Package 'pomdp'

August 5, 2021

Title Solver for Partially Observable Markov Decision Processes (POMDP)

Version 0.99.3

Date 2021-08-05

Description Provides the infrastructure to define and analyze the solutions of Partially Observable Markov Decision Processes (POMDP) models. The package includes pomdp-solve to solve POMDPs using a variety of exact and approximate value iteration algorithms. Smallwood and Sondik (1973) <doi:10.1287/opre.21.5.1071>.

Classification/ACM G.4, G.1.6, I.2.6

URL https://github.com/mhahsler/pomdp

BugReports https://github.com/mhahsler/pomdp/issues

Depends R (>= 3.5.0)

Imports igraph

Suggests knitr, rmarkdown, testthat, Ternary, visNetwork, sarsop

VignetteBuilder knitr

Encoding UTF-8

LazyData true

License GPL (>= 3)

Copyright pomdp-solve is Copyright (C) Anthony R. Cassandra; LASPack is Copyright (C) Tomas Skalicky; lp-solve is Copyright (C) Michel Berkelaar, Kjell Eikland, Peter Notebaert; all other code is Copyright (C) Hossein Kamalzadeh and Michael Hahsler.

RoxygenNote 7.1.1

NeedsCompilation yes

Author Michael Hahsler [aut, cph, cre], Hossein Kamalzadeh [aut, cph], Anthony R. Cassandra [ctb, cph]

Maintainer Michael Hahsler <mhahsler@lyle.smu.edu>

Repository CRAN

Date/Publication 2021-08-05 11:40:02 UTC

R topics documented:

`MDP` *Define an MDP Problem*

Description

Defines all the elements of a MDP problem and formulates them as a POMDP where all states are observable.

Usage

```
MDP(
  states,
  actions,
  transition_prob,
  reward,
  discount = 0.9,
  horizon = Inf,
  terminal_values = 0,
  start = "uniform",
  max = TRUE,
  name = NA
)
```

Arguments

states	a character vector specifying the names of the states.
actions	a character vector specifying the names of the available actions.
transition_prob	Specifies the transition probabilities between states.
reward	Specifies the rewards dependent on action, states and observations.
discount	numeric; discount rate between 0 and 1.
horizon	numeric; Number of epochs. Inf specifies an infinite horizon.
terminal_values	a vector with the terminal values for each state.
start	Specifies in which state the MDP starts.
max	logical; is this a maximization problem (maximize reward) or a minimization (minimize cost specified in reward)?
name	a string to identify the MDP problem.

Details

The MDP is formulated as a POMDP where all states are completely observable. This is achieved by defining one observation per state with identity observation probability matrices.

More details on specifying the parameters can be found in the documentation for POMDP.

Value

The function returns an object of class POMDP which is list with an element called model containing a list with the model specification. solve_POMDP reads the object and adds a list element called solution.

Author(s)

Michael Hahsler

See Also

POMDP, solve_POMDP.

Examples

```
## Michael's Sleepy Tiger Problem is an MDP with perfect observability

Tiger_MDP <- MDP(
  name = "Michael's Sleepy Tiger Problem",
  discount = 1,

  states = c("tiger-left" , "tiger-right"),
  actions = c("open-left", "open-right", "do-nothing"),
  start = "tiger-left",
```

```
  transition_prob = list(
    "open-left" =  "uniform",
    "open-right" = "uniform",
    "do-nothing" = "identity"),

  # the rew helper expects: action, start.state, end.state, observation, value
  reward = rbind(
    R_("open-left",  "tiger-left",  v = -100),
    R_("open-left",  "tiger-right", v =   10),
    R_("open-right", "tiger-left",  v =   10),
    R_("open-right", "tiger-right", v = -100),
    R_("do-nothing",                v =    0)
  )
)

Tiger_MDP

# do 5 epochs with no discounting
s <- solve_POMDP(Tiger_MDP, method = "enum", horizon = 5)
s

# value function and policy
plot_value_function(s)
policy(s)
```

`optimal_action` *Optimal action for a belief*

Description

Determines the optimal action for a policy (solved POMDP) for a given belief at a given epoch.

Usage

```
optimal_action(model, belief, epoch = 1)
```

Arguments

`model`	a solved POMDP model.
`belief`	The belief (probability distribution over the states) as a vector or a matrix with multiple belief states as rows.
`epoch`	what epoch of the policy should be used.

Value

The name of the optimal action.

Author(s)

Michael Hahsler

See Also

POMDP

Examples

```
data("Tiger")
Tiger

sol <- solve_POMDP(model = Tiger)

# these are the states
sol$model$states

# belief that tiger is to the left
optimal_action(sol, c(1, 0))
optimal_action(sol, "tiger-left")

# belief that tiger is to the right
optimal_action(sol, c(0, 1))
optimal_action(sol, "tiger-right")

# belief is 50/50
optimal_action(sol, c(.5, .5))
optimal_action(sol, "uniform")

# the POMDP is converged, so all epoch give the same result.
optimal_action(sol, "tiger-right", epoch = 10)
```

---

plot_belief_space *Plot a 2D or 3D Projection of the Belief Space*

Description

Plots the optimal action, the node in the policy graph or the reward for a given set of belief points on a line (2D) or on a ternary plot (3D). If no points are given, points are sampled using a regular arrangement or randomly from the (projected) belief space.

