POMDP: Intorduction to Partially Observable Markov Decision Processes

Hossein Kamalzadeh, Michael Hahsler

2018-11-30

In this document we will go through various features of the POMDP package, provide explanations on how the package functions, what functions the package provide the users with, how to implenet the package and its functions on a real case, how to define arguments of the functions, and what each function's output have to say. We will also give a real case example and solve it and visualze the output using the package. We will also provide a very brief introduction on POMDPs to facilitate the understanding of how this package works.

This package utilizes the 'pomdp-solve' program (written in C) to solve problems that are formulated as partially observable Markov decision processes, a.k.a. POMDPs [1]. The solver actually have the option to use various POMDP solution algorithms to solve problems but the pomdp function in this R package uses only the Finite Grid method [2] to do so. The function provides user with various ways of defining the pomdp components such as states, actions, transitions and rewards and then transforms all the inputs into pomdp-friendly structures and feeds them to 'pomdp-solver'. Given there is an optimal solution, the function provides the optimal solution including the optimal policy.

Introduction on POMDPs

A partially observable Markov decision process (POMDP) is a combination of an MDP and a hidden Markov model. At each time point, the agent gets to make some observations that depend on the state. The agent only has access to the history of observations and previous actions when making a decision. It cannot directly observe the current state. The POMDP framework is general enough to model a variety of real-world sequential decision processes. Applications include robot navigation problems, machine maintenance, and planning under uncertainty in general. The general framework of Markov decision processes with incomplete information was described by Karl Johan Åström in 1965 in the case of a discrete state space, and it was further studied in the operations research community where the acronym POMDP was coined. It was later adapted for problems in artificial intelligence and automated planning by Leslie P. Kaelbling and Michael L. Littman [4].

A POMDP is a 7-tuple (S , A , T , R , Ω , O , λ), where

- S: is a set of states,
- A: is a set of actions,
- T: is a set of conditional transition probabilities between states,
- R: S $\times$ A $\rightarrow$ R is the reward function.
- Ω: is a set of observations,
- O: is a set of conditional observation probabilities, and
- $\lambda \in$: [0 , 1] is the discount factor.

At each time period, the environment is in some state $s \in$ S. The agent takes an action $a \in$ A, which causes the environment to transition to state s' with probability $T(s' \mid s, a)$. At the same time, the agent receives an observation $o \in \Omega$ which depends on the new state of the environment with probability $O(o \mid s', a)$. Finally, the agent receives a reward r equal to $R(s, a)$. Then the process repeats. The goal is for the agent to choose actions at each time step that maximize its expected future discounted reward.

In the next section we will see how this package implements all the componenets of a POMDP model in a single function and then in what form it provides the solution to the problem.

Package Functionality

This package has two sets of functions. One is used to provide all the inputs (corresponding to the POMDP components defined above) and the other is used to provide different features of the solution to the defined problem. The first set contains only one function under the name `pomdp` and takes all the components of a POMDP, creates the model, solves it and then provides the solution in an object of class POMDP. The second set of functions, contains `alpha`, `belief`, `belief.proportion`, `initial.node`, `model`, `pg`, `plot`, `solver.output`, and `total.expected.reward`. Every one of these functions provides a single feature of the solution provided by the main `pomdp` function. We will here go through each function and their arguments one by one and provide of how we can define arguments in different ways.

First, we will go through all the arguments of `pomdp` function. The `pomdp` function has the following arguments, each corresponds to one of the POMDP components.

```
pomdp(discount, states, actions, observations, start, transition_prob, observation_prob,
reward, values = "reward", grid_size, verbose = FALSE)
```

Here are the description of each of these arguments:

- **`discount`:** This argument corresponds to the discount factor (λ) in POMDPs. It should be a numeric, a real number in $[0, 1]$.

```
discount <- 0.9
```

- **`states`:** This argument simply is a way to define S or the set of all the possible states of a POMDP. It should be a vector of strings specifying the names of the states as you see in the example below.

```
states <- c("state1" , "state2" , "state3")
```

- **`actions`:** This argument is for defining A or the set of all possible actions in our model. It should be a vector of strings specifying the names of the actions available.

```
actions <- c("action1" , "action2")
```

- **`observations`:** This argument just like the two previous arguments is for defining Ω, the set of all possible observation in the model. It should be a vector of strings specifying the names of the observations.

```
observations <- c("obs1" , "obs2")
```

