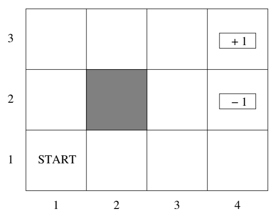
Sequential decision problems

In **sequential decision problems** the utility of agent's actions do not depend on single decisions, expressed with the state, which the agent would have gotten into, as the result of this decision, but rather on the whole sequence of agent's action.

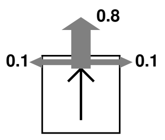
EXAMPLE: an agent is in the field START, and can move in any direction between the field. Its actions ends when it reaches one of the fields (4,2) or (4,3), with the result marked in those fields.



If the problem was fully deterministic — and the agent's knowledge of its position was complete — then the problem would be reduced to action planning. For example, for the above example the correct solution would be the action plan: U-U-R-R-R. Equally good would be the plan: R-R-U-U-R. If the single actions did not cost anything (ie. only the final state did matter), then equally good would also be the plan: R-R-R-L-L-L-U-U-R-R-R, and many others.

The uncertainty of agent's action effects

However, considering the uncertainty, the result of the agent's actions corresponds with its intentions only with some probability. For example, we may assume that the action U (Up) transfers the agent to the desired position with probability 0.8, and with probabilities 0.1 takes the agent left or right. It is only certain that the agent will not end up in the direction opposite to the intended one. For simplicity let us also assume that the presence of the walls does not change this probability distribution, and only makes the agent stay in place, if it turned out that it should move into the wall.

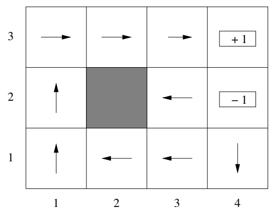


We can now compute the expected values of the sequences of agent's moves. In general, there is no guarantee, that after executing any of the above sequences, the agent will indeed end up in the desired terminal state.

The agent's policy

As opposed to the action planning algorithms, here the agent should work out its strategy not as a specific sequence of actions, but as its **policy**, which is a scheme determining actions the agent should take for any specific state it would find itself in.

We can determine the optimal policy for the previous example problem. Note that at point (3,2) the policy makes the agent move left, which may seem wrong, but allows the agent to avoid ending up in state (4,2). Similarly in state (4,1).

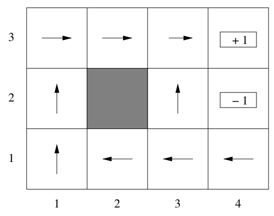


This policy obviously results from the assumption of zero cost of the moves. If the agent's outcome depended not only on the final state, but also on the number of moves, then such conservative policy would probably no longer be optimal.

Considering the cost of the moves

Considering a positive cost of the moves, lowers the result achieved in the final states by the cumulative cost of all the moves. This certainly affects the agent's optimal policy.

For example, the following diagram shows the optimal policy when the cost of the moves equals 1/25 of a unit. Let's note that in the states (4,1) and (3,2) the optimal policy dictates the move toward the state (4,3), despite the risk. However, in the states (2,1) and (3,1) the policy still recommends going around.



Formally, the cost of the moves is introduced as a **reward** function R for states, so $R(s) = -0.04$ for all the nonterminal states.

Markov decision problems

Computing the policy as a complete mapping from states to the set of actions is called the **Markov decision problem** (MDP), if the probabilities of transitions resulting from the agent's actions depend only on the current state of the agent, and not on its history. Such problems are said to have the **Markov property**.

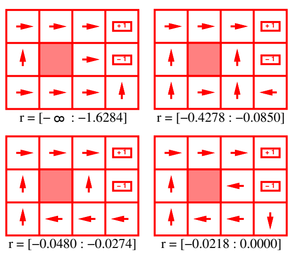
Formally, a Markov decision problem is defined by the following elements:

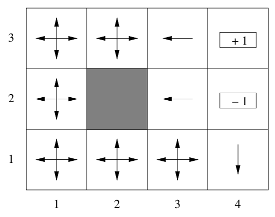
- the set of states with the starting state s_0,
- the set $\text{ACTIONS}(s)$ of actions allowed in state s,
- transition model $P(s'|s, a)$,
- reward function $R(s)$ (also possible is: $R(s, a), R(s, a, s')$).

