Technical University of Crete
Electrical and
Computer Engineering

Reinforcement Learning and Dynamic Optimization
COMP 423
Network Friendly Recommendations
Project Report

<u>Group 33:</u>
Ιωαννίδης Χρήστος (2018030006)
Παπαματθαιάκη Ηλέκτρα-Δέσποινα (2018030106)

The goal of this project is to develop a recommendation system using MDP (Markov Decision Process) and Q-learning methods. The aim is to optimize recommendations and minimize the average total cost of items watched during a user session.

Policy Iteration

Problem modeling: For the environment of this problem each state is represented by an item, and each action is represented by the items that are going to be recommended (e.g. state:4 ,action: 5-7 , 435-999, if N=3 then 1-5-8 etc). Every time an item that is not cached is chosen as the next state it inflicts a cost of -1. Everything is modeled as a dictionary whose structure is as follows:

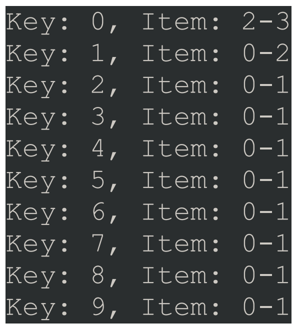
{state1 { action1 : [(probability_of_transition,next_state,cost , is_it_terminal),(probabil……)] action2 : [(probability………)]} state2 { action1 : [(p……..….)]}

The transition probabilities are calculated iteratively by using the following parameters as base:

- q : the probability the user ends the viewing session (for transition to terminal state)
- α: the probability the user picks one of the N recommended items (with equal probability)

Testing optimality: In order to test the algorithm's optimality we created a case where the relativity of all items u_ij was 1 with item 0 and 0 elsewhere. If the algorithm works optimally we expect the actions of the final policy to all include item zero (except for state 0 because it cant transition again to the same state).

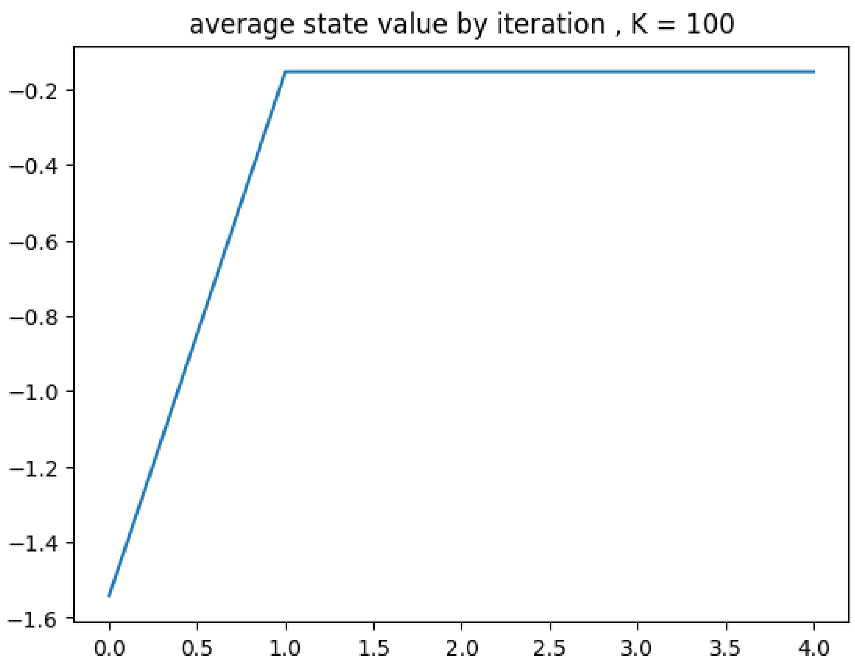
Results: sure enough the results confirm the hypothesis



