# POMDP: Introduction to Partially Observable Markov Decision Processes

*Hossein Kamalzadeh, Michael Hahsler*

*2019-05-14*

The R package **pomdp** provides an interface to 'pomdp-solve', a solver (written in C) for Partially Observable Markov Decision Processes (POMDP). The package enables the user to simply define all components of a POMDP model and solve the problem using several methods. The package also contains functions to analyze and visualize the POMDP solutions (e.g., the optimal policy).

In this document we will give a very brief introduction to the concept of POMDP, describe the features of the R package, and illustrate the usage with a toy example.

## Introduction on POMDPs

A partially observable Markov decision process (POMDP) is a combination of an MDP to model system dynamics with a hidden Markov model that connects unobservant system states to observations. The agent can perform actions which affect the system (i.e., may cause the system state to change) with the goal to maximize a reward that depends on the sequence of system state and the agent's actions. However, the agent cannot directly observe the system state, but at each discrete point in time, the agent makes observations that depend on the state. The agent uses these observations to form a belief of in what state the system currently is. This belief is called a belief state and is expressed as a probability distribution over the states. The solution of the POMDP is a policy prescribing which action is optimal for each belief state.

The POMDP framework is general enough to model a variety of real-world sequential decision-making problems. 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 (Littman 2009).

A discrete-time POMDP can formally be described as a 7-tuple ( $S$ , $A$ , $T$ , $R$ , $\Omega$ , $O$ , $\lambda$), where

- $S = \{s_1, s_2, \dots, s_n\}$ is a set of states,
- $A = \{a_1, a_2, \dots, a_m\}$ is a set of actions,
- $T$ is a set of conditional transition probabilities $T(s' \mid s, a)$ for the state transition $s \to s'$.
- $R : S \times A \to \mathbb{R}$ is the reward function,
- $\Omega = \{o_1, o_2, \dots, o_k\}$ is a set of observations,
- $O$ is a set of conditional observation probabilities $O(o \mid s', a)$, and
- $\lambda \in [0, 1]$ is the discount factor.

At each time period, the environment is in some state $s \in S$. The agent chooses an action $a \in A$, which causes the environment to transition to state $s' \in S$ 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 $R(s, a)$. Then the process repeats. The goal is for the agent to choose actions at each time step that maximizes its expected future discounted reward

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

.

# Package Functionality

Solving a POMDP problem with the **pomdp** package consists of two steps:

1. Define a POMDP problem using the function `POMDP`, and

2. solve the problem using `solve_POMDP`.

## Defining a POMDP Problem

The `POMDP` function has the following arguments, each corresponds to one of the elements of a POMDP.

```
str(args(POMDP))
```

```
## function (discount = 0, states, actions, observations, start = "uniform",
##     transition_prob, observation_prob, reward, values = "reward",
##     name = NA)
```

Next we describe the arguments in detail and give examples:

- **`discount`:** Is the used discount factor $\lambda$, a real number in the range $[0, 1]$.

```
r   discount = 0.9
```

- **`states`:** Defines the set of states $S$ using a vector of strings.

```
r   states = c("state1" , "state2" , "state3")
```

- **`actions`:** Defines the set of actions $A$ using a vector of strings.

```
r   actions = c("action1" , "action2")
```

- **`observations`:** Defines the set of observations $\Omega$ using a vector of strings.

```
r   observations = c("obs1" , "obs2")
```

- **`start`:** The initial probability distribution over the system states $S$. It can be specified in several ways.
    - A vector of $n$ probabilities in $[0, 1]$, that add up to 1, where $n$ is the number of states.

    ```
    r   start = c(0.5 , 0.3 , 0.2)
    ```

    - The string ‘ “uniform” ’ for a uniform distribution over all states.

    ```
    r   start = "uniform"
    ```

    - A vector of integer indices specifying a subset as start states. The initial probability is uniform over these states. For example, only state 3 or state 1 and 3:

    ```
    r   start = 3   start = c(1, 3)
    ```

    - A vector of strings specifying a subset as start states.

