PPS: User Manual

Krishnendu Chatterjee, Martin Chmelik, Raghav Gupta, and Ayush Kanodia

IST Austria (Institute of Science and Technology Austria), Klosterneuburg, Austria

In this section we describe the tool features, user manual including the input file format, and a detailed description of the examples used.

0.1 Features

The tool loads the input file with the POMDP and the parity objective. Depending on the arguments, the PPS tool can perform the following operations (see `http://pub.ist.ac.at/pps` for further details).

– Visualize the textual input POMDP using the DOT language.
– Reduce a POMDP with a parity objective, to an equivalent POMDP with a coBüchi objective (a parity objective with only 2 priorities), and visualize the POMDP.
– Given a POMDP, with a coBüchi objective, it constructs its belief-observation POMDP $\widehat{G}$, and visualizes it.
– Given the belief-observation POMDP $\widehat{G}$, if there exists an almost-sure winning strategy, it outputs a finite-state machine, where the states encode the memory elements, and transitions labeled by actions and observations stand for memory updates. As before the tool can visualize the finite-memory strategy.

Note, that it is possible to visualize the POMDP, using the DOT language, after performing any of the reductions, to get the intermediate results.

0.2 User Manual

Dependencies. The following software will be required to run the PPS tool:

– Java Runtime Environment or Java Development Kit (version 1.7 onwards). The PPS tool has been developed using Java 1.7. Download from `http://www.oracle.com/technetwork/java/javase/downloads/jre7-downloads-1880261.html`
– JGraphT. This is the graph handling library that has been used to work with the POMDPs, treated as directed multigraphs. The `JGraphT.jar` library is required to run the software. Download from `http://jgrapht.org/`
– GraphViz. This software is required to convert the files that are output by the PPS tool in the dot format for graphs, into visual graphs. Download from `http://www.graphviz.org/Download.php`. The command for converting a dot file into its corresponding graph (in the png format) `dot -Tpng source > destination`. Detailed documentation for GraphViz can be found at `http://www.graphviz.org/Documentation.php`

Running the PPS tool. To run the software, download the jar file `pps.jar` from `http://pub.ist.ac.at/pps/pps.jar`. Install all dependencies. Now issue the command `"java -jar pps.jar"` from the command line terminal. The user interface will now open. A detailed example of how to use the tool is presented later.

File Format. We now present the input file format.

- *name-of-input-POMDP* → `[<Name>]#`
- *observations* → `[<obs_1> ,<obs_2> , ....., <obs_j>] #`
- *states* → `[<state_1>, <state_2> , ....., <state_n>]#`
- *state* → `{ <state_name>; <priority_value>; <state_observation>; T (if this is a start state) or F (if this is not a start state)} #`
- *actions* → `[<action_1> ,<action_2> , ....., <action_k>] #`
- *transitions* → `[<source_state> ,<action> , { <target_state_1>; <target_state_2>; .....; <target_state_m> }]#`

The input file is treated a sequence of tokens, separated by the '#' character. As the exact probability of a transition is not important in qualitative analysis, the file format is designed in way that it stores only the support of the transition function.

Sample POMDP Input File. In this part we present an example of a POMDP in the input file format and provide detailed a description. Figure 1 shows the corresponding POMDP generated using 'GraphViz'.

```
NAME: [Demo POMDP] #
OBSERVATIONS: [o1,o2,o3] #
STATES: [{x; 1; o1; F}, {y; 2; o2; F}, {z; 1; o2; F},
{w; 2; o3; T}] #
ACTIONS: [a,b] #
TRANSITIONS: [x,a,{w}] #
TRANSITIONS: [x,b,{z;w}] #
TRANSITIONS: [y,a,{z}] #
TRANSITIONS: [y,b,{x}] #
TRANSITIONS: [z,a,{x;y}] #
TRANSITIONS: [z,b,{x;y;z}] #
TRANSITIONS: [w,a,{w}] #
TRANSITIONS: [w,b,{w;z}] #
```

We now explain the above POMDP:

- The name of the input POMDP is "`Demo POMDP`"
- The observations are "`o1`", "`o2`", "`o3`"
- The states are
  - "`x`", priority `1`, observation `o1`.
  - "`y`", priority `2`, observation `o2`.
  - "`z`", priority `1`, observation `o2`.
  - "`w`", priority `2`, observation `o3`, which is a start state.

