[INFO-F409] Learning Dynamics
Third assignment

BUI QUANG PHUONG Quang Linh
Université libre de Bruxelles - ULB ID : 000427796
MA1 Computer Sciences

December 2018

Contents

1 First game: N-Armed Bandit 2
1.1 Exercise 1 . . . 2
1.1.1 Average reward according to the algorithm . . . 2
1.1.2 Evolution of the Q-value for each arm . . . 3
1.1.3 Frequency of each action being selected . . . 6
1.2 Exercise 2 . . . 6
1.2.1 Average reward according to the algorithm . . . 6
1.2.2 Evolution of the Q-value for each arm . . . 7
1.2.3 Frequency of each action being selected . . . 9
1.3 Exercise 3 . . . 10
1.3.1 Average reward according to the algorithm . . . 10
1.3.2 Evolution of the Q-value for each arm . . . 10
1.3.3 Frequency of each action being selected . . . 12
1.4 Conclusion . . . 13

2 Second game: Windy Gridworld 14
2.1 Plot of an execution of Windy Gridworld . . . 14
2.2 Best actions and path taken . . . 14
2.2.1 Observation . . . 14
2.3 Statistics . . . 14
2.3.1 Observation . . . 15
2.4 Question 1: Changing ϵ . . . 16
2.4.1 Result . . . 16
2.5 Question 2: Changing γ . . . 16
2.5.1 Result . . . 16

3 Third game: Graphical Coordination Game 17
3.1 Payoff matrix . . . 17
3.2 Convergence time . . . 17
3.2.1 What is the role of α for the Win-Stay-Lose-probabilistic-Shift algorithm? . . . 18
3.2.2 How is the convergence time affected by the value of α? . . . 18
3.2.3 Which α value(s) result in the fastest convergence time for your setting? . . . 19
3.2.4 For which α values the algorithm fails to converge? . . . 19
3.2.5 How is the convergence time affected by the number of actions? . . . 19

1 First game: N-Armed Bandit

Context

The first game that we discuss is the N-Armed Bandit game. In this problem, the player has the choice between 4 actions (i.e 4 arms) which give for each action a different reward calculated according to the reward distribution table presented in Figure 1 [1]. The rewards are subject to noise according to a normal probability distribution with mean Q^*_{ai} and standard deviation σ_i. The results presented for this game in the document are based on **1000 simulations**.

Action	$Q^*_{a_i}$	σ_i
action #1	2.4	0.9
action #2	1.3	2
action #3	1	0.4
action #4	2.2	0.6

Figure 1 – Reward distributions for each action

1.1 Exercise 1

Statement

Run the algorithms and provide the requested plots. Comment the results. Which algorithms arrive close to the best arm after 1000 iterations? Which one arrives there faster? Explain your findings.

1.1.1 Average reward according to the algorithm

The Figure 2 shows the average reward earned for each algorithm used. The figure is proving that picking the action randomly is the worst choice to optimise the reward while using the ϵ-Greedy strategy with an ϵ value of 0 is the best choice. Indeed, $\epsilon = 0$ means that the player is **always** selecting the arm with the potential best reward. Here, the best arm has a reward of 2.4. The average reward by using this strategy is then approaching this value.

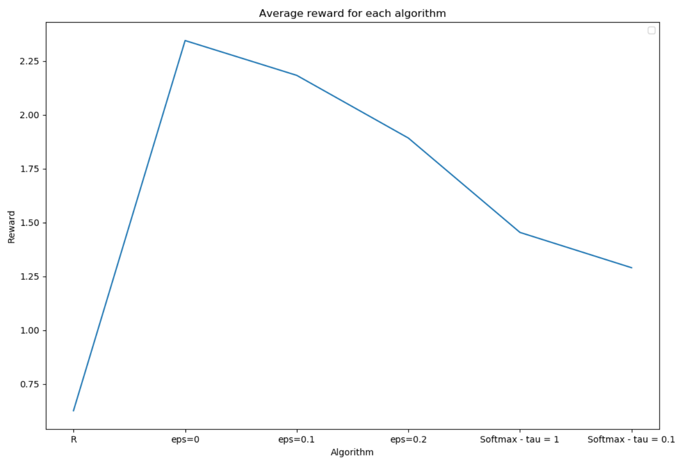

Figure 2 – Average reward for each algorithm

[1] My ULB enrolment number is 000427796 and is then ending by a 6. The table selected is then the Table 4 in the statement

1.1.2 Evolution of the Q-value for each arm

The evolution of the Q matrix along time by using the different strategy are shown in the Figure 3 for the first arm, Figure 4 for the second arm, Figure 5 for third arm and Figure 6 for the fourth arm. We remind that these results are obtained by running the simulation 1000 times. What we can observe from the results is that the general behaviour of each algorithm for each arm is that the Q-values are converging to the average payoff[2] of the arm. Moreover, we see that the evolution of the Q-values for the first arm is quite more steady than others. This can be explained because all the algorithms tend to take the optimal action, i.e the action with the best average payoff (action 1 with a payoff of 2.4). Now let's analyse the result for each arm one by one.

Arm 1: In this case, no matter which strategy you chose, all the Q-values seem to converge to the average reward of 2.4. A stationary state can be found after 100-150 simulations. At this time the values variate less and begin to be steady. As explained before, this trend is due to the fact that the first arm is in average the best action to pick.

Arm 2, 3 and 4: Concerning the other arms, what we observe is that the Softmax algorithm with $\tau = 0.1$ has a constant null estimation of Q-values. To explain such a quirk, we need to explain what does τ means in the Softmax algorithm. τ is defined by what we call the *computational temperature.* This feature means that for a high temperature ($\rightarrow \infty$), all actions have nearly the same probability while for a low temperature ($\rightarrow 0$), the action with the best average/expected reward has nearly a probability of 1 to be selected. In our case, when $\tau = 0.1$, it means that the temperature is really low. Therefore, the action with the best expected payoff, i.e action 1, has a probability of 1. Thus, it explains why for every other arms than arm 1, the evaluation of the Q-values for the Softmax algorithm with $tau = 0.1$ is equals to zero. Concerning the other algorithms, their behaviour is quite similar to Arm 1, the Q-values are evolving such that they tend to converge to the average reward of the arm despite they are more variables. Their stationary state is then later than for Arm 1 (i.e 200-250 simulations for Arm 2, 400-450 simulations for Arm 3 and 500-550 for Arm 4).

In summary, what are the differences between each algorithm ? The algorithms are separated into two categories : the random and the greedy actions. The first one is simply selecting the action for each simulation randomly. The efficiency of this algorithm is somehow bad. It would chose the optimal action with a probability of 1/4. On the second hand, we have the greedy actions. The meaning of this type of actions is that you are **exploiting** your current knowledge of the values of the actions while if you are playing randomly, then non-greedy, you are **exploring** the environment. The goal of playing greedy is to improve along time your action's choice from your previous actions. Thus, exploitation is the good method to adopt to optimize your choice. However, one drawback is that when it explores it chooses equally among all actions. This means that it is as likely to choose the action with the worst expected reward as it is to choose the action with the best expected reward. In the case where the worst actions are very bad, this may be unsatisfactory. The solution of this is given by the Softmax algorithm which gives a higher probability to the best actions and a lower probability to the bad actions. This algorithm is characterized by τ, the computational temperature. As said before, this feature spreads the probabilities equally or not according to the value of τ. The probabilities are computed following this formula :

$$\frac{e^{\frac{Q_t(a)}{\tau}}}{\sum_{b=1}^{n} e^{\frac{Q_t(a)}{\tau}}}$$

where a is the action to play, n the number of actions and τ the computational temperature.