Usage

```
plot_belief_space(
  model,
  projection = NULL,
  epoch = 1,
```

```
  sample = "regular",
  n = 100,
  what = c("action", "pg_node", "reward"),
  legend = TRUE,
  pch = 20,
  col = NULL,
  ...
)
```

Arguments

model	a solved POMDP.
projection	a vector with state IDs or names to project on. Allowed are projections on two or three states. NULL uses the first two or three states.
epoch	display this epoch.
sample	a matrix with belief points as rows or a character string specifying the method used for sample_belief_space.
n	number of points sampled.
what	what to plot.
legend	logical; add a legend? If the legend is covered by the plot then you need to increase the plotting region of the plotting device.
pch	plotting symbols.
col	plotting colors.
...	additional arguments are passed on to plot for 2D or TerneryPlot for 3D.

Value

Returns invisibly the sampled points.

Author(s)

Michael Hahsler

See Also

sample_belief_space

Examples

```
# two-state POMDP
data("Tiger")
sol <- solve_POMDP(Tiger)

plot_belief_space(sol)
plot_belief_space(sol, n = 10)
plot_belief_space(sol, n = 10, sample = "random")
```

```
# plot the belief points used by the grid-based solver
plot_belief_space(sol, sample = sol$solution$belief_states)

# plot different measures
plot_belief_space(sol, what = "pg_node")
plot_belief_space(sol, what = "reward")

# three-state POMDP
# Note: If the plotting region is too small then the legend might run into the plot
data("Three_doors")
sol <- solve_POMDP(Three_doors)
sol

plot_belief_space(sol)
plot_belief_space(sol, sample = "random")
plot_belief_space(sol, what = "pg_node")
plot_belief_space(sol, what = "reward")

# plot the belief points used by the grid-based solver
plot_belief_space(sol, sample = sol$solution$belief_states)

# plot the belief points obtained using simulated trajectories (we use n = 50 to save time).
plot_belief_space(sol, sample = simulate_POMDP(Three_doors, n = 50, horizon = 100,
  random_actions = TRUE, visited_beliefs = TRUE))
```

`plot_policy_graph` *Visualize a POMDP Policy Graph*

Description

The function plots the POMDP policy graph in a POMDP. It uses `plot` in **igraph** with appropriate plotting options.

Usage

```
plot_policy_graph(
  x,
  belief = TRUE,
  legend = TRUE,
  engine = c("igraph", "visNetwork"),
  col = NULL,
  ...
)
```

Arguments

`x`	object of class POMDP containing a solved POMDP problem.

belief	logical; display belief proportions as a pie chart in each node. This requires belief space sampling and may be slow.
legend	logical; display a legend for colors used belief proportions?
engine	The plotting engine to be used.
col	colors used for the states.
...	plotting options passed on to the plotting engine (see Details section).

Details

The function currently only plots converged policy graphs.

The policy graph nodes represent segments in the value function. Each segment represents one or more believe states. The pie chart in each node (if available) represent the average belief proportions of the belief states belonging to the node/segment.

The built in plotting engines are **igraph** and **visNetwork**. The additional arguments specified in ... are passed on to the engine plotting function. For **igraph** this is plot.igraph (see plot.common for available options). For **visNetwork** this is visIgraph.

Other plotting libraries can be used by creating a policy graph (as an igraph object) using policy_graph and converting it into a suitable representation for that library.

See Also

solve_POMDP, policy_graph.

From: **igraph** plot.igraph, igraph_options, plot.common.

From **visNetwork**: visIgraph.

Examples

```
data("Tiger")
sol <- solve_POMDP(model = Tiger)
sol

## policy graph
policy_graph(sol)

## visualization
plot_policy_graph(sol)

## use a different graph layout (circle and manual; needs igraph)
library("igraph")
plot_policy_graph(sol, layout = layout.circle)
plot_policy_graph(sol, layout = rbind(c(1,1), c(1,-1), c(0,0), c(-1,-1), c(-1,1)))

## hide labels and legend
plot_policy_graph(sol, edge.label = NA, vertex.label = NA, legend = FALSE)

## add a plot title
plot_policy_graph(sol, main = sol$model$name)
```

```
## custom larger vertex labels (A, B, ...)
plot_policy_graph(sol,
  vertex.label = LETTERS[1:nrow(policy(sol)[[1]])],
  vertex.label.cex = 2,
  vertex.label.color = "white")

## plotting using the graph object
## (e.g., using the graph in the layout and to change the edge curvature)
pg <- policy_graph(sol)
plot(pg,
  layout = layout_as_tree(pg, root = 3, mode = "out"),
  edge.curved = curve_multiple(pg, .2))

## changes labels
plot(pg,
  edge.label = abbreviate(E(pg)$label),
  vertex.label = sol$solution$pg$action,
  vertex.size = 10)

## plot interactive graphs using the visNetwork library
plot_policy_graph(sol, engine = "visNetwork")

## add smooth edges and a layout (note, engine can be abbreviated)
plot_policy_graph(sol, engine = "vis", layout = "layout_in_circle", smooth = TRUE)
```

`plot_value_function` *Plot the Value Function of a POMDP Solution*

Description

Plots the value function of a POMDP solution as a line plot. The solution is projected on two states (i.e., the belief for the other states is held constant at zero).

Usage

```
plot_value_function(
  model,
  projection = 1:2,
  epoch = 1,
  ylim = NULL,
  legend = TRUE,
  col = NULL,
  lwd = 1,
  lty = 1,
  ...
)
```

Arguments

model	a solved POMDP.
projection	index or name of two states for the projection.
epoch	the value function of what epoch should be plotted? Ignored for infinite-horizon solutions.
ylim	the y limits of the plot.
legend	logical; add a legend?
col	potting colors.
lwd	line width.
lty	line type.
...	additional arguments are passed on to line.

