Resit Project 2

Jan van Ruijven 2006698

January 26, 2024

Question 1

1

The first greedy strategy is to go for streets whenever the odds seem adequete. An overview of this algorithm can be find in the appendix at algorithm 1. The second greedy algorithm is to go for combinations of three when there are two of the same dices. An overview of this algorithm can be found in the appendix at algorithm 2. The final algorithm consist of both the combination algorithm and the streets algorithm, by applying them both in that particular order. Important to note is that the dices are always sorted from low to high.

2

In this game there are $6 * 6 * 6 = 216$ possible states the game can be in and in each state the same list of 8 actions is available:

Throw first dice

Throw second dice

Throw last dice

Throw first and second dice

Throw first and last dice

Throw second and last dice

Throw all three dices

Throw no dice (stop the game)

A Q table is produced with 216 states and initialized with 8 zero's (one for each action). The reward for each action is equal to the change that is created after the action. So if the actions leads to an improvement the reward is positive and vice versa. This also means that the last action (doing nothing) results in a reward of 0.

3

Because each throw is independed of that what has happend before, gamma is set to one, so future rewards are just as important as rewards in the beginning of the game. For alpha, a relatively low value has to be chosen for proper learning (0.01 in this case), this is due to to fact that we are dealing with a game where the statistical average of a decicion is very important. If alpha would be set to a high value, we might just end up with a Q table with values that are only high by change, not because they are consitently outperforming the other options. Epsilon is set to 0.1, the algorithm should try new actions from time to time, but overall it is able to find the most suitable action, even when epsilon is very close to 0. This is due to the fact that the starting values are equal to 0, so taking wrong decicions will lead to negative values in the Q table, therefore the actions with a reward of 0 associated, will eventually be chosen.

4

After applying the greedy algorithms and letting the Q learning algorithm learn over 1.000.000 throws. The Q learning is able to outperform all of the greedy stategies. An overview of the results can be found in table 1 in the appendix.

5

The agent goes for streets whenever there are two dices with a difference of one and there are no combinations between dices. Whenever there are no combinations and no options for streets, it rethrows the highest and lowest number, so with dices [1, 4, 6], the first and last dice are thrown. This seems to be very a good option because it is able to get a streets afterwards in the example found. When there is a combination of two dices it throws for the third equal dice. In this case we are dealing with a dice-game, but simmilar scenarios can happen in economical games. Where there are certain probabilites applied to taking certain actions, the probabilites are the same each time the action is taken and a reward is given.

Question 2

1

Take the following instance weights = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11] and capacity = 13. The greedy algorithm would give the following solution:

S1 = 11 + 1 + 2

S2 = 10 + 3

S3 = 9 + 4 + 5

S4 = 8 + 6

S5 = 7 < 13

Which has a value of four in total. This is not optimal, since a value of five can be achieved in the following manner:

S1 = 11 + 2

S2 = 10 + 3

S3 = 9 + 4

S4 = 8 + 5

S5 = 6 + 1 + 7

2

To make the instance hard enough for gurobi to take atleast 5 seconds, the following instances are generated:

Number of items: 200

Capacity: 200

Weights: Uniformly distributed intergers ranging from 1 to 200

3

The stopping criterion is the amount of items that we want to put in a subset. Since gurobi is able to outperform the greedy (with 5 seconds time limit), it seems logical that the stopping criterion should be met quite soon. Letting half of the work be done by the greedy algorithm will not give better results. The question is how much exactly. By keeping track of the number of large and small items, where large corresponds to greater than 150 and small means smaller than 50, we can create a Q table where each state correspond to the quarter percentile the numer of large/small items fall in (binomaly distributed). With quarter percentiles, there are 4 states the number of large items can be in and the same hold for the small items. This means there are $4 * 4 = 16$ states in total. In each state we can take an action which corresponds to the number of items to process in the greedy part. The amount chosen is $160 + a * 10$, where a corresponds to the action taken (1, 2, 3 or 4), so four actions in total. For completely diffrent instances, the difrent types of actions

could be found and the definition of a large or small item could be changed aswell. The alpha by which the reward is multiplied, corresponding to each action in each state can be found in equation 1.

$$\alpha = 1/N_s^a \tag{1}$$

Here, N_s^a corresponds to the number of times the action a is taken in state s. This would imply that each the value in the Q tables will slowly converge to a limit. This implies that the actions with the highest reward should perform the best in each particular state. The Q table is initialized with zeros.

4

The instances generated are completely similar to the ones described in question 2.

5

Gamma is equal to 1, there is not a future reward to be taking into account. Alpha is slowly decreasing for each time an action is taken, as shown in question 2. Epsilon is set to 0.1, such that each action is tried atleast once.

6

An overview of the average results obtained by both the greedy, ILP solver and Hyper heuristic can be found in table 2 in the appendix. The average is calculated over 100 instances, the hyper-heuristic does give a small performance increase of the ILP, increasing the training time could bring better results.

7

The agent prefers to apply the greedy to 30-20 items in most states, however, when there are many large items and a small amount of small items, it seems to prefer only 10 items decided by the greedy. This seems logical, since it is harder to make a perferct fit when there are many large items and not so much small items. The overall benifit this hyper-heuristic has that it is able to bring a performance boost to the ILP-solver when applied correctly and because the greedy algorithm is so fast (especially for only 10 items), it does not really affect the total execution time.

Algorithms

Algorithm 1: Greedy steets Algorithm

Data: $D_i \quad i = 1, 2, 3 \quad C_j = [D_2 - D_1, D_2 - D_3]$
if $\sum_{j=1}^{2} C_j <= 1$ **then**
| Do nothing. We have combinations.
end
else if $C = [2, 1]$ **then**
| Rethrow first dice
end
else if $C = [1, 2]$ **then**
| Rethrow last dice
end

Algorithm 2: Greedy Combinations Algorithm

Data: $D_i \quad i = 1, 2, 3 \quad C_j = [Dice_2 - Dice_1, Dice_2 - Dice_3]$
if $\sum_{j=1}^{2} C_j <= 1$ **then**
| Do nothing. We have combinations.
end
else if $C[0] = 0$ **then**
| Rethrow last dice
end
else if $C[1] = [0]$ **then**
| Rethrow first dice
end

Tables

Algorithm	**Average Value**
Only combos	22.61
First combos then streets	25.7
Only streets	24.44
Q testing	34.92

Table 1: Results question 1

Scenario	**Average Value**
Q testing	86.74
Full greedy	80.81
ILP Solver	86.2

Table 2: Results question 2