[2]see Figure 1

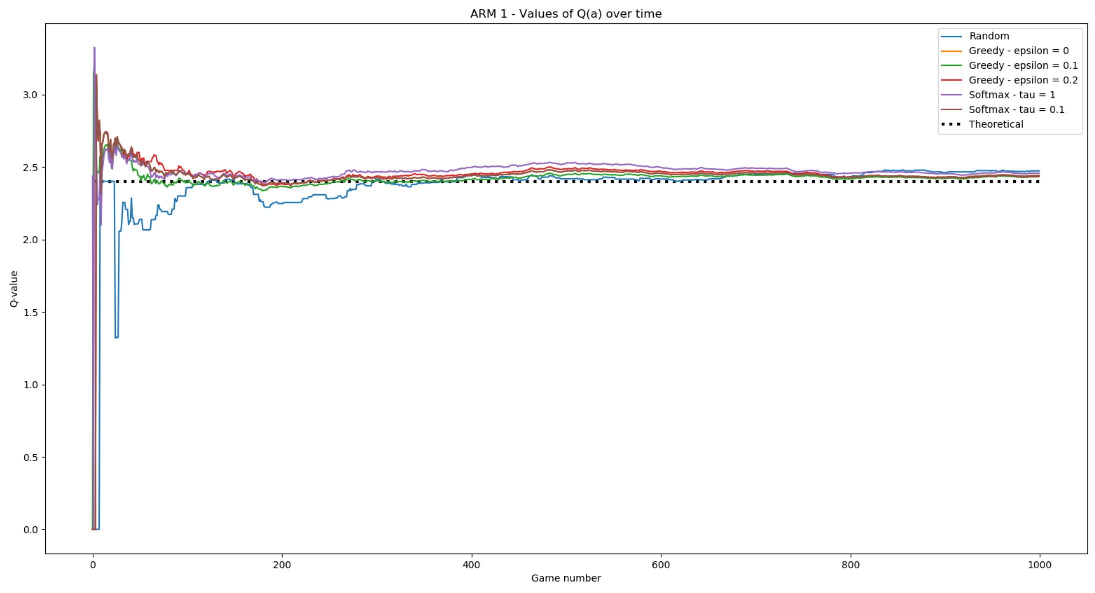

Figure 3 – Evolution of the Q-value for Arm 1

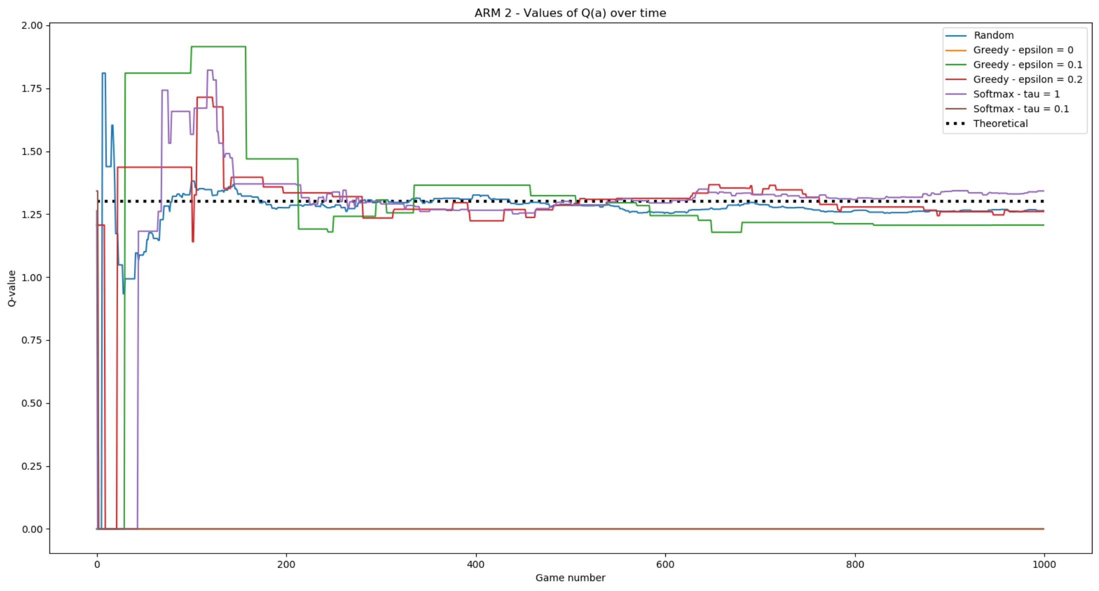

Figure 4 – Evolution of the Q-value for Arm 2

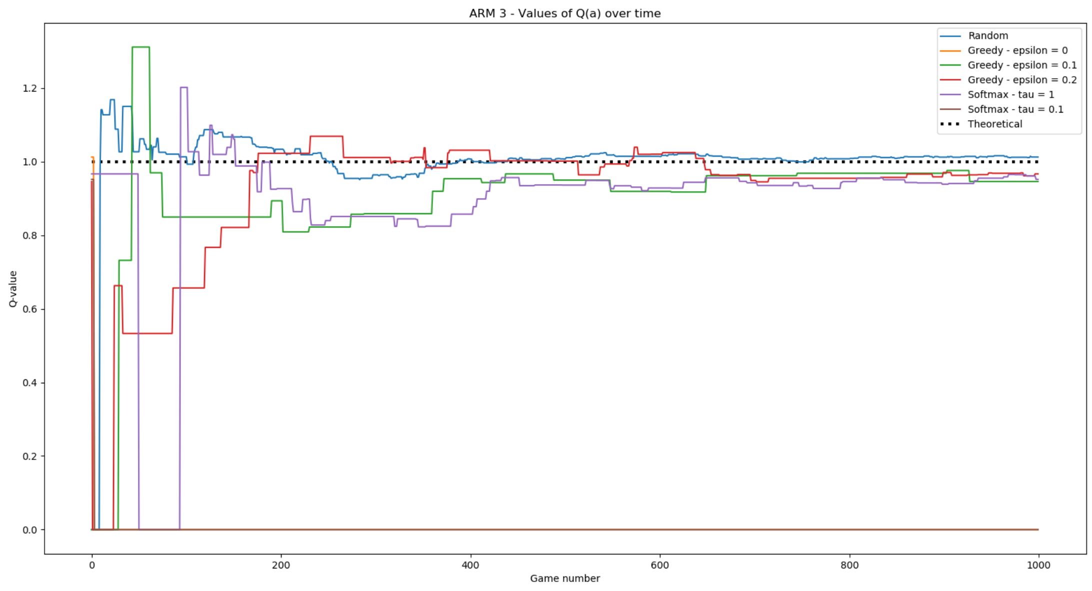

Figure 5 – Evolution of the Q-value for Arm 3

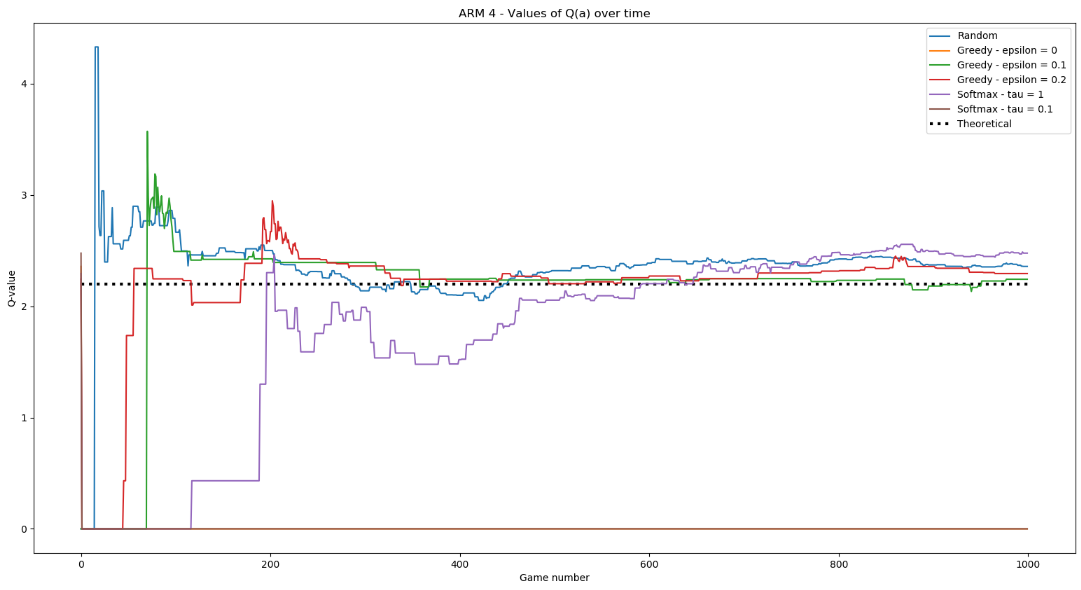

Figure 6 – Evolution of the Q-value for Arm 4

1.1.3 Frequency of each action being selected

The number of times that an action is selected on 1000 simulations for every algorithm is shown in the Figure 7. In the first row from left to right, the algorithms represented are Random, Greedy with $\epsilon = 0$, Greedy with $\epsilon = 0.1$, in the second row, Greedy with $\epsilon = 0.2$, Softmax with $\tau = 1$ and Softmax with $\tau = 0.1$. For the random strategy, we observe that the selecting rate of each action is somehow equivalent while for the other strategies, the first action is hardly dominating the other actions. In summary, this is mainly due to the fact that all the strategies except the random one is tending to select the best action. This is explained in details in the previous section.