Author(s)

Michael Hahsler

Examples

```
data("Tiger")
sol <- solve_POMDP(model = Tiger)
sol

plot_value_function(sol, ylim = c(0,20))

## finite-horizon
sol <- solve_POMDP(model = Tiger, horizon = 3, discount = 1,
  method = "enum")
sol

plot_value_function(sol, epoch =1, ylim = c(-5, 25))
plot_value_function(sol, epoch =2, ylim = c(-5, 25))
plot_value_function(sol, epoch =3, ylim = c(-5, 25))
```

---

policy *Extract the Policy from a POMDP/MDP*

Description

Extracts the policy from a solved POMDP/MDP.

Usage

```
policy(x)
```

Arguments

x	A solved POMDP object.

Details

A list (one entry per epoch) with the optimal policy. For converged, infinite-horizon problems solutions, a list with only the converged solution is produced. The policy is a data frame consisting o:

Part 1: The value function with one column per state. (For MDPs this is just one column with the state).

Part 2: One column with the optimal action.

Value

A list with the policy for each epoch.

Author(s)

Michael Hahsler

Examples

```
data("Tiger")
sol <- solve_POMDP(model = Tiger)
sol

# policy with value function, optimal action and transitions for observations.
policy(sol)
```

---

policy_graph *Extract the Policy Graph (as an igraph Object)*

Description

Convert the policy graph in a POMDP solution object into an igraph object.

Usage

```
policy_graph(x, belief = TRUE, col = NULL)
```

Arguments

x	A POMDP object.
belief	logical; add belief proportions as a pie chart in each node of the graph? If belief points are provided by the solver, then these are used. If a number is specified, then a random sample of that size is used instead to calculate belief proportions.
col	colors used for the states in the belief proportions.

Value

An object of class igraph containing a directed graph.

Author(s)

Hossein Kamalzadeh, Michael Hahsler

See Also

solve_POMDP

Examples

```
data("Tiger")
sol <- solve_POMDP(model = Tiger)
sol

pg <- policy_graph(sol)

plot(pg)
```

POMDP *Define a POMDP Problem*

Description

Defines all the elements of a POMDP problem including the discount rate, the set of states, the set of actions, the set of observations, the transition probabilities, the observation probabilities, and rewards.

Usage

```
POMDP(
  states,
  actions,
  observations,
  transition_prob,
  observation_prob,
  reward,
  discount = 0.9,
  horizon = Inf,
  terminal_values = 0,
  start = "uniform",
  max = TRUE,
  name = NA
)
```

```
O_(action = "*", end.state = "*", observation = "*", probability)

T_(action = "*", start.state = "*", end.state = "*", probability)

R_(action = "*", start.state = "*", end.state = "*", observation = "*", value)
```

Arguments

`states`	a character vector specifying the names of the states.
`actions`	a character vector specifying the names of the available actions.
`observations`	a character vector specifying the names of the observations.
`transition_prob`	Specifies action-dependent transition probabilities between states. See Details section.
`observation_prob`	Specifies the probability that an action/state combination produces an observation. See Details section.
`reward`	Specifies the rewards structure dependent on action, states and observations. See Details section.
`discount`	numeric; discount factor between 0 and 1.
`horizon`	numeric; Number of epochs. `Inf` specifies an infinite horizon.
`terminal_values`	a vector with the terminal values for each state or a matrix specifying the terminal rewards via a terminal value function (e.g., the alpha component produced by solve_POMDP). A single 0 specifies that all terminal values are zero.
`start`	Specifies the initial probabilities for each state (i.e., the initial belief), typically as a vector or the string `"uniform"` (default). This belief is used to calculate the total expected cumulative reward. It is also used by some solvers. See Details section for more information.
`max`	logical; is this a maximization problem (maximize reward) or a minimization (minimize cost specified in `reward`)?
`name`	a string to identify the POMDP problem.
`action, start.state, end.state, observation, probability, value`	Values used in the helper functions `O_()`, `R_()`, and `T_()` to create an entry for `observation_prob`, `reward`, or `transistion_prob` above, respectively. The default value `"*"` matches any action/state/observation.

Details

POMDP problems can be solved using `solve_POMDP`. More details about the available specifications can be found in [1].

In the following we use the following notation. The POMDP is a 7-duple $(S, A, T, R, \Omega, O, \gamma)$. S is the set of states; A is the set of actions; T are the conditional transition probabilities between states; R is the reward function; Ω is the set of observations; O are the conditional observation

probabilities; and γ is the discount factor. We will use lower case letters to represent a member of a set, e.g., s is a specific state. To refer to the size of a set we will use cardinality, e.g., the number of actions is $|A|$.

Specification of transition probabilities

Transition probability to transition to state s' from s given action a is $T(s'|s, a)$. The transition probabilities can be specified in the following ways:

- A data frame with 4 columns, where the columns specify action a, start.state s, end.state s' and the transition probability $T(s'|s, a)$, respectively. The first 3 columns can be either character (the name of the action or state) or integer indices. You can use `rbind()` with helper function `T_()` to create this data frame.
- A named list of $|A|$ matrices. Each matrix is square of size $|S| \times |S|$. Instead of a matrix, also the strings `"identity"` or `"uniform"` can be specified.