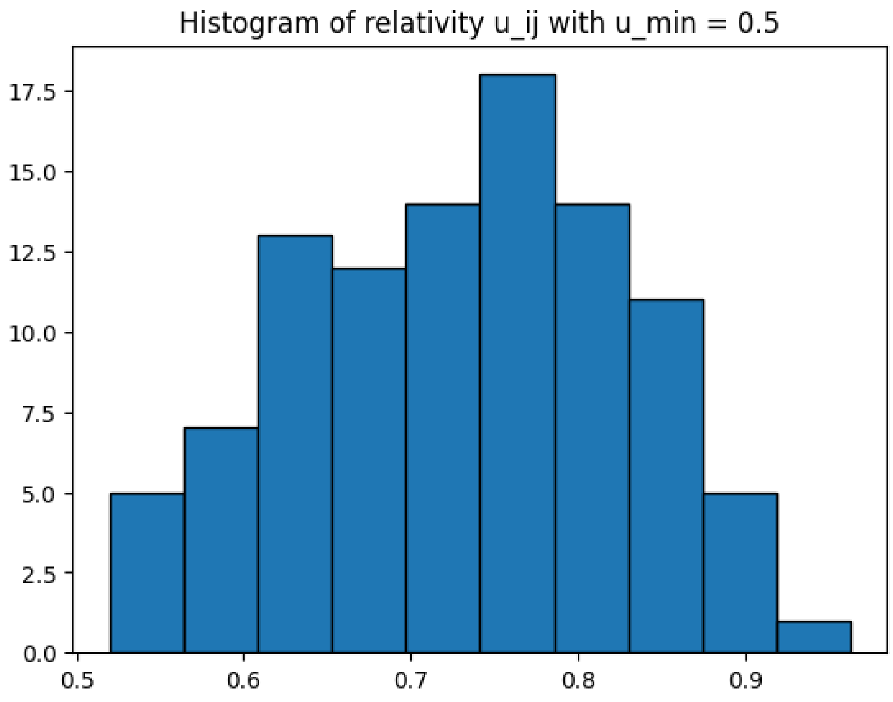
```
Key: 0, Item: 2-3
Key: 1, Item: 0-2
Key: 2, Item: 0-1
Key: 3, Item: 0-1
Key: 4, Item: 0-1
Key: 5, Item: 0-1
Key: 6, Item: 0-1
Key: 7, Item: 0-1
Key: 8, Item: 0-1
Key: 9, Item: 0-1
```

Where ‘Key’ represents a state and ‘Item’ the action proposed by the policy.

Running with random u_ij: Running the algorithm normally, with randomized relativities we can observe that with each iteration the cost of the value function moves closer to 0, first sharply and slowly afterwards. X-axis represents the iteration number and Y-axis the average cost of the value function. A typical scenario with parameters u_min=0,5, a=0.9 , q=0.02 , γ=0.5.



Also below is a histogram including the average u_ij values of the items in the actions selected.



Parameters effect in final average cost:

Below a matrix is presented which includes different combinations of the parameters K, u_min, a, q and the respective average of the value function in the final iteration. As it is not possible to present data for every case, we will focus on the ones deemed most important.

Variance in the number of items

K=50	u_min=0,5	a=0.9	q=0.02
average final cost = -0.152			
K=30	u_min=0,5	a=0.9	q=0.02
average final = -0,160			
K=10	u_min=0,5	a=0.9	q=0.02
average final cost = -1,33			

We can observe it affects the final cost only for small values.

Variance in u_min

K=50	**u_min=0**	a=0.9	q=0.02
average final cost = **-0.152**			
K=50	**u_min=1**	a=0.9	q=0.02
average final cost = **-1.534 !!**			

We can observe from the above fringe values of u_min that for case u_min = 0 the cost is not affected as expected, however in the case of u_min = 1 the value essentially forces the algorithm to act randomly, having a huge impact on the cost.

Changing quit probability

K=50	u_min=0,5	a=0.9	**q=0.2**
average final cost = -0.106			

No notable change here.

When we act deterministically upon choosing the next item

K=50	u_min=0,5	**a=1**	q=0.02
average final cost = **0**			

It basically nullifies randomness in the selection which seems to also nullify the cost in this case