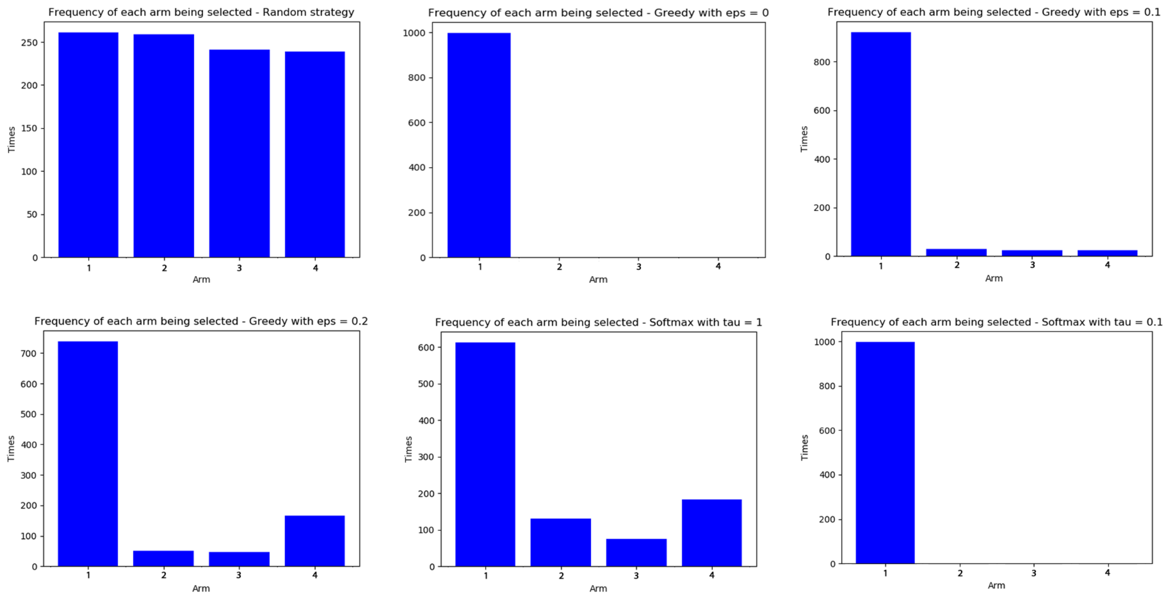

Figure 7 – Frequency of each arm being selected for 1000 time steps

1.2 Exercise 2

Statement

Re-run Exercise 1 doubling the standard deviations of each arm, and comment the plots and results. Discuss the results. Is this problem harder or easier to learn? Does the performance of the algorithms change significantly? Which of the above performs best now?

1.2.1 Average reward according to the algorithm

The graph obtained for the average reward for each algorithm has the same appearance than Figure 2. Modifying the value of the standard deviation has apparently no real impact on the graph.

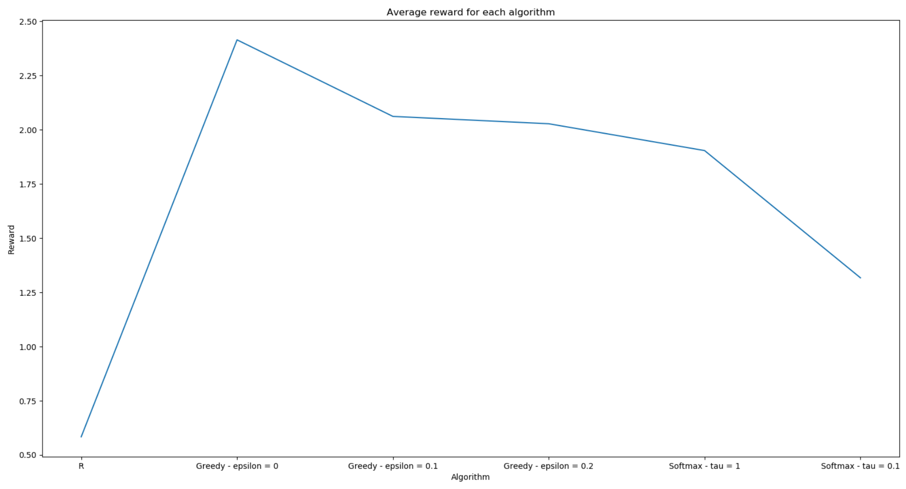

Figure 8 – Average reward for each algorithm with 2σ

1.2.2 Evolution of the Q-value for each arm

Doubling the standard deviation means that the noise is 2 times bigger. In theory, the results at the end of the 1000 simulations should be the same because the mean has not been modified. Nevertheless, at the beginning the values should be more noisy and should take more time to find the mean value. It's exactly what we can observe from the results presented in the figures. At the end of the 1000 simulations, the values are somehow the same, they are all converging to the corresponding expected reward of the arm. However, the stationary state is reached a bit later compared to the figures obtained before doubling the value of σ. For instance, the evolution of the Q-value for the first arm is really noisy at the beginning and is reaching a stationary state near the 350-400th simulation while it was reached before towards 100-150 simulations.