Specification of observation probabilities

The POMDP specifies the probability for each observation o given an action a and that the system transitioned to a specific state s', $O(o|s', a)$. These probabilities can be specified in the following ways:

- A data frame with 4 columns, where the columns specify the action a, the end.state s', the observation o and the probability $O(o|s', a)$, respectively. The first 3 columns could be either character (the name of the action, state, or observation), integer indices, or they can be `"*"` to indicate that the observation probability applies to all actions or states. You can use `rbind()` with helper function `O_()` to create this data frame.
- A named list of $|A|$ matrices. Each matrix is of size $|S| \times |\Omega|$. The name of each matrix is the action it applies to. Instead of a matrix, also the string `"uniform"` can be specified.

Specification of the reward function

The reward function $R(s, s', o, a)$ can be specified in the following ways:

- a data frame with 5 columns, where the columns specify action a, start.state s, end.state s', observation o and the associated reward $R(s, s', a)$, respectively. The first 4 columns could be either character (names of the action, states, or observation), integer indices, or they can be `"*"` to indicate that the reward applies to all transitions. Use `rbind()` with helper function `R_()` to create this data frame.
- a named list of $|A|$ lists. Each list contains $|S|$ named matrices representing the start states s. Each matrix is of size $|S| \times |\Omega|$, representing the end states s' and observations.

Start Belief

This belief is used to calculate the total expected cumulative reward printed with the solved model. The function `reward` can be used to calculate rewards for any belief.

Some methods use this belief to decide which belief states to explore (e.g., the finite grid method). The default initial belief is a uniform distribution over all states. No initial belief state can be used by setting `start = NULL`.

Options to specify the start belief state are:

- a probability distribution over the states. That is, a vector of $|S|$ probabilities, that add up to 1.

- the string "uniform" for a uniform distribution over all states.
- an integer in the range 1 to n to specify the index of a single starting state.
- a string specifying the name of a single starting state.

Time-dependent POMDPs

Time dependence of transition probabilities, observation probabilities and reward structure can be modeled by considering a set of episodes representing epoch with the same settings. The length of each episode is specified as a vector for horizon, where the length is the number of episodes and each value is the length of the episode in epochs. Transition probabilities, observation probabilities and/or reward structure can contain a list with the values for each episode. See solve_POMDP for more details and an example.

Value

The function returns an object of class POMDP which is list with an element called model containing a list with the model specification. solve_POMDP reads the object and adds a list element called solution.

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: http://www.pomdp.org

See Also

solve_POMDP

Examples

```
## The Tiger Problem

Tiger <- POMDP(
  name = "Tiger Problem",

  discount = 0.75,

  states = c("tiger-left" , "tiger-right"),
  actions = c("listen", "open-left", "open-right"),
  observations = c("tiger-left", "tiger-right"),

  start = "uniform",

  transition_prob = list(
    "listen" =     "identity",
    "open-left" =  "uniform",
    "open-right" = "uniform"),
```

```
  observation_prob = list(
    "listen" = rbind(c(0.85, 0.15),
                     c(0.15, 0.85)),
    "open-left" =  "uniform",
    "open-right" = "uniform"),

  # the reward helper expects: action, start.state, end.state, observation, value
  reward = rbind(
    R_("listen",                    v =   -1),
    R_("open-left",  "tiger-left",  v = -100),
    R_("open-left",  "tiger-right", v =   10),
    R_("open-right", "tiger-left",  v =   10),
    R_("open-right", "tiger-right", v = -100)
  )
)

Tiger

Tiger$model
```

---

reward *Calculate the Reward for a POMDP Solution*

Description

This function calculates the expected total reward for a POMDP solution given a starting belief state.

Usage

```
reward(x, belief = NULL, epoch = 1)
```

Arguments

x	a POMDP solution (object of class POMDP).
belief	specification of the current belief state (see argument start in POMDP for details). By default the belief state defined in the model as start is used.
epoch	return reward for this epoch. Default is the first epoch.

Details

The value is calculated using the value function stored in the POMDP solution.

Value

A list with the components

reward	the total expected reward given a belief and epoch.
belief_state	the belief state specified in belief.
pg_node	the policy node that represents the belief state.
action	the optimal action.

Author(s)

Michael Hahsler

See Also

POMDP, solve_POMDP

Examples

```
data("Tiger")
sol <- solve_POMDP(model = Tiger)

# if no start is specified, a uniform belief is used.
reward(sol)

# we have additional information that makes us believe that the tiger
# is more likely to the left.
reward(sol, belief = c(0.85, 0.15))

# we start with strong evidence that the tiger is to the left.
reward(sol, belief = "tiger-left")

# Note that in this case, the total discounted expected reward is greater
# than 10 since the tiger problem resets and another game staring with
# a uniform belief is played which produces additional reward.
```

---

round_stochastic *Round a stochastic vector or a row-stochastic matrix*

Description

Rounds a vector such that the sum of 1 is preserved. Rounds a matrix such that the rows still sum up to 1.

Usage

```
round_stochastic(x, digits = 3)
```

Arguments

x	a stochastic vector or a row-stochastic matrix.
digits	number of digits for rounding.

Details

Rounds and adjusts one entry such that the rounding error is the smallest.

Value

The rounded vector or matrix.

See Also

round

Examples

```
x <- c(0.25, 0.25, 0.5)
round_stochastic(x, 2)
round_stochastic(x, 1)
round_stochastic(x, 0)
```

---

sample_belief_space *Sample from the Belief Space*

Description

Sample randomly (uniform) or regularly spaced points from the projected belief space.