- **`start`:** The system needs to start in a state and this initial state should be specified for the model. It can be a single state or a probability distribution over a set of states. This argument will take care of that. Based on the ways we can define this initial state, the options are:
  - a vector of n numbers each in $[0, 1]$, that add up to 1, where n is the number of states, or

```
start <- c(0.5 , 0.3 , 0.2) # probabilities of being in each state
```

  - character, a string with the word "uniform" (a uniform distribution over all states), or

```
start <- "uniform" # equal probabilities for all the states
```

  - numeric, an integer in 1 to n (to specify a single starting state by its corresponding number, remember that the numbers start from 0), or

```
start <- 1 # initial state is "state2"
```

  - character, a string specifying the name of a single state (to specify a single starting state by its name), or

```
start <- "state2" # initial state is "state2"
```

  - character, a vector of strings, specifying a subset of states (a uniform distribution over a subset of states) (the first element of the vector should be "-" if the following subset of states is supposed to be excluded).

```
start <- c("state1" , "state3") # uniform distribution over "state1" and "state3"
start <- c("-" , "state2") # uniform distribution over "state1" and "state3" ("state2" excluded)
```

- **transition_prob:** This argument contains the information associated with T or the transition probabilities between states. As we demonstrated earlier these probabilities are *action-start state-end state-dependent* as we see in $T(s' \mid s, a)$. It means depending on what state the system is in, what action is taken and what state the system ends in, the probability might differ. Thus one way to provide all these prbabilities is to mention each single one along with its dependents. The other ways is to provide all these probabilities in action-dependent matrices. Given that, options are:

  - a dataframe with 4 columns, where the columns specify action, start-state, end-state and the probability respectively. The frist 3 columns could be either character (the name of the action or state) or numeric with an integer starting from 0 (the number associated with the action or state e.g., 0 as the first action or 2 as the third state).

```
transition_prob <- data.frame("action" = c("action1" , "action1" , "action1" ,
                                           0 , 0 , "action1" ,
                                           0 , 0 , 0 ,
                                           "action2" , "action2" , 1 ,
                                           1 , 1 , 1 ,
                                           1 , 1 , 1),
                              "start-state" = c("state1" , "state1" , 0 ,
                                                "state2" , "state2" , "state2" ,
                                                "state3" , 2 , 2 ,
                                                "state1" , "state1" , 0 ,
                                                "state2" , "state2" , "state2" ,
                                                "state3" , 2 , 2),
                              "end-state" = c("state1" , "state2" , 2 ,
                                              "state1" , "state2" , "state3" ,
                                              "state1" , 1 , 2 ,
                                              "state1" , "state2" , 2 ,
                                              "state1" , "state2" , "state3" ,
                                              "state1" , 1 , 2),
                              "probability" = c(0.1 , 0.4 , 0.5 ,
                                                0 , 0.7 , 0.3 ,
                                                0.4 , 0.4 , 0.2 ,
                                                0 , 0.6 , 0.4 ,
                                                0.1 , 0.9 , 0 ,
                                                0.7 , 0.3 , 0))
```

  - a list of m matrices where m is the number of actions, each matrix is square of size n where n is the number of states (each matrix should have a name in the list and the name should be one of the actions). Also each matrix can be defined using "identity" or "uniform".

```
transition_prob <- list("action1" = matrix(c( 0.1 , 0.4 , 0.5 ,
                                              0 , 0.7 , 0.3 ,
                                              0.4 , 0.4 , 0.2) ,
                                           nrow = 3 , byrow = TRUE) ,
```

```
                        "action2" = matrix(c( 0 , 0.6 , 0.4 ,
                                              0.1 , 0.9 , 0 ,
                                              0.7 , 0.3 , 0 ) ,
                                           nrow = 3 , byrow = TRUE))

transition_prob <- list("action1" = "identity" ,
                        "action2" = "uniform")

transition_prob <- list("action1" = matrix(c( 0.1 , 0.4 , 0.5 ,
                                              0 , 0.7 , 0.3 ,
                                              0.4 , 0.4 , 0.2) ,
                                           nrow = 3 , byrow = TRUE) ,
                        "action2" = "uniform")
```