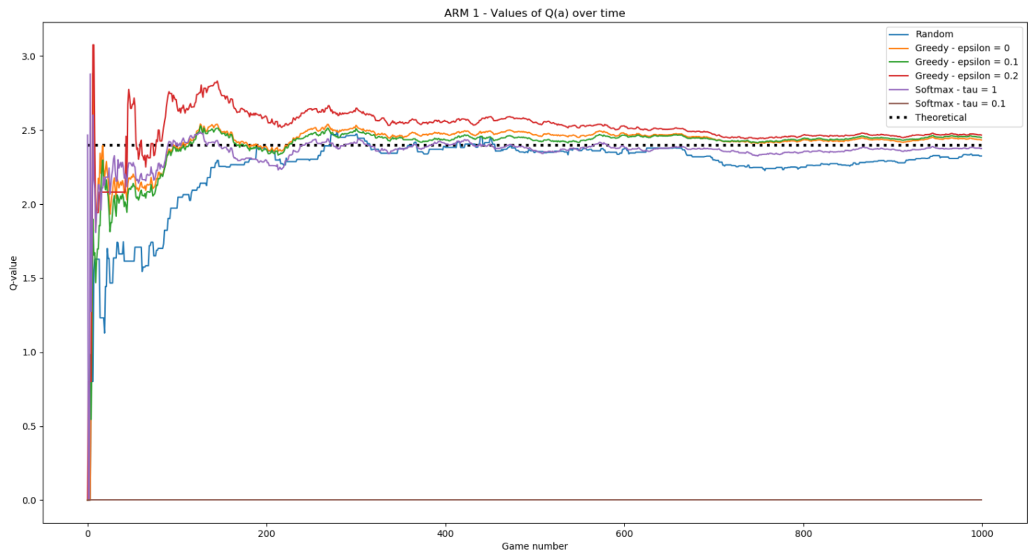

Figure 9 – Evolution of the Q-value for Arm 1 with 2σ

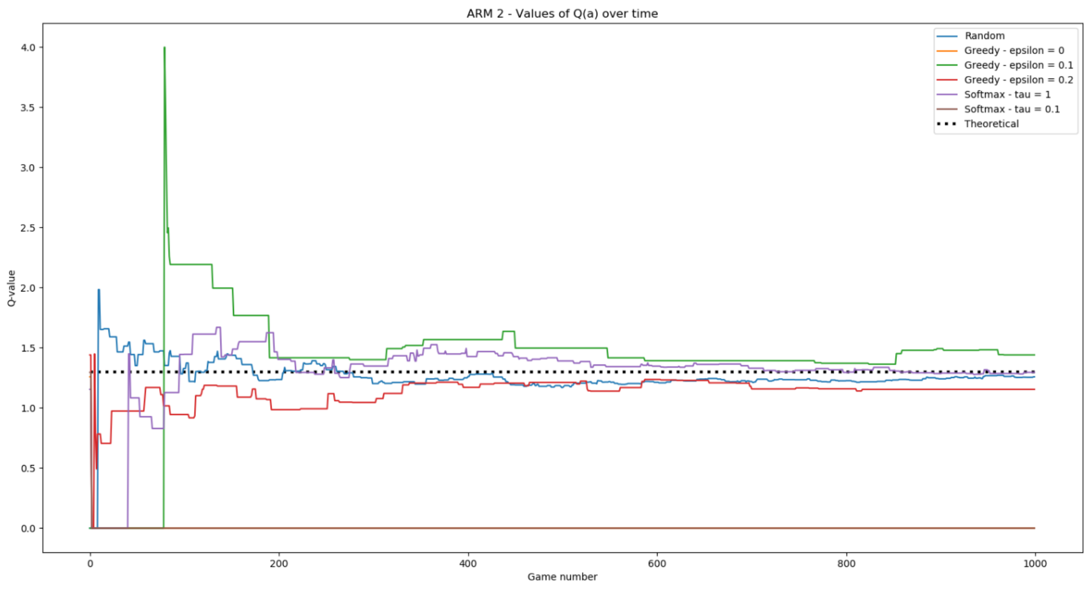

Figure 10 – Evolution of the Q-value for Arm 2 with 2σ

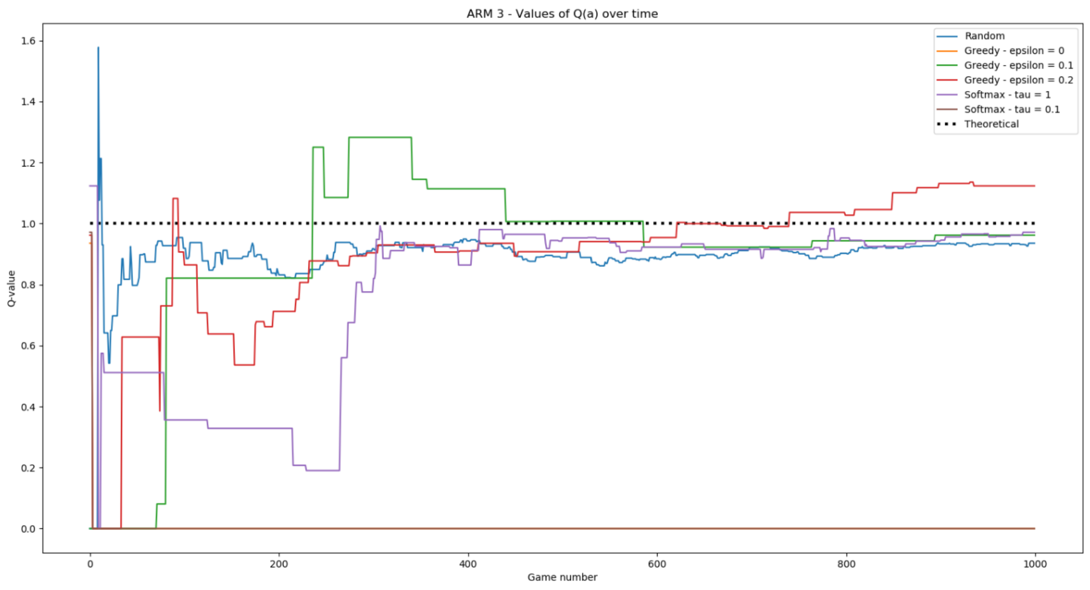

Figure 11 – Evolution of the Q-value for Arm 3 with 2σ

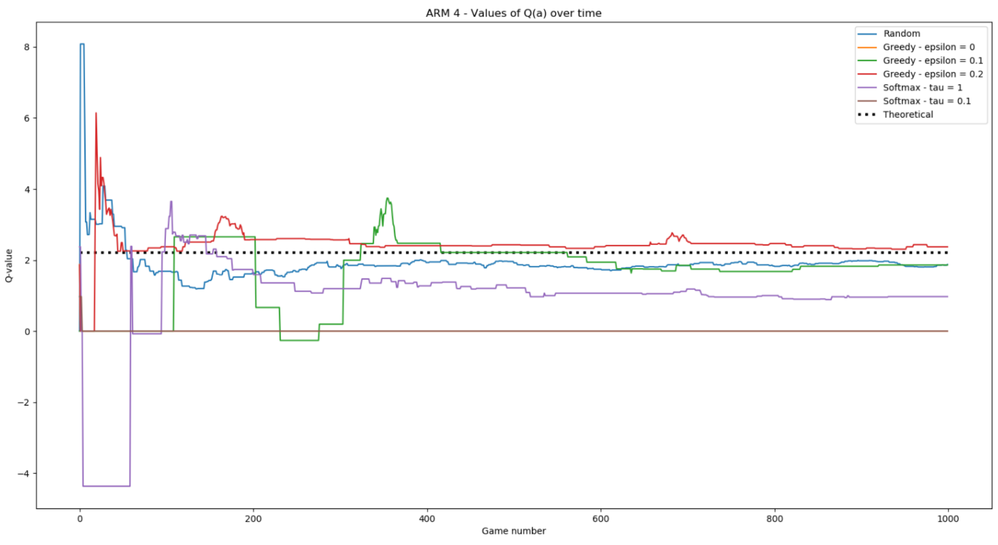

Figure 12 – Evolution of the Q-value for Arm 4 with 2σ

1.2.3 Frequency of each action being selected

The general observation in these results is that they really look the same as Figure 7. However, we can see that for the frequencies of the greedy strategies > 0 the actions 2, 3 and 4 are being selected a bit more than Figure 7. This shows one more time that the noise is more present which means that it takes more simulations to reach the average reward. To summarize, doubling the standard deviation is decreasing the simplicity to learn because of the noise augmentation despite the performance is not changing significantly. Nevertheless, keeping the initial standard deviation is still the best choice to perform the best.

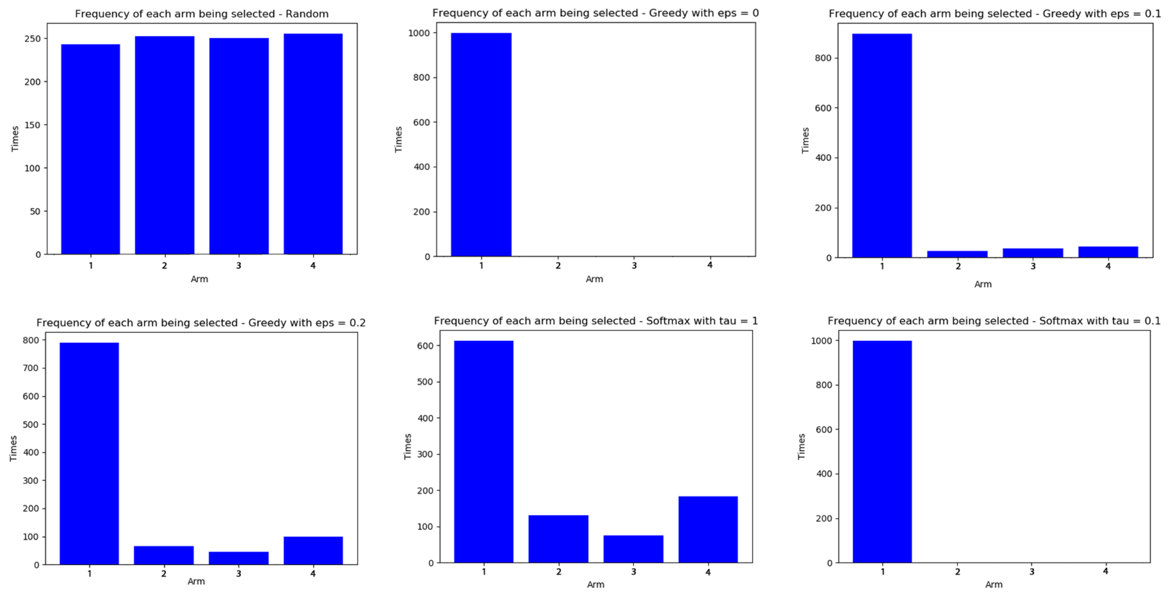

Figure 13 – Frequency of each arm being selected for 1000 time steps with 2σ

1.3 Exercise 3

Statement

Re-run exercise 1 with two additional algorithms, and plot their results. The new algorithms have a time-varying parameter, which depends on the iteration number $t = 1, \ldots, 1000$. Discuss the results. Are these algorithms better than the ones with a fixed parameter?