Usage

```
sample_belief_space(model, projection = NULL, n = 1000, method = "random")
```

Arguments

model	a unsolved or solved POMDP.
projection	Sample in a projected belief space. All states not included in the projection are held at a belief of 0. NULL means no projection.
n	size of the sample.
method	character string specifying the sampling strategy. Available are "random", "regular", and "vertices".

Details

Method random samples uniformly sample from the projected belief space using the method described by Luc Devroye. Method regular samples points using a regularly spaced grid. This method is only available for projections on 2 or 3 states. Method vertices only samples from the vertices of the belief space.

Value

Returns a matrix. Each row is a sample from the belief space.

Author(s)

Michael Hahsler

References

Luc Devroye, Non-Uniform Random Variate Generation, Springer Verlag, 1986.

Examples

```
data("Tiger")

sample_belief_space(Tiger, n = 5)
sample_belief_space(Tiger, n = 5, method = "regular")

# sample and calculate the reward for a solve POMDP
sol <- solve_POMDP(Tiger)
reward(sol, belief = sample_belief_space(sol, n = 5, method = "regular"))
```

---

simulate_POMDP *Simulate Trajectories in a POMDP*

Description

Simulate several trajectories through a POMDP. The start state for each trajectory is randomly chosen using the specified belief. For solved POMDPs the optimal actions will be chosen, for unsolved POMDPs random actions will be used.

Usage

```
simulate_POMDP(
  model,
  n = 100,
  belief = NULL,
  horizon = NULL,
  visited_beliefs = FALSE,
```

```
  random_actions = FALSE,
  digits = 7,
  verbose = FALSE
)
```

Arguments

model	a POMDP model.
n	number of trajectories.
belief	probability distribution over the states for choosing the starting states for the trajectories. Defaults to the start belief state specified in the model or "uniform".
horizon	number of epochs for the simulation. If NULL then the horizon for the model is used.
visited_beliefs	logical; Should all belief points visited on the trajectories be returned? If FALSE then only the belief at the final epoch is returned.
random_actions	logical; should randomized actions be used instead of the policy of the solved POMDP? Randomized actions can be used for unsolved POMDPs.
digits	round belief points.
verbose	report used parameters.

Value

A matrix with belief points as rows. Attributes containing action counts, and rewards may be available.

Author(s)

Michael Hahsler

See Also

POMDP()

Examples

```
data(Tiger)

# solve the POMDP for 5 epochs and no discounting
sol <- solve_POMDP(Tiger, horizon = 5, discount = 1, method = "enum")
sol
policy(sol)

## Example 1: simulate 10 trajectories, only the final belief state is returned
sim <- simulate_POMDP(sol, n = 100, verbose = TRUE)
head(sim)

# plot the final belief state, look at the average reward and how often different actions were used.
```

```
plot_belief_space(sol, sample = sim)

# additional data is available as attributes
names(attributes(sim))
attr(sim, "avg_reward")
colMeans(attr(sim, "action"))


## Example 2: look at all belief states in the trajectory starting with an initial start belief.
sim <- simulate_POMDP(sol, n = 100, belief = c(.5, .5), visited_beliefs = TRUE)

# plot with added density
plot_belief_space(sol, sample = sim, ylim = c(0,3))
lines(density(sim[,1], bw = .05)); axis(2); title(ylab = "Density")


## Example 3: simulate trajectories for an unsolved POMDP using randomized actions
sim <- simulate_POMDP(Tiger, n = 100, horizon = 5, random_actions = TRUE, visited_beliefs = TRUE)
plot_belief_space(sol, sample = sim, ylim = c(0,6))
lines(density(sim[,1], bw = .05)); axis(2); title(ylab = "Density")
```

`solve_POMDP_parameter` *Solve a POMDP Problem using pomdp-solver*

Description

This function utilizes the C implementation of 'pomdp-solve' by Cassandra (2015) to solve problems that are formulated as partially observable Markov decision processes (POMDPs). The result is an optimal or approximately optimal policy.

Usage

```
solve_POMDP_parameter()

solve_POMDP(
  model,
  horizon = NULL,
  discount = NULL,
  terminal_values = NULL,
  method = "grid",
  digits = 7,
  parameter = NULL,
  verbose = FALSE
)
```

Arguments

model	a POMDP problem specification created with POMDP. Alternatively, a POMDP file or the URL for a POMDP file can be specified.
horizon	an integer with the number of epochs for problems with a finite planning horizon. If set to Inf, the algorithm continues running iterations till it converges to the infinite horizon solution. If NULL, then the horizon specified in model will be used. For time-dependent POMDPs a vector of horizons can be specified (see Details section).
discount	discount factor in range [0, 1]. If NULL, then the discount factor specified in model will be used.
terminal_values	a vector with the terminal values for each state or a matrix specifying the terminal rewards via a terminal value function (e.g., the alpha component produced by solve_POMDP). If NULL, then the terminal values specified in model will be used.
method	string; one of the following solution methods: "grid", "enum", "twopass", "witness", or "incprune". The default is "grid" implementing the finite grid method.
digits	precision used when writing POMDP files (see write_POMDP).
parameter	a list with parameters passed on to the pomdp-solve program.
verbose	logical, if set to TRUE, the function provides the output of the pomdp solver in the R console.

Details

solve_POMDP_parameter() displays available solver parameter options.

Horizon: Infinite-horizon POMDPs (horizon = Inf) converge to a single policy graph. Finite-horizon POMDPs result in a policy tree of a depth equal to the smaller of the horizon or the number of epochs to convergence. The policy (and the associated value function) are stored in a list by epoch. The policy for the first epoch is stored as the first element.