- **observation_prob:** This argument contains the information associated with O or the observation probabilities between states and observations. As we demonstrated earlier these probabilities are *action-end state-observation-dependent* as we see in $O(o \mid s', a)$. It means depending on what action is taken and what state the system ends in and what observation is observed, the probability might differ. Thus one way to provide all these prbabilities is to mention each single one along with its dependents. The other ways is to provide all these probabilities in action-dependent matrices. Given that, options are:
  - a dataframe with 4 columns, where the columns specify action, end-state, observation and the probability respectively. The frist 3 columns could be either character (the name of the action, state, or observation) or numeric with an integer starting from 0 (the number associated with the action, state or observation e.g., 0 as the first action or 2 as the third state) or they can be "*" to indicate the independency.

```
observation_prob <- data.frame("action" = c( "*" , "*" , "*" , "*" , "*" , "*") ,
                               "end-state" = c("state1" , "state1" , "state2" ,
                                               "state2" , "state3" , "state3") ,
                               "observation" = c("obs1" , "obs2" , "obs1" ,
                                                 "obs2" , "obs1" , "obs2"),
                               "probability" = c(0.1 , 0.9 , 0.3 , 0.7 , 0.4 , 0.6))
```

  - a list of m matrices where m is the number of actions, each matrix is of size $n \times o$ where n is the number of states and o is the number of observations (each matrix should have a name in the list and the name should be one of the actions). Also each matrix can be defined using "uniform"

```
observation_prob <- list("action1" = matrix(c(0.1 , 0.9 ,
                                              0.3 , 0.7 ,
                                              0.4 , 0.6) , nrow = 3 , byrow = TRUE) ,
                         "action2" = matrix(c(0.1 , 0.9 ,
                                              0.3 , 0.7 ,
                                              0.4 , 0.6 ) , nrow = 3 , byrow = TRUE))

observation_prob <- list("action1" = "uniform" ,
                         "action2" = "uniform")

observation_prob <- list("action1" = "uniform" ,
                         "action2" = matrix(c(0.1 , 0.9 ,
                                              0.3 , 0.7 ,
                                              0.4 , 0.6 ) , nrow = 3 , byrow = TRUE))
```

As we see above, both observation probability matrices are the same and it indicates that the observation probabilities are independent of the action taken. Thus we can use the dataframe format using asterisks

to make it more concise. Asterisks here indicates that the probabilities are independent of the actions. "*" could be used anytime where there is independency.

- **reward:** This argument corresponds to R or the reward function of the POMDP model. The reward function in its most general form is *action-start state-end state-observation-dependent. Thus one way to provide these rewards to the function is to mention wach reward with its dependents. Sometimes rewards are only dependent on actions, end state and observations thus we can use action dependent matrices to provide them. Given that the options are:
  - a dataframe with 5 columns, where the columns specify action, start-state, end-state, observation and the reward respectively. The frist 4 columns could be either character (the name of the action, state, or observation) or numeric with an integer starting from 0 (the number associated with the action, state, observation e.g., 0 as the first action or 2 as the third state) or they can be "*" to indicate the independency

```
reward <- data.frame("action" = c("action1" , "action1" , "action1" ,
                                  "action2" , "action2" , "action2") ,
                     "start-state" = c("*" , "*" , "*" , "*" , "*" , "*") ,
                     "end-state" = c("state1" , "state2" , "state3" ,
                                     "state1" , "state2" , "state3") ,
                     "observation" = c("*" , "*" , "*" , "*" , "*" , "*") ,
                     "reward" = c(10000 , 2000 , 50 , 150 , 2500 , 100))
```

  - numeric, a matrix of size $n \times o$ where n is the number of states and o is the number of observations.

- **values:** This argument indicates whether the problem is minimization or a maximization. If the values are costs then the problem is a minimization and if they are rewards then it is a maximization. It mainly depends on the nature of the problem. The argument should be a string with either "reward" or "cost". The default is reward.

```
values <- "cost"
values <- "reward"
```

- **grid_size:** As we mentione earlier one of the fast methods to solve a POMDP problem is *Finite-Grid* method. Grid-based algorithms comprise one approximate solution technique. In this approach, the value function is computed for a set of points in the belief space, and interpolation is used to determine the optimal action to take for other belief states that are encountered which are not in the set of grid points. The `grid_size` argument here is actually the number of points in the grid which is also known as the grid size. It should be a numeric, an integer and you should know that the larger the grid, the longer it takes to solve the problem.

```
grid_size <- 10
```

- **verbose:** This argument simply indicates whether or not the outputs of the 'pomdp-solve' program should be printed in the R console. It should be logical, if set to true, the function provides the output of the pomdp solver in R console. If set to False still the output of the solver will be available in the POMDP class object.

We talked about the first group of functions in the package but there is still another group of functions that as we mentioned earlier provide features of the solution provided by the main `pomdp` function. Since it would be easier and more understandable to explain about this functions given an example, we will go through them in the next section using a real world problem.

