

Lab 3

Assigned: Oct 25

Due: Nov 15, 10pm

Assignment

Write a generic Markov process solver.

The program should take 4 flags (which will be understood better later) and an input file.

- -df : a float discount factor [0, 1] to use on future rewards, defaults to 1.0 if not set
- -min : minimize values as costs, defaults to false which maximizes values as rewards
- -tol : a float tolerance for exiting value iteration, defaults to 0.01
- -iter : an integer that indicates a cutoff for value iteration, defaults to 100

e.g.

```
mdp -df .9 -tol 0.0001 some-input.txt
```

Grading

out of 100:

- 90 points for correct behavior
- 10 points for well written code: clear data structures, methods, control flow, etc.

Markov process solver

You will implement the algorithm described in class:

```
 $\pi$  = initial policy (arbitrary)
V = initial values (perhaps using rewards)
for {
    V = ValueIteration( $\pi$ ) // computes V using stationery P
     $\pi'$  = GreedyPolicyComputation(V) // computes new P using latest V
    if  $\pi == \pi'$  then return  $\pi$ , V
     $\pi = \pi'$ 
}
```

Where:

- *ValueIteration* computes a transition matrix using a fixed policy, then iterates by recomputing values for each node using the previous values until either:
 - no value changes by more than the 'tol' flag,
 - or -iter iterations have taken place.
- *GreedyPolicyComputation* uses the current set of values to compute a new policy. If -min is not set, the policy is chosen to maximize rewards; if -min is set, the policy is chosen to minimize costs.

The value of an individual state is computed using the Bellman equation for a Markov property

$$v(s) = r(s) + df * P * v$$

Where:

- v on the RHS is the previous values for each state
- df is the -df discount factor flag applied to future rewards
- P is the transition probability matrix computed using the policy and the type of node this is (see below)
- $r(s)$ is the reward/cost for being in this particular state.

Input file and State types

Node/state names should be alphanumeric

The input file consists of 4 types of input lines:

- Comment lines that start with # and are ignored (as are blanklines)
- Rewards/costs lines of the form 'name = value' where value is an integer
- Edges of the form 'name : [e1, e2, e3]' where each $e\#$ is the name of an out edge from name
- Probabilities of the form 'name % p1 p2 p3' (more below)

These lines may occur in any order and do not have to be grouped, so this is valid:

```
A = 7
B % .9
C : [ B, A]
C=-1
A : [B, A]
A % .2 .8
B : [A, C]
```

Probability entries:

- A node with the same number of probabilities as edges is a *chance node*, with synchronized positions. e.g.

```
A : [ B, C, D]
A % 0.1 0.2 0.7
```

Indicates that from node A there is a 10% chance of transitioning to node B, 20% to node C and 70% to node D.

Note: these probabilities should sum to 1 or your program should indicate an error.

- A node with a single probability 'name : p' is a *decision node*, with the given success rate. As in the maze example, failures are split evenly amongst the other edges. e.g.

```
F : [C, E, G]
F % .8
```

Given a policy of $F \rightarrow E$, transition probabilities would be $\{C=.1 \ E=.8 \ G=.1\}$, noting that this changes with the policy.

Note: with $p < 1$ this is Q-learning where $\alpha=1-p$

Nodes:

- If a node has edges but no probability entry, it is assumed to be a decision node with $p=1$
- If a node has edges but no reward entry, it is assumed to have a reward of 0
- If a node has no edges it is terminal. A probability entry for such a node is an error.
- If a node has a single edge it always transitions there. (this is useful for capturing some reward on the way)
- A node referenced as an edge must separately have one of the three entries to be valid
- Therefore to create a 0 value terminal node you must do 'name = 0'

What the fuck does this mean?

Output

Your program should output the optimal policy (if applicable, a problem with only chance nodes has no policy), and the values of each node/state (under that policy if applicable).

See examples for the format.

Examples

1. [Maze from slides](#) produces [solution](#).
2. [Publishing Decision Tree](#) produces [solution](#).
3. [Restaurant Decision Tree](#) run with '-min' produces [solution](#).
4. [Student Markov Process](#) produces [solution](#).
5. [Student MDP](#) produces [solution](#).
6. [CMU Teacher example](#) with -df=.9 produces [solution](#).

Submission

On Brightspace, a zip file containing:

- Code for the program
- Makefile/BUILD/pom.xml/etc needed to build your code
- README indicating how to compile and run your code
- In your README include any classmates you consulted with
- As previously indicated, the code should build on department linux machines