COMP3702 - Assignment 3

Roy Portas - 43560846

October 27, 2016

1 Question 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.

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

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 profit of the transactions
    profit = prob * item_prices
```

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

2 Question 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 to calculate the profit for the action
3. For each result in the result list, repeat step 1 with the new result as the new root node
4. Break the loop when the number of weeks is reached
5. Find the highest profit node and backtrack up the tree to find the order to make

2.2 Online Implementation

The online method will involve getting the current store stock at a given week and calculating the best outcome based on the given probabilities. It will do this by using the value function to calculate the profit for every possible outcome.

1. Get the current stock of the store
2. Using the current stock, find every possible combination of returning and buying items
3. Apply the value function to every combination
4. Select the state with the highest profit

3 Question 3: Algorithm Analysis

The two algorithms to compare are the Monte Carlo search tree, which was used for the offline implementation, and the base value iteration method, which was used for the online implementation.

The two algorithms can be compared based on running time and profit generated, these are shown in the tables below.

Base Value Interation, Small Store		
Run	Running time	Profit
1	8ms	$1349.56
2	9ms	$1322.18
3	10ms	$1311.05
4	9ms	$1531.78
avg	9ms	$1378.64

Monte Carlo Search Tree, Small Store		
Run	Running time	Profit
1	20711ms	$817.99
2	20748ms	$521.06
3	21802ms	$1146.53
4	21458ms	$1267.68
avg	21179.75ms	$938.315

As shown in the tables, the base value interation method is both faster and yields more profits, thus this is the method used in the code.

This is due to the estimates made by the algorithm, the base value interation method has a more accurate estimate, as it is only predicting a single step ahead, whereas the Monte Carlo Search Tree is attempting to look ahead until the total number of weeks are reached, thus it's estimate become inaccurate the more into the future it guesses.

Additionally the base value iteration method scales better with larger inputs. This is because larger input have larger possible combinations, thus the Monte Carlo Search Tree will have many more nodes at each level, which will drastically increase the running time of the application.

However the base value iteration method only looks one step ahead at most, thus will be affected by larger inputs far less.