COMP767 - Experiences with the Q(sigma) algorithm in a stochastic environnement with risk

Vincent Antaki

March 4, 2017

Abstract

The Q(σ) algorithm is presented in [1] as an algorithm that unifies the classical reinforcement learning approaches In this homework, we study the effect of different functions to choose the value of sigma on the algorithm performance and compare them with more classical approaches (Qlearning, Expected Sarsa, Treebackup).

1 Environnement

Our environnement to evaluate our algorithms is called the WindyCliffWorld. Our agent move around a grid map with the objective to reach the goal tile. There are four different types of tiles : normal, wall, windy and cliff (we consider the start and goal to be attributes of tiles and not proper type). The agent cannot move in the direction of wall tiles; doing so would result in the agent hitting the wall and not moving. The same idea applies if the agent tries to move out of the grid in the Undefined. When ending his turn on a windy tile, the agent has 50% chance to be moved in a random direction. This effect can be applied multiples times if the agent gets pushed by the wind on another windy tile. At each iteration, the agent receive a -0.5 reward as an incitative to go faster. Moving on a cliff tile brings the agent back to the starting position and gives a -2 reward (the cliff is not that steep). Reaching the goal state gives a 100 points reward.

We test the algorithms on four instances of this environment. Figure 1 offer visualization of the four maps.

The first and third maps are extensions of the cliffworld example previously seen in class. A cliff separates the start and end goal. Several windy lanes are next to the cliff; to be on the one closest to the cliff is a big risk since there is a 12.5% that the agent gets pushed on a cliff tile (without counting the probability of being pushed by down the cliff by the next windy tile). The agent can take the safe path, no stochasticity involved, all around the grid to get to the goal.

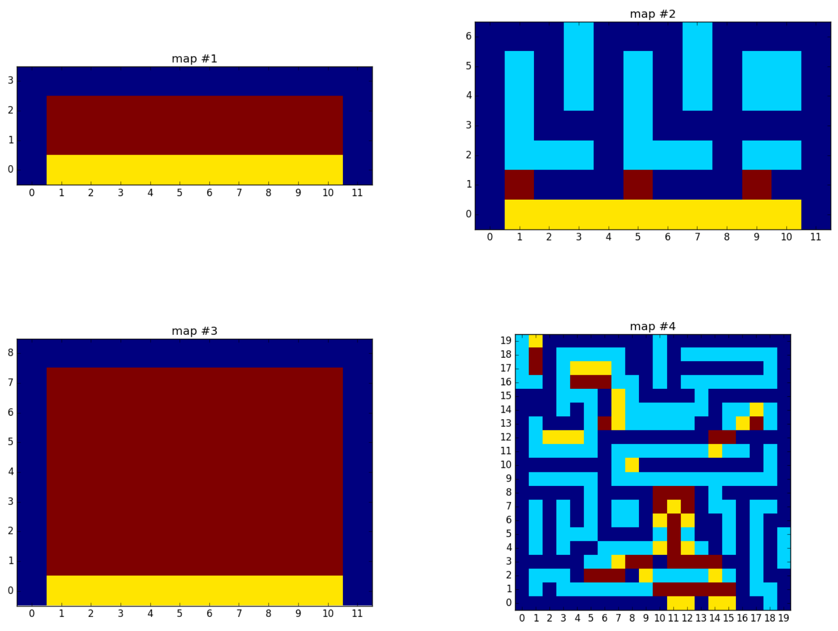


Figure 1: The maps we test our algorithms on. Yellow tiles are cliff, dark blue tiles are regular tiles, brown tiles are windy tiles and light blue tiles are walls. The agent always starts at the bottom-left corner and the goal is to reach the bottom-right corner

However, the agent can try to cut through the windy lanes to get faster to the end goal. Such endeavour is not riskless although the associated risk is decreased exponentially the further from the cliff the agent is attempting to cross.

The second map contains drastically less windy tiles; in it, the agent is confronted to a sequence of dichotomal path choices. All along the way to the goal, the agent either use shortcuts which implies walking on a windy tile next to a cliff or goes through the safe but longer path. The risk associated with walking on a windy tile next to a cliff is greater the closer the agent is to the goal. We are curious to see how well and how quick our algorithms can assess risk-value

The fourth map is a 20 by 20 tiles maze with 6 main path to get to the end. The bottom 3 path are quicker but riskier, the one above has a tiny bit of risk associated with and the two top ones are riskless, longer and have no stochasticity involved.

2 Methodology, implementation details and other notices

Unless otherwise specified, hyperparameters are, if they apply to the algorithm, set as follows : the discount factor $\gamma = 0.9$, the learning rate $\alpha = 0.05$, $\epsilon = 0.05$, the number of episodes is a thousand, the maximum number of iterations per episode allowed is 750. On map 4, the number of episodes is two thousands, and the maximum number of iterations per episode allowed is 1500. We initialize our qvalue matrix to be filled with ones. When we derive action probabilities from our qmatrix, we beforehand normalize the action values with a softmax function with temperature 10. The Q(sigma) variants and the treebackup algorithm are always evaluated with $n = 2$.

Map#1 and map#3, particularly #3, expose the agent to high stochasticity in transitions by having a dense area of windy tiles. Since we allow multiple application of wind effect in one turn, the agent can possibly get shuffled by the wind quite a lot. That stochasticy has an effect on the stability of the qvalue learned by the agent. To address that, we decided to run each training loop three times. Since it seemed like good practice for analysis of learning in stochastic context, we ran three time all experiments.

Section 7 contains 4 graphs for every combination of algorithm/map that we're experimenting with. The two first graph, respectively the maximum qvalue associated with each state and the greedy policy learned after 1000 episodes, are derived from the average qvalue by state-action pair through 3 runs. The two last graph are plots of the total reward obtained and iterations takens in function of the number of episodes completed. In those, each run is represented with a transparent grey line while their average is of darker tone. Furthermore, all rewards and iterations taken lines are smoothed with a moving average of 10 episodes since there is a lot of short-term variance in the algorithm's performance.

3 Qlearning, expected sarsa and 2-step treebackup

We set our hyperparameters as specified in the previous section. Table 1, 2, 3 and 4 in the next section provide the average total number of iteration needed to complete all the episodes and the average reward for the last 100 episodes.

On map#1, all three algorithm converge to sensibly to the same reward, however 2-steps backup uses approximatively 3000 less iterations to complete the 1000 episodes. Expected Sarsa seems to experience less variance in its last episodes; this could account for it having an slightly above average reward for the last 100 episodes.

On map#2, the Qlearning algorithm learns and the 2-step backup learns to use all the shortcuts. Expected sarsa learns to take the safe way around, resulting in a lower average reward than the two other algorithms and a higher number of iteration taken to complete the thousand episodes. Curiously, it does learn that if the agent was to be right after the first windy tile encountered it should take all the subsequent shortcuts. This is rather suprising considering our initial idea which suggested that an agent should be more risk-averse in regards of cliffs closer its goal. The qvalues of the states in top-right corner of this map, an useless detour, remains badly defined after a thousand episodes for all three algorithms.

On map#3, expected sarsa seems to be more affected by variance in its training than Qlearning and treebackup; it takes approximatively 15k more iterations to complete the thousand episodes and on average approximately 6 more steps complete an episode (on the last 100 episodes). Qleaning learns converge faster and seems more stable, although treebackup is a very close second.

On map#4, the Qlearning algorithm and the 2-step backup learns to use the path with the low-risk shortcut while expected sarsa learns to take the shortest safe way around and has a lower average total reward by episode. Curiously, the Qlearning algorithm's greedy policy shows a useless 2 iterations "detour" around the windy tile. This does not seems to be significantly reflected in the average total reward obtained by the algorithm. Again, Qlearning is the algorithm which complete the prescribed 2 thousands episodes the quickest. Another interesting notice is that all algorithms seems to experience higher variance on this map than on the other; we suggest that it might be because of the bigger, more complex state space.

4 Q(σ)

4.a Behavior policy choice

Initially, we considered sampling a uniform distribution between all possible action as behavior policy (μ). This revealed itself to be insufficiant for learning the maps; the algorithm was not able to properly explore and learn qvalues. We opted for having the behavior policy be an ϵ-greedy sampling of the Q matrix values from the classic Q learning algorithm with $\epsilon = 0.05$. Our policy π is also egreedy with respect to the Qvalues but, considering we do not want to do on-policy learning (which would induces a $\rho == 1$ at every episode of the algorithm), we set epsilon for π to always be half of the one for μ.

4.b Sigma function choice

Multiple functions are considered to generate the sigma value.

1. Fixed value 0.1
2. Fixed value 0.5
3. Fixed value 0.9
4. Binomial samples with p=0.5
5. Uniform samples from a uniform distribution between 0 and 1.
6. Truncated Gaussian samples : Sampling from a normal distribution with mean 0.5 and standard deviation of 0.5. Values under 0 or above 1 are truncated to 0 or 1.
7. Type-informed values : sigma=0.1 if the agent is on a windy tile, sigma=0.9 if not.

4.c Results

Figure 2, 3, 4 and 5 shows the performance of the different variants of the Q(σ) algorithm on the four different maps. For each figures, the two first graphs represents the number of iteration taken and the reward in function of the number of episodes completed normalized with a moving average over 10 episodes. The third graph is the reward in function of the number of iteration completed. Since our points are not located at a regular interval on the x axis, no moving average was applied. This resulted into drastically reduced readability of the graph. However, tables 1, 2, 3 and 4 contains the average total number of iteration taken to complete all episodes and the average reward of the last 100 episodes. These tables give us insight on speed of convergence and variance inherent to the q(sigma) variants.

On map#1, all algorithms seems to converge to approximately the same value. Interestingly, the fix0.1 and the fix0.9 variants underperform the fix0.5 variant on both total number of iteration taken and last 100 episodes reward. The type-informed sigma function achieves convergence sligtly faster than 2-steps treebackup, making it the fastest converging algorithm experience on this map.

On map#2, all variants except fix0.1 learn to use all the shortcuts. However, they all take at least 5 thousands more iteration than the Qlearning to complete all episodes. The fix0.9 and the type-informed variant seems to be the q(sigma) variant which learns the best path faster. The fix0.1 variant learns, like expected sarsa, to take the safe path around. It takes it 3000 more iteration to do so. All variants except the fix0.1, fix0.9 and type-informed variants seem to be the subject of higher variance.