Examples

In this section we will provide real world problems and examples which can be formulated and solved using POMDPs. We model very simple and famous examples in a POMDP context and solve them using the

pomdp package. In the first example, we will also talk about the functions of the package that somehow deal with the solution provided by the `pomdp` function in an object of class POMDP.

The Tiger Example

Consider this situation in which you are standing in front of two closed doors, behind one there is a tiger and behind the other one there is a treasure. If you open the door behind which the tiger is, you will get hurt and injured by the tiger (bodily damage, negative rewrads) and if you open the door to the treasure you will be rewarded with the treasure (positive rewards). Instead of opening the doors, you also he option of listening to see what is behind the door. But listening is not free and it is not accurate either. You might hear the tiger behind the left door while it is actually behind the right one and vice versa (there is a certain chane to that).

In this situation the states of the system are, tiger behind the left door (tiger-left) and tiger behind the right door (tiger-right). The available actions that you can take are to open the left door (open-left), to open the right door (open-right) and to listen and check (listen). The rewards associated with these actions depending on what state you end in are +10 for opening the correct door (the door with treasure), -100 for opening the door behind which the tiger is and -1 for just listening. As a result of listening there are only two observations, either you hear the tiger on the right (tiger-right) or you hear it on the left (tiger-left). The problem will be reset as soon as you open one of the door, means you will again be standing in front of the closed doors and the tiger will be randomly put behind of one of the doors with equal probability.

In order to solve this problem we also need to see how the system changes from one state to the other and what are the chances of observing a certain observation given the actual state of the system. We know that listening does not change the state of the system (means everytime you listen, the tiger does not move). When we listen, given the tiger is behind the left door (we are in state tiger-left) there is a 0.85 chance that we hear the tiger behind the left door (we observe observation tiger-left) and 0.15 chance that we hear the tiger behind the right door (we observe observation tiger right). This is totally opposite given the tiger is behind the right door (we are in state tiger-right). And regardless of what state the system is in (where the tiger is) when we open the door (either left or right), the probability of observing either observations is 0.5. Also by choosing left or right actions, the system will transitiion into either states with probability of 0.5 (literally reseting the problem as we said before) [3].

Now let's formulate the problem using the given information and solve it using the `pomdp` function.

```
discount <- 0.75
values <- "reward"
states <- c("tiger-left" , "tiger-right")
actions <- c("listen" , "open-left" , "open-right")
observations <- c("tiger-left" , "tiger-right")
start <- "uniform"
grid_size <- 10
transition_prob <- list("listen" = "identity" ,
                        "open-left" = "uniform" ,
                        "open-right" = "uniform")
observation_prob <- list("listen" = matrix(c(0.85 , 0.15 ,
                                             0.15 , 0.85) , nrow = 2 , byrow = TRUE) ,
                         "open-left" = "uniform" ,
                         "open-right" = "uniform")
reward <- data.frame("action" = c("listen" , "open-left" , "open-left" ,
                                  "open-right" , "open-right") ,
                     "start-state" = c("*" , "tiger-left" , "tiger-right" ,
                                       "tiger-left" , "tiger-right") ,
                     "end-state" = c("*" , "*" , "*" , "*" , "*") ,
                     "observation" = c("*" , "*" , "*" , "*" , "*") ,
                     "reward" = c(-1 , -100 , 10 , 10 , -100))
```

```
result <- pomdp::pomdp(discount, states, actions, observations, start, transition_prob,
                observation_prob, reward, values = "reward", grid_size, verbose = FALSE)
```

Now let's see what the outcome of the `pomdp` function is.

```
result
```

```
## POMDP Object with attributes including belief, belief proportions, alpha, pg, total expected reward,
```

As you see, the output is an object of class POMDP with a couple of attributes that we are going to dig into and extract using the second set of functions we mentioned earlier. This set of functions include `alpha`, `belief`, `belief.proportion`, `initial.node`, `model`, `pg`, `plot`, `solver.output`, and `total.expected.reward`. Every one of these functions provides a single feature of the solution provided by the main `pomdp` function.

- **`result$model`:** This feature is a list of all the inputs defined by the user but transformed into POMDP-friendly structure. This feature despite having no new information can come in handy since it has a all the user inputs in a clean pomdp-friendly format and easy to read and understand.