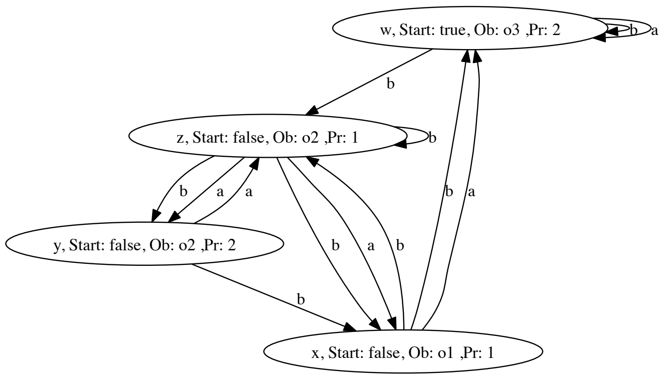

w, Start: true, Ob: o3 ,Pr: 2
z, Start: false, Ob: o2 ,Pr: 1
y, Start: false, Ob: o2 ,Pr: 2
x, Start: false, Ob: o1 ,Pr: 1
a
b

Fig. 1. The `Demo POMDP` generated using GraphViz

- The actions are "a" and "b".
- The following are the set of transitions on given actions, for every state of the POMDP
  - In state "x", on action "a", the next state is "w" with probability 1.
  - In state "x", on action "b", the next state is with positive prob. "z", or "w".
  - In state "y", on action "a", the next state is "z" with probability 1.
  - In state "y", on action "b", the next state is "x" with probability 1.
  - In state "z", on action "a", the next state is with positive prob. "x", or "y".
  - In state "z", on action "b", the next state is with positive prob. "x", or "w", "z".
  - In state "w", on action "a", the next state is "w" with probability 1.
  - In state "w", on action "b", the next state is with positive prob. "w", or "z".

Sample Tool Usage. In this part we show a sample tool walkthrough. We will use `AlternatePOMDP.txt` example to illustrate the walkthrough. This input file is also available on the website along with the PPS tool on `http://pub.ist.ac.at/pps/examples/AlternatePOMDP.txt`. Figure 2 shows the corresponding POMDP generated using the GraphViz tool.

```
NAME: [POMDP Alternate] #
OBSERVATIONS: [o1] #
STATES: [ {x; 2; o1; F}, {y; 2; o1; F}, {z; 1; o1; T}] #
ACTIONS: [a,b] #
TRANSITIONS: [x,a,{y}] #
TRANSITIONS: [x,b,{z}] #
TRANSITIONS: [y,a,{z}] #
TRANSITIONS: [y,b,{x}] #
```

```
TRANSITIONS: [z,a,{x;y;z}] #
TRANSITIONS: [z,b,{x;y;z}] #
```

There are three states in the POMDP, `x`, `y` and the initial state `z`. All the states have the same observation `o1`, therefore the Player does not receive any useful information from the play. There are two actions, `a` and `b`. In state `x`, taking action `a` (resp. `b`) brings the POMDP deterministically to state `y` (resp. `z`), and in state `y`, taking action `b` (resp. `a`) brings the POMDP deterministically to state `x` (resp. `z`). Playing any action in state `z` brings the POMPD to all three states with positive probability. States `x` and `y` have priority 2 and state `z` has priority 1. Therefore, the goal in the POMDP is to play a strategy that visits state `z` only finitely often with probability 1.

Note, that once the POMDP reaches state `x`, if the player plays the strategy that alternates between playing `a` followed by `b`, then the POMDP will always stay in states `x` and `y`, both of which have priority priority 2. Similarly, once the POMDP reaches state `y`, if the strategy alternates between playing `b` followed by `a`, then the POMDP will always stay in states `y` and `x`.

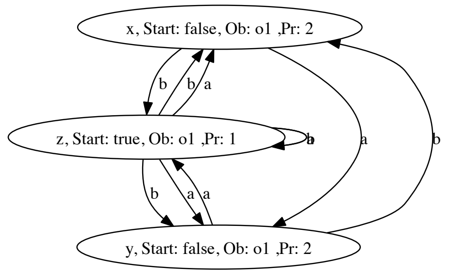
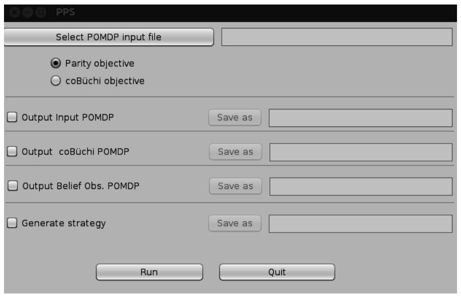

x, Start: false, Ob: o1 ,Pr: 2
z, Start: true, Ob: o1 ,Pr: 1
y, Start: false, Ob: o1 ,Pr: 2
b
b
a
a
b
b
a
a
b

Fig. 2. `POMDP Alternate` generated using GraphViz

In the following part we enumerate the required steps in order to solve whether there exists a finite-memory strategy ensuring the objective with probability 1.