On map#3, all variants except fix0.1 converge to similar greedy policy and have performances for the last episodes similar to the Qlearning algorithm (our best performing algorithm on this map). Fix0.1 does perform slightly worst than expected sarsa (which is the second worst of all algorithms tried). However, all q(sigma) variants except the fix0.9 take at least 10k more iteration than the Qlearning algorithm to complete all episodes. The fix0.9 variant performs similarly to treebackup, i.e. takes a few thousand more iteration to complete all episodes.

On map#4, no q(sigma) variant is able to learn as quick as the Qlearning or the treebackup algorithm. However, the fix0.9 and the type-informed variant perform as well as qlearning on the average total reward of the last 100 episodes. On an interesting note, the fix0.1 variant seems to experience some traumatic events near the 1500th episode in 2 out of 3 run. These traumatic events push the qvalues in a regime where they become, after a few hundred more episodes, unable to guide the policy to the end goal. This accounts for the poor performance of the fix0.1 variant. It can be supposed that some element of stochastic nature might be behind such unfolding.

Looking at the resulting greedy policies, we see that the fix0.9, binomial and the truncated gaussian variant seems to learn to use the low-risk shortcut while all the other variants seems to learn to use the shortest safe path. The difference in the average last episodes performance is mostly due to algorithm stability; sampling based variants and fix0.5 variant seem to experience less stable convergence on this map.

While looking at values in table 4, please remember that the fourth map has an increased maximum number of iteration per episode (1500 instead of 750) and an increased number episode to complete (2000) .

Table 1: Average total number of iterations taken to complete 1000 episodes and average reward for last 100 episodes by algorithm on map#1

Algorithm	avg total it. taken	avg. reward last episodes
Q learning	29504.33	89.24
Expected Sarsa	29910.0	**91.57**
2-steps treebackup	26744.33	90.25
fix0.1	33221.67	88.95
fix0.9	30818.33	88.38
fix0.5	28226.0	90.43
uniform	28167.0	90.78
binomial0.5	28870.0	89.83
truncatedGauss	29667.67	89.09
TypeBased	**26259.33**	91.2

Figure 2:

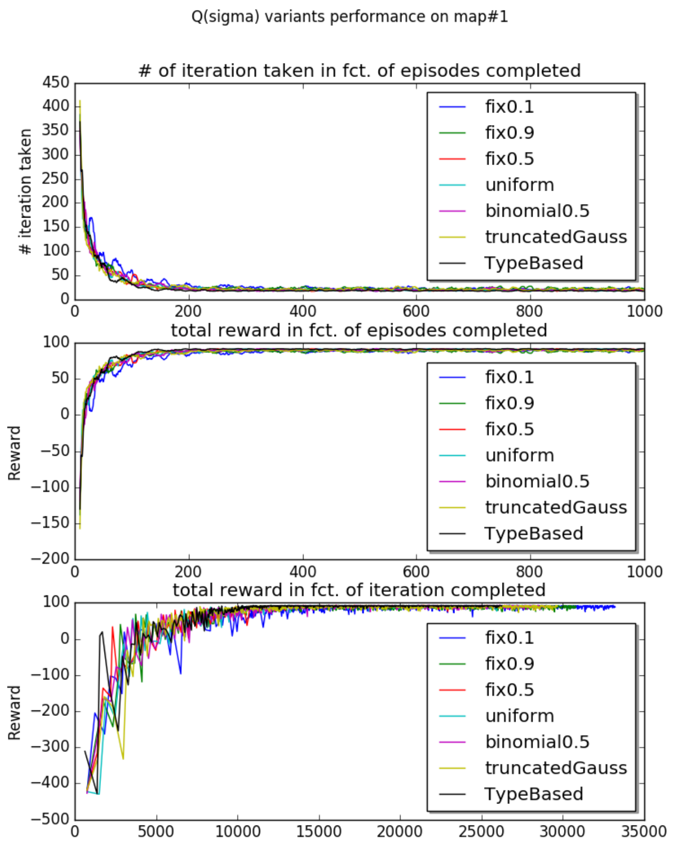

Q(sigma) variants performance on map#1
# of iteration taken in fct. of episodes completed
# iteration taken
total reward in fct. of episodes completed
Reward
total reward in fct. of iteration completed
Reward
fix0.1
fix0.9
fix0.5
uniform
binomial0.5
truncatedGauss
TypeBased

Figure 3:

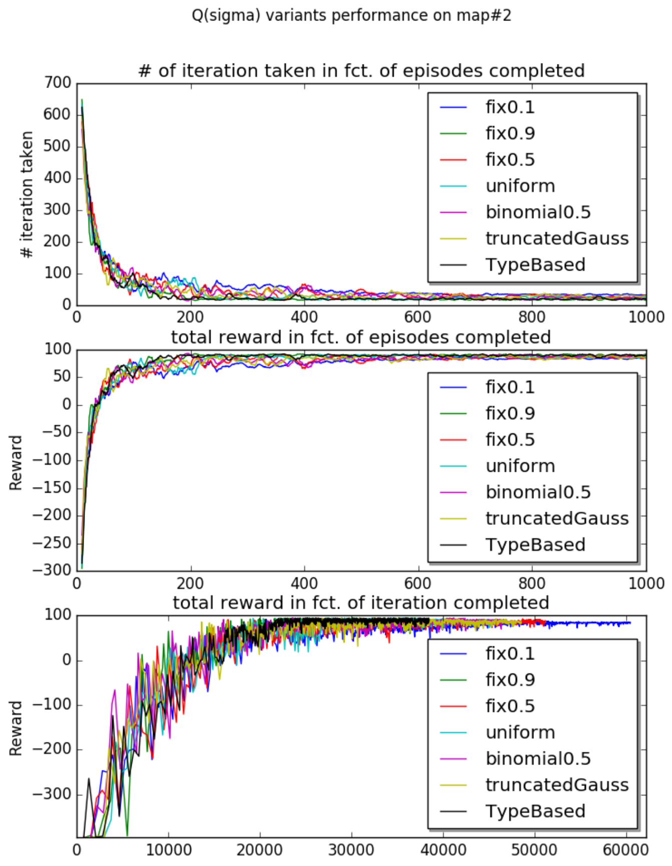

Q(sigma) variants performance on map#2
# of iteration taken in fct. of episodes completed
# iteration taken
total reward in fct. of episodes completed
Reward
total reward in fct. of iteration completed
Reward
fix0.1
fix0.9
fix0.5
uniform
binomial0.5
truncatedGauss
TypeBased

Figure 4:

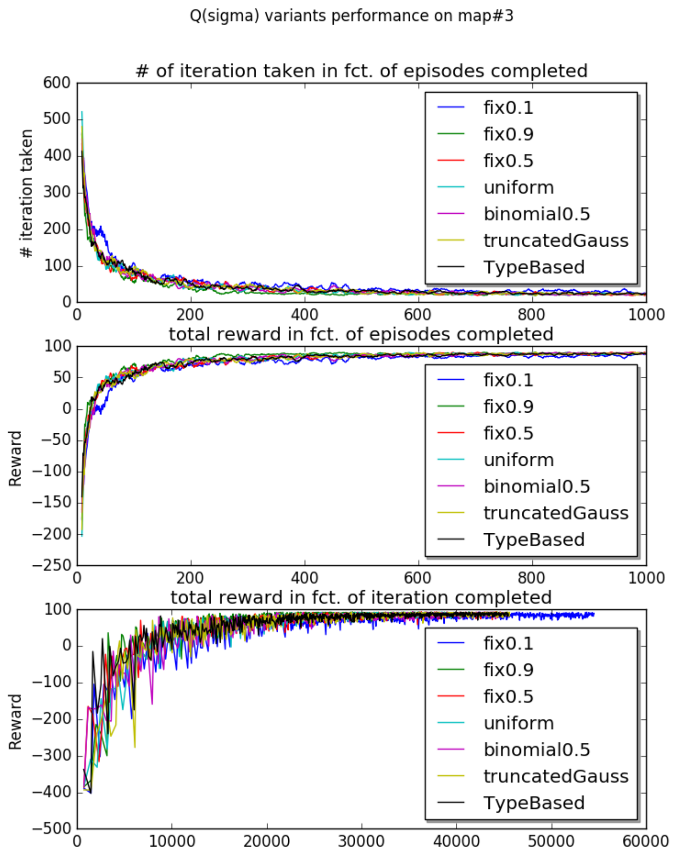

Q(sigma) variants performance on map#3
# of iteration taken in fct. of episodes completed
# iteration taken
total reward in fct. of episodes completed
Reward
total reward in fct. of iteration completed
Reward
fix0.1
fix0.9
fix0.5
uniform
binomial0.5
truncatedGauss
TypeBased

Figure 5:

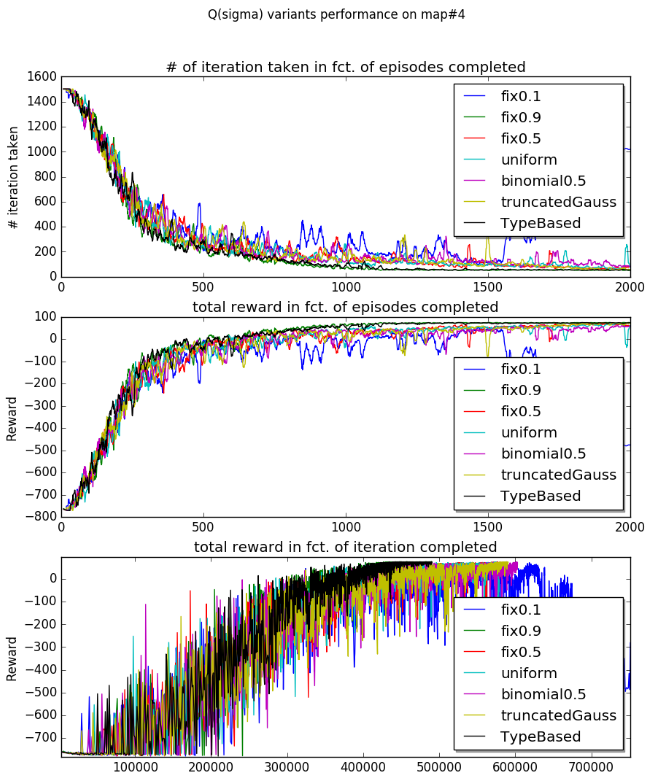

Q(sigma) variants performance on map#4
# of iteration taken in fct. of episodes completed
# iteration taken
fix0.1
fix0.9
fix0.5
uniform
binomial0.5
truncatedGauss
TypeBased
total reward in fct. of episodes completed
Reward
total reward in fct. of iteration completed
Reward