Q-Learning Algorithm

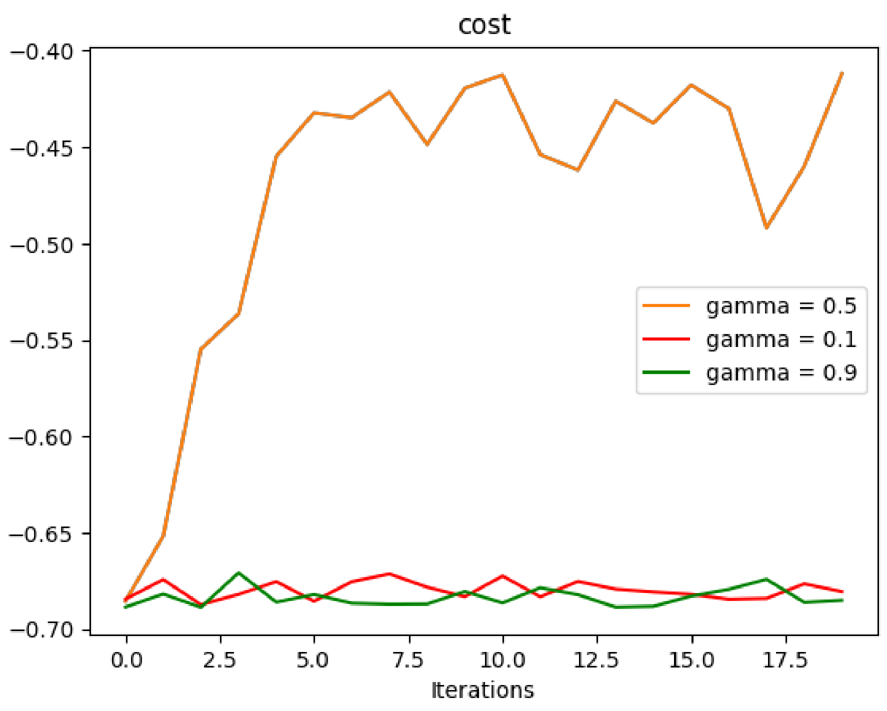
This algorithm is implemented when parameters α and u_min are unknown. It uses the Q-learning technique to learn and optimize the recommendation policy. It updates the Q-table based on the rewards obtained and the maximum expected future rewards.

Problem modeling: The environment of this problem is the same as in policy iteration, each state is represented by an item, and each action is represented by the items that are going to be recommended (e.g. state:4 ,action: 5-7 , 435-999, if N=3 then 1-5-8 etc).

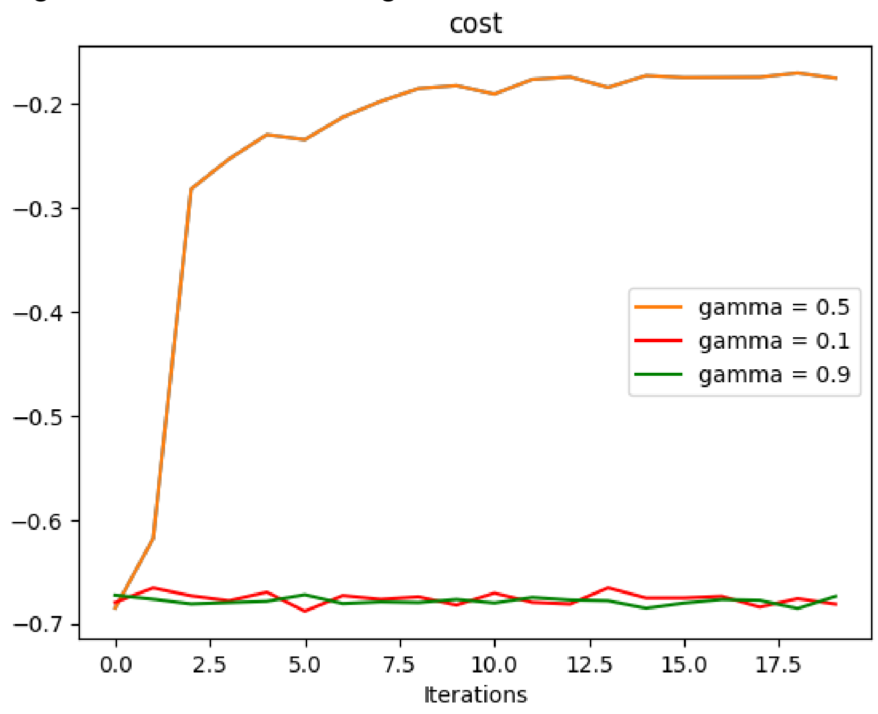
Important Note: In the below graphs for the X-axis the unit **'iterations'** is used. It is just a unit invented to simplify the numbers and help with the analysis of the data by dividing it in slices. Each iteration is a slice of the total episodes in this case 1 iteration = 1000 episodes.

In order to decide on the best possible parameter values a few tests were run. The Learning Rate was set to 0.1. 0.3 and 0.7 while the discount Factor (γ) was set to 0.5, 0.1 and 0.9 .

