Dynamic Programming & Reinforcement Learning 2020 **(Due: 6 nov 2020)**

Submission Assignment #1

Instructor: Ger Koole *Names:* Mick IJzer & Kimberly Boersma, *Netids:* 2552611 &

This assignment considers an airline revenue management problem. Given are 3 prices ($f = (500, 300, 200)$), a maximum capacity ($c = 100$), and a time limit ($T = 600$). Furthermore, the probability of a customer coming in that is willing to pay f_i is given by equation 0.1, where $\mu_i = (0.001, 0.015, 0.05)$ and $v_i = (0.01, 0.005, 0.0025)$.

$$\lambda_t(i) = \mu_i \exp^{v_i t} \tag{0.1}$$

1 Question A

The first step to determine the total expected revenue and the optimal policy by implementing dynamic programming is to declare the constants and initialize the value matrix and alpha matrix. The constants (see the introduction) are used to compute a probability matrix of seeing a certain customer type at all timepoints. This results in a 3x600 matrix, where each row corresponds to a customer type and each column corresponds to a timepoint. The probability of seeing a customer that is willing to pay more than the ticket price is added to each probability, because a customer that is willing to pay 500 for a ticket is also willing to pay 300 or 200 for the same ticket. Concretely this means that the first row is the probability that someone is willing to pay 500, the second row is the probability that someone is willing to pay 300 or 500, and the third row indicates the probability that someone is willing to pay 200, 300, or 500 for a ticket.

Next, the value matrix and alpha matrix are initialized. Both have a shape of 100(capacity)x600(timepoints). The value when there are no tickets available anymore is set to 0 and the corresponding action (alpha) is set to nothing. This makes sense, since if there are no tickets the airline cant make more revenue or sell tickets anymore.

Now that everything is properly prepared, the dynamic programming approach can start. The first iteration is over the reversed timepoints, within that loop there is another loop over all possible states. In the inner loop the expected revenue at that timepoint is computed by multiplying a slice of probability matrix at that timepoint with the prices. Next, the expected value of the next timepoint is added. Lastly, at that timepoint and that state the maximum resulting value is added in the value matrix, and the corresponding action (price of the ticket) is added to the alpha matrix. In the resulting value matrix the total expected revenue can be found at capacity = 100 and time = 0. For this specific revenue management problem the total expected revenue is 30.190,90. What this means is that the average ticket price is around 302.

Algorithm 1: Dynamic Porgramming Algorithm

```
Result: Optimal Value Matrix and Action Matrix
Initialize(Probability, Value, and Alpha Matrices);
for each t in reversed(time) do
    for each c in capacity do
        values = probability_matrix[:, t] * prices;
        values += probability_matrix[:, t] * value_matrix[c - 1, t + 1];
        values += (1 - probability_matrix[:, t]) * value_matrix[c, t + 1];
        value_matrix[c, t] = max(values);
        action_matrix[c, t] = prices[argmax(values)];
    end
end
```

2 Question B

The alpha matrix that resulted from section 1 can be used to visualize the optimal policy. This optimal policy is shown in figure 1. What this policy shows is that at time 0 seats are sold for 200. If enough tickets are sold,

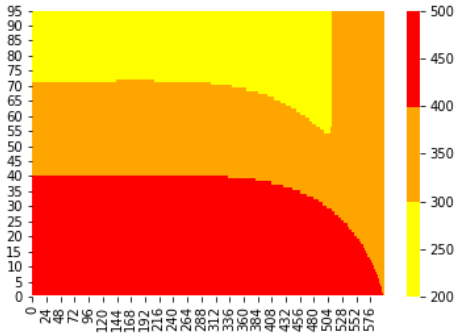

Figure 1: Optimal Policy

the price will go up. This happens again for the boundary between 500 and 300. However, as the last timepoint comes closer the policy changes to pricing the seats 300. This makes sense from a real-life point of view, since at first prices are low and when the demand rises the prices will rise as well. And when the departure date draws near and there are still tickets left, airlines tend to do a last minute sale.