1. Run the command `java -jar pps.jar`. A dialog box opens, as shown in Figure 3.
2. Select the POMDP input file `AlternatePOMDP.txt`, by clicking the button. This will open a dailog box as in Figure 4.
3. In the case of `AlternatePOMDP.txt` the objective is specified only by priorities 1 and 2, therefore we can select the radio button corresponding to coBüchi objectives to speed up the computation. In case the objective has also other priorities, select the radio button corresponding `Parity objective`.
4. To generate the dot format for the input POMDP, check the first option. Click the corresponding `Save As` button to specify the location and the name of the file to



[✓ Parity objective] coBüchi objective

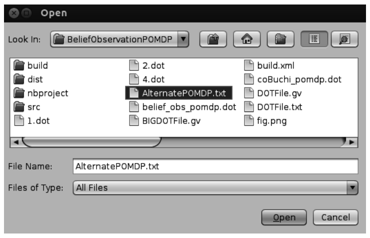
☐ Output Input POMDP

☐ Output coBüchi POMDP

☐ Output Belief Obs. POMDP

☐ Generate strategy

Fig. 3. Opening Screen



Look In: BeliefObservationPOMDP

build, dist, nbproject, src, 1.dot, 2.dot, 4.dot, AlternatePOMDP.txt, belief_obs_pomdp.dot, BIGDOTFile.gv, build.xml, coBuchi_pomdp.dot, DOTFile.gv, DOTFile.txt, fig.png

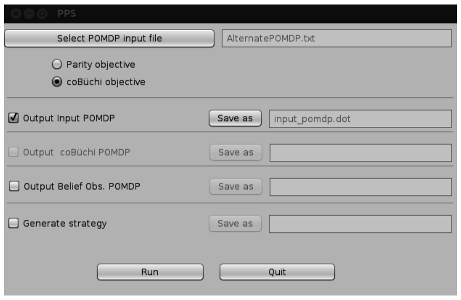
File Name: AlternatePOMDP.txt

Files of Type: All Files

Fig. 4. Select Input POMDP

be saved to. Figure 5 illustrates the same. One can then use GraphViz to generate an image of the file from the dot format. Figure 2 was generated in this way.

5. In case the input POMDP is specified with a parity objective, select the Output coBüchi POMDP checkbox to generate the dot format of an equivalent POMDP with a coBüchi objective. Click the corresponding `Save As` button to specify the location and the name of the file to be saved to.
6. To generate the dot format for the belief-observation POMDP which is generated as part of the algorithm, select the check box next to Output Belief Obs. POMDP. Click the corresponding `Save As` button to specify the location and name of the file to be saved to. This is illustrated in Figure 6



AlternatePOMDP.txt

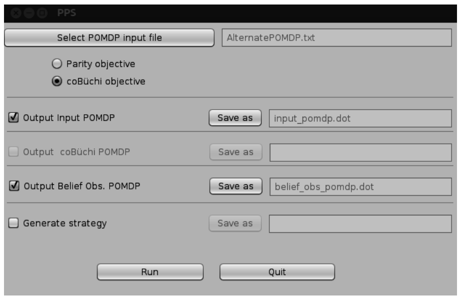
Parity objective [✓ coBüchi objective]

☑ Output Input POMDP — input_pomdp.dot

☐ Output coBüchi POMDP

☐ Output Belief Obs. POMDP

☐ Generate strategy



AlternatePOMDP.txt

Parity objective [✓ coBüchi objective]