    ```
    r   start <- "state3"   start <- c("state1" , "state3")
    ```

    - A vector of strings starting with `"-"` specifying which states to exclude from the uniform initial probability distribution.

    ```
    r   start = c("-" , "state2")
    ```

- **`transition_prob`:** Defines the conditional transition probabilities $T(s' \mid s, a)$. The probabilities depend on the `end-state` $s'$, the `start-state` $s$ and the `action` $a$. The set of conditional transition probabilities can be specified in several ways:

  - A data frame with 4 columns, where the columns specify `action`, `start-state`, `end-state` and the `probability`, respectively. The first 3 columns are either the names or the index of the action or states.

```
r   transition_prob = data.frame(     "action" = c(            "action1" , "action1"
, "action1",              "action1" , "action1" , "action1",              "action1"
, "action1" , "action1",             "action2" , "action2" , "action2",              "action2"
, "action2" , "action2",             "action2" , "action2" , "action2"),     "start-state"
= c(             "state1" , "state1" , "state1",             "state2" , "state2"
, "state2",             "state3" , "state3" , "state3",             "state1" ,
"state1" , "state1",             "state2" , "state2" , "state2",             "state3"
, "state3" , "state3"),       "end-state" = c(            "state1" , "state2"
, "state3",             "state1" , "state2" , "state3",             "state1" ,
"state2" , "state3",             "state1" , "state2" , "state3",             "state1"
, "state2" , "state3",             "state1" , "state2" , "state3"),       "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 named list of $m$ matrices where each matrix represents one of the $m$ actions. Each matrix is square of size $n \times n$ where $n$ is the number of states. Matrices can also be defined as `"identity"` or `"uniform"`.

“‘r 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” = 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`:** Specifies the conditional observation probabilities $O(o \mid s', a)$. The probabilities depend on the `action`, the `end-state`, and the `observation`. The set of conditional observation probabilities can be specified in several ways:

  - A data frame with 4 columns, where the columns specify `action`, `end-state`, `observation` and the `probability`, respectively. The first 3 columns could be either the name or the index of the action, state, or observation. The special character ’“*“’ can be used to indicate that the probability applies for all actions, states or observations.

```
r   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 named list of $m$ matrices, one for each 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). Matrices can also be defined as `"identity"` or `"uniform"`.

“‘r 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” = matrix(c(0.1, 0.9, 0.3, 0.7, 0.4, 0.6), nrow = 3, byrow = TRUE)) “‘

- **`reward`:** This argument corresponds to the reward function $R$. The reward function in its most general form depends on `action`, `start-state`, `end-state` and the `observation`. The reward function can be specified several ways:

  - A data frame with 5 columns, where the columns specify `action`, `start-state`, `end-state`, `observation` and the reward, respectively. The first 4 columns could be either the name or the index of the action, state, or observation. The special character ’“*“’ can be used to indicate that the same reward applies for all actions, states or observations.

```
r   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))
```

  - A named list of $m$ lists, where $m$ is the number of actions (names should be the actions). Each list contains $n$ named matrices where each matrix is of size $n \times o$, in which $n$ is the number of states and $o$ is the number of observations. Names of these matrices should be the name of states.

```
r   reward = list(     "action1" = list(        "state1" = matrix(c(1, 2, 3, 4,
5, 6) , nrow = 3 , byrow = TRUE),        "state2" = matrix(c(3, 4, 5, 2, 3, 7) ,
nrow = 3 , byrow = TRUE),        "state3" = matrix(c(6, 4, 8, 2, 9, 4) , nrow = 3 ,
byrow = TRUE)),     "action2" = list(        "state1" = matrix(c(3, 2, 4, 7, 4, 8)
, nrow = 3 , byrow = TRUE),        "state2" = matrix(c(0, 9, 8, 2, 5, 4) , nrow = 3
, byrow = TRUE),        "state3" = matrix(c(4, 3, 4, 4, 5, 6) , nrow = 3 , byrow =
TRUE)))
```

- **`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. The default is reward.

  ```
  r  values = "cost"  values = "reward"
  ```

- **`name:`** This argument can be used to name the POMDP problem defined by the user. This way the user can keep track of the POMDP problems he defines.

  ```
  r  name = "Test Problem"
  ```