Table 2: Average total number of iterations taken to complete 1000 episodes and average reward for last 100 episodes by algorithm on map#2

Algorithm	avg total it. taken	avg. reward last episodes
Q learning	**32057.0**	**91.03**
Expected Sarsa	57471.67	85.82
2-steps treebackup	33004.0	89.12
fix0.1	60449.67	83.53
fix0.9	37073.33	89.41
fix0.5	51183.0	87.11
uniform	45577.33	86.99
binomial0.5	46968.67	87.53
truncatedGauss	48523.0	85.89
TypeBased	38471.0	88.98

Table 3: Average total number of iterations taken to complete 1000 episodes and average reward for last 100 episodes by algorithm on map#3

Algorithm	avg total it. taken	avg. reward last episodes
Q learning	**35508.0**	**89.92**
Expected Sarsa	52571.67	87.02
2-steps treebackup	37519.67	88.85
fix0.1	54505.67	85.57
fix0.9	38757.0	88.49
fix0.5	44717.67	88.93
uniform	44555.33	87.96
binomial0.5	45380.0	88.7
truncatedGauss	45611.0	88.09
TypeBased	45498.0	87.85

Table 4: Average total number of iterations taken to complete 2000 episodes and average reward for last 100 episodes by algorithm on map#4

Algorithm	avg total it. taken	avg. reward last episodes
Q learning	**457928.0**	73.2
Expected Sarsa	577554.67	63.69
2-steps treebackup	469420.67	71.8
fix0.1	1000374.33	-478.28
fix0.9	490042.67	**73.33**
fix0.5	582841.0	64.33
uniform	579639.33	54.63
binomial0.5	602890.67	52.76
truncatedGauss	590555.67	69.37
TypeBased	490338.0	72.6

5 Discussions

While these results fail to find a way to have sigma values that leads to better or quicker learning than the more classical methods, they do show that the algorithm is capable of performing as good as Qlearning (which is not that much of a suprise to be honest). It also shows the versatility of the algorithm and the crucial importance associated with the choice of a sigma function. Also, we discovered that choosing sigma value from sampling with a distribution that has a lots of variance tend to introduce instability especially in late training stages.

With regards to these results, what could be some other approaches worth investigating to generate the sigma value? We suggest as future experiments to push the notion of adaptivity in functions to generate sigma. Here are some ideas of how could that notion be introduced.

1. We could learn a model to approximate $P(S_{t+1}|A_t S_t)$, feed that model in entry to the algorithm. Then, we maintain a state-wise metric of accuracy in our ability to predict what the next state will be in function of the (current state, action) pair and vary sigma to be more like expected sarsa when believe that our actions have less impact on what the next state will be. Furthermore, the learning of the transition model could potentially need to be done concurrently to the execution of our Q(σ) algorithm since it is not given that we have a start with a good enough exploration policy to learn that the relevant transition probabilities.

2. Considering that our approaches with sampled sigma seems to induce instability in particular in later episodes, we could schedule the decrease of variance in our sampling function with regards to the number of episodes completed.

6 Reference

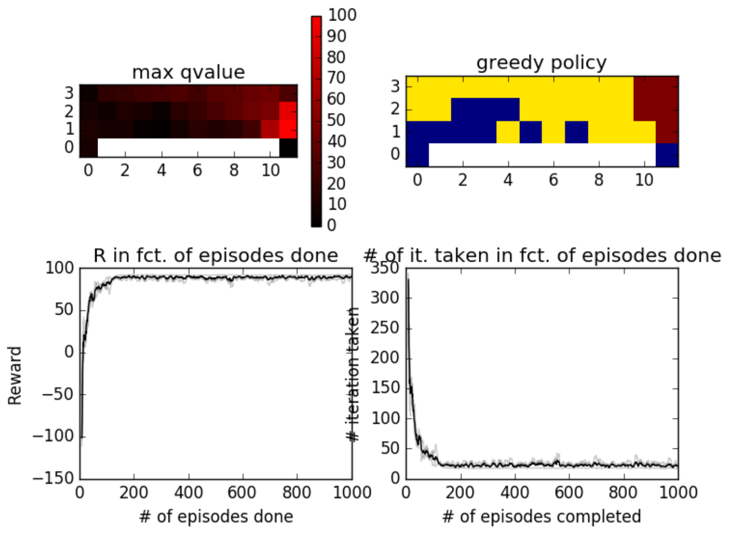
References

[1] Richard S. Sutton and Andrew G. Barto, *Reinforcement Learning: An Introduction*, Bookdraft september 2016.