The solution to an MDP is the policy $\pi(s)$ mapping from states to actions. Let's note that under uncertainty each pass of the agent through the environment according to the policy may result in a different sequence of states, and possibly different outcome. The **optimal policy** $\pi^*(s)$ is the policy achieving the greatest expected utility.

Influence of rewards on the optimal policy

Varying the reward values results in the changes of the optimal policy for a problem. With very high negative rewards the policy recommends going directly to the final state, regardless which one. With the rewards approaching zero the initial policy gradually returns.





In case of positive rewards, it is no longer profitable for the agent to terminate the game, so the optimal policy tells it to avoid the terminal states. Executing actions is profitable, so the agent should keep running, and avoid termination.

The horizon problem

In MDP problems, states do not have utilities, except terminal states. We could speak of the utility of a sequence (history) of states $U_h([s_0, s_1, ..., s_n])$, if it corresponds to an actual sequence of agent's actions, and leads to a final state. It then equals the final result obtained.

Previously we have defined the optimal policy based on the expected utility of a sequence of states. But determining the optimal policy depends on one important factor: do we have an infinite time **horizon**, or is it limited to some finite number of steps? In the latter case, the specific horizon value will likely affect the optimal policy. If it is so, then we say the optimal policy is **nonstationary**. For infinite horizon problems the optimal policy is stationary.

Computing the optimal policies for finite horizon problems is harder, and we will consider only infinite horizon problems.

Discounting

As can be seen in our simple example, infinite action sequences are possible, and can even be optimal policies for an agent. Considering infinite, or very long, sequences is sometimes necessary, eg. if the problem does not have terminal states, or if the agent is not expected to reach them. Such computation is difficult, however, as the reward sums are infinite, and it is hard to compare them then.

To deal with this, a technique of **discounting** is used, which works by reducing the effective utilities of future states by some factor $0 < \gamma < 1$. The utility of a sequence of states H is then defined as $U(H) = \Sigma_i \gamma^i R_i$, or:

$$U_h([s_0, s_1, ..., s_n]) = R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + ... + \gamma^n R(s_n)$$

For $\gamma < 1$ and $R \leq R_{max}$ the utilities so defined are always finite.

The discounting technique has an intuitive justification in many real life problems. It reflects the smaller value of very distant rewards. Likewise, the technique of discounting is used in economy to evaluate the investments.

Proper policies and averaging

In the case of infinite action sequences there are other approaches possible, beside discounting. For example, the **average reward**, computed for a single step, can be used as the utility of a sequence.

If the problem has terminal states, then it is possible to determine a policy which guarantees bringing the agent to one of these states. Then, it is not necessary to analyze infinite sequences. The policies guaranteeing bringing the agent to one of the terminal states are termed **proper**.

The properties of the utilities of state sequences

A utility function for the sequences of states is called **separable**, if:

$$U([s_0, s_1, ..., s_n]) = f(s_0, U([s_1, ..., s_n]))$$

In our example 4×3 problem the utility function is separable since we can compute it using the formula:

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

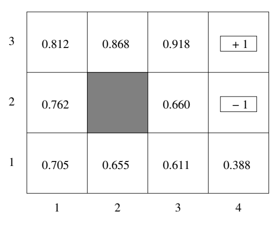
We call a utility function **additive** if it has the following property:

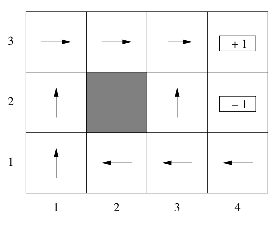
$$U([s_0, s_1, ..., s_n]) = R(s_0) + U([s_1, ..., s_n])$$

It turns out that for many practical cases the utility functions are additive. For example, when considering the cost function in the state space search, we implicitly assumed that they are additive. The incurred cost in a state was simply the cost in the previous state, plus the cost of the move.

Computing the optimal policy — utilities of the states

In order to determine the optimal policy it would be useful to have state utilities, such as these marked in the diagram on the left (the question where these came from we shall postpone until later). We could then employ the MEU (Maximum Expected Utility) principle, and for each state designate the move, which maximizes the expected utility.





However, in MDP problems states do not have utilities, except for the final states. The "utility" of a state (intermediate) depends on the agent's policy, ie. what it intends to do in that state. At the same time, the agent's policy depends on the "utilities" of the states.