Policy: Each policy is a data frame where each row representing a policy graph node with an associated optimal action and a list of node IDs to go to depending on the observation (specified as the column names). For the finite-horizon case, the observation specific node IDs refer to nodes in the next epoch creating a policy tree. Impossible observations have a NA as the next state.

Value function: The value function is stored as a matrix. Each row is associated with a node (row) in the policy graph and represents the coefficients (alpha vector) of a hyperplane. An alpha vector contains one value per state and is the value for the belief state that has a probability of 1 for that state and 0s for all others.

Precision: The POMDP solver uses various epsilon values to control precision for comparing alpha vectors to check for convergence, and solving LPs. Overall precision can be changed using parameter = list(epsilon = 1e-3).

Methods: Several algorithms for dynamic-programming updates are available:

- Enumeration (Sondik 1971).
- Two pass (Sondik 1971).

- Witness (Littman, Cassandra, Kaelbling, 1996).
- Incremental pruning (Zhang and Liu, 1996, Cassandra et al 1997).
- Grid implements a variation of point-based value iteration to solve larger POMDPs (PBVI; see Pineau 2003) without dynamic belief set expansion.

Details can be found in (Cassandra, 2015).

Note on method grid: The grid method implements a version of Point Based Value Iteration (PBVI). The used belief points are by default created using points that are reachable from the initial belief (`start`) by following all combinations of actions and observations. The size of the grid can be set via `parameter = list(fg_points = 100)`. Alternatively, different strategies can be chosen using the parameter `fg_type`. In this implementation, the user can also specify manually a grid of belief states by providing a matrix with belief states as produced by `sample_belief_space` as the parameter `grid`.

To guarantee convergence in point-based (finite grid) value iteration, the initial value function must be a lower bound on the optimal value function. If all rewards are strictly non-negative, an initial value function with an all zero vector can be used and results will be similar to other methods. However, if there are negative rewards, lower bounds can be guaranteed by setting a single vector with the values $min(reward)/(1 - discount)$. The value function is guaranteed to converge to the true value function, but finite-horizon value functions will not be as expected. `solve_POMDP` produces a warning in this case.

Time-dependent POMDPs: Time dependence of transition probabilities, observation probabilities and reward structure can be modeled by considering a set of episodes representing epoch with the same settings. In the scared tiger example (see Examples section), the tiger has the normal behavior for the first three epochs (episode 1) and then becomes scared with different transition probabilities for the next three epochs (episode 2). The episodes can be solved in reverse order where the value function is used as the terminal values of the preceding episode. This can be done by specifying a vector of horizons (one horizon for each episode) and then lists with transition matrices, observation matrices, and rewards. If the horizon vector has names, then the lists also need to be named, otherwise they have to be in the same order (the numeric index is used). Only the time-varying matrices need to be specified. An example can be found in Example 4 in the Examples section. The procedure can also be done by calling the solver multiple times (see Example 5).

Note: The parser for POMDP files is experimental. Please report problems here: https://github.com/mhahsler/pomdp/issues.

Value

The solver returns an object of class POMDP which is a list with the model specifications (`model`), the solution (`solution`), and the solver output (`solver_output`).

Author(s)

Hossein Kamalzadeh, Michael Hahsler

References

Cassandra, A. (2015). pomdp-solve: POMDP Solver Software, http://www.pomdp.org.

Sondik, E. (1971). The Optimal Control of Partially Observable Markov Processes. Ph.D. Dissertation, Stanford University.

Cassandra, A., Littman M.L., Zhang L. (1997). Incremental Pruning: A Simple, Fast, Exact Algorithm for Partially Observable Markov Decision Processes. UAI'97: Proceedings of the Thirteenth conference on Uncertainty in artificial intelligence, August 1997, pp. 54-61.

Monahan, G. E. (1982). A survey of partially observable Markov decision processes: Theory, models, and algorithms. Management Science 28(1):1-16.

Littman, M. L.; Cassandra, A. R.; and Kaelbling, L. P. (1996). Efficient dynamic-programming updates in partially observable Markov decision processes. Technical Report CS-95-19, Brown University, Providence, RI.

Zhang, N. L., and Liu, W. (1996). Planning in stochastic domains: Problem characteristics and approximation. Technical Report HKUST-CS96-31, Department of Computer Science, Hong Kong University of Science and Technology.

Pineau J., Geoffrey J Gordon G.J., Thrun S.B. (2003). Point-based value iteration: an anytime algorithm for POMDPs. IJCAI'03: Proceedings of the 18th international joint conference on Artificial Intelligence. Pages 1025-1030.