1.3.1 Average reward according to the algorithm

In this section, we are using two new algorithms to run our simulations. These two algorithms are using time-varying parameters :

- ϵ-Greedy with parameter $\epsilon = \frac{1}{\sqrt{t}}$
- Softmax with parameter $\tau = 4 \cdot \frac{1000-t}{1000}$

such that t is the current simulation time.

The two additional algorithms, even the ϵ-Greedy with $\epsilon = \frac{1}{\sqrt{t}}$ or the softmax strategy with $\tau = 4 \cdot \frac{1000-t}{1000}$ seem to be really efficient. Indeed, their average reward is approaching a lot the optimal expected reward.

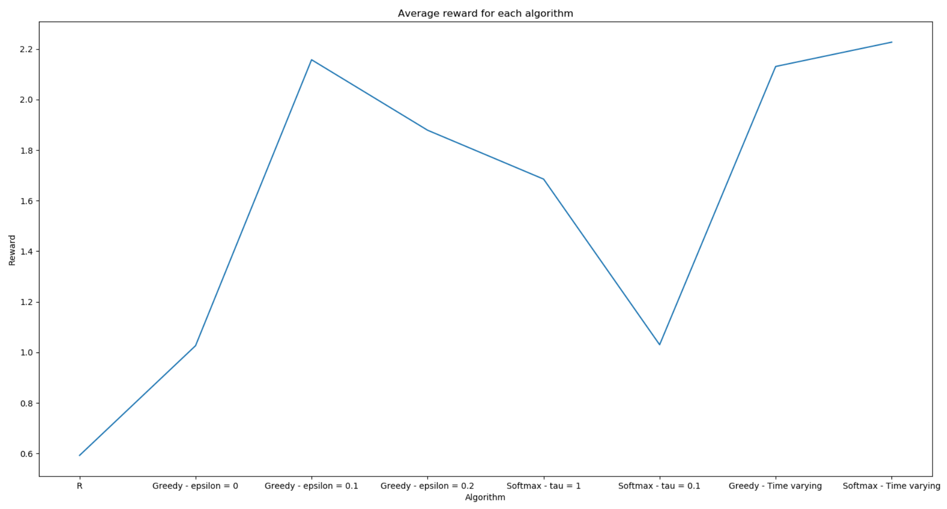

Figure 14 – Average reward for each algorithm

1.3.2 Evolution of the Q-value for each arm

In the following figures, what we can observe from the new algorithms is that the result are quite similar than using these algorithms with fixed parameter. In general, they look somehow steady and their noise level, i.e exploration time, is not really huge. Indeed, their stationary level for the first arm for instance is around 100 rounds as using the fixed parameter. However, what we should observe is that the time exploration is decreasing according to the time step simulation which means that the noise should be a little more present at the beginning than with a fixed parameter. Indeed, the formula which is doing the variation for ϵ means that ϵ is decreasing from 1 to 0.03 in 1000 rounds while τ is decreasing from 4 to 0 in after all the simulations. We can remark that the decrease is going way much faster for the softmax algorithm with τ.