We can introduce state utilities based on policies.

The utilities of states

The utility of a state with respect to a policy can be defined as the expected value of the rewards obtained by initiating actions from this state:

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

By S_t we denote here the random variable signifying the state the agent will be at step t after starting from state s and executing the policy π.

It turns out that, even though theoretically the optimal policy $\pi^* = \underset{\pi}{\operatorname{argmax}}\, U^{\pi}(s)$ may depend on the choice of the starting state, for the decision processes with the Markov property, for infinite sequences with discounting, there is no such dependence. The agent's optimal policy, determining all her actions, is independent on the starting point.

For the utility of a state $U(s)$ we will take its utility computed with respect to the optimal policy $U^{\pi^*}(s)$.

Dynamic programming

The optimal policy $\pi^\star$, being a function defined on the set of states, may then be associated with the (yet unknown) state utility function:

$$\pi^*(s) = \operatorname*{argmax}_a \sum_{s'} P(s'|s,a)U(s')$$

where $P(s'|s,a)$ is the probability that the agent will reach the state s' if she executes the action a in the state s.

Since we want to express the utility of a state as the expected value of a discounted sum of rewards of a sequence of states, then it can be tied to the utilities of its descendant states with the following equation (Bellman, 1957):

$$U(s) = R(s) + \gamma \max_a \sum_{s'} P(s'|s,a)U(s')$$

For the n states we thus obtain n equations — unfortunately nonlinear, due to the $\max$ term — with n unknowns. Solving these equations is called **dynamic programming**.

n-step decision problems

If in some problem the final states were achieved with known utilities in exactly n steps, then we could solve the Bellman equation by first determining the utilities for the states at step $n-1$, then at step $n-2$, etc., until reaching the start state. Problems of such type are called **n-step decision problems**, and solving them is relatively easy.

Unfortunately, in most practical cases we may not assume to have constant, n-step sequences, due, for example, to looping.

The value iteration algorithm

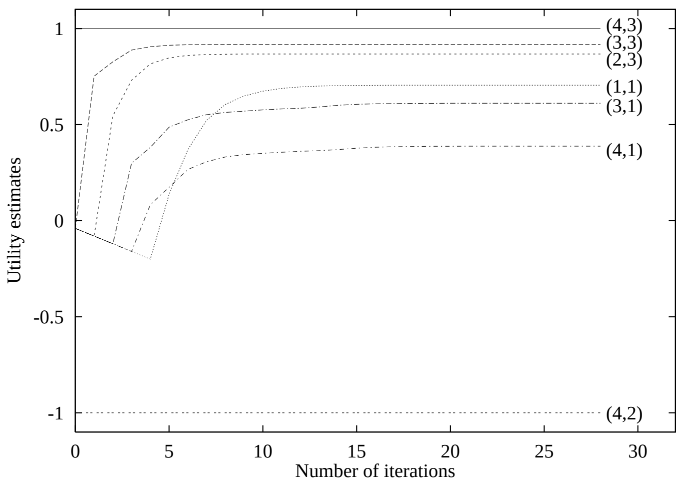
For problems, which cannot be stated as the above n-step decision problems, we can compute approximate values of the state utilities in an iterative procedure called the **value iteration**:

$$U_{t+1}(s) = R(s) + \gamma \max_a \sum_{s'} P(s'|s,a)U_t(s')$$

At step $(t=0)$ we assume arbitrary state utility values, and at the subsequent steps of the algorithm we compute their subsequent approximations.

The algorithm may be terminated by comparing the obtained utilities and estimating the error. Note that the optimal agent's policy may be precisely determined by approximate utility values, even before they converge.

The value iteration algorithm: an example



Convergence of value iteration

As we saw in the example the value iteration procedure converged nicely in all states. The question is, is it always this way?

It turns out that it is. The value iteration algorithm always leads to stable values of state utilities, which are the sole solutions of the Bellman equation. The number of iterations of the algorithm needed to reach an arbitrary error level ϵ is given by the following formula, where $R_{\max}$ is the upper bound on the reward values:

$$N = \lceil \log(2R_{\max}/\epsilon(1-\gamma))/\log(1/\gamma) \rceil$$

Convergence of value iteration — further remarks

- In practice, the following termination condition can be used for the value iteration: $||U_{i+1} - U_i|| < \epsilon(1-\gamma)/\gamma$
- In practice, the optimal policy can be reached much further than the utility values stabilize with desired small errors.
- N grows unboundedly when γ approaches one. The convergence can be sped up by decreasing γ, although this implies shortening the agent's horizon, and neglecting the long-term effects.
- For $\gamma = 1$, if there exist terminal states, similar convergence criteria and error estimates can be derived.

