RL Grocery Shopping Solutions

Aayushi Malik (2015003)
Manasi Malik (2015146)

Problem Statement

When we want to purchase our monthly grocery, there are multiple store options with different pricings and availability. There is also a cost attached to travelling to these stores. Using reinforcement learning, we model these variables to learn the best strategy to purchase all the items we want.

Objective:

- To simplify grocery shopping-given a list of items to be bought and a list of shops with each shop having different pricing and item-availability patterns.
- The goal of our agent will be to find an optimal sequence of shops to visit to be able buy all the items in a given list - in the least amount of time.

Problem Description

- shops=[S_1, S_2,...,S_n]
- buying_status=[I_1,I_2,...,I_m] where $I_i \in \{Y,N\}$ depicting whether the item i has been bought or not
- States={current_shop,current_buying_status}
- State Space = No. of shops x 2^(no. Of items to buy) = n * (2^m)
- Actions=[S_1, S_2,...,S_n]

Problem Description

Transition Probabilities and Rewards (for 1 item to buy):

1. If item has been bought (current_status=Y) then, session terminates
2. If chosen to try same shop again (action = current_shop, next_shop=current_shop)

   P[current_shop][current_status][action][next_shop][next_status] = availability_in_shop(next_shop,next_status)

   a. Item Found : *R[current_shop][current_status][action][next_shop][next_status] = reward_buying + distance_penalty(distance(current_shop,next_shop)) + price_penalty(Price[next_shop])*

   b. Item Not Found: *R[current_shop][current_status][action][next_shop][next_status] = distance_penalty(distance(current_shop,next_shop))*

3. If chosen to go to another shop

   a. If you end up in the chosen shop (next_shop=action)
      *P[current_shop][current_status][action][next_shop][next_status] = 0.9*availability_in_shop(next_shop,next_status)*
      i. Item Found: *R[current_shop][current_status][action][next_shop][next_status] = reward_buying + distance_penalty(distance(current_shop,next_shop)) + price_penalty(Price[next_shop])*
      ii. Item Not Found: *R[current_shop][current_status][action][next_shop][next_status] = distance_penalty(distance(current_shop,next_shop))*
   b. If you end in some shop not chosen (next_shop!=action)
      *P[current_shop][current_status][action][next_shop][next_status] = 0.1*availability_in_shop(next_shop,next_status)*M(next_shop,current_shop)*
      i. Item Found: *R[current_shop][current_status][action][next_shop][next_status] = reward_buying + distance_penalty(distance(current_shop,action) + distance(action,next_shop))+price_penalty(Price[next_shop])*
      ii. Item Not Found: *R[current_shop][current_status][action][next_shop][next_status] = distance_penalty(distance(current_shop,action) + distance(action,next_shop))*

Models

Item Availability Model

Function: availability_in_shop

We will use a Bernoulli Distribution for each item in each shop. Either the item is available or it's not. (X=0 or 1)

This function returns the probability P(X=next_status) using the distribution for that particular item and shop.

M function

The M function used above ensures that if action Si is chosen, then the probability of ending up in shop Sj is greater than that of ending up in shop Sk if and only if Sj is closer to Si than Sk.

The function is described as:

$$M(i,j) = \frac{\left(\sum_{p=1}^{n}\sum_{q=1}^{n} distance(S_p, S_q)\right) - distance(S_i, S_j)}{(n-1)\sum_{p=1}^{n}\sum_{q=1}^{n} distance(S_p, S_q)}$$

Price of Item Model

Function: price_penalty

We will use a Gaussian Distribution for price of each item in each shop. The is the MRP, the price fluctuates slightly around the MRP. Based on the price a penalty (cost) is added.

