COMP3702 - Assignment 3

Roy Portas - 43560846

October 25, 2016

1 Problem Definition

1.1 State Space

The state space will be a set of tuples of integers from 0 to the maximum number of items the shop can stock. The size of the tuple is the number of items a shop can stock. Thus for a tiny store it will be $\{(0,0),(0,1),(1,0),\ldots,(3,0),(0,3)\}$. Note the sum of the elements in the tuple must be less than or equal to the number of items a store can stock.

1.2 Action Space

The actions space will be the set of all actions that can be performed by the customers. Thus for a tiny store, it will look like the following:

```
{
    (buy 0 of item 1, buy 0 of item 2),
    (buy 1 of item 1, buy 0 of item 2),
    ...,
    (buy 2 of item 1, buy 1 of item 2),
    (buy 3 of item 1, buy 0 of item 2)
}
```

Note that returning an item is represented as buying a negative number of items.

1.3 Transition Function

The transitions of items are indepedant from each other, meaning that buying one item type does not influence the probability of buying another item type. Thus the transition function for a tiny store can be represented as follows:

$$T((i_1,i_2),(t_1,t_2),i'_1,i'_2) = T_{i_1}(i_1,t_1,i_1)\cdot T_{i_2}(i_2,t_2,i_2)$$

The transition function will be a matrix for each item type where the rows and columns are the number of items the store stocks.

Item 1	0	1	2	3
0	0.2	0.3	0.2	0.3
1	0.4	0.1	0.2	0.3
2	0.1	0.5	0.2	0.2
3	0	0.8	0.2	0

Item 2	0	1	2	3
0	0.4	0.1	0.2	0.3
1	0.2	0.3	0.2	0.3
2	0.1	0.5	0.2	0.2
3	0.1	0.9	0	0

In this table, the rows represent the current state and the columns represent the next state.

1.4 Reward Function

The reward function will be the net profit made by the store after performing an action. As with the transition function, the reward for one item type is independant to the reward of other item types.

1.5 Discount Factor

The discount factor is given in the input file.

1.6 Value Function

A value function associates a state with the value, or 'worth' of being in that state.

$$V_\pi(s) = \sum_{s'} T(s, \pi(s), s')\big[R(s, \pi(s)s') + dV_\pi(s')\big]$$

However since the reward is independant from the next state, the reward function can be represented as $R(s,a)$, yielding:

$$V_\pi(s) = R(s, \pi(s)) + d\sum_{s'} T(s, \pi s, s')V_\pi(s')$$

1.6.1 Calculating the value function

```
def value(action):
    # Calculate the expected number of items sold
    prob = action * transition

    # Calculate the net value of the transactions
    value = prob * item_prices
```

Where action is the tuple containing the the number of items.

2 Algorithm Description

2.1 Offline Implementation

1. Generate every possible action the customer can take and put it into an array, calculating the cost of any returns done
2. Run each array through the value function (transition function?) to calculate the profit for the action
3. Depending on if online or offline, continue building the tree

- Monte Carlo Tree, edges will be actions and nodes will be profit value
- Choose the best node and set that as the root
- Monte Carlo Tree seems to converge to a point?

2.2 Online Implementation