☑ Output Input POMDP — input_pomdp.dot

☐ Output coBüchi POMDP

☑ Output Belief Obs. POMDP — belief_obs_pomdp.dot

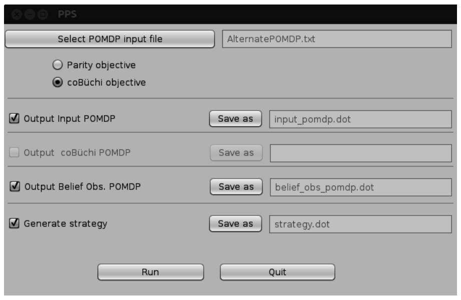
☐ Generate strategy

Fig. 5. Generate Input POMDP

Fig. 6. Generate belief-observation POMDP

7. To generate the dot format file for the final strategy which is generated, select the check box next to Generate Strategy. Click the corresponding `Save As` button to specify the location and name of the file to be saved to. This is illustrated in Figure 7.
8. Now click the `Run` button. After waiting for program execution, you will see a dialog box. An example of the same is illustrated in Figure 8. This provides information about whether there exists a finite-memory almost-sure winning strategy, and the execution time for the program. The dot format files which were selected

for generation will be found at the specified locations. This completes program execution.


PPS

Select POMDP input file: AlternatePOMDP.txt

( ) Parity objective
[✓ coBüchi objective]

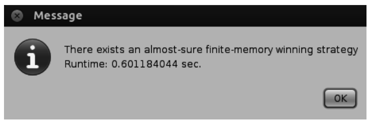
[✓] Output Input POMDP — Save as: input_pomdp.dot

[ ] Output coBüchi POMDP — Save as:

[✓] Output Belief Obs. POMDP — Save as: belief_obs_pomdp.dot

[✓] Generate strategy — Save as: strategy.dot

Fig. 7. Generate Strategy Graph


Message

There exists an almost-sure finite-memory winning strategy
Runtime: 0.601184044 sec.

Fig. 8. Result Screen

Now, 'GraphViz' (See 'Dependencies') must be used in order to visualize the graphs, which have been output in the dot format.

An Explanation for the Strategy. After the execution the tool outputs that there exists a finite-memory almost-sure winning strategy and outputs the strategy graph. Here, we provide an intuitive explanation for the strategy which is given by the PPS tool, for the input POMDP shown in Figure 2. The output strategy is shown in Figure 9. The finite-memory strategy $\sigma = (\sigma_u, \sigma_n, M, m_0)$ corresponding to the Figure is defined as follows:

- The memory set M contains four memory elements $\{m1, m2, m3, m4\}$. The initial memory is the memory element that is successor of the `start` action from the node labeled as `StartState`, i.e., the initial memory element is $m1$.
- The action selection function σ_n for memory m selects uniformly all actions outgoing from node m in the graph, i.e., $\sigma_n(m1)$ selects action `a` with probability 0.5 and action `b` with probability 0.5, $\sigma_n(m3)$ selects action `a` with probability 1, etc.
- The memory update function σ_u given memory element m, action a and observation o chooses the new memory uniformly from the successor states of state m by edges labeled by action a and target states labeled with o, i.e., $\sigma_u(m1, \mathtt{a}, o1)$ updates with uniform probability to $m2$, $m3$, and $m4$. In case of the memory element $m3$ we have $\sigma_u(m3, \mathtt{a}, o1)$ updates with probability 1 to $m4$.

One can see that by playing the strategy σ the Player reaches the memory elements $m3$ and $m4$ with probability 1. From these memory elements the strategy prescribes to alternate actions `a` and `b`. As was mentioned before, this strategy ensures that the initial state `z` is visited only finitely often almost-surely.

0.3 Example - Space Shuttle

The space shuttle example originally comes from [1], and along with the original POMDP we also consider slight variants of the model presented in [2]. We will de-

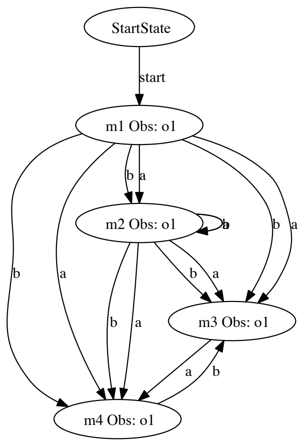


Fig. 9. Finite-memory almost-sure winning strategy

scribe the easiest variant of the problem, the remaining variants will be described in the end of the example. It models a simple space shuttle docking problem, where the shuttle must dock by backing up into one of the two space stations. The goal is to visit both stations infinitely often. Figure 10 originally comes from [2] and shows a schematic representation of the model. The left most and right most states in Figure 10 are the docking stations, the most recently visited docking station is labeled with MRV, and the least recently visited docking station is labeled with LRV. The property of the model is that whenever a LRV station is visited it automatically changes its state to MRV. Both of the actions go forward and turn around are deterministic.