The policy iteration algorithm

Just because the optimal policy is often relatively insensitive to the specific values of the utilities, it can be computed by a similar iterative process, called the **policy iteration**. It works by selecting an arbitrary initial policy π_0, and initial utilities, and then updating the utilities determined by the policy, according to the following formula:

$$U_{t+1}(s) = R(s) + \gamma \sum_{s'} P(s'|s, \pi_t(s)) U_{t+1}(s')$$

alternating it with subsequent update of the policy

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

In the above formulas $\pi_t(s)$ denotes the action designated by the current policy for the state s. The first formula gives a set of linear equations, which can be solved exactly for U_{t+1} in $O(n^3)$ time (they are the exact utilities for the current approximate policy).

The policy iteration algorithm (cntd.)

The policy iteration algorithm terminates when the policy update step make no change. Since for a finite space there exist a finite number of policies, the algorithm is certain to terminate.

For small state spaces (n in $O(n^3)$) the above procedure is often most efficient. For large spaces, however, the $O(n^3)$ causes it to run very slowly. In these cases a **modified policy iteration** can be used, which works by iteratively updating the utilities — instead of computing it exactly each time — using a simplified Bellman updating given by the formula:

$$U_{t+1}(s) = R(s) + \gamma \sum_{s'} P(s'|s, \pi_t(s)) U_t(s')$$

Compared with the original Bellman equation the determination of the optimal action has been dropped here, since the actions are determined here by the current policy. Thus this procedure is simpler, and even several such update steps can be made before the next policy iteration step (updating the policy).

The case of uncertain state information — POMDP

In a general case the agent may not be able to determine the state it ended up in after taking an action, or rather can only determine it with a certain probability. Such problem are called **partially observable Markov decision problems (POMDP)**. In these problems the agent must compute the expected utility of its actions, taking into account both their various possible outcomes, and various possible new information (still incomplete), that it may acquire, depending on the state it ends up in.

The solution to the problem can be obtained by computing the probability distribution over all possible states that the agent can possibly be in, by considering the uncertain information about its environment that it was able to accumulate. In the general case, however, this computation is made difficult by the fact, that undertaking an action causes the agent to acquire new information, which may change its knowledge of the environment in ways difficult to consider. In practice, the agent must take into account new information it may acquire, along with the states it may transfer to. This may involve computing the value of information, which was covered earlier.

POMDP — formal description

A POMDP problem is defined by the following four elements:

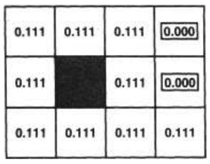
- the set of states, but with no starting state s_0,
- the set $\textsc{Actions}(s)$ of actions allowed in state s,
- the transition function: $P(s'|s, a)$, which is a probability distribution of reaching the state s' after executing action a in state s,
- reward function: $R(s)$,
- sensor model: $P(e|s)$, which is a probability distribution of receiving the evidence e (partially erroneous) in state s,
- initial belief state: b_0.

In POMDP problems there is no assumption about knowing the initial state. Instead, the agent has to work with the **belief state** $b(s)$, which is a probability distribution over all possible states s. Initially, we only know the initial belief state b_0.

The task is to compute such policy, that would allow the agent to reach the goal with the highest probability. During the course of action the agent will change her belief state, due both to the newly received information, and to executing the actions by itself.

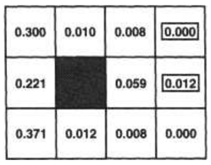
POMDP: an example

Let us again consider the agent in the 4x3 environment, except in this case the agent does not know which initial state it is in, and must assume equal probabilities $\frac{1}{9}$ of being in each of the nonterminal states.
What would the agent's optimal policy be now?