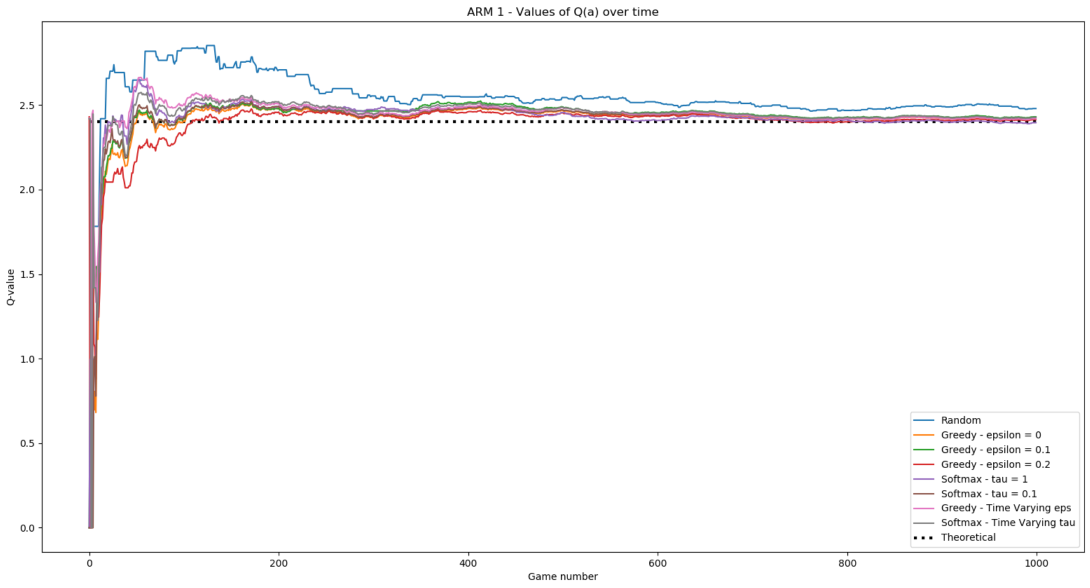

Figure 15 – Evolution of the Q-value for Arm 1

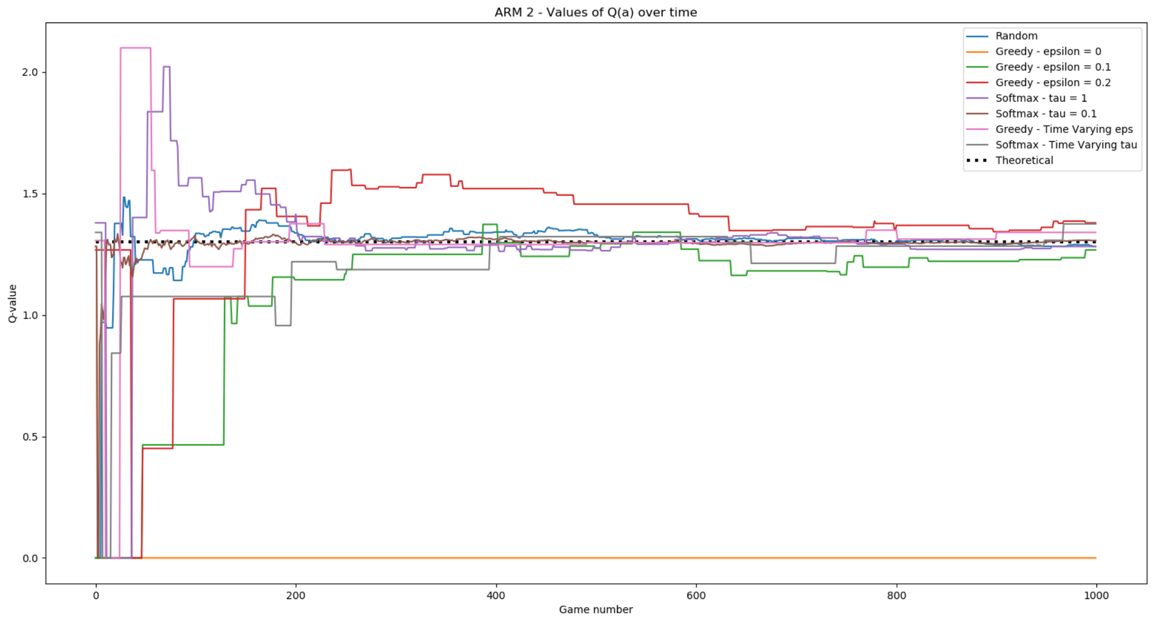

Figure 16 – Evolution of the Q-value for Arm 2

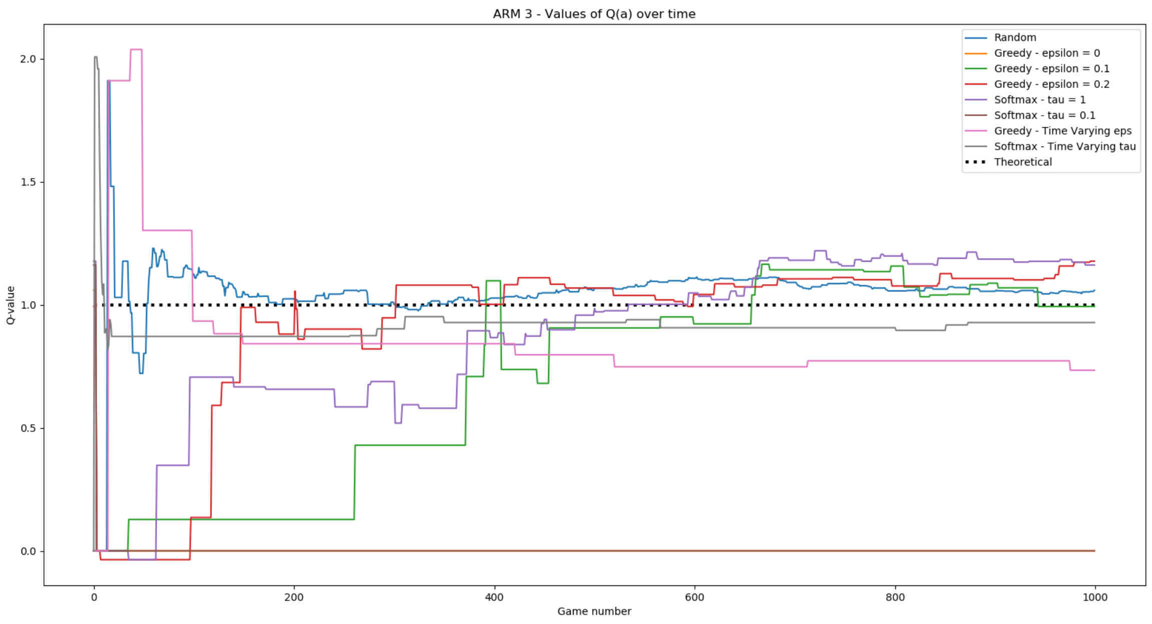

Figure 17 – Evolution of the Q-value for Arm 3

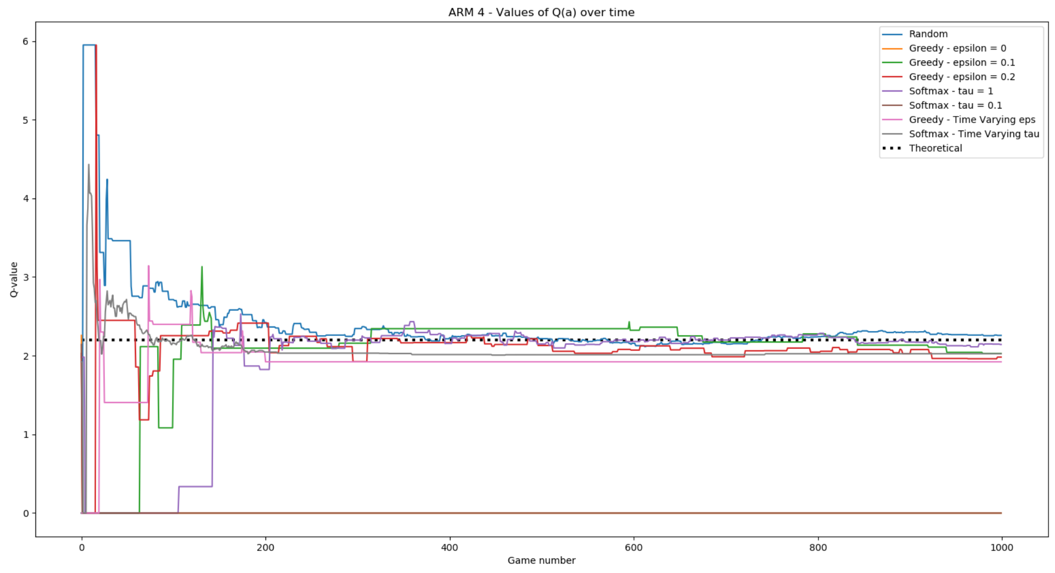

Figure 18 – Evolution of the Q-value for Arm 4

1.3.3 Frequency of each action being selected

The Figure 19 is proving that the exploration is more present for the case of Softmax using time varying parameter comapared to a fixed parameter $\tau = 0.1$ where the exploration was null. On the other hand, for the ϵ-greedy with $\epsilon(t)$, it looks like an average of the frequency computed for fixed parameter $\epsilon = 0, 0.1$ and 0.2 which is normal because in 1000 simulations, the algorithm is covering all the values between 1 and 0.03. Finally, what we can say about these 2 new algorithms is that they are quite good and even looks a bit more robust than the ones with a fixed parameter.

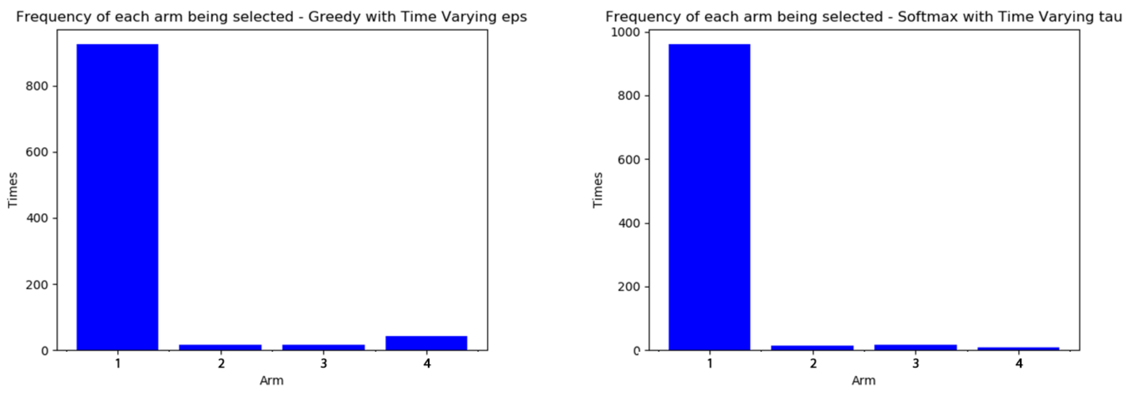

Figure 19 – Frequency of each arm being selected using greedy $\epsilon(t)$ and softmax $\tau(t)$ for 1000 time steps

1.4 Conclusion

After observing the results with multiple conditions, what we can conclude is that the best results are obtained with an exploration phase. Chosing an action randomly is never a good solution whether the condition. The softmax and ϵ-greedy algorithms are then the best method to adopt.

2 Second game: Windy Gridworld

2.1 Plot of an execution of Windy Gridworld

2.2 Best actions and path taken

An execution of the Q-learning algorithm for the Windy Gridworld problem is presented in the Figure 20. In the figure, the start point was $(3, 0)$ and the goal point is represented by a red point and is $(9, 3)$ in this case. Each cell is showing its best action. The red arrows are representing the path taken to reach the goal. In this run, $\alpha = 0.1$, $\gamma = 0.9$ and $\epsilon = 0.2$. The Q-values have been computed over 5000 episodes.