Examples

```
################################################################
# Example 1: Solving the simple infinite-horizon Tiger problem
data("Tiger")
Tiger

sol <- solve_POMDP(model = Tiger)
sol

# look at the model
sol$model

# look at solver output
sol$solver_output

# look at the solution
sol$solution

# policy (value function (alpha vectors), optimal action and observation dependent transitions)
policy(sol)

# plot the policy graph of the infinite-horizon POMDP
plot_policy_graph(sol)

# value function
plot_value_function(sol, ylim = c(0,20))

# display available solver options which can be passed on to the solver as parameters.
solve_POMDP_parameter()

################################################################
# Example 2: Solve a problem specified as a POMDP file
#            using a grid of size 10
```

```
sol <- solve_POMDP("http://www.pomdp.org/examples/cheese.95.POMDP",
  method = "grid", parameter = list(fg_points = 10))
sol

policy(sol)

# Example 3: Solving a finite-horizon POMDP using the incremental
#            pruning method (without discounting)
sol <- solve_POMDP(model = Tiger,
  horizon = 3, discount = 1, method = "incprune")
sol

# look at the policy tree
policy(sol)
# note: it does not make sense to open the door in epochs 1 or 2 if you only have 3 epochs.

reward(sol) # listen twice and then open the door or listen 3 times
reward(sol, belief = c(1,0)) # listen twice (-2) and then open-left (10)
reward(sol, belief = c(1,0), epoch = 3) # just open the right door (10)
reward(sol, belief = c(.95,.05), epoch = 3) # just open the right door (95% chance)

################################################################
# Example 3: Using terminal values
#
# Specify 1000 if the tiger is right after 3 (horizon) epochs
sol <- solve_POMDP(model = Tiger,
  horizon = 3, discount = 1,  method = "incprune",
  terminal_values = c(0, 1000))
sol

policy(sol)
# Note: the optimal strategy is never to open the left door, because we think the
# tiger is there then we better wait to get 1000 as the terminal value. If we think
# the Tiger is to the left then open the right door and have a 50/50 chance that the
# Tiger will go to the right door.

################################################################
# Example 4: Model time-dependent transition probabilities

# The tiger reacts normally for 3 epochs (goes randomly two one
# of the two doors when a door was opened). After 3 epochs he gets
# scared and when a door is opened then he always goes to the other door.

# specify the horizon for each of the two differnt episodes
Tiger_time_dependent <- Tiger
Tiger_time_dependent$model$name <- "Scared Tiger Problem"
Tiger_time_dependent$model$horizon <- c(normal_tiger = 3, scared_tiger = 3)
Tiger_time_dependent$model$transition_prob <- list(
  normal_tiger = list(
    "listen" = "identity",
    "open-left" = "uniform",
    "open-right" = "uniform"),
  scared_tiger = list(
```

```
    "listen" = "identity",
    "open-left" = rbind(c(0, 1), c(0, 1)),
    "open-right" = rbind(c(1, 0), c(1, 0))
  )
)

Tiger_time_dependent

sol <- solve_POMDP(model = Tiger_time_dependent, discount = 1,  method = "incprune")
sol

policy(sol)

################################################################
# Example 5: Alternative method to solve time-dependent POMDPs

# 1) create the scared tiger model
Tiger_scared <- Tiger
Tiger_scared$model$transition_prob <- list(
    "listen" = "identity",
    "open-left" = rbind(c(0, 1), c(0, 1)),
    "open-right" = rbind(c(1, 0), c(1, 0))
  )

# 2) Solve in reverse order. Scared tiger without terminal values first.
sol_scared <- solve_POMDP(model = Tiger_scared,
  horizon = 3, discount = 1,  method = "incprune")
sol_scared
policy(sol_scared)

# 3) Solve the regular tiger with the value function of the scared tiger as terminal values
sol <- solve_POMDP(model = Tiger,
  horizon = 3, discount = 1, method = "incprune",
  terminal_values = sol_scared$solution$alpha[[1]])
sol
policy(sol)
# note: it is optimal to mostly listen till the Tiger gets in the scared mood. Only if we are
# extremely sure in the first epoch, then opening a door is optimal.

################################################################
# Example 6: PBVI with a custom grid

# Create a search grid by sampling from the belief space in
#   10 regular intervals
custom_grid <- sample_belief_space(Tiger, n = 10, method = "regular")
custom_grid

# Visualize the search grid
plot_belief_space(sol, sample = custom_grid)

# Solve the POMDP using the grid for approximation
sol <- solve_POMDP(Tiger, method = "grid", parameter = list(grid = custom_grid))
sol
```

`solve_SARSOP` *Solve a POMDP Problem using SARSOP*

Description

This function uses the C++ implementation of the SARSOP algorithm by Kurniawati, Hsu and Lee (2008) interfaced in package **sarsop** to solve infinite horizon problems that are formulated as partially observable Markov decision processes (POMDPs). The result is an optimal or approximately optimal policy.

Usage

```
solve_SARSOP(
  model,
  horizon = Inf,
  discount = NULL,
  terminal_values = NULL,
  method = "sarsop",
  digits = 7,
  parameter = NULL,
  verbose = FALSE
)
```

Arguments

`model`	a POMDP problem specification created with `POMDP`. Alternatively, a POMDP file or the URL for a POMDP file can be specified.
`horizon`	need to be `Inf`.
`discount`	discount factor in range [0, 1]. If `NULL`, then the discount factor specified in `model` will be used.
`terminal_values`	needs to be `NULL`. SARSOP does not use terminal values.
`method`	string; there is only one method available called `"sarsop"`.
`digits`	precision used when writing POMDP files (see `write_POMDP`).
`parameter`	a list with parameters passed on to the function `sarsop` in package **sarsop**.
`verbose`	logical, if set to `TRUE`, the function provides the output of the solver in the R console.

Value

The solver returns an object of class POMDP which is a list with the model specifications (`model`), the solution (`solution`), and the solver output (`solver_output`).

Author(s)

Michael Hahsler

References

Carl Boettiger, Jeroen Ooms and Milad Memarzadeh (2020). sarsop: Approximate POMDP Planning Software. R package version 0.6.6. https://CRAN.R-project.org/package=sarsop

H. Kurniawati, D. Hsu, and W.S. Lee (2008). SARSOP: Efficient point-based POMDP planning by approximating optimally reachable belief spaces. In Proc. Robotics: Science and Systems.