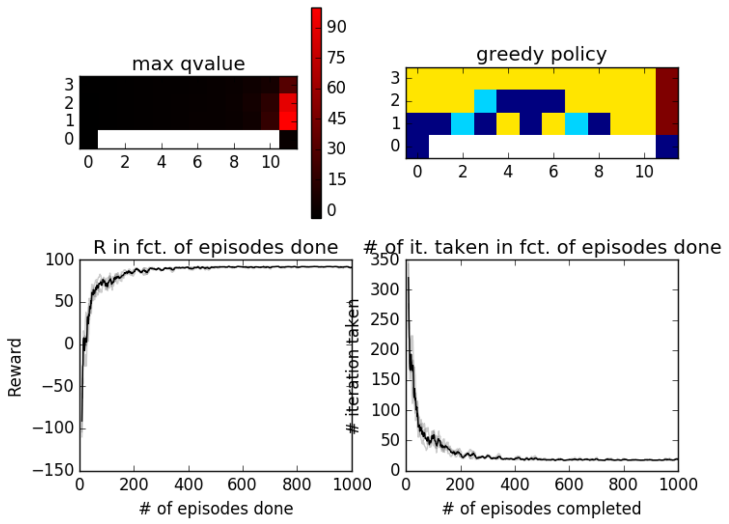
7 Figures

All figures in this section correspond to specific experiences

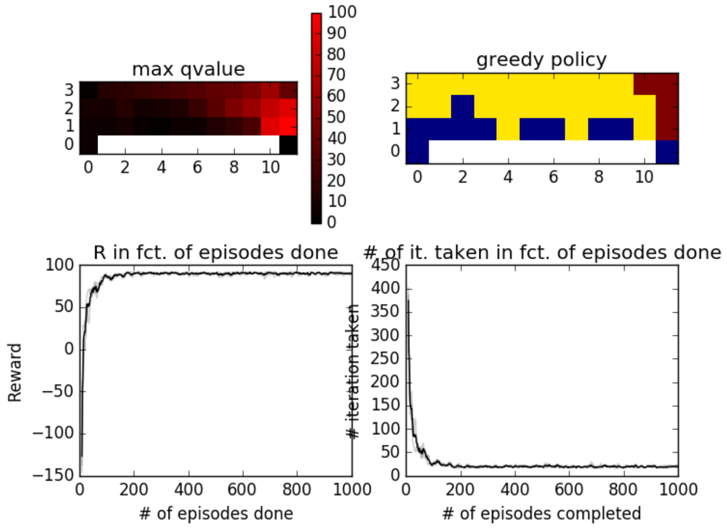
Q learning - map#1



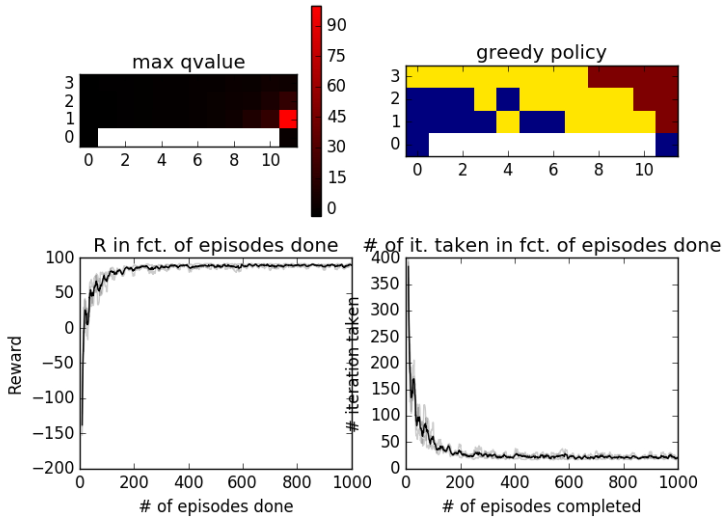
Expected Sarsa - map#1

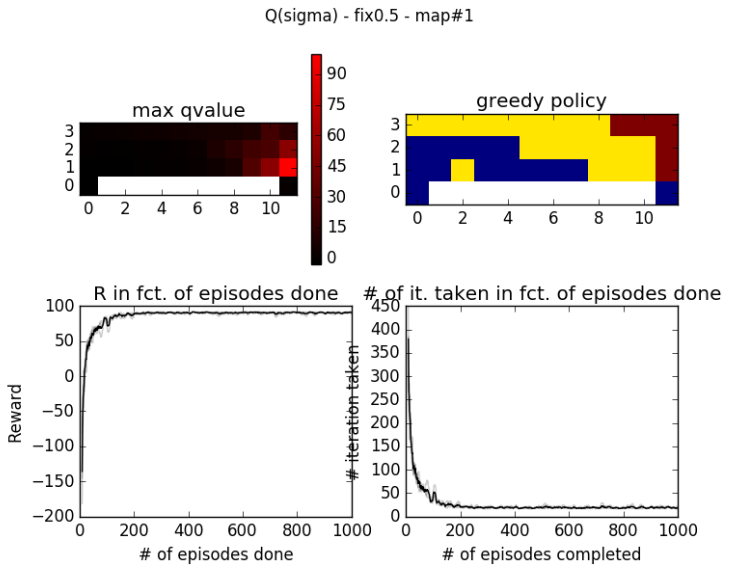


2-steps treebackup - map#1

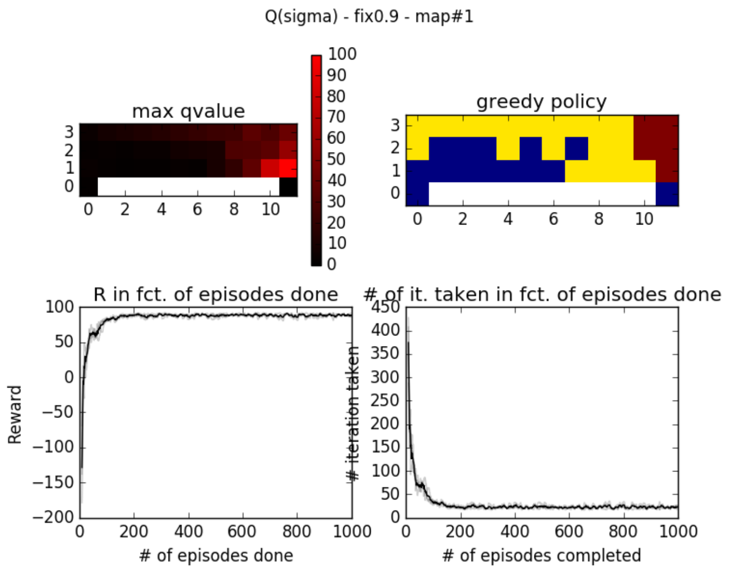


Q(sigma) - fix0.1 - map#1



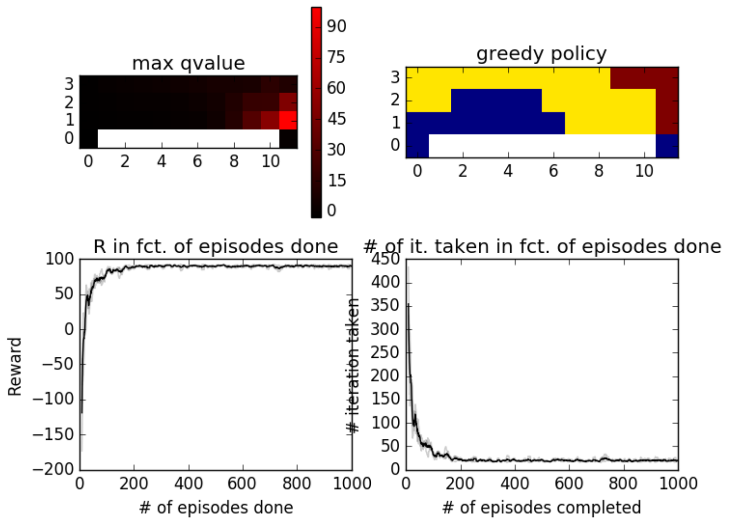


Q(sigma) - fix0.5 - map#1
max qvalue
greedy policy
R in fct. of episodes done
Reward
# of episodes done
# of it. taken in fct. of episodes done
# iteration taken
# of episodes completed

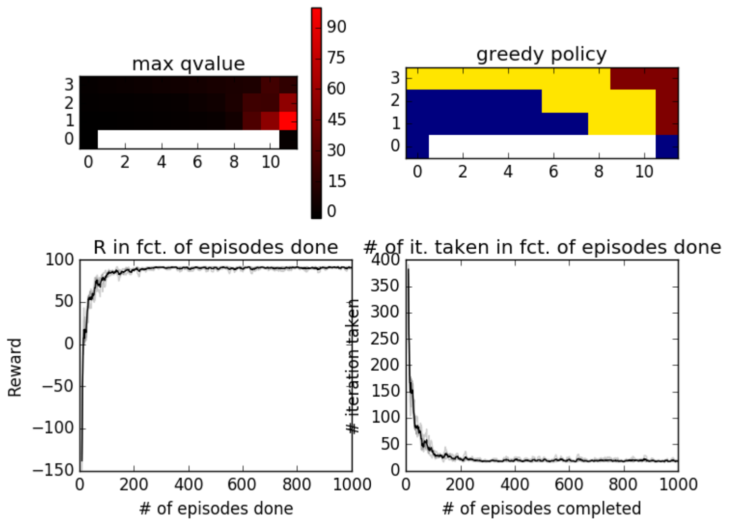


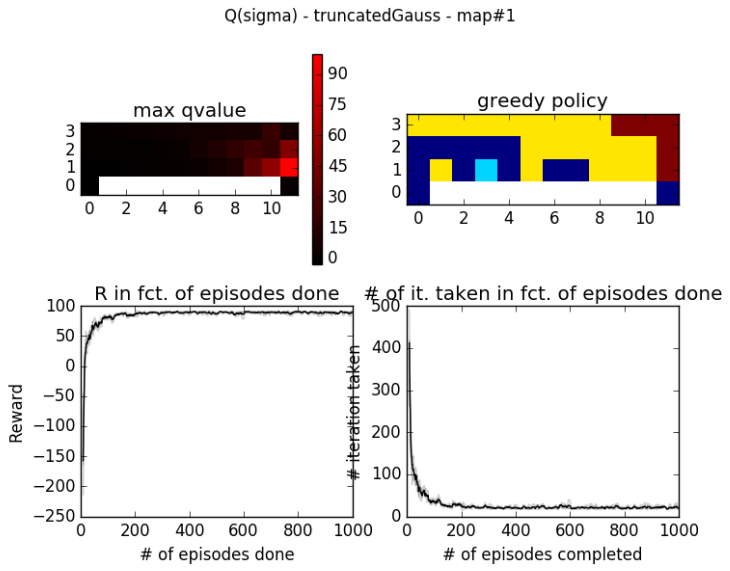
Q(sigma) - fix0.9 - map#1
max qvalue
greedy policy
R in fct. of episodes done
Reward
# of episodes done
# of it. taken in fct. of episodes done
# iteration taken
# of episodes completed

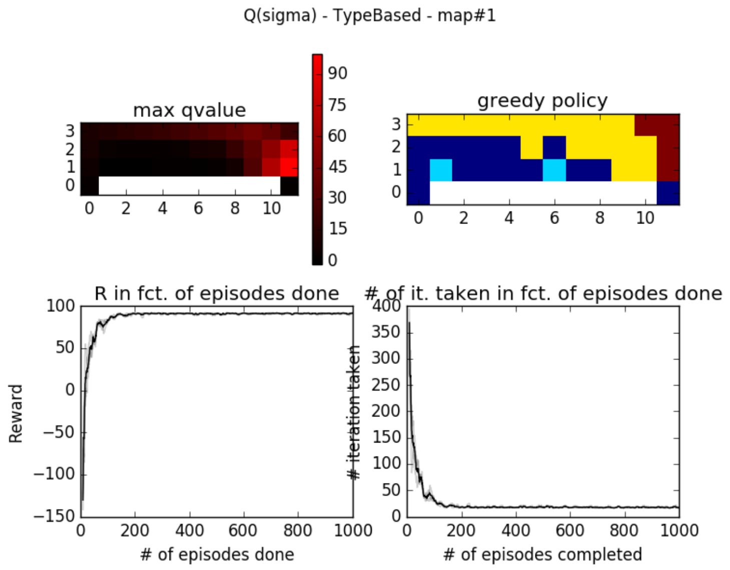
Q(sigma) - binomial0.5 - map#1

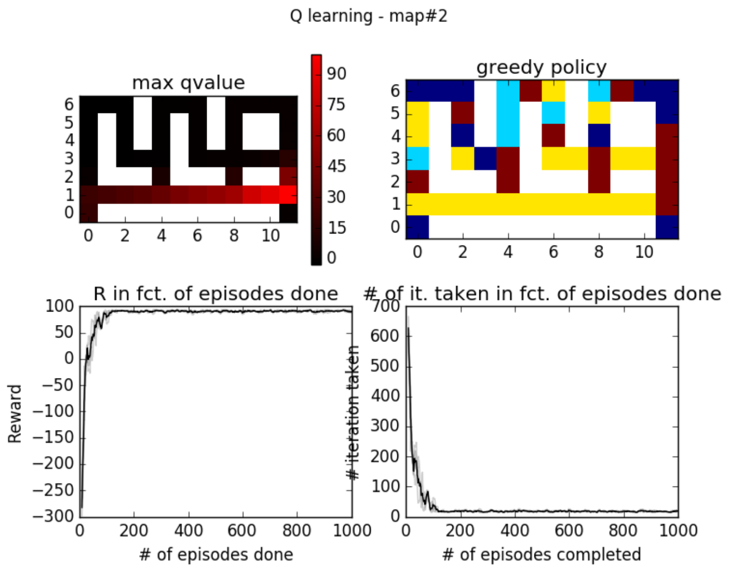


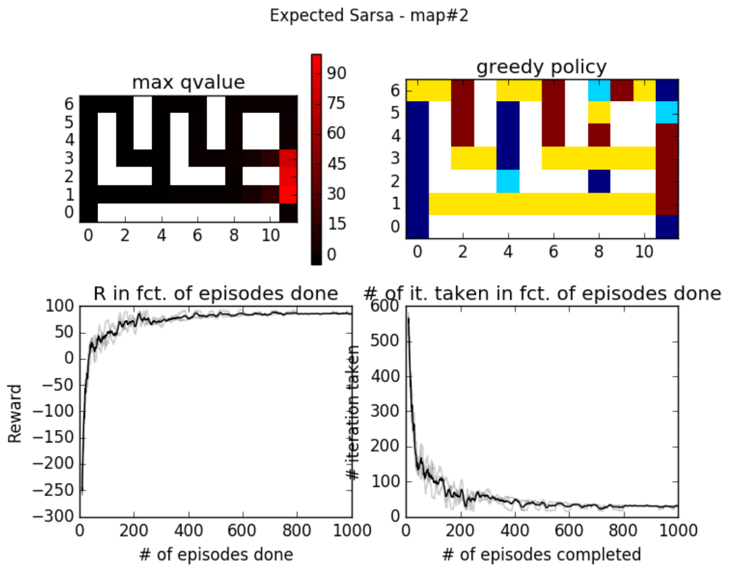
Q(sigma) - uniform - map#1



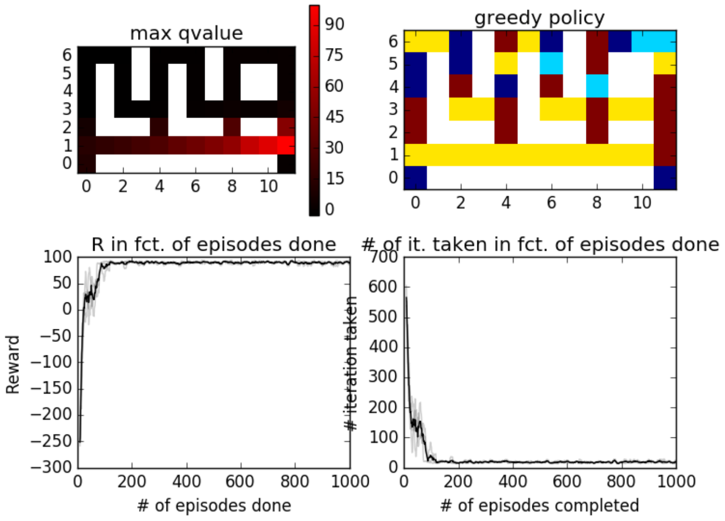

Q(sigma) - truncatedGauss - map#1
max qvalue
greedy policy
R in fct. of episodes done
# of it. taken in fct. of episodes done
Reward
# iteration taken
# of episodes done
# of episodes completed