States: There are 11 states in the POMDP, corresponding to the position of the shuttle: `0` - docked in LRV; `1` - just outside space station MRV, front of ship facing station; `2` - space, facing MRV; `3` - just outside space station LRV, back of ship facing station; `4` - just outside space station MRV, back of ship facing station; `5` - space, facing LRV; `6` - just outside space station LRV, front of ship facing station; `7` - docked in MRV, the initial state; `8` - successful delivery; `9` - bump into LRV; `10` - bump into MRV.

Observations: There are 7 observations corresponding to what can be seen from the shuttle: `o0` see LRV forward, `o1`, see MRV forward, `o2` docked in MRV, `o3` see nothing, `o4` docked in LRV, `o5` bumping into a docking station, and finally `o6` is observed upon a successful delivery.

Actions: There are three actions that can be chosen: go forward (`f`), turn around (`a`), and backup (`b`).

Transition relation: If the shuttle is facing a station (states `1` and `6`) the backup action succeeds only with probability 0.3, has no effect with probability 0.4, and with probability 0.3 acts like a turn around action. Whenever in space (states `2` and `5`) the backup actions succeeds with probability 0.8, has no effect with probability 0.1, and with the remaining probability 0.1 has the same effect as an combination of turning around and a backup action. Finally, when the shuttle is adjacent to a station and facing away (states `3` and `4`), it has a probability of 0.7 of actually docking to a station, and with the remaining probability 0.3 has no effect. In the remaining states `8`, `9`, and `10` the action effect is deterministic.

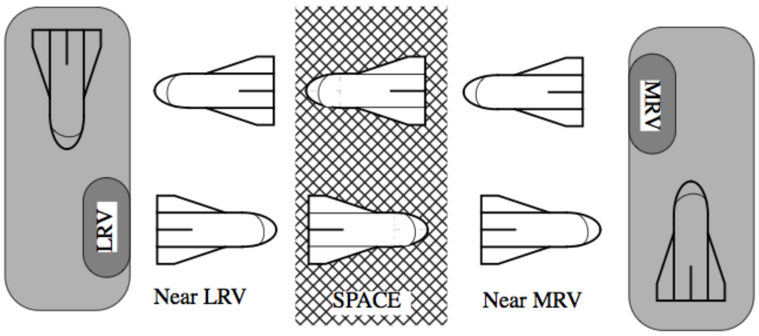


Fig. 10. Space shuttle docking problem

Objective. The parity objective of the model is defined by the priority assignment function as follows: all states from `1` to `7` have priority 3; the state `8` which represents a successful delivery into a least recently visited station, has priority 2. Whenever the shuttle is facing a station, it should try to backup into the station as trying to move forward results into a bump into a station represented with states `9` and `10` with priority 1. Therefore, the objective can be intuitively explained as trying to visit both of the stations infinitely often, while trying to bump only finitely often with probability 1.

The input file for the POMDP can be downloaded here `http://pub.ist.ac.at/pps/examples/Space_shuttle_small.txt`. The more complex variants of this model differ in the number of states that are required to travel through the space, and therefore have higher uncertainty about the position of the shuttle.

0.4 Example - Cheese Maze

The maze is shown in Figure 11 is introduced in [2]. We will describe only the smallest variant in detail. The goal of the player is to reach a goal state while trying to avoid poison in bad states. The player is only partially informed, its observation corresponds to what would be seen in all four directions immediately adjacent to the location. After the goal state is reached the player is respawned with positive probability in multiple states of the maze and the game is restarted.

States: There are 11 states in the POMDP that are illustrated on Figure 11. The game starts in state `6`. The poison is placed in states `8` and `9`, and `10` is the goal state.

Observations: There are 7 observations corresponding to what would be seen in all four directions immediately adjacent to the location, i.e., states `5`, `6`, and `7` do have the same observation. The observations are as follows: `o0`, the walls are NW; `o1`, the walls are NS; `o2`, the wall is N; `o3`, the walls are NE; `o4`, the walls are WE; `o5`, a poisoned state; and `o6`, the goal state.

Actions: There are four actions available corresponding to the movement in the four compass directions (north `n`, east `e`, south `s`, west `w`).

Transition relation: Actions that attempt to move outside of the maze have no effect on the position. The rest of the moves is deterministic in all 4 actions.