Examples

```
## Not run:
# Solving the simple infinite-horizon Tiger problem with SARSOP
# You need to install package "sarsop"
data("Tiger")
Tiger

sol <- solve_SARSOP(model = Tiger)
sol

# look at solver output
sol$solver_output

# policy (value function (alpha vectors), optimal action and observation dependent transitions)
policy(sol)

# value function
plot_value_function(sol, ylim = c(0,20))

# plot the policy graph
plot_policy_graph(sol)

# reward of the optimal policy
reward(sol)

# Solve a problem specified as a POMDP file
sol <- solve_SARSOP("http://www.pomdp.org/examples/cheese.95.POMDP")
sol

## End(Not run)
```

Tiger *Tiger Problem POMDP Specification*

Description

The model for the Tiger Problem [1].

Format

An object of class POMDP.

Details

The original Tiger problem was published in [1]. A tiger is put with equal probability behind one of two doors represented by the states tiger-left and tiger-right, while treasure is put behind the other door. You are standing in front of the two closed doors and need to decide which one to open. If you open the door with the tiger, you will get hurt by the tiger (a negative reward of -100), but if you open the door with the treasure, you receive a positive reward of 10. Instead of opening a door right away, you also have the option to wait and listen for tiger noises producing an observation (tiger-left or tiger-right). But listening is neither free (reward of -1) nor entirely accurate. You might hear the tiger behind the left door while it is actually behind the right door and vice versa. Once you open a door (actions open-left or open-right), you receive the appropriate reward and the problem is reset (i.e., the tiger is randomly assigned to a door and the belief is set to 50/50).

The three doors problem is an extension of the Tiger problem where the tiger is behind one of three doors represented by three states (tiger-left, tiger-center, and tiger-right) and treasure is behind the other two doors. There are also three observations for listening.

References

[1] Anthony R. Cassandra, Leslie P Kaelbling, and Michael L. Littman (1994). Acting Optimally in Partially Observable Stochastic Domains. In Proceedings of the Twelfth National Conference on Artificial Intelligence, pp. 1023-1028.

Examples

```
data("Tiger")
Tiger

Tiger$model

data("Three_doors")
Three_doors

Three_doors$model
```

---

transition_matrix *Extract the Transition, Observation or Reward Matrices from a POMDP*

Description

Converts the description of transition probabilities and observation probabilities in a POMDP into a list of matrices, one for each action. Rewards are converted into a list (actions) of lists (start states) of matrices.

Usage

```
transition_matrix(x, episode = 1)

observation_matrix(x, episode = 1)

reward_matrix(x, episode = 1)
```

Arguments

x	A POMDP object.
episode	Episode used for time-dependent POMDPs (see POMDP).

Value

A list or a list of lists of matrices.

Author(s)

Michael Hahsler

See Also

POMDP

Examples

```
data("Tiger")

# transition matrices for each action in the from states x states
transition_matrix(Tiger)

# observation matrices for each action in the from states x observations
observation_matrix(Tiger)

# reward matrices for each matrix and (start) state in
# the form (end) state x observation
reward_matrix(Tiger)

# Visualize transition matrix for action 'open-left'
library("igraph")
g <- graph_from_adjacency_matrix(transition_matrix(Tiger)$"open-left", weighted = TRUE)
edge_attr(g, "label") <- edge_attr(g, "weight")

igraph.options("edge.curved" = TRUE)
plot(g, layout = layout_on_grid, main = "Transitions for action 'open=left'")
```

update_belief *Belief Update*

Description

Update the belief given a taken action and observation.

Usage

```
update_belief(
  model,
  belief = NULL,
  action = NULL,
  observation = NULL,
  episode = 1,
  digits = 7,
  drop = TRUE
)
```

Arguments

model	a POMDP model.
belief	the current belief state. Defaults to the start belief state specified in the model or "uniform".
action	the taken action.
observation	the received observation.
episode	Use transition and observation matrices for the given episode for time-dependent POMDPs (see POMDP).
digits	round decimals.
drop	logical; drop the result to a vector if only a single belief state is returned.

Details

Update the belief state b (belief) with an action a and observation o. The new belief state b' is:

$$b'(s') = \eta O(o|s', a) \sum_{s \in S} T(s'|s, a) b(s)$$

where $\eta = 1/\sum_{s' \in S}[O(o|s', a) \sum_{s \in S} T(s'|s, a) b(s)]$ normalizes the new belief state so the probabilities add up to one.

Author(s)

Michael Hahsler

See Also

POMDP(), simulate_POMDP()

Examples

```
data(Tiger)

update_belief(c(.5,.5), model = Tiger)
update_belief(c(.5,.5), action = "listen", observation = "tiger-left", model = Tiger)
update_belief(c(.15,.85), action = "listen", observation = "tiger-right", model = Tiger)
```

---

write_POMDP *Read and write a POMDP Model to a File in POMDP Format*

Description

Reads and write a POMDP file suitable for the pomdp-solve program. Note: read POMDP files are intended to be used in solve_POMDP() and do not support all auxiliary functions. Fields like the transition matrix, the observation matrix and the reward structure are not parsed.

Usage

```
write_POMDP(model, file, digits = 7)

read_POMDP(file)
```

Arguments

model	an object of class POMDP_model.
file	a file name.
digits	precision for writing numbers (digits after the decimal point).

Value

read_POMDP returns a POMDP object.

Author(s)

Hossein Kamalzadeh, Michael Hahsler

References

POMDP solver website: http://www.pomdp.org

See Also

POMDP

Index