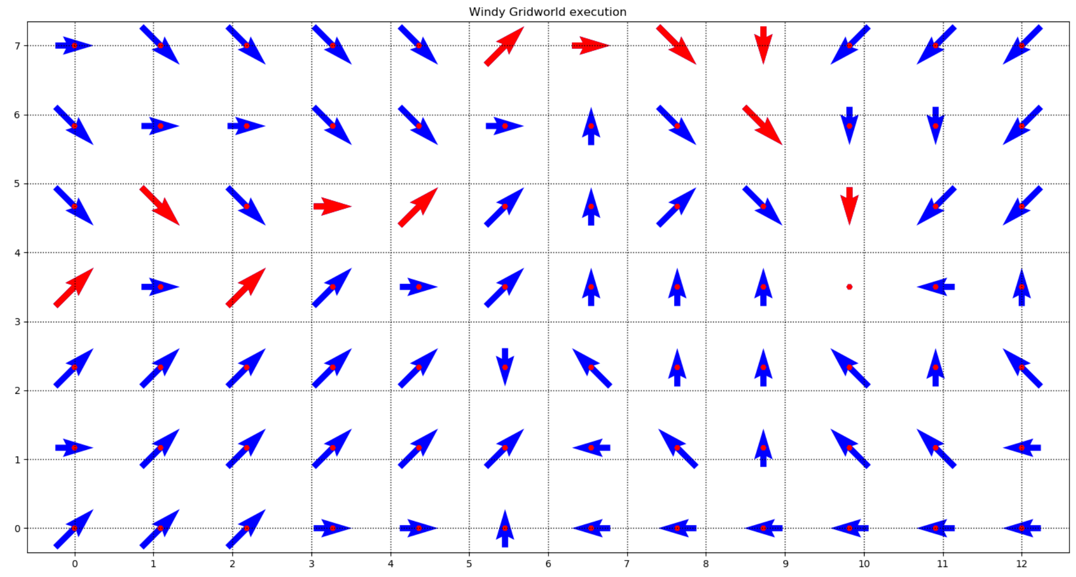

Figure 20 – Execution of Windy Gridworld

2.2.1 Observation

The first observation is that generally all the best actions chosen by each cell is not really accurate, even they are mostly converging to the goal point, we can see that the exploration is kinda weak such that some arrows are completely not pointing the goal even they're beside the goal point. For instance, the arrow left to the goal is pointing to the north while the goal is just one cell to its right. This is explained by the small value chosen for ϵ.

2.3 Statistics

The two statistics that we are interested of are the total collected reward per episode and the number of steps to reach the goal per episode. The first one is presented in the Figure 21 and the second one in the Figure 22.

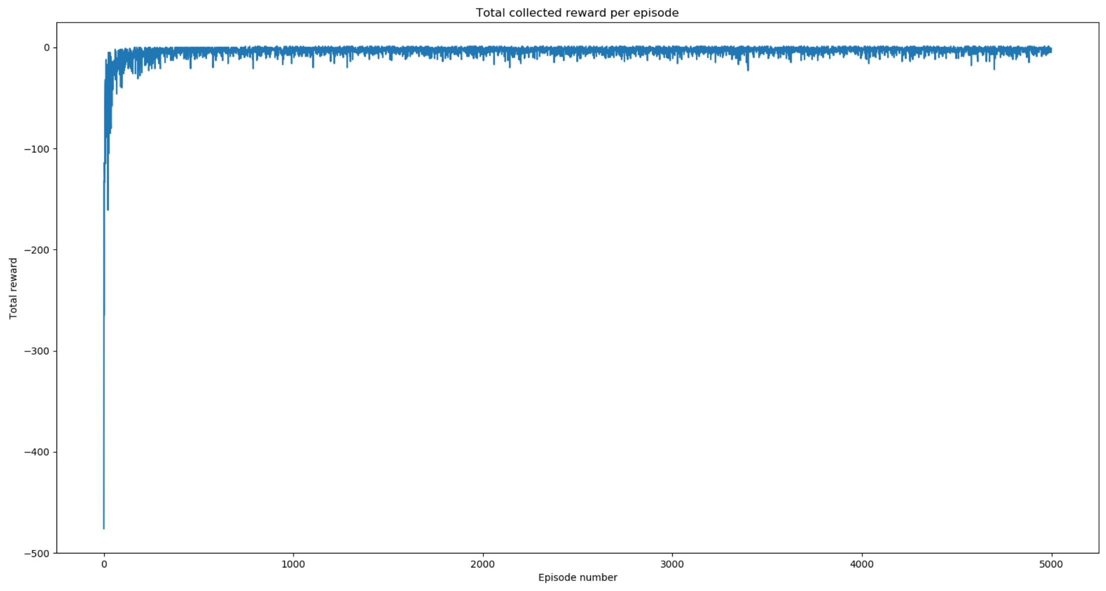

Figure 21 – Total reward per episode

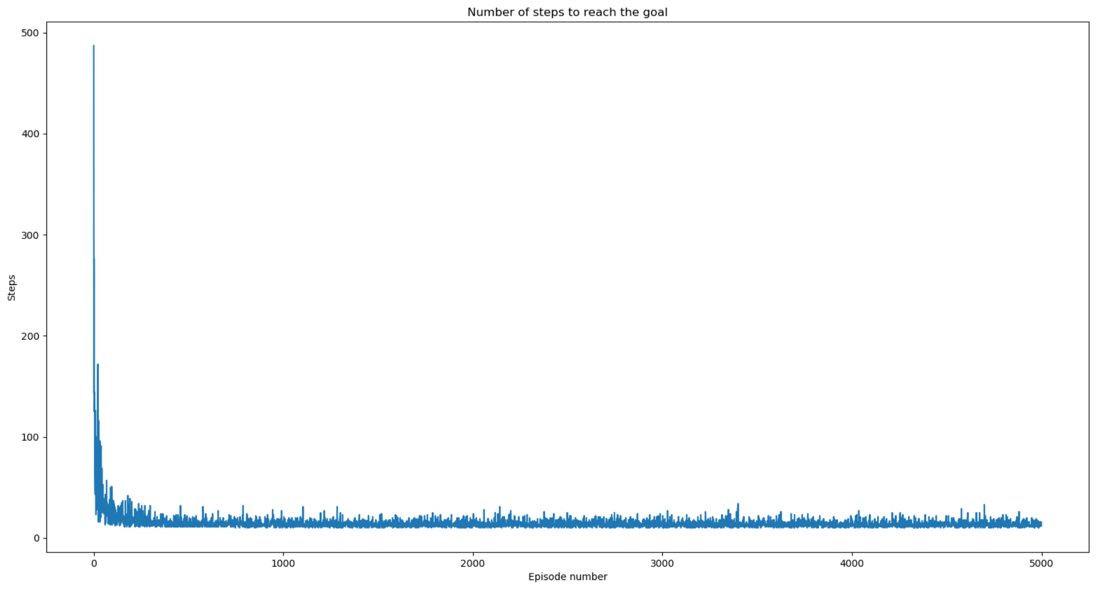

Figure 22 – Number of steps to reach the goal per episode

2.3.1 Observation

What we first observe is that the total reward and the number of steps to reach the goal is proportional. Indeed, the first steps are taking a long time to reach the goal (e.g 500 for the first one) and impacts then the total reward because each step is decreasing the reward of 1 while the goal isn't reached (reward is increased by 10 when reached). Nevertheless, the learning done at the beginning seems to be good because after 250-300 episodes the steps number to find the goal is stationary on an average of 15-20 steps.

2.4 Question 1: Changing ϵ

Statement

How will the learning process change if we make $\epsilon = 0$, while keeping the other parameters the same?

2.4.1 Result

As saw in the previous game, ϵ is representing the rate of exploration. By putting ϵ to zero, we define that no exploration is done, only exploitation which means that from the beginning to the end, the same action is always picked. However, what we can observe from the results shown in Figure 23 is that some exploration are done at the beginning for 400-500 episodes. In fact, what is happening is that the ϵ-greedy algorithm is always taking the maximum value of $Q(s, a)$. As saw before, when the goal isn't reached, the reward will be negative. Thereby, by initializing the Q-values to all zeros, the ϵ-greedy algorithm will pick randomly an action which its Q-value equals zero. The exploration will end when all the actions have been explored once. At this time, the maximum Q-value will be the same for every other rounds.

