Planning and Optimization

G. Röger, T. Keller
G. Francès

University of Basel
Fall Semester 2018

Exercise Sheet G

Due: December 16, 2018

The files required for this exercise are in the directory `exercise-g` *of the course repository (*`https://bitbucket.org/aibasel/planopt-hs18`*). All paths are relative to this directory. Update your clone of the repository with* `hg pull -u` *to see the files.*

Exercise G.1 ($1 + 3$ marks)

In this exercise, we will work again with the Berkeley Pacman framework and extend it with some of the algorithms discussed in the lecture. Pull the Pacman code *from our repository*, in `exercise-g/pacman`, as we have done some modifications to the original to make things simpler.

(a) Modify your implementation of Value Iteration from the previous exercise sheet so that instead of running Value Iteration for a fixed number of iterations, you run the standard version of VI where the updates of the value vector V^i stop only when the residual $Res^{i+1} = \max_{s \in S} \left| V^{i+1}(s) - V^i(s) \right|$ is smaller than a certain value $\epsilon > 0$. Your new version of Value Iteration should be in class `ResidualValueIterationAgent`, in `pacman/basel/offlineAgents.py`. You can run your algorithm with `python2 gridworld.py -a resvi`. Remember that you can explore the effect of different parameters in your algorithm, and use different grids. `python2 gridworld.py -h` will give you an overview of the different options.

(b) Value Iteration usually wastes large efforts computing the value function for regions of the state space which might be completely irrelevant if all we want is to reach the goal from a certain initial state. Real-Time Dynamic Programming (RTDP) tackles this by focusing the search on those states that seem more relevant for that task.

- Complete the implementation of an RTDP agent in class `RTDPAgent`, in `pacman/basel/offlineAgents.py`. As discussed in the lecture, there are different possible termination conditions for RTDP, but to simplify things, your agent should keep running RTDP trials until the residual Res^{i+1} computed *over all states visited during the last trial by the greedy policy* π_{V^i} is smaller than a certain value $\epsilon > 0$.
- The convergence of RTDP can be improved by using an admissible heuristic to initialize the state value function on states that have not been seen before. Think on what could be an adequate heuristic in the context of Pacman's `Gridworld`, and justify briefly why your choice is indeed admissible. Then, change the implementation of `GridworldHeuristic` to use the estimates provided by your heuristic. You can test your RTDP agent e.g. with `python2 gridworld.py -d 0.95 -a rtdp -i 100`

Exercise G.2 ($3 + 2 + 2 + 2$ marks)

Let us now tackle a slightly more interesting problem from the Pacman framework, where the Pacman wants to eat all dots in the maze without being caught by a number of ghosts that move randomly around the maze. Note that this can be seen as a SSP problem with rewards instead of costs and with one single goal state $s_\star$. The transition probabilities of the SSP are such that any state where all food dots have been eaten can only transition to $s_\star$, with a reward of approximately $+500$ (the exact number is not too relevant), and any state where the Pacman has been eaten by some ghost can only transition to $s_\star$, with a reward of approximately -500.

We ask you to implement some Monte Carlo Tree Search (MCTS) agents in `pacman/basel/mctsAgents.py` with different tree and default policies, as detailed below. Note that the exact formulation of the SSP is not absolutely needed to implement the agents, since a simulator of the environment is already implemented for you (see `pacman/basel/simulator.py`). In all cases, you should run MCTS rollouts for a number of iterations, as specified in the code comments, and then return the action that maximizes the expected reward according to the information gathered by the MCTS rollouts. The different variations we ask you to implement are:

(a) An MCTS agent with ϵ-greedy tree policy and a random default policy (a default policy that selects actions uniformly at random among the available actions at every state), in `EpsilonGreedyMCTSAgent`. You can invoke this agent with e.g.

```
python pacman.py -p EpsilonGreedyMCTSAgent -l mediumGrid.
```

(b) An MCTS agent with the UCB1 tree policy and a random default policy, in `UCB1MCTSAgent`. You can invoke this agent with e.g.

```
python pacman.py -p UCB1MCTSAgent -l mediumGrid.
```

(c) For both agents above, change the random default policy for a more sensible policy that takes into account the characteristics of the Pacman domain. You should invoke your agent with an additional command-line option `-a default_policy=pacman`, e.g., for the ϵ-greedy agent:

```
python pacman.py -p EpsilonGreedyMCTSAgent -l mediumGrid
-a default_policy=pacman.
```

(d) How does the performance of your agent change when using the different strategies above? How does the exploration constant in the UCB1 action selection algorithm affect this performance? Try out different values for this constant (e.g. by adding command-line option `-a exploration_factor 0.6`) and report on any change, if any, in the behaviour of the agent and the average accumulated reward. Remember that you can try out different grid layouts with command-line option `-l`.

Exercise G.3 (4 marks)

One of the earliest successes of Monte Carlo Tree Search (MCTS) methods and of UCT in particular was achieved a few years ago in the ancient game of Go. An accessible overview of those early attempts, on which later methods such as AlphaGo build, can be found in:

> Gelly, S., Kocsis, L., Schoenauer, M., Sebag, M., Silver, D., Szepesvári, C., & Teytaud, O. The grand challenge of computer Go: Monte Carlo tree search and extensions. In *Communications of the ACM*, vol. 55:3, pp. 106–113, 2012.

Read the paper and summarize its main ideas *in your own words* (between 300 and 600 words).[1] These are some of the points that your answer could discuss:

- What is the main difference between the use of MCTS discussed in the lecture and that discussed in the paper, in terms of the context where the algorithm is used. Is Go a deterministic or a stochastic game?
- What are the relative advantages and disadvantages of using evaluation functions, playout policies and stochastic playout policies?
- How exactly does UCT achieve "consistency"?
- How do the authors suggest that offline knowledge about the problem being tackled, in this case Go, can be used to improve the performance of the algorithm?

[1] We remind you that you can quote material from the original paper or other sources if you think it is necessary (although that is not the objective of this exercise), but using literal sentences in your summary without proper attribution through the use of quotes or otherwise will be considered plagiarism.

The exercise sheets can be submitted in groups of three students. Please submit one single copy of the exercises per group (only one member of the group does the submission), and provide all student names on the submission.