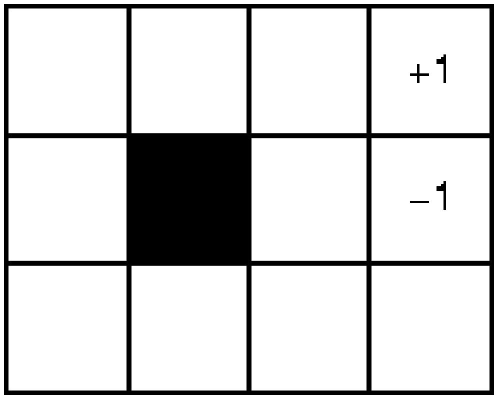


2. Calculate Utilities.

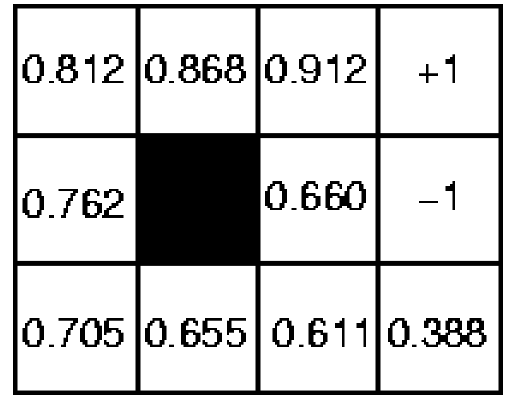


3. Extract optimal policy.

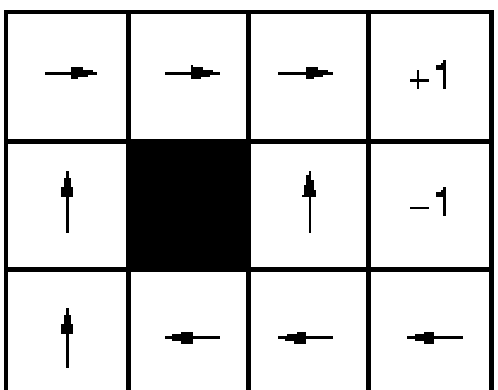
Example: Value Iteration



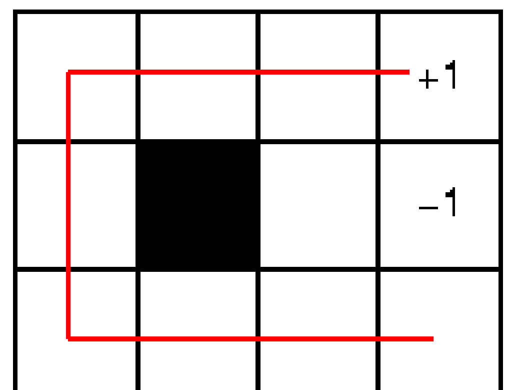
1. The given environment.



2. Calculate Utilities.



3. Extract optimal policy.

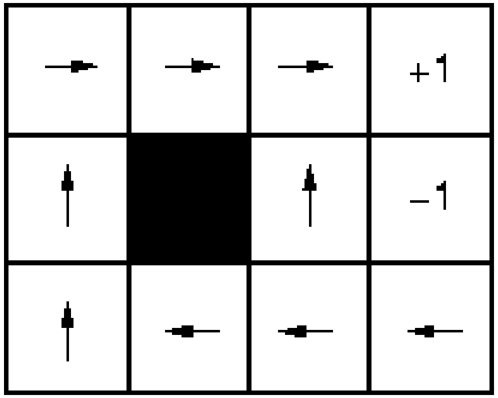


4. Execute actions.

Example: Value Iteration

0.812	0.868	0.912	+1
0.762		0.660	−1
0.705	0.655	0.611	0.388

The Utilities.