## Solving a POMDP

POMDP problems are solved with the function `solve_POMDP` with the following arguments.

```r
str(args(solve_POMDP))
```

```
## function (model, horizon = NULL, method = "grid", parameter = NULL,
##     verbose = FALSE)
```

The `model` argument is a POMDP problem created using the `POMDP` function. The `horizon` argument specifies the finite time horizon (i.e, the number of time steps) considered in solving the problem. If the horizon is unspecified (i.e., `NULL`), then the algorithm continues running iterations till it converges to the infinite horizon solution (Anthony Rocco Cassandra 1998). The `method` argument specifies what algorithm the solver should use. Available methods including `"grid"`, `"enum"`, `"twopass"`, `"witness"`, and `"incprune"`. Further solver parameters can be specified as a list as `parameters`. The list of available parameters can be obtained using the function `solve_POMDP_parameter()`. Finally, `verbose` is a logical that indicates whether the solver output should be shown in the R console or not. The output of this function is an object of class POMDP.

## Helper Functions

The package offers several functions to help with managing POMDP problems and solutions.

The functions `model`, `solution`, and `solver_output` extract different elements from a POMDP object returned by `solve_POMDP()`.

The package provides a plot function to visualize the solution’s policy graph using the package **igraph**. The graph itself can be extracted from the solution using the function `policy_graph()`.

# The Tiger Problem Example

We will demonstrate how to use the package with the Tiger Problem (Anthony R. Cassandra, Kaelbling, and Littman 1994).

A tiger is put with equal probability behind one of two doors, while treasure is put behind the other one. 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 (negative reward), but if you open the door with the treasure, you receive a positive reward. Instead of opening a door right away, you also have the option to wait and listen for tiger noises. But listening is neither free nor entirely accurate. You might hear the tiger behind the left door while it is actually behind the right door and vice versa.

The states of the system are the tiger behind the left door (tiger-left) and the tiger behind the right door (tiger-right).

Available actions are: open the left door (open-left), open the right door (open-right) or to listen (listen).

Rewards associated with these actions depend on the resulting state: +10 for opening the correct door (the door with treasure), -100 for opening the door with the tiger. A reward of -1 is the cost of listening.

As a result of listening, there are two observations: either you hear the tiger on the right (tiger-right), or you hear it on the left (tiger-left).

The transition probability matrix for the action listening is identity, i.e., the position of the tiger does not change. Opening either door means that the game restarts by placing the tiger uniformly behind one of the doors.

## Specifying the Tiger Problem

The problem can be specified using function `POMDP()` as follows.

```r
library("pomdp")

TigerProblem <- 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" = 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))
  )

TigerProblem
```

```
## POMDP model: Tiger Problem
##
## $name
## [1] "Tiger Problem"
##
## $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
##
## $values
## [1] "reward"
```

## Solving the Tiger Problem

Now, we can solve the problem using the default algorithm (finite grid, a form of point-based value iteration).

```
tiger_solved <- solve_POMDP(TigerProblem)
tiger_solved
```

```
## Solved POMDP model: Tiger Problem
##      method: grid
##      belief states: 5
##      total expected reward: 1.933439
```