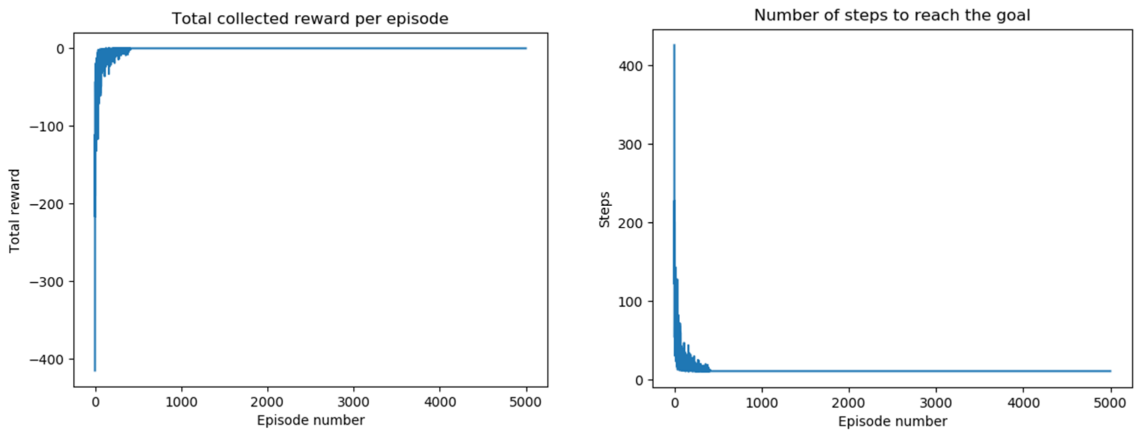

Figure 23 – Results with $\epsilon = 0$

2.5 Question 2: Changing γ

Statement

How will the learning process change if we make $\gamma = 1$, while keeping the other parameters the same?

2.5.1 Result

γ is the discount factor which means that γ is valuing rewards received earlier higher than those received later (reflecting the value of a "good start"). By putting γ to 1, it means that the future values will have the same impact weight than the first received values. The learning is then taking in account fairly the current and the future values.

3 Third game: Graphical Coordination Game

3.1 Payoff matrix

First of all, let's have a look on the payoff matrix considering 2 and 3 actions. The reward for each player is symmetric. Indeed, an agent is rewarded only if the other player plays the same action than him, otherwise whether the action played, if the other agent plays another action, the payoff of both player will be zero. In other words, the agents have to be coordinated to win.

Agent 2 / Agent 1	Action 1	Action 2
Action 1	1	0
Action 2	0	1

Agent 2 / Agent 1	Action 1	Action 2	Action 3
Action 1	1	0	0
Action 2	0	1	0
Action 3	0	0	1

Table 1 – Payoff matrices for 2 and 3 actions game

3.2 Convergence time

The average value of the convergence time according to the value of α is given in Table 2 for 2 actions and Table 3 for 3 actions. Note that we use arbitrary 50000 as value of T_{max} which means that it never converges when exceeding this threshold value. Thereby, in the Figure 24, an average convergence time of 50000 means a non-converging game for this α value. The Figure 25 shows the same graph as Figure 24 without the non-converging α.

α	Convergence time (average)
0	Never converge
0.1	8664
0.2	4566
0.3	3780
0.4	2485
0.5	2173
0.6	2026
0.7	2098
0.8	2265
0.9	3329
1	Never converge

Table 2 – Average convergence time for 2 actions

α	Convergence time (average)
0	Never converge
0.1	11936
0.2	5201
0.3	3601
0.4	3208
0.5	2787
0.6	2326
0.7	2270
0.8	2625
0.9	3713
1	Never converge

Table 3 – Average convergence time for 3 actions

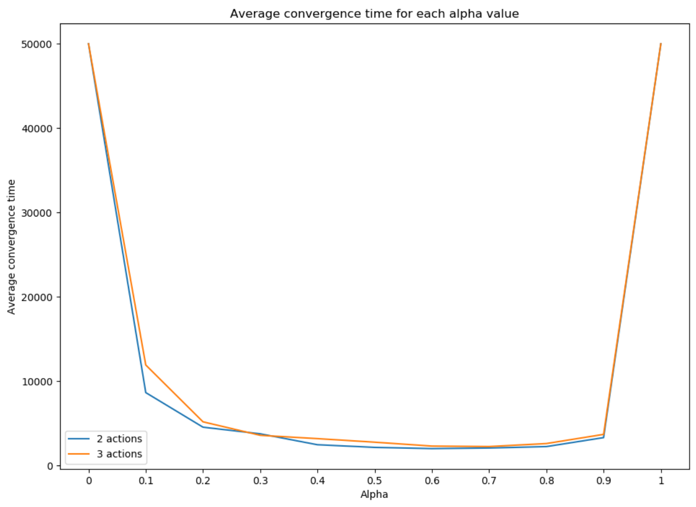

Figure 24 – Average convergence time for each α value

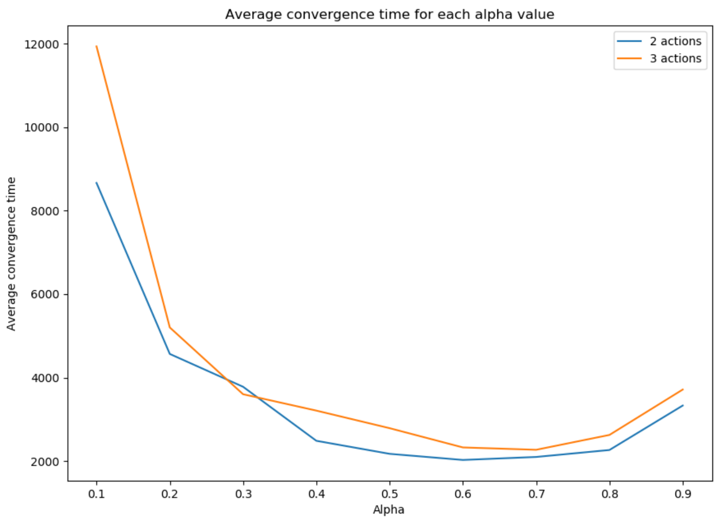

Figure 25 – Average convergence time for converging α value

3.2.1 What is the role of α for the Win-Stay-Lose-probabilistic-Shift algorithm?

The α is the lose-probabilistic-shift. The action change depends of this value of α. An agent will keep his action with a probability of 1-α. A small value of α means that an agent is flexible and will change quite often his action while a large value of α means that the agent will mostly keep playing his action.

3.2.2 How is the convergence time affected by the value of α?

As Figure 25 shows, if α? is too small, the agents will really often change their actions in every round which means that the actions of the agents keep changing. Getting coordinated becomes then more difficult. From $\alpha = 0.5$ to $\alpha = 0.8$, it becomes really fast to get the coordinated state (around 2000 rounds).

3.2.3 Which α value(s) result in the fastest convergence time for your setting?

Generally, the fastest convergence will be between $\alpha = 0.5$ and $\alpha = 0.8$. In this interval, the probability to change action is not to big and not to small. These values are then ideal to get the convergence quickly.

3.2.4 For which α values the algorithm fails to converge?

It exists 2 values of α which the algorithm will always fail to converge : $\alpha = 0$ and $\alpha = 1$. It is logic because in the first case ($\alpha = 0$), the agents will change their action in every round which means that if their basic configuration isn't respecting the convention, i.e the same action for every agent, it becomes impossible to get this convergence. Same logic for $\alpha = 1$ which means that the agents will never change their action. The only case where it converges is that the random action choice at first round is the same for every agent. At this time, it will converges immediately after 10 iterations. Nevertheless, this case is somehow impossible with that number of agents or really rare.

3.2.5 How is the convergence time affected by the number of actions?

Obviously, less is the number of actions, less the convergence time will be. It is a simple combinatoric problem which shows that exists more combination with more actions, thus it exists more possibilities when assigning a random action to each agent. Indeed, the numbers of combinations is :

$$C_n^k = \binom{n}{k} = \frac{n!}{(n-k)! \cdot k!}$$

where n is the number of actions and k the number of agents.

For 26 agents, it means that for 2 actions, it exists 325 combinations possible for the assignment of actions in the first iterations and 2600 for 3 actions. Thus, this difference is mostly marked for the first iterations but will of course also impact the sequel of iterations.