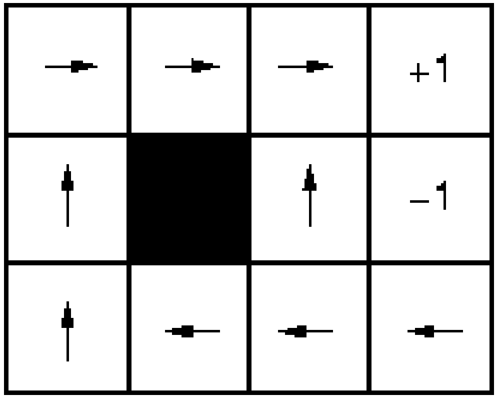
The optimal policy.

- (3,2) has higher utility than (2,3). Why does the polity of (3,3) points to the left?

Example: Value Iteration

0.812	0.868	0.912	+1
0.762		0.660	−1
0.705	0.655	0.611	0.388

The Utilities.



The optimal policy.

- (3,2) has higher utility than (2,3). Why does the polity of (3,3) points to the left?
- Because the Policy is **not** the gradient! It is:

$$policy^*(i) = \underset{a}{\text{argmax}} \sum_j M_{ij}^a \cdot U(j)$$

Convergence of Policy and Utilities

- In practice: policy converges faster than the utility values.

- After the relation between the utilities are correct, the policy often does not change anymore (because of the argmax).

- Is there an algorithm to compute the optimal policy faster?

Policy Iteration

- **Idea** for faster convergence of the policy:
  1. Start with one policy.
  2. Calculate utilities based on the current policy.
  3. Update policy based on policy formula.
  4. Repeat Step 2 and 3 until policy is stable.

The Policy Iteration Algorithm

```
function POLICY-ITERATION(M, R) returns a policy
   inputs: M, a transition model
           R, a reward function on states
   local variables: U, a utility function, initially identical to R
                    P, a policy, initially optimal with respect to U

   repeat
      U ← VALUE-DETERMINATION(P, U, M, R)
      unchanged? ← true
      for each state i do
         if max_a Σ_j M^a_ij U[j] > Σ_j M^{P[i]}_ij U[j] then
            P[i] ← arg max_a Σ_j M^a_ij U[j]
            unchanged? ← false
      end
   until unchanged?
   return P
```

Value-Determination Function (1)

- 2 ways to realize the function **`VALUE-DETERMINATION`**.
- 1st way: use modified Value Iteration with:

$$U_{t+1}(i) = R(i) + \sum_j M_{ij}^{Policy(i)} \cdot U_t(j)$$

- Often needs a lot if iterations to converge (because policy starts more or less random).

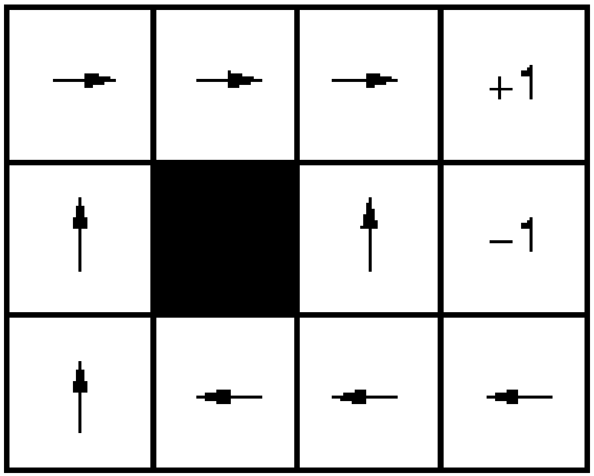
Value-Determination Function (2)

- 2nd way: compute utilities directly. Given a fixed policy, the utilities obey the eqn:

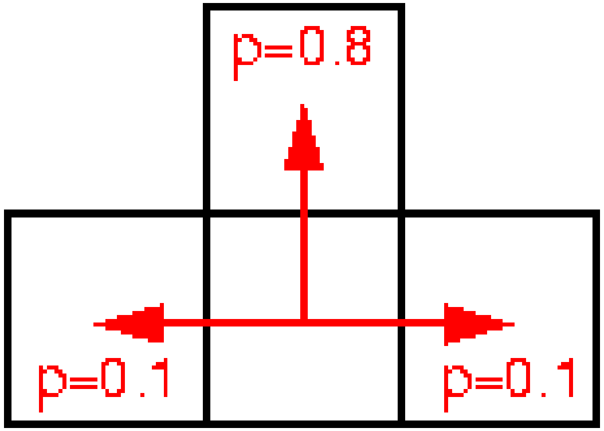
$$\forall i \in S : U(i) = R(i) + \sum_j M_{ij}^{Policy(i)} \cdot U_t(j)$$

- Solving the set of equations is often the most efficient way for small state spaces.

Value-Determination Example



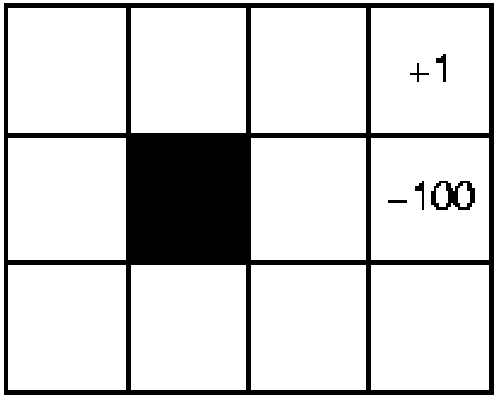
Policy



Transition Probabilities

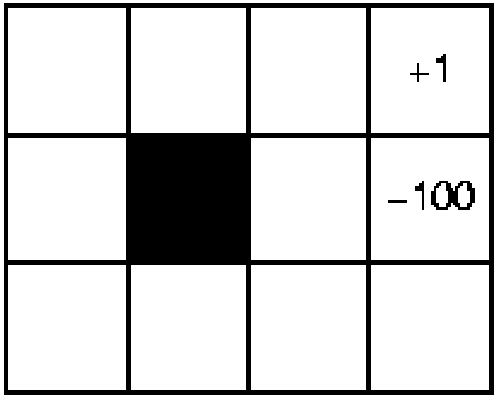
$$
\begin{aligned}
U_{(1,3)} &= 0.8U_{(1,2)} + 0.1U_{(1,3)} + 0.1U_{(2,3)} \\
U_{(1,2)} &= 0.8U_{(1,1)} + 0.2U_{(1,2)} \\
&\dots
\end{aligned}
$$

Value/Policy Iteration Example

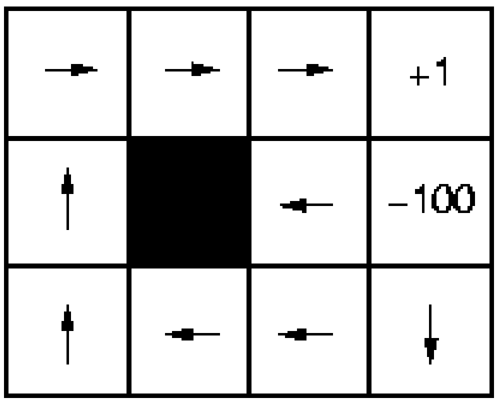


- Consider such a situation. How does the optimal policy look like?

Value/Policy Iteration Example



- Consider such a situation. How does the optimal policy look like?



- Try to move from (4,3) and (3,2) by bumping to the walls. Then entering (4,2) has probability 0.

What's next?   POMDPs!

- Extension to MDPs.
- POMDP = MDP in not or only partly accessible environments.
- State of the system is not fully observable.
- "Partially Observable MDPs".
- POMDPs are extremely hard to compute.
- One must integrate over all possible states of the system.
- Approximations MUST be used.
- We will not focus on POMDPs in here.

Approximations to MDPs?

- For real-time applications even MDPs are hard to compute.
- Are there other way to get the a good (nearly optimal) policy?
- Consider a “nearly deterministic” situation. Can we use techniques like A^*?