Q(sigma) - TypeBased - map#1
max qvalue
greedy policy
R in fct. of episodes done
# of it. taken in fct. of episodes done
Reward
# iteration taken
# of episodes done
# of episodes completed

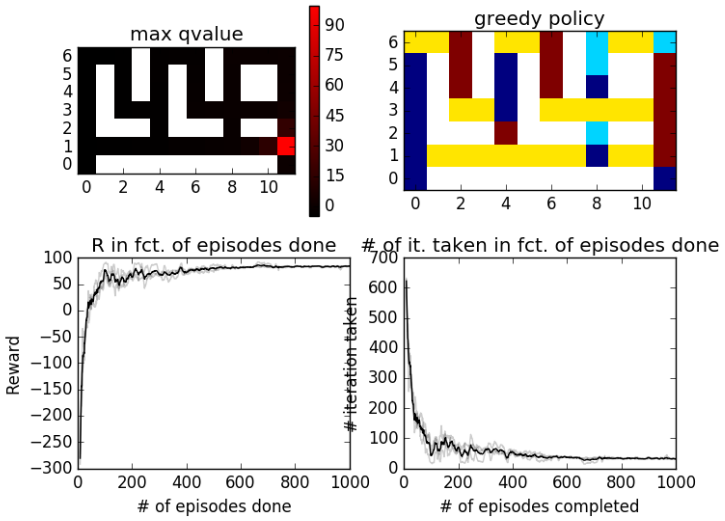

Q learning - map#2
max qvalue
greedy policy
R in fct. of episodes done
Reward
# of episodes done
# of it. taken in fct. of episodes done
# iteration taken
# of episodes completed

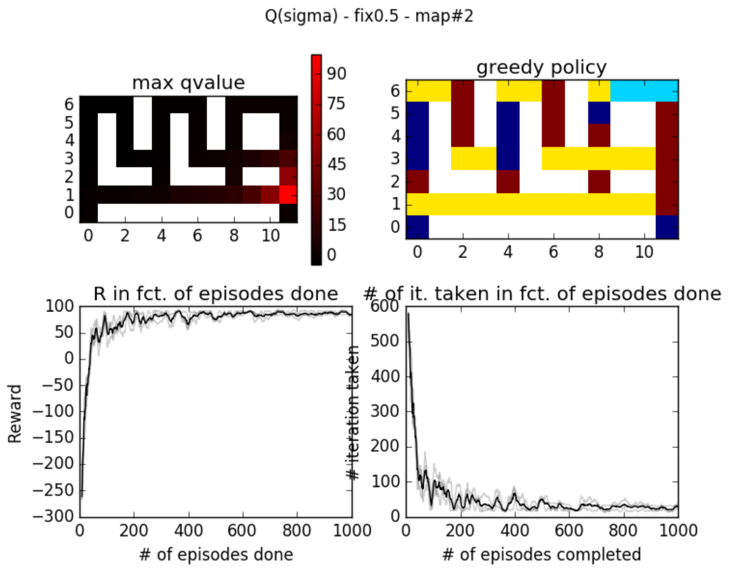

Expected Sarsa - map#2
max qvalue
greedy policy
R in fct. of episodes done
Reward
# of episodes done
# of it. taken in fct. of episodes done
# iteration taken
# of episodes completed

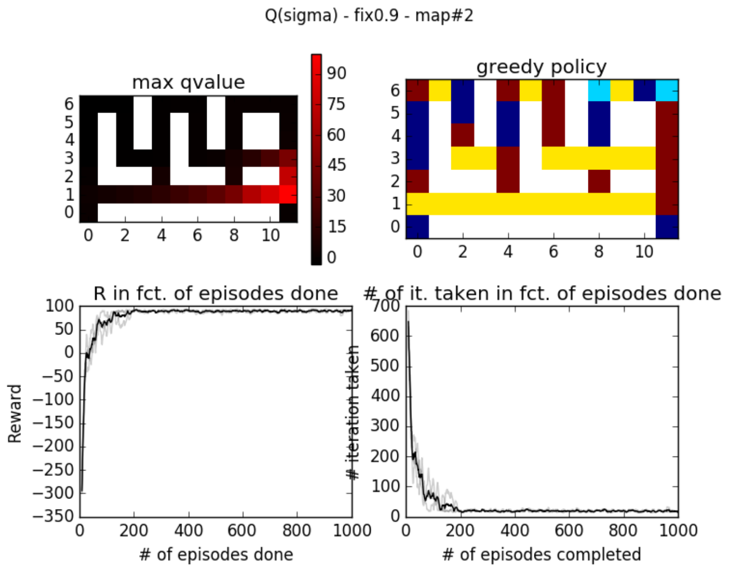
2-steps treebackup - map#2



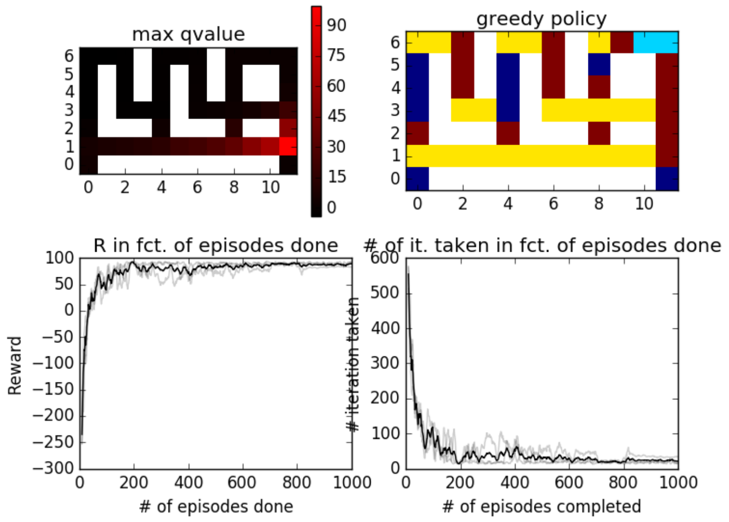
Q(sigma) - fix0.1 - map#2



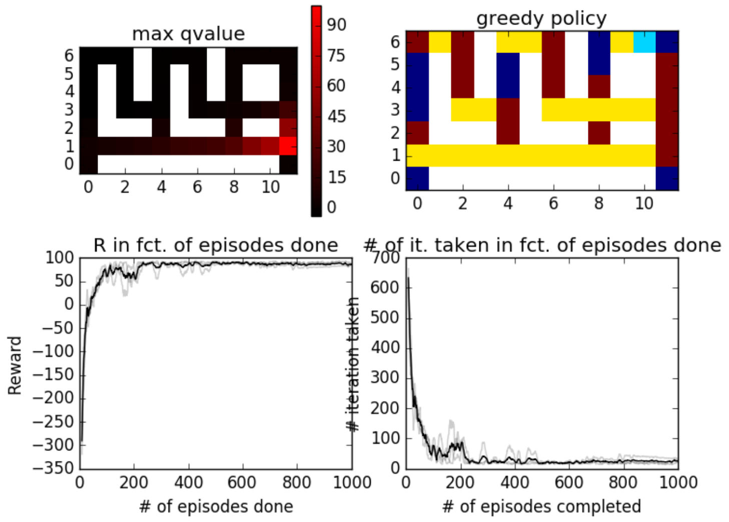

Q(sigma) - fix0.5 - map#2
max qvalue
greedy policy
R in fct. of episodes done
# of it. taken in fct. of episodes done
Reward
# iteration taken
# of episodes done
# of episodes completed


Q(sigma) - fix0.9 - map#2
max qvalue
greedy policy
R in fct. of episodes done
# of it. taken in fct. of episodes done
Reward
# iteration taken
# of episodes done
# of episodes completed