Distance Travelled Model

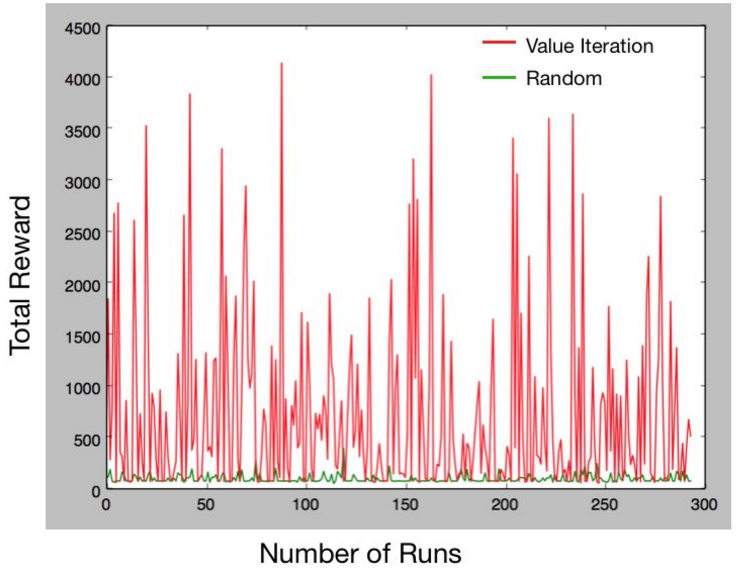
Function: distance_penalty

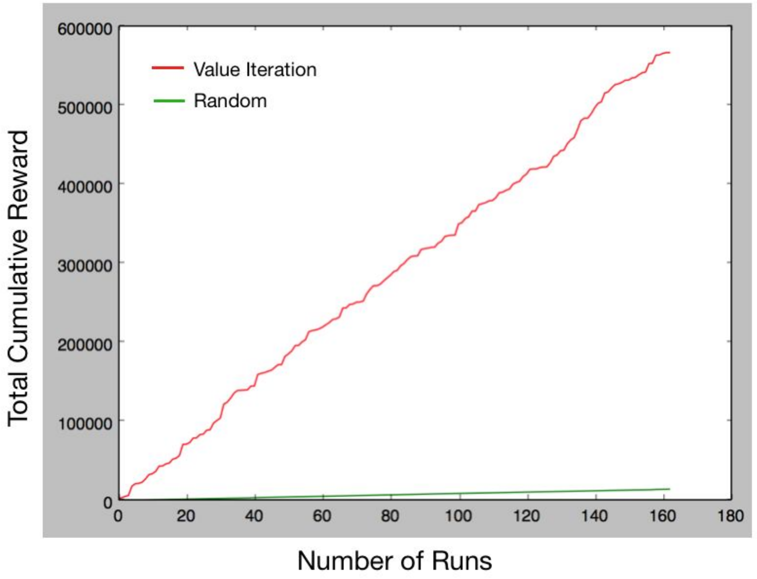
We will have fixed distances between the shops. To account for real world conditions like traffic congestions, we use a Poisson Distribution - parameter passed is distance. A penalty(cost) is associated with the distance (+noise) travelled.

What has been done:

- Creating the environment from scratch
- Implemented Value Iteration
- Implemented Q Learning
- Implemented Representation Policy Iteration (Approximation Method)

Value Iteration





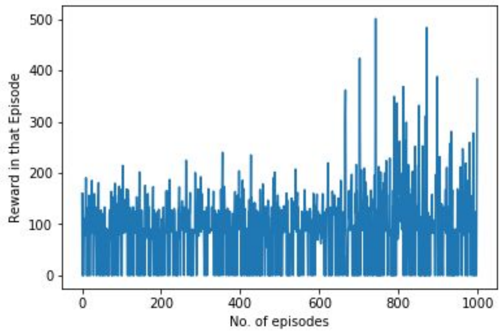
It can be seen that value iteration shows much better performance than simply selecting actions randomly.

Q Learning

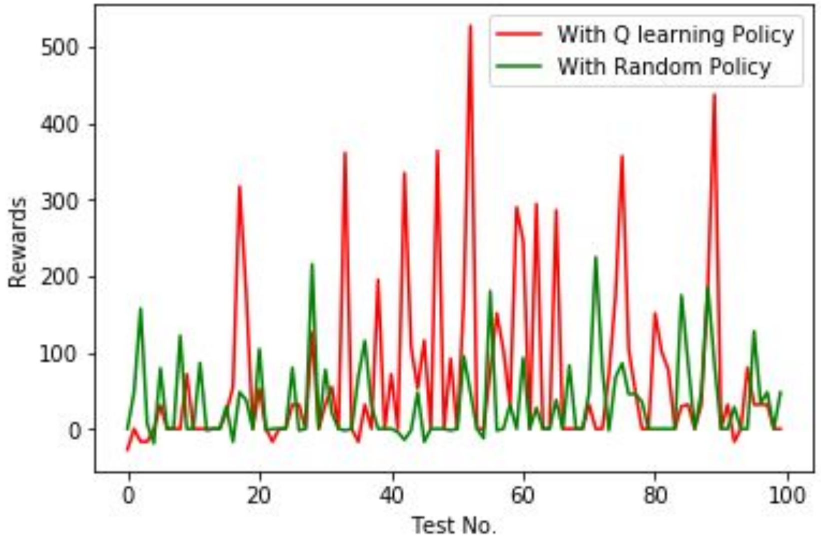
```
Best Policy for each state

State (2, (1, 1)) Best Action End
State (1, (0, 0)) Best Action 1
State (0, (1, 0)) Best Action 0
State (1, (1, 0)) Best Action 1
State (3, (1, 0)) Best Action 3
State (3, (0, 0)) Best Action 0
State (2, (0, 1)) Best Action 3
State (0, (0, 1)) Best Action 0
State (2, (0, 0)) Best Action 0
State (3, (1, 1)) Best Action End
State (1, (1, 1)) Best Action End
State (1, (0, 1)) Best Action 0
State (3, (0, 1)) Best Action 0
State (0, (1, 1)) Best Action End
State (0, (0, 0)) Best Action 0
State (2, (1, 0)) Best Action 1
```

Rewards



Rewards



Representation Policy Iteration

- For number of shops = 10 and number of items = 10, the size of the state space becomes $10*2^{10}$ = 10240 states.
- We used Representation Policy Iteration to obtain an approximate value function.
- We used PVF to construct the basis vectors, which uses the Laplacian operator on the graph representation of the states and then finds the eigenfunctions. We had 20 basis vectors.
- We used Least Squares Projected Equations method to obtain the parameter vector r.
- However, as we have created the environment ourselves, we were getting some errors which we could not tackle in the given time frame.