  You can use `model()` function to easily access this feature in the form of a list of all the inputs. Here is the model of the tiger example.

```
pomdp::model(result)
```

```
## $discount
## [1] 0.75
##
## $states
## [1] "tiger-left"  "tiger-right"
##
## $actions
## [1] "listen"     "open-left"  "open-right"
##
## $observations
## [1] "tiger-left"  "tiger-right"
##
## $start
## [1] "uniform"
##
## $transition_prob
## $transition_prob$listen
## [1] "identity"
##
## $transition_prob$`open-left`
## [1] "uniform"
##
## $transition_prob$`open-right`
## [1] "uniform"
##
##
## $observation_prob
## $observation_prob$listen
##      [,1] [,2]
## [1,] 0.85 0.15
## [2,] 0.15 0.85
##
## $observation_prob$`open-left`
## [1] "uniform"
```

```
##
## $observation_prob$`open-right`
## [1] "uniform"
##
##
## $reward
##       action start.state end.state observation reward
## 1     listen           *         *           *     -1
## 2  open-left  tiger-left         *           *   -100
## 3  open-left tiger-right         *           *     10
## 4 open-right  tiger-left         *           *     10
## 5 open-right tiger-right         *           *   -100
```

Using this feature you can always keep the inputs with the outputs and also check if the inputs are correct.

Now let's move to the actual outputs of the function in terms of the solution of the given problem. The package provides many solution features and details for a given POMDP problem, each can be extracted from the POMDP object using a proper function provided in the package.

- **result$belief:** This feature is a dataframe of all the belief states (rows) used while solving the problem. There is a column at the end that indicates which line (vector) provides the best value for the given belief state. The coefficients of these lines are given in the *alpha* feature of the POMDP object. You can use `belief()` function to get this dataframe directly.

```
pomdp::belief(result)
```

```
##      tiger-left  tiger-right line
## 1  0.5000000000 0.5000000000    3
## 2  0.8500000000 0.1500000000    4
## 3  0.1500000000 0.8500000000    2
## 4  0.9697986577 0.0302013423    5
## 5  0.0302013423 0.9697986577    1
## 6  0.9945344130 0.0054655870    5
## 7  0.0054655870 0.9945344130    1
## 8  0.9990311237 0.0009688763    5
## 9  0.0009688763 0.9990311237    1
## 10 0.9998288853 0.0001711147    5
## 11 0.0001711147 0.9998288853    1
```

This dataframe would usually have as many rows as we defined earlier in `grid_size` variable.

- **result$belief_proportions:** This feature is a dataframe that includes the probabilities of being in each actual state for each node of the policy graph (rows). Use `belief.proportions()` function to get this dataframe directly.

```
pomdp::belief.proportions(result)
```

```
##   tiger-left tiger-right
## 1 0.00920173  0.99079827
## 2 0.15000000  0.85000000
## 3 0.50000000  0.50000000
## 4 0.85000000  0.15000000
## 5 0.99079827  0.00920173
```

We will further see how this dataframe correlates to the policy graph we visualize later.

- **result$alpha:** This feature is a dataframe that includes all the coefficients of the optimal hyperplanes. Use `alpha()` function to get this dataframe directly.

```
pomdp::alpha(result)
```

```
##   coeffecient 1 coeffecient 2
## 1    -98.549921     11.450079
## 2    -10.854299      6.516937
## 3      1.933439      1.933439
## 4      6.516937    -10.854299
## 5     11.450079    -98.549921
```

- **result$pg:** This feature with no doubt is the most important element of the solution. It provides you with the optimal decisions you need to make to achieve what you were trying to achieve. It is a dataframe of all the nodes (rows) and arcs in the policy graph and how they are connected as well as the optimal action associated with each node in the graph. Use `pg()` function to get this dataframe directly. We will further visualize this table as a graph.

```
pomdp::pg(result)
```

```
##   belief     action tiger-left tiger-right
## 1      1  open-left          3           3
## 2      2     listen          3           1
## 3      3     listen          4           2
## 4      4     listen          5           3
## 5      5 open-right          3           3
```

- **result$total_expected_reward:** This feature is a numeric value indicating the total expected reward of the optimal solution. Use `total.expected.reward()` function to get this value directly.

```
pomdp::total.expected.reward(result)
```

```
## [1] 1.933439
```

- **result$initial_node:** This feature is an integer number indicating the initial node of the policy graph. Use `initial.node()` function to get this node number directly. This humber shows in what row number of the policy table or in which node of the policy graph you start.

```
pomdp::initial.node(result)
```

```
## [1] 3
```