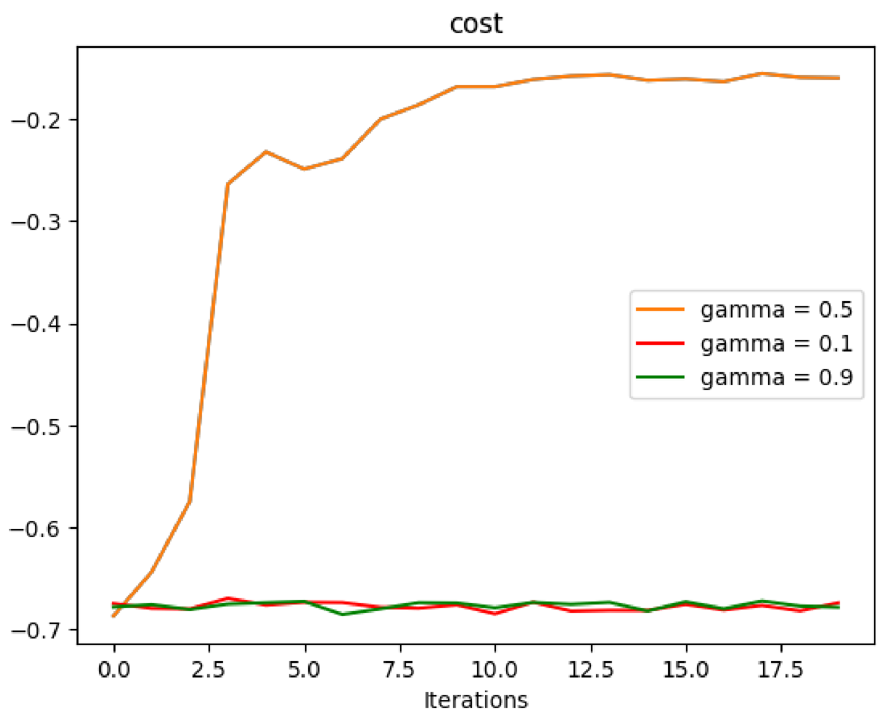
- Learning Rate = 0.7 with different gamma values:



- Learning Rate = 0.3 with different gamma values:



- Learning Rate = 0.1 with different gamma values:



Observations:

It is obvious by looking at the graphs that when the learning rate is at 0.7, it takes way more time for the algorithm to find the optimal solution. When it is set to 0.3 the outcome is not faulty therefore it is easily seen that when the value is 0.1 after around 7.5 iterations it is the Q-table converges and the values do not significantly change. As far as the discount factor is concerned the one set to 0.5 shows much better and clear results in every graph.

Also, the run time of the code had to be tested. Hence, the value K (the number of the content items) was changed several times to determine the point where the code starts going too slow. One important piece of information is that the code was tested only for parameters: K = 50, γ = 0.5, learning rate = 0.1 and not for every value of the discount factor. The results can be observed in this table:

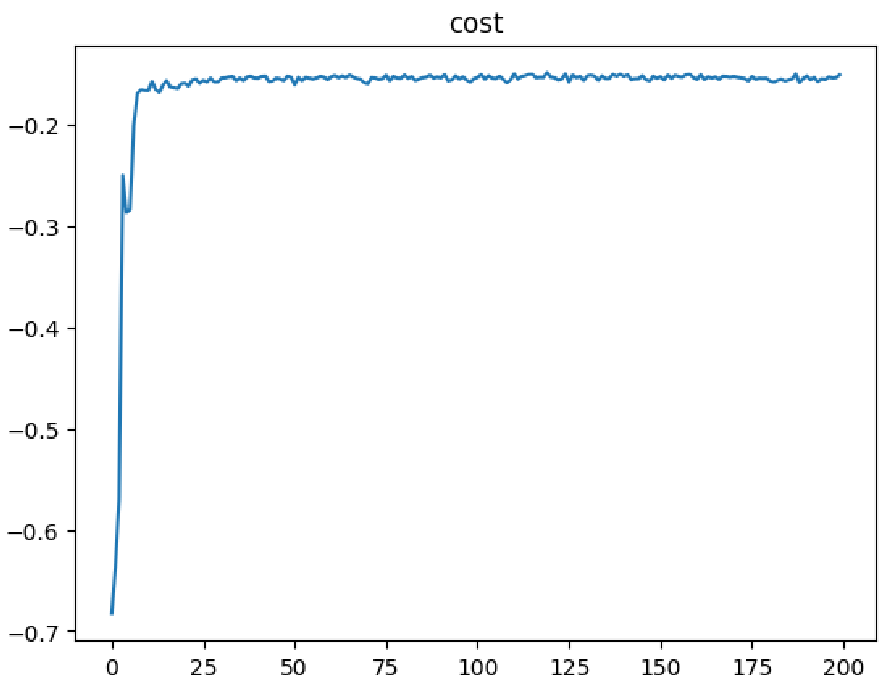
K	50	100	200	300	400	700	2200
Total Run Time	39s	1 min 3s	2 min 7s	4 min 19s	6 min 35s	15 min	>1h