The output is an object of class POMDP which contains the solution.

```
solution(tiger_solved)
```

```
## POMDP solution
##
## $method
## [1] "grid"
##
## $parameter
## NULL
##
## $alpha
##   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
##
## $pg
##   belief_state     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
##
## $belief
##      tiger-left  tiger-right belief_state
## 1  5.000000e-01 5.000000e-01            3
## 2  8.500000e-01 1.500000e-01            4
## 3  1.500000e-01 8.500000e-01            2
## 4  9.697987e-01 3.020134e-02            5
## 5  3.020134e-02 9.697987e-01            1
## 6  9.945344e-01 5.465587e-03            5
## 7  5.465587e-03 9.945344e-01            1
## 8  9.990311e-01 9.688763e-04            5
## 9  9.688763e-04 9.990311e-01            1
## 10 9.998289e-01 1.711147e-04            5
## 11 1.711147e-04 9.998289e-01            1
## 12 9.999698e-01 3.020097e-05            5
## 13 3.020097e-05 9.999698e-01            1
## 14 9.999947e-01 5.329715e-06            5
## 15 5.329715e-06 9.999947e-01            1
## 16 9.999991e-01 9.405421e-07            5
## 17 9.405421e-07 9.999991e-01            1
## 18 9.999998e-01 1.659782e-07            5
## 19 1.659782e-07 9.999998e-01            1
## 20 1.000000e+00 2.929027e-08            5
## 21 2.929027e-08 1.000000e+00            1
## 22 1.000000e+00 5.168871e-09            5
## 23 5.168871e-09 1.000000e+00            1
## 24 1.000000e+00 9.121536e-10            5
## 25 9.121536e-10 1.000000e+00            1
##
## $belief_proportions
##    tiger-left tiger-right
## 1 0.003349418 0.996650582
## 2 0.150000000 0.850000000
## 3 0.500000000 0.500000000
## 4 0.850000000 0.150000000
## 5 0.996650582 0.003349418
##
## $total_expected_reward
## [1] 1.933439
##
## $initial_belief_state
## [1] 3
```

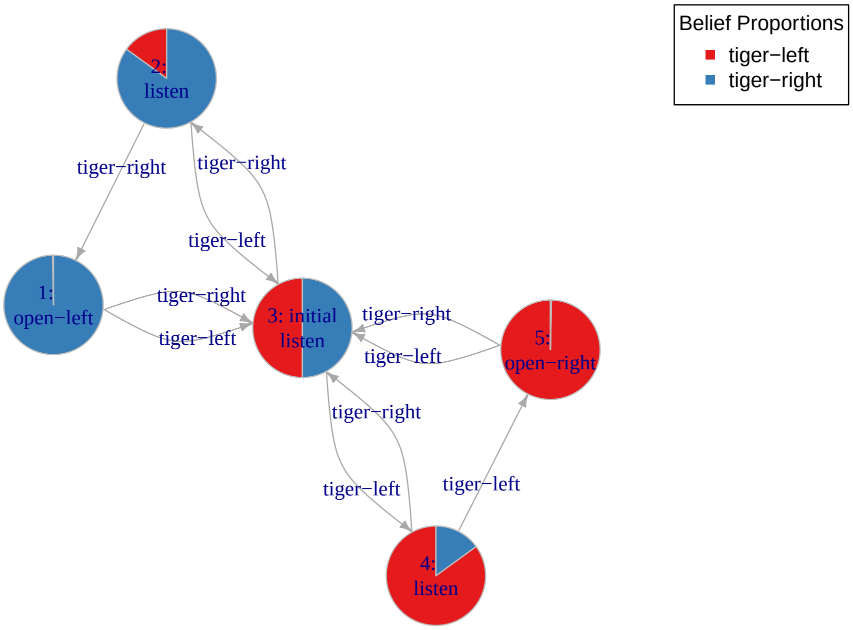
The solution contains the following elements:

- **`belief`:** A data frame of all the belief states (rows) used while solving the problem. There is a column at the end that indicates which hyperplane (specified in **alpha** below) provides the best value for the given belief state.
- **`belief_proportions`:** A data frame with the probability distribution for each belief state.
- **`alpha`:** A data frame with the coefficients of the optimal hyperplanes.
- **`pg`:** A data frame containing the optimal policy graph. Rows are belief states. Column two indicates the optimal action for the belief stare. Columns three and after represent the transitions from belief state to belief state depending on observations.
- **`total_expected_reward`:** The total expected reward of the optimal solution.
- **`initial_node`:** The index of the initial belief state in the policy graph.

