POMDP framework implementation in Pacman Project Report

Shengliang Xu
slxu@cs.washington.edu

Problem Statement

Pacman is a small yet powerful code base for AI educations. The code base has already implemented frameworks for a bunch of important AI algorithms, including basic search, competitive search, Markov Decision Process(MDP), reinforcement learning and particle filtering, in the form of interesting games. Although students already are able to learn a lot from this current code base, it may not sufficient if the students want to learn more advanced models and algorithms such POMDP.

This project aims to fill this up, i.e. implementing the general framework for POMDP in Pacman. Specifically, the project implemented several functional parts:

1. Extend the MDP interface in existing code base to provide POMDP problem modeling
2. Implement a MDP module for Pacman. Existing code base provides the MDP module only for gridworld. And the Pacman MDP module is required to implement Pacman POMDP.
3. Implement a basic adaptive POMDP algorithm, which is a simple adaptation of MDP to POMDP.
4. Implement the PBVI [1] algorithm for Pacman POMDP. The current implementation still has problems which are caused by some difficulties I meet. The details are discussed in the Difficulties Meet section.

Implementations

To support POMDP, I extend the basic MarkovDecisionProcess in mdp.py to include two more methods which are required by the POMDP model. The code is in pomdp.py.

The MDP module for Pacman lies in mdpAgents.py. The key point here is to have a correct state modeling and correctly implementation of the state transition probability distributions. The built-in game state is not correct here because it records scores the Pacman currently earned as its state. Current implementation use the agent positions, and the food status as the state (PacmanMDPState). No capsules are allowed for simplification. To implement the state transition probability distributions, we need to know the ghost model before the MDP process runs. The current implementation uses only random ghost modeling for simplicity.

To fit the MDP module into the Pacman framework, PacmanMDPAgent is implemented to give out directives (actions) to the Pacman during running. It does value iteration before the actual game starts. The command to run the pacman game using MDP is:

```
python pacman.py -p PacmanMDPAgent -a
iterations=$ITERATION,discount=$DISCOUNT -l $LAYOUT
```

For POMDP, in addition to all the modeling in the MDP part, we need an extra observation model. Currently, the observation model is fixed as the following for both the adaptive POMDP algorithm and the PBVI algorithm:

> The Pacman can always directly observe its own position and the current food state. It cannot directly observe where the ghost is. But it can observe the Manhattan distance between itself and the ghost.

This observation model is actually very simple because the mapping from any state, in which the Pacman position and the ghost position are both given, to the Manhattan distance is deterministic.

The adaptive POMDP algorithm is implemented in distObservableAgents.py. It works as the following.

1. Compute an MDP model for the current game setting
2. At each step, first compute the posterior belief of the ghost positions using the observation. And then choose the position with the highest updated belief (i.e. MAP estimation of ghost positions). And at last, assume that the ghost is actually at the MAP position and use the pre-computed MDP model to find out the action to be taken.

The command to run the adaptive POMDP algorithm is:

```
python pacman.py -p AdaptivePOMDPAgent -a
iterations=$ITERATION,discount=$DISCOUNT -l $LAYOUT
```

The PBVI algorithm is a point-based approximate algorithm to solve POMDP problems. The reason of implementing PBVI rather than exact POMDP solving algorithms is that in Pacman the number of states can be very large even for a tiny layout. For example, the following 7X7 layout ():

```
%%%%%%%
%  P  %
% % % %
% % % %
% %.% %
% .G. %
%%%%%%%
```

has different number of states as many as (16 Pacman possible non-food positions) * (19 ghost possible positions) * (2^3 different food status) + (3 Pcaman possible food positions)* (19 ghost possible positions) * (2^2 different food status without the food right under Pacman) = 2660 different states. Note that the number of states grows exponentially as the number of food grows. Therefore if the above grid is full of food, the number of states can be as large as about 2^{19} = 524288. This is way too many for exact POMDP solvers.