Observations:

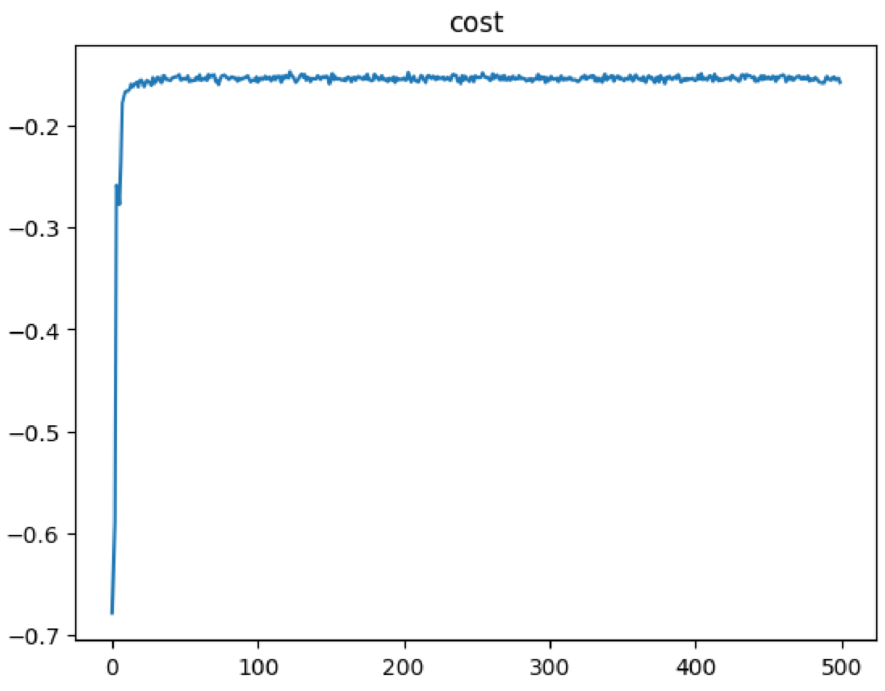
The algorithm is fast when processing item quantities of 50, 100, and 200. It is performing fast even at a scale of 300 and 400 items however, a decrease in speed is noticable. The runtime increased significantly when it came to 700 items, taking 15 minutes to process, which can be considered extremely slow. In a final "bold" test with 2200 items, the execution time exceeded one hour, which is why the process was terminated since there was no point in waiting more. In conclusion, when more than 2200 items are used the Q-Learning algorithm is starting to go too slow.

Afterwards, with K = 50, γ = 0.5, learning rate = 0.1, the number of executions of the algorithm were changed in order to check the convergence.

- Executed Q Learning Algorithm 200 times:



- Executed Q Learning Algorithm 500 times:



Observations:

It is easily observed that in both scenarios the Q-Values have converged to the optimal ones and stay static after 20-30 iterations, this concludes that 20 iterations used in the above graph are enough to represent the whole.

Parameters effect in final average cost:

After performing the same tweaks in the algorithm's base parameters as in policy iteration (K, u_min, a, q) , we can observe similar response patterns concerning the cost, something expected as the base function of those parameters is the same.

Computational efficience comparison between the 2 algorithms:

In contrast to policy iteration in which computational time can increase exponentially with the size of the state space, Q-learning uses a sample-based estimate approach which works far better with larger state spaces computationally, while giving good enough results. We can clearly notice the difference by observing the linear increase in time compared to K in Q-learning while at the same time with policy iteration, even a state space as small as 100 states the computation time becomes unbearably high.