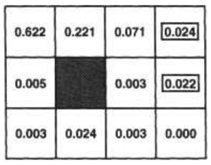


0.111	0.111	0.111	$\boxed{0.000}$
0.111	■	0.111	$\boxed{0.000}$
0.111	0.111	0.111	0.111

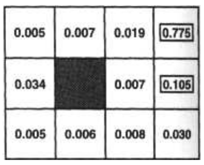
The figures below show subsequent probability distributions of the agent position after executing subsequently five of each of the following actions: Left, Up, and Right. It is an extremely cautious and conservative policy, although quite wasteful. Even though the agent ends up in the "good" terminal state with probability 0.775, the expected utility of such sequence of moves is only 0.08.



0.300	0.010	0.008	$\boxed{0.000}$
0.221	■	0.059	$\boxed{0.012}$
0.371	0.012	0.008	0.000



0.622	0.221	0.071	$\boxed{0.024}$
0.005	■	0.003	$\boxed{0.022}$
0.003	0.024	0.003	0.000



0.005	0.007	0.019	$\boxed{0.775}$
0.034	■	0.007	$\boxed{0.105}$
0.005	0.006	0.008	0.030

Solving POMDP

The key to solving the POMDP is the understanding that the optimal action depends only on the agent's belief state. Since that agent does not know her actual state (and will never learn it in fact), her optimal policy must be a mapping $\pi^{\star}(b)$ from belief states to actions. The subsequent belief states can be computed using the formula:

$$b'(s') = \alpha P(e|s') \sum_s P(s'|s,a) b(s)$$

where $P(e|s')$ is the probability of receiving the observation e in state s', and α is an auxiliary constant normalizing the sum of the belief states to 1.

The work cycle of a POMDP agent, assuming she has already computed her complete optimal policy $\pi^{*}(b)$, is then as follows:

1. For the current belief state b, execute the action $\pi^{*}(b)$.
2. Receive the observation e.
3. Move to the belief state $b'(s')$, and repeat the cycle.

The belief state space

Since the MDP model considers the probability distributions and permits to solve such problems, a POMDP problem can be transformed to an equivalent MDP problem defined in the belief space. In this space we operate on the probability distribution of the agent reaching the set of beliefs b' where she currently has the set of beliefs b and executes the action a. For a problem with n states, b are n-element real valued vectors.

Note that the belief state space, which we obtained while studying the POMDP problems, is a continuous space, unlike the original problem. Furthermore, it is typically multi-dimensional. For example, for the 4×3 environment it has 11-dimensions.

The above value iteration and policy iteration algorithms are not applicable to such problems. Solving them is computationally very hard (PSPACE-hard).

Converting POMDP to MDP

$$\begin{aligned}
P(e|a,b) &= \sum_{s'} P(e|a,s',b) P(s'|a,b) \\
&= \sum_{s'} P(e|s') P(s'|a,b) \\
&= \sum_{s'} P(e|s') \sum_s P(s'|s,a) b(s)
\end{aligned}$$

$$\begin{aligned}
P(b'|b,a) &= P(b'|a,b) = \sum_e P(b'|e,a,b) P(e|a,b) \\
&= \sum_e P(b'|e,a,b) \sum_{s'} P(e|s') \sum_s P(s'|s,a) b(s)
\end{aligned}$$

where

$$P(b'|e,a,b) = \begin{cases} 1 & \text{if } b'(s') = \alpha P(e|s') \Sigma_s P(s'|s,a) b(s) \\ 0 & \text{otherwise} \end{cases}$$

The above equation can be taken as a definition for the transition model for the belief state space. We need to redefine the reward function:

$$\rho(b) = \sum_s b(s) R(s)$$

and all the elements defined above constitute a totally observable Markov decision process (MDP) over the belief state space.

It can be proved, that the optimal policy $\pi^{*}(b)$ for this MDP is also the optimal policy for the original POMDP problem.

Computing optimal policies for POMDP's

A sketch of the algorithm: we define a policy $\pi(b)$ for regions of the belief space, where for one region the policy designates a single action. An iterative process analogous to the value or policy iteration then updates the region boundaries, and may introduce new regions.

The optimal policy computed with this algorithm for the above example is:

[L, U, U, R, U, U, (R, U, U)*]

(the cyclically repeating R-U-U sequence is necessary due to the uncertainty of the agent ever reaching the terminal state). The expected effect of this solution is 0.38, which is significantly better than for the naive policy proposed earlier (0.08).