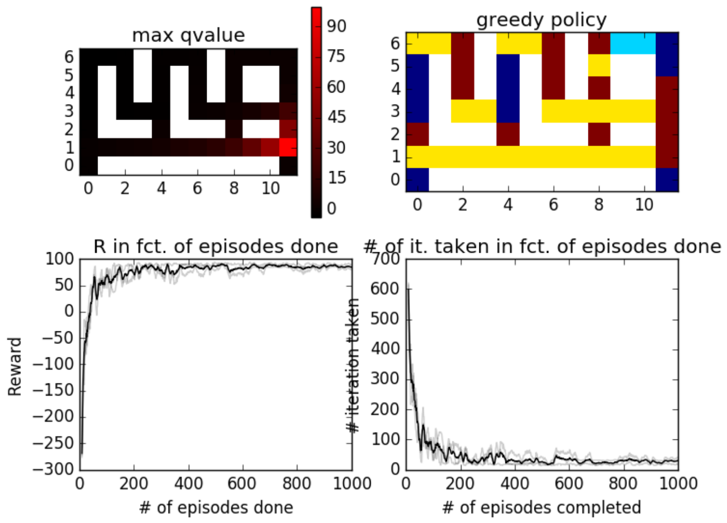
Q(sigma) - binomial0.5 - map#2



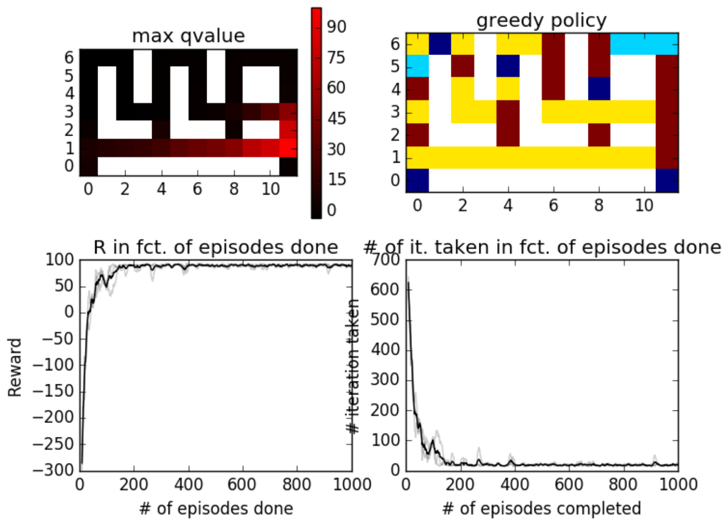
Q(sigma) - uniform - map#2



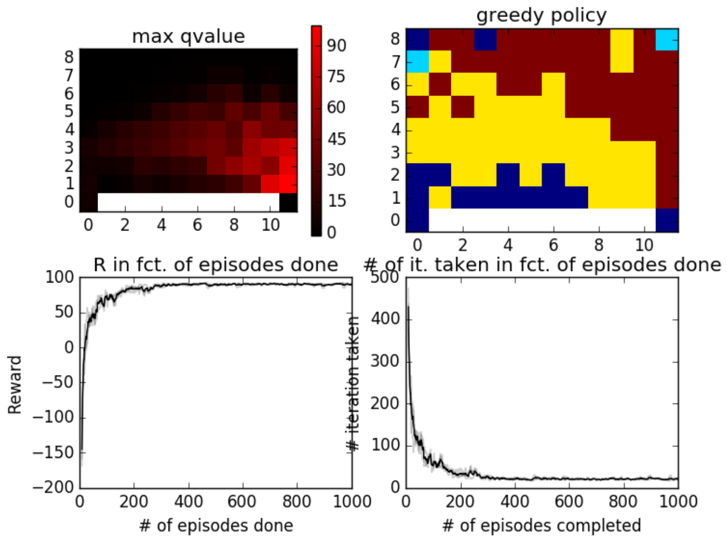
Q(sigma) - truncatedGauss - map#2



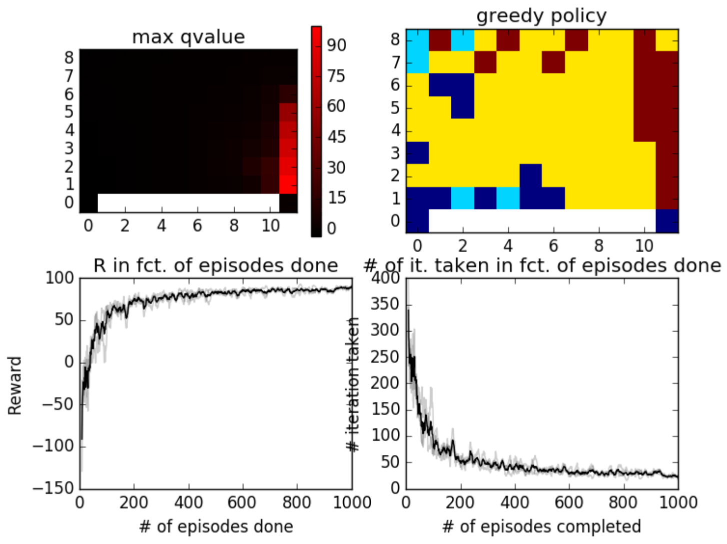
Q(sigma) - TypeBased - map#2



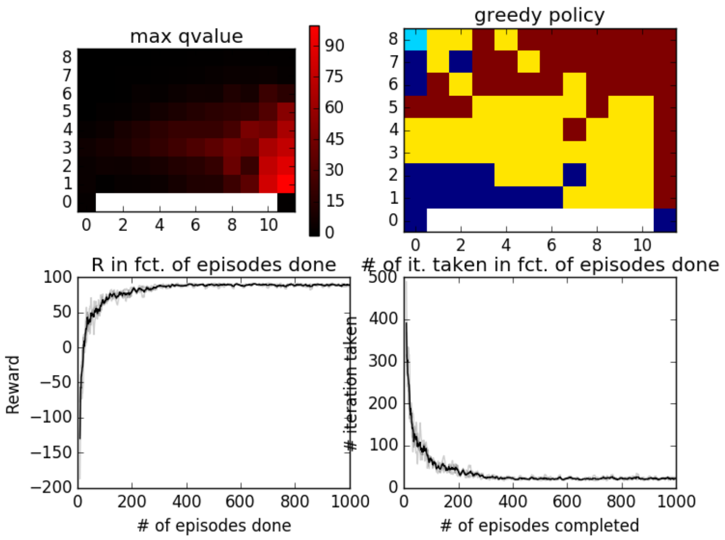
Q learning - map#3



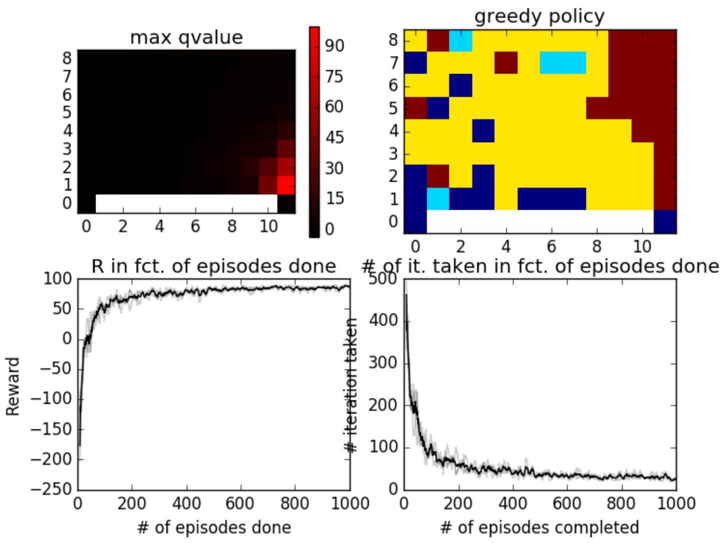
Expected Sarsa - map#3



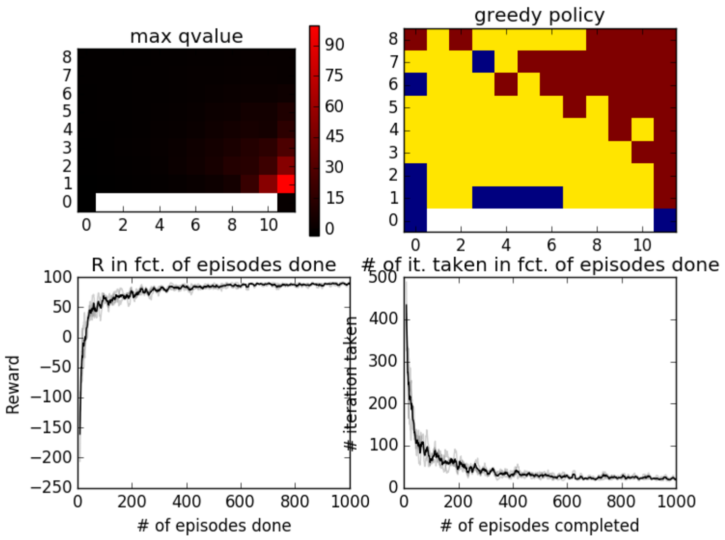
2-steps treebackup - map#3



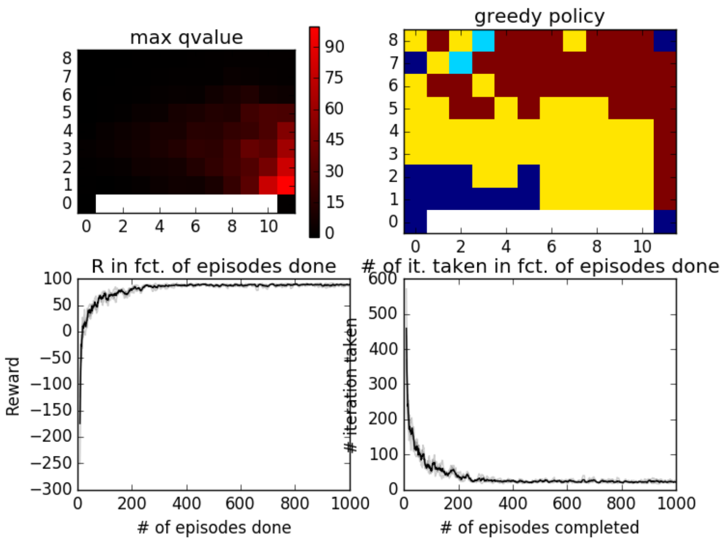
Q(sigma) - fix0.1 - map#3