MDP-Approximation in Robotics

- A robot is assumed to be localized.
- Often the correct motion commands are executed (but no perfect world!).
- Often a robot has to compute a path based on an occupancy grid.

- Example for the path planning task:
  Goals:
  - Robot should not collide.
  - Robot should reach the goal fast.

Convolve the Map!

- Obstacles are assumed to be bigger than in reality.
- Perform a A* search in such a map.
- Robots keeps distance to obstacles and moves on a short path!

Map Convolution

- Consider an occupancy map. Than the convolution is defined as:

$$P(occ_{x_i,y}) = \frac{1}{4} \cdot P(occ_{x_{i-1},y}) + \frac{1}{2} \cdot P(occ_{x_i,y}) + \frac{1}{4} \cdot P(occ_{x_{i+1},y})$$

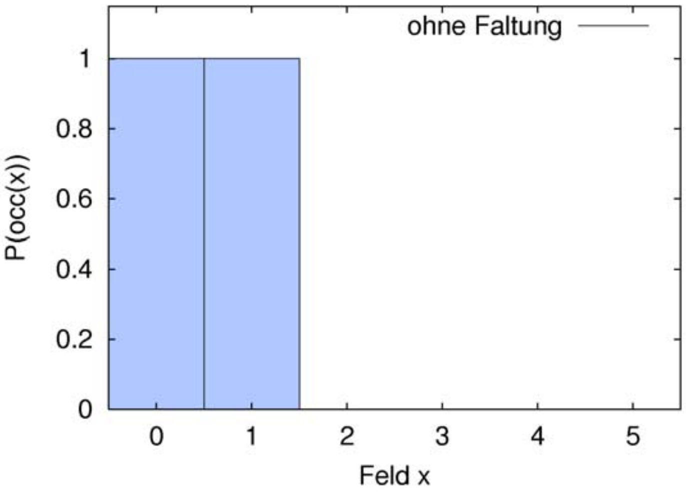
$$P(occ_{x_0,y}) = \frac{2}{3} \cdot P(occ_{x_0,y}) + \frac{1}{3} \cdot P(occ_{x_1,y})$$

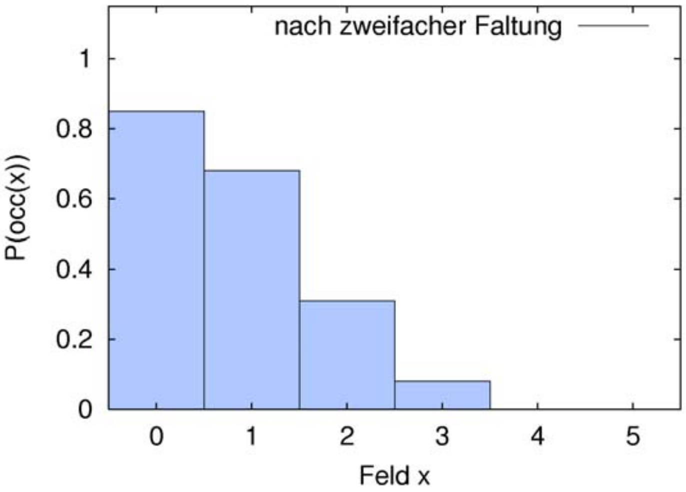
$$P(occ_{x_{n-1},y}) = \frac{1}{3} \cdot P(occ_{x_{n-2},y}) + \frac{2}{3} \cdot P(occ_{x_{n-1},y})$$

- This is done for each row and each column of the map.

Example: Map Convolution

- 1-d environment, cells c_0, ..., c_5





- Cells before and after 2 convolution runs.

A* in Convolved Maps

- The costs are a product of path length and occupancy probability of the cells.
- Cells with higher probability (e.g. caused by convolution) are shunned by the robot.
- Thus, it keeps distance to obstacles.
- This technique is fast and quite reliable.

Literature

This course is based on:

Russell & Norvig: AI – A Modern Approach (Chapter 17, pages 498-)