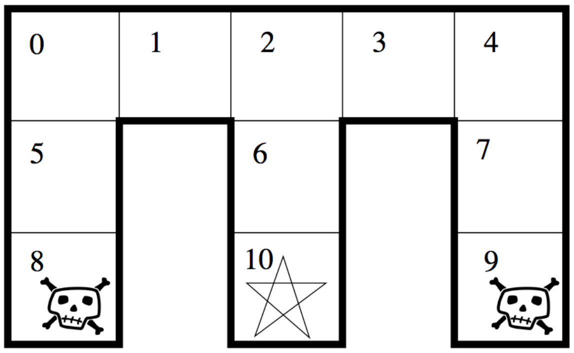


Fig. 11. Cheese maze problem

Objective: The objective is to visit the goal state `10` infinitely often, while getting poisoned in states `8` and `9` only finitely often. This is encoded as a parity objective with 3 priorities. State `10` has priority 2, states `8` and `9` have priority 1, and every other state has priority 3. Whenever the goal state is reached, the maze is restarted with probability $1/3$ to state `0`, with probability $1/3$ to state `2`, and with probability $1/3$ to state `4` in the easiest variant of the problem.

The other variants are medium and difficult, and differ in the number of states the maze can restart to, intuitively increasing the uncertainty in the POMDP and also result into longer running times. We also consider a more difficult setting of the problem by constructing an intermediate and large size mazes with more states but based on the same principle. The input file for the POMDP can be downloaded here `http://pub.ist.ac.at/pps/examples/Small_cheese_maze_easy.txt`.

0.5 Example - Grid

We will describe only the grid 4×4, the other larger variants of the grid only differ in size. The problem consists of a 4 by 4 grid of locations. There is a single goal state, and multiple trap states that are placed beforehand but not known to the player. Whenever a goal state is reached the game is restarted. The goal of the player is to learn the position of the traps and visit the goal infinitely often, while visiting the trap state only finitely often with probability 1.

States: There are 33 states in the POMDP. The uncertainty about the placement of the traps is modeled by two grids of states 4×4 and an additional initial state `start` that has transition to both of the grids. The coding of the states is as follows, the state `ijk` corresponds to a state in the i-th copy, j-th row, and k-th column. The goal state in each grid is the lower right corner, i.e., state `033` and `133` (rows and columns are numbered 0-3).

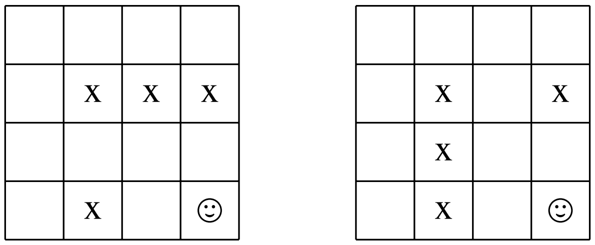


Fig. 12. Trap placement

Observations: There are 4 observations, the initial state has observation `o0`, the trap states have observation `o2`, the goal state has observation `03`, and all the remaining states have observation `o1`.

Actions: As in the previous example, there are 4 actions available corresponding to the movement in the four compass directions (north `n`, east `e`, south `s`, west `w`).

Transition relation: The initial state of the POMDP is the state `start`, no matter what action is played the next state is with probability 0.5 the upper left corner in one of the grids, and with the remaining probability the upper left corner of the second grid. In the grid all the actions are deterministic and attempts to move outside of the grid have no effect on the position. Whenever a goal state is reached the game is restarted the the upper left corner of the same grid (the trap states stay in the same position).

Objective: The objective is to learn in which grids the player is and as in the previous examples try to reach the goal state infinitely often while visiting the trap states only finitely often with probability 1. This is encoded as a parity objective with 3 priorities, the goal state has priority 2, the trap states have priority 1, and all the remaining states have priority 3.

The placement of the trap states we have considered for the 4×4 grid is depicted on Figure 12. The other variants differ in the size of the grid and placements of the trap states. The input file for the POMDP can be downloaded here `http://pub.ist.ac.at/pps/examples/4x4_grid.txt`.

References

1. Lonnie Chrisman. Reinforcement learning with perceptual aliasing: The perceptual distinctions approach. In *AAAI*, pages 183–188. Citeseer, 1992.
2. M. L. Littman, A. R. Cassandra, and L. P. Kaelbling. Learning policies for partially observable environments: Scaling up. In *ICML*, volume 95, pages 362–370. Citeseer, 1995.