3 Question C

Based on the information in sections 1 and 2 the process is simulated. At each timepoint a sample is drawn from a uniform distribution. Since a cumulative probability matrix is used it is easy to relate the drawn sample with the probabilities at a certain timepoint. The result is the amount that the customer is willing to pay. Of course there is also a probability that there will be no customer at a timepoint. Next, the optimal policy is used to get the price that a ticket will be sold for. If the price is larger than the amount that the customer is willing to pay the seat will not be sold. Otherwise the seat is sold for the price of the policy. At each timepoint the cumulative revenue, current capacity, the ticket price, and the sales are stored. The simulation is repeated for 500 times to obtain a reliable average over time. The stored information is shown in figures 2, 3, 4, and 5.

4 Question D

As can be seen in 1 the price can go down in certain states and at certain times. Sometimes cooperations do not want to reduce prices over time, so that prices can only increase. To implement this in a dynamic programming setting the computation of the values in algorithm 1 have to be updated. This is done by adding a penalty

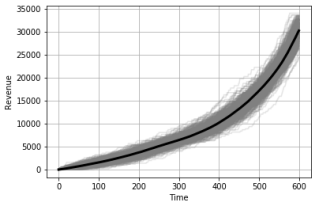

Figure 2: Simulated Revenue

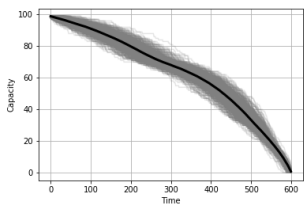

Figure 3: Simulated Capacity

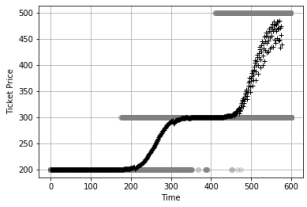

Figure 4: Simulated Ticket Prices

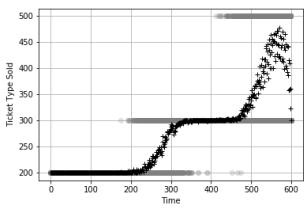

Figure 5: Simulated Ticket Sales

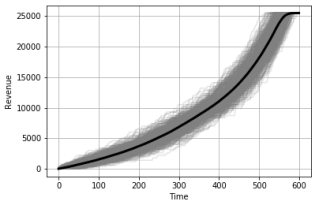

Figure 6: Simulated Revenue

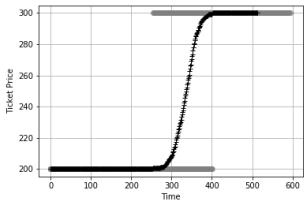

Figure 8: Simulated Ticket Prices

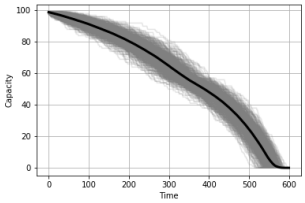

Figure 7: Simulated Capacity

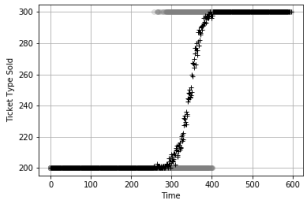

Figure 9: Simulated Ticket Sales

when the next time/state combination has a lower price. The penalty is simply multiplying the future expected reward by either 1 if the price is the same or higher and by 0 when the price is lower in the future state. This penalty will make sure that the price will not go down, since the future expected rewards have a big impact on determining the maximum expected value.

The impact of this penalty is that the total expected revenue will become lower, namely 25.499,75. When the optimal policy is inspected it becomes clear that the ticket price 500 doesn't occur at any point. The reason for this is that at time 600 and capacity 1 the optimal action is a price of 300. Any point before this with a higher price will receive the penalty as described earlier. The same simulation of section 3 is repeated with the new penalty. The result is shown in figures 6, 7, 8, and 9.