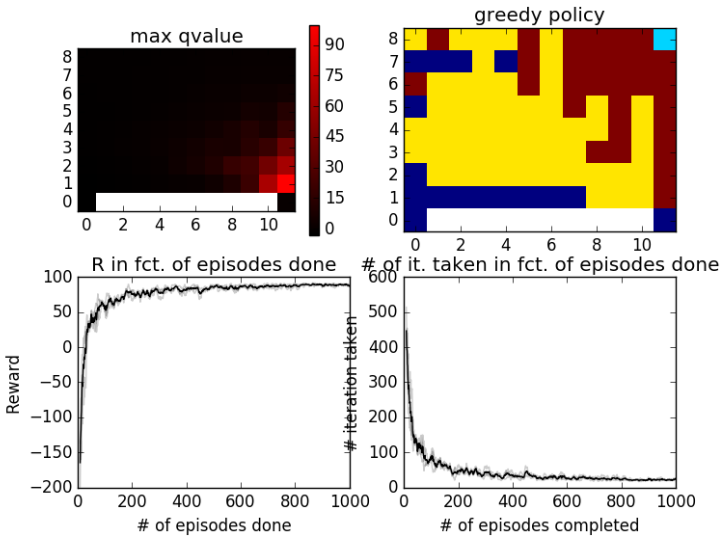
Q(sigma) - fix0.5 - map#3



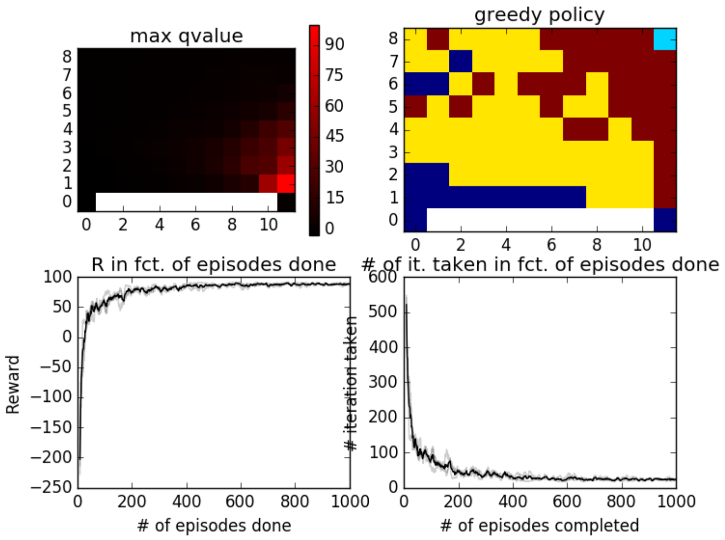
Q(sigma) - fix0.9 - map#3

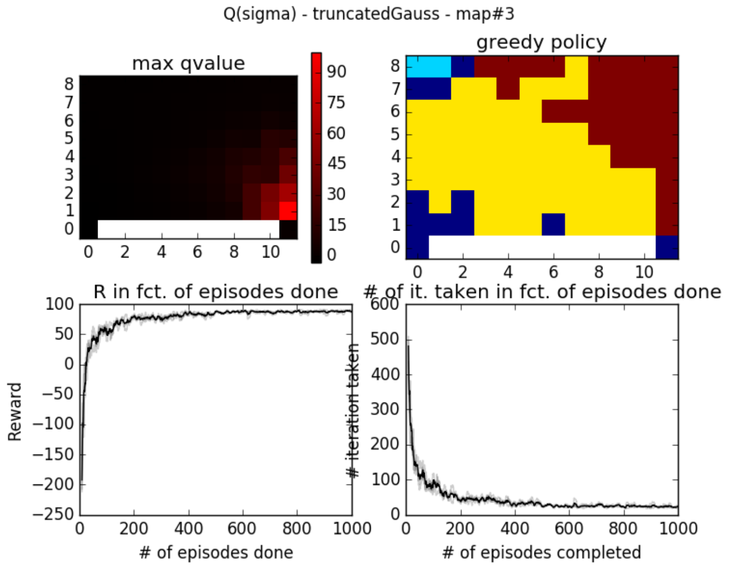


Q(sigma) - binomial0.5 - map#3

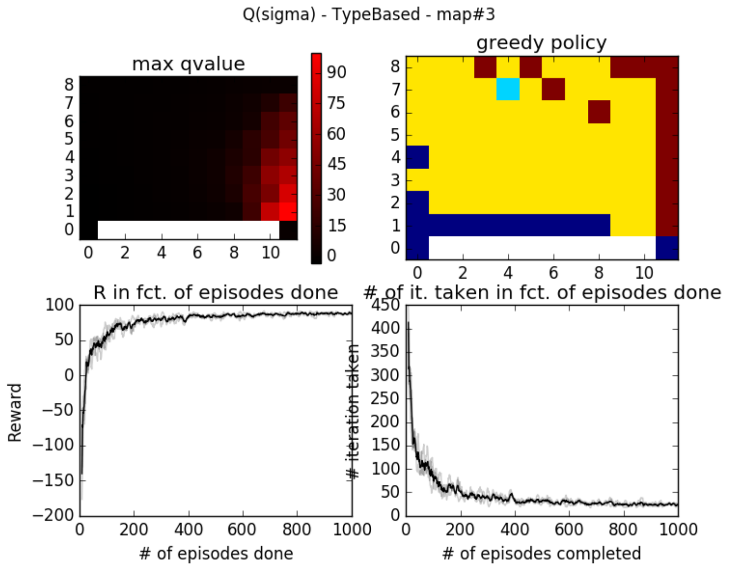


Q(sigma) - uniform - map#3

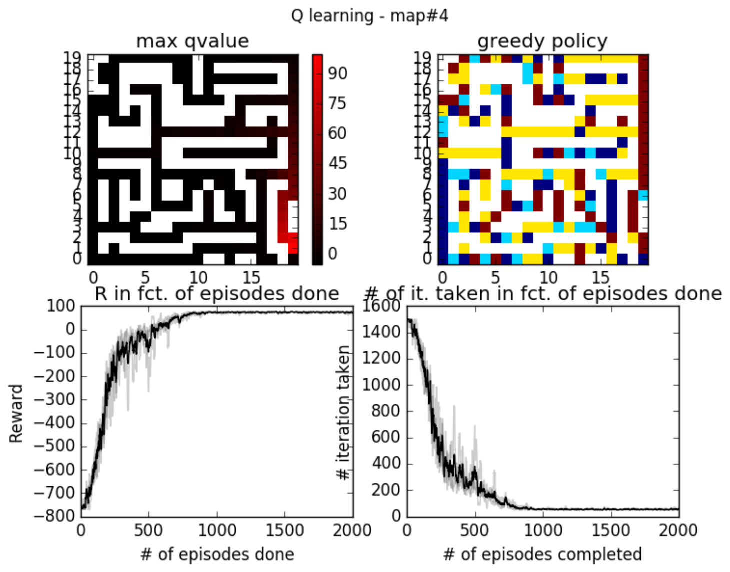


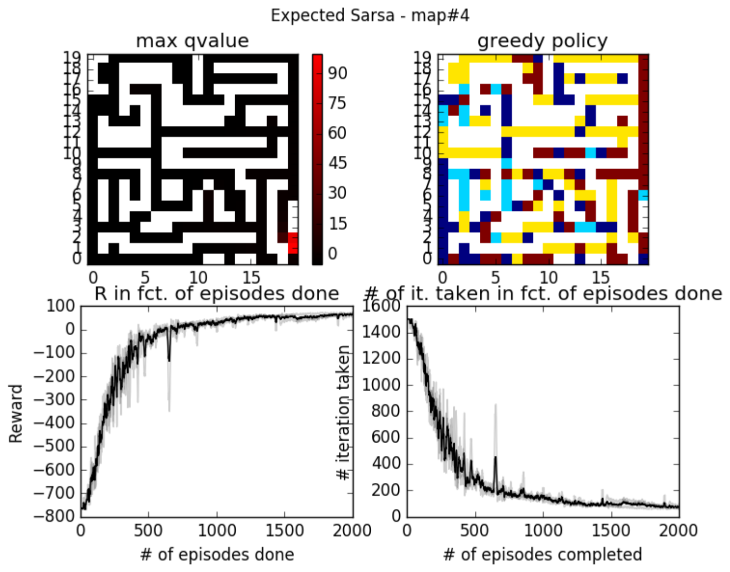


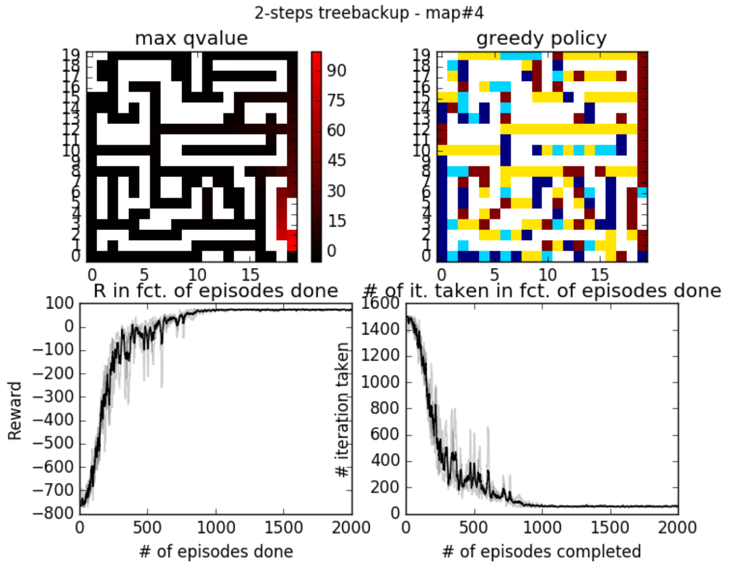
Q(sigma) - truncatedGauss - map#3
max qvalue
greedy policy
R in fct. of episodes done
# of it. taken in fct. of episodes done
Reward
# iteration taken
# of episodes done
# of episodes completed