### Visualization

In this section, we will visualize the policy graph provided in the solution by the `solve_POMDP` function.

```
plot(tiger_solved)
```



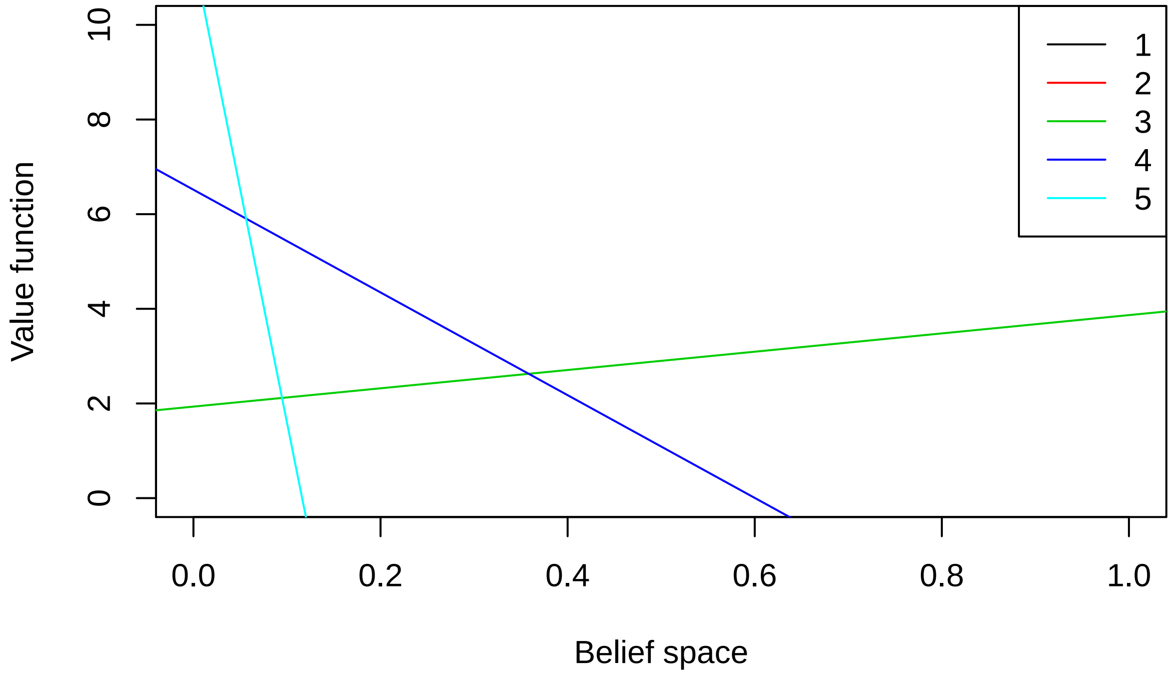
The policy graph can be easily interpreted. Without prior information, the agent starts at the belief state marked with "initial." In this case the agent beliefs that there is a 50-50 chance that the tiger is behind ether door. The optimal action is displayed inside the state and in this case is to listen. The observations are labels on the arcs. Let us assume that the observation is "tiger-left", then the agent follows the appropriate arc and ends in a belief state that has a very high probability of the tiger being left. However, the optimal action is still to listen. If the agent again hears the tiger on the left then it ends up in a belief state that has `open-right` as the optimal action. There are arcs back to the initial state which reset the problem.

Since we only have two states, we can visualize the piecewise linear convex value function as a simple plot.

```
alpha <- solution(tiger_solved)$alpha
alpha
```

```
##   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
```

```
plot(NA, xlim = c(0, 1), ylim = c(0, 10), xlab = "Belief space", ylab = "Value function")
for(i in 1:nrow(alpha)) abline(a = alpha[i,1], b= alpha[i,2], col = i)
legend("topright", legend = 1:nrow(alpha), col = 1:nrow(alpha), lwd=1)
```



## References

Cassandra, Anthony R., Leslie Pack Kaelbling, and Michael L. Littman. 1994. "Acting Optimally in Partially Observable Stochastic Domains." In *Proceedings of the Twelfth National Conference on Artificial Intelligence*. Seattle, WA.

Cassandra, Anthony Rocco. 1998. "Exact and Approximate Algorithms for Partially Observable Markov Decision Processes." PhD thesis, Providence, RI, USA: Brown University.

Littman, Michael L. 2009. "A Tutorial on Partially Observable Markov Decision Processes." *Journal of Mathematical Psychology* 53 (3): 119–25. doi:10.1016/j.jmp.2009.01.005.