The PBVI algorithm is implemented in pbviAgents.py. The algorithm alternates between a belief point expansion step and a finite-horizon value iteration step. The algorithm adopts the same observation model and the same ghost model as in AdaptivePOMDPAgent. The number of alternates is controlled by the model parameter *iterations*, and the horizon is controlled by model parameter *horizon*.

The command to run the PBVI Pacman algorithm is:

```
python pacman.py -p PBVIAgent -a
iterations=$ITERATION,discount=$DISCOUNT,horizon=$HORIZON -l $LAYOUT
```

Currently, the algorithm still have problems. The details are discussed in the Difficulties Meet section.

Experiments

I conducted some simple experiments to test the correction of the algorithms and the relative effectiveness of the different algorithms.

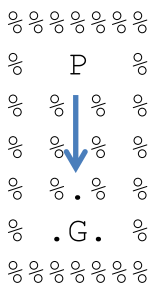
Because the PBVI algorithm still fails to give the correct answers, I only did experiments using PacmanMDPAgent and AdaptiveMDPAgent. For either model, I set the number of iterations to be 20 and run 100 games after the MDP models are computed. The layout used by both algorithms is *smallPOMDP*.

The experiment result is as the following:

Algorithm	Win Rate	Average Score
PacmanMDPAgent	100/100 (1.0)	518.65
AdaptiveMDPAgent	59/100 (0.59)	102.52

The result is just as expected. PacmanMDPAgent is expected to win every time because it is able to observe everything and is able to plan as good as we want. 20 iterations are actually way too many for this algorithm. 5 iterations are enough for the Pacman to win every time.

The behavior of AdaptiveMDPAgent is actually fixed. The Pacman always go down the middle road as the following:



```
%%%%%%%
%  P  %
% % % %
% % % %
% %.% %
% .G. %
%%%%%%%
```

This is because every time the algorithm decides that according MAP estimation, the ghost should be either on the left path or on the right path. The middle path is always safe to go. And because the ghost model is random, the most probable situation in which Pacman may win is that the Ghost goes either the left path or the right path, which is about 2/3=0.67 of them time. The number is very close to 0.59.

Difficulties Meet

The difficulties I meet all lies in implementing the PBVI algorithm. Most importantly, there are two differences between the Pacman environment and the standard POMDP environment assumption used by PBVI.

1. Not all the actions are defined for each state. There are 5 actions defined in Pacman, i.e. EAST, SOUTH, WEST, NORTH, and STOP. But if in the current state, the Pacman is next to a wall or at a corner, some of the actions cannot be conducted. This problem makes the semantic of expected-discounted-reward hyper-planes in POMDP solutions unclear.
2. The second problem is related to the first one but is specific to Point-Based solvers. PBVI computes a set of hyper-planes on a finite set of belief points to approximate the true value functions. But because different state has different subset of valid actions. It happens that given the current belief point b_n and the belief point b_c that is closest to b_n, the action which optimize b_c may not defined for any of the possible states in b_n. Especially in our problem, each belief is actually a belief over the possible positions of the ghost. All the states in any belief will always have the same Pacman position.

Currently, I'm still thinking about the possible solutions to the above problems.

Conclusions and Future Work

To sum up, as the experiments show, the PacmanMDPAgent and the AdaptiveMDPAgent work as expected. The partially observability in POMDP make simple adaptation of MDP unacceptable.

I read a lot trying to understand the Point-based POMDP algorithms. During this process, I've learned a lot. I find POMDP and reinforcement learning very interesting.

In my opinion, they can be good models for real life. I’ll read more on these topics in the future. And I’ll continue working on the PBVI algorithm and definitely will solve the problems discussed in the previous section.

References

[1] Pineau, Joelle, Geoff Gordon, and Sebastian Thrun. "Point-based value iteration: An anytime algorithm for POMDPs." International joint conference on artificial intelligence. Vol. 18. LAWRENCE ERLBAUM ASSOCIATES LTD, 2003.