- **result$solver_output:** This feature is a character that contains the output of the *'pomdp-solve'* code including its iterations. Use `solver.output()` function to get this output directly.

```
pomdp::solver.output(result)[c(1:5 , 60:70)]
```

```
##  [1] " //****************\\\\"
##  [2] "||   pomdp-solve    ||"
##  [3] "||     v. 5.3       ||"
##  [4] " \\\\****************//"
##  [5] "      PID=5836"
##  [6] "[Initial policy has 1 vectors.]"
##  [7] "+++++++++++++++++++++++++++++++++++++++"
##  [8] "Epoch: 1...3 vectors in 0.00 secs. (0.00 total) (err=1.10e+02)"
##  [9] "Epoch: 2...5 vectors in 0.00 secs. (0.00 total) (err=7.85e+01)"
## [10] "Epoch: 3...5 vectors in 0.00 secs. (0.00 total) (err=5.89e+01)"
## [11] "Epoch: 4...5 vectors in 0.00 secs. (0.00 total) (err=4.42e+01)"
## [12] "Epoch: 5...5 vectors in 0.00 secs. (0.00 total) (err=3.27e+01)"
## [13] "Epoch: 6...5 vectors in 0.00 secs. (0.00 total) (err=2.45e+01)"
## [14] "Epoch: 7...5 vectors in 0.00 secs. (0.00 total) (err=1.84e+01)"
## [15] "Epoch: 8...5 vectors in 0.00 secs. (0.00 total) (err=1.37e+01)"
```

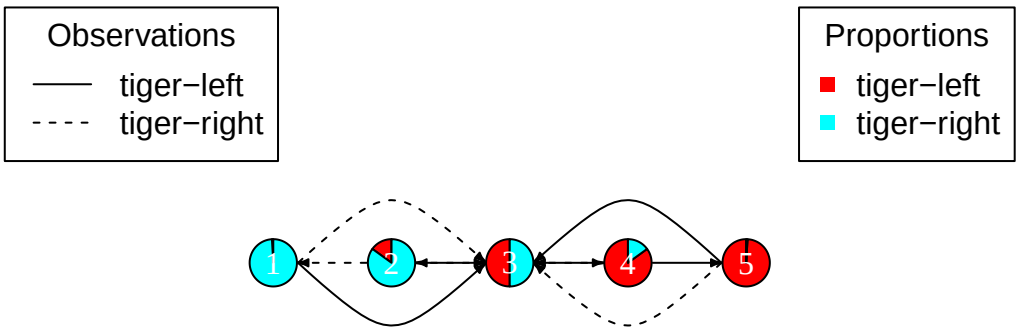
```
## [16] "Epoch: 9...5 vectors in 0.00 secs. (0.00 total) (err=1.02e+01)"
```

Above is only a part of the solver's output.

Visualization

In this section we will visualize the policy graph provided in by the `pomdp()` function through `pg()` function. THe plocy table provided by `pg()` function is easy to read and follow but has no information on the actual state the system might be in at each node. We can visualize this using the information provided by `belief.proportion()` easily by ploting an object of class POMDP.

```
plot(result)
```



The policy graph above demonstrates the way the decision maker should act given the observation and the current place he is. The policy can be read very easily. Using the `initial.node()` function we know the node number we stand at the beginning. At each decision epoch, based on what observation we make, we move using the arcs associated with each observation. and every time we arrive to a node, we take the decision associated with the node.

Note

'pomdp-solve' program uses the basic dynamic programming approach, solving one stage at a time working backwards in time. It does finite horizon problems with or without discounting. It will stop solving if the answer is within a tolerable range of the infinite horizon answer, and there are a couple of different stopping conditions (requires a discount factor less than 1.0). Alternatively you can solve a finite horizon problem for some fixed horizon length.

Author(s)

Hossein Kamalzadeh, Michael Hahsler

References

[1] For further details on how the POMDP solver utilized in this R package works check the following website: www.pomdp.org

[2] Cassandra, A. Rocco, Exact and approximate algorithms for partially observable markov decision processes, (1998). https://dl.acm.org/citation.cfm?id=926710

[3] Kaelbling, L.P., Littman, M.L., & Cassandra, A.R. (1998). Planning and Acting in Partially Observable Stochastic Domains. Artif. Intell., 101, 99-134. http://doi.org/10.1016/S0004-3702%2898%2900023-X

[4] M. L. Littman, (2009). A tutorial on partially observable Markov decision processes," J. Math. Psychol., vol. 53, no. 3, pp. 119–125.