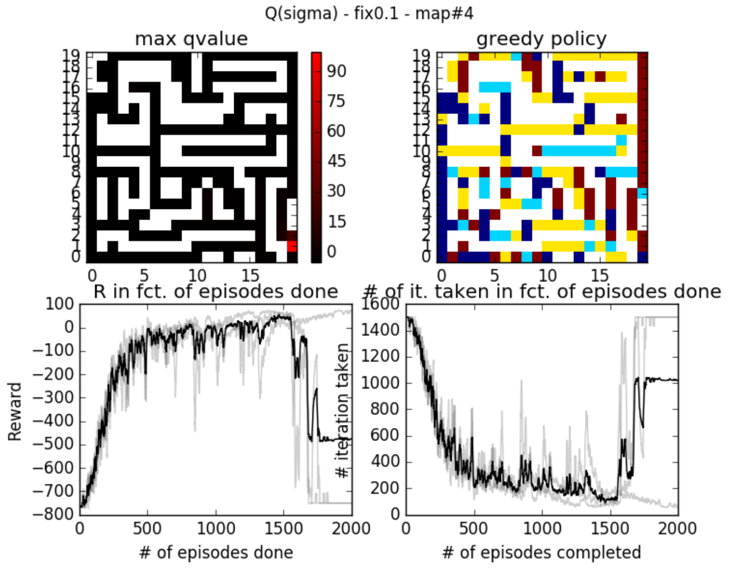


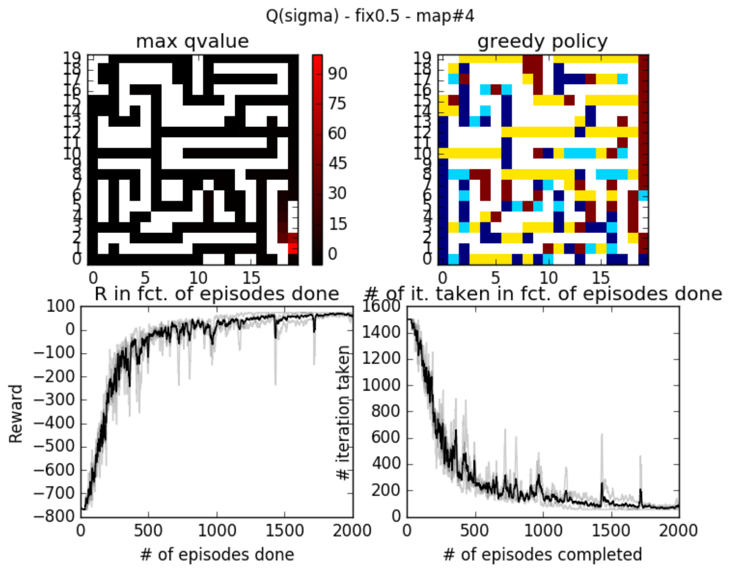
Q(sigma) - TypeBased - map#3
max qvalue
greedy policy
R in fct. of episodes done
# of it. taken in fct. of episodes done
Reward
# iteration taken
# of episodes done
# of episodes completed


Q learning - map#4
max qvalue
greedy policy
R in fct. of episodes done
Reward
# of episodes done
# of it. taken in fct. of episodes done
# iteration taken
# of episodes completed

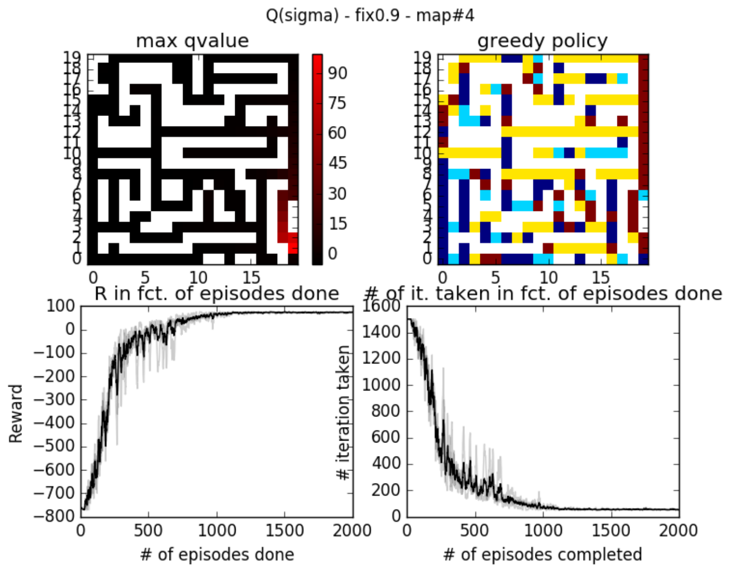

Expected Sarsa - map#4
max qvalue
greedy policy
R in fct. of episodes done
Reward
# of episodes done
# of it. taken in fct. of episodes done
# iteration taken
# of episodes completed


2-steps treebackup - map#4
max qvalue
greedy policy
R in fct. of episodes done
# of it. taken in fct. of episodes done
Reward
# iteration taken
# of episodes done
# of episodes completed

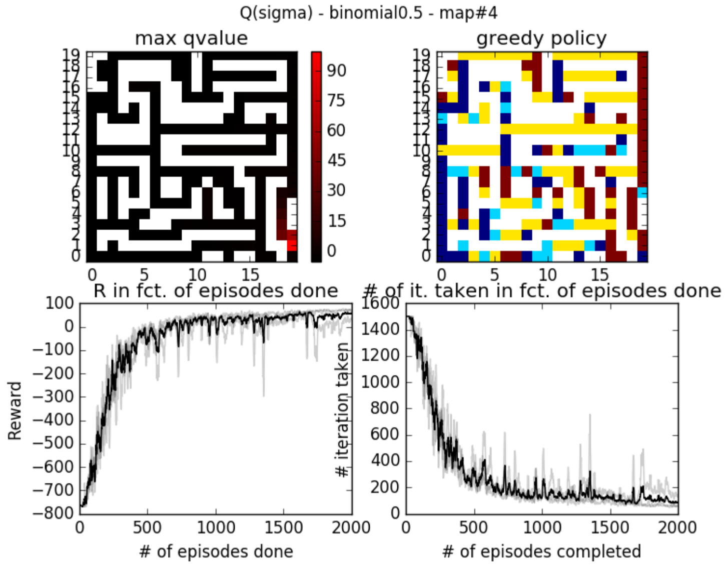

Q(sigma) - fix0.1 - map#4
max qvalue
greedy policy
R in fct. of episodes done
# of it. taken in fct. of episodes done
Reward
# iteration taken
# of episodes done
# of episodes completed

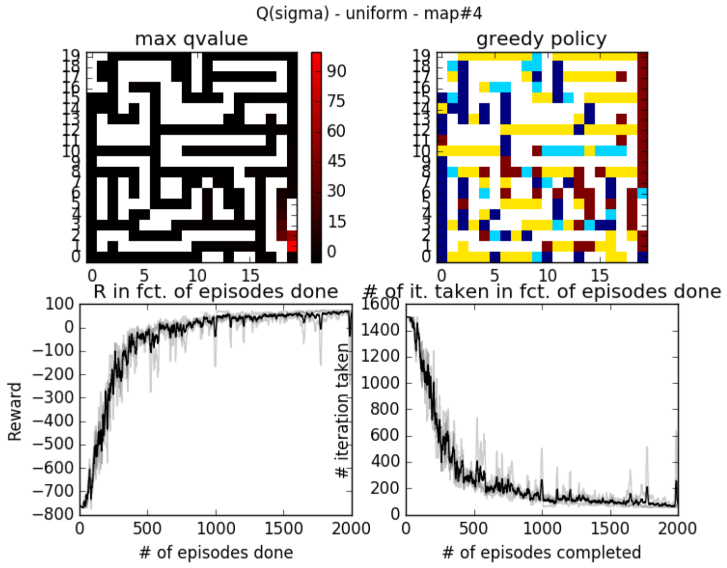


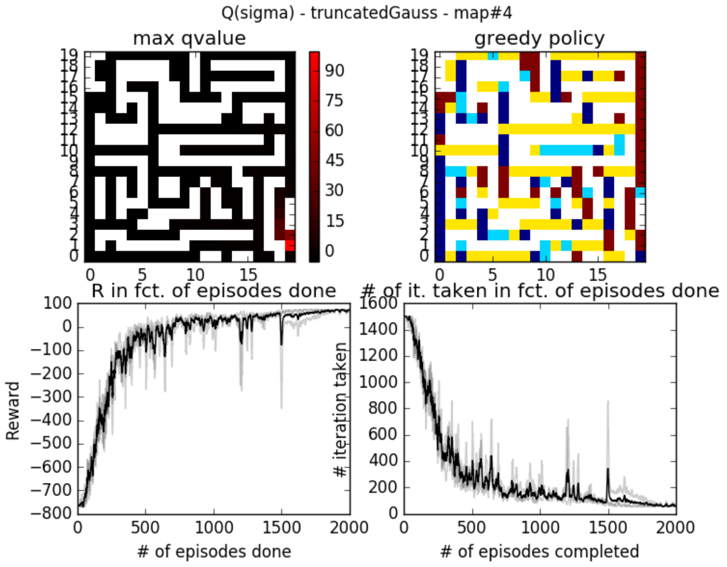
Q(sigma) - fix0.5 - map#4
max qvalue
greedy policy
R in fct. of episodes done
# of it. taken in fct. of episodes done
Reward
# iteration taken
# of episodes done
# of episodes completed



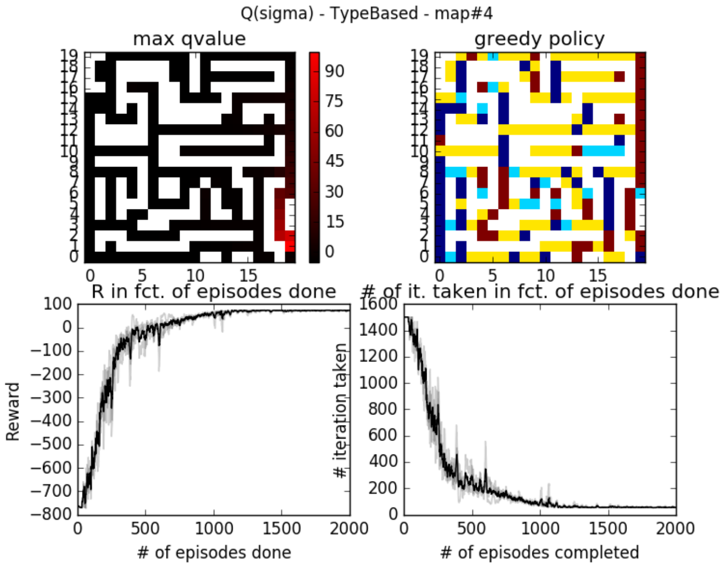
Q(sigma) - fix0.9 - map#4
max qvalue
greedy policy
R in fct. of episodes done
# of it. taken in fct. of episodes done
Reward
# iteration taken
# of episodes done
# of episodes completed


Q(sigma) - binomial0.5 - map#4
max qvalue
greedy policy
R in fct. of episodes done
# of it. taken in fct. of episodes done
Reward
# iteration taken
# of episodes done
# of episodes completed


Q(sigma) - uniform - map#4
max qvalue
greedy policy
R in fct. of episodes done
# of it. taken in fct. of episodes done
Reward
# iteration taken
# of episodes done
# of episodes completed



Q(sigma) - truncatedGauss - map#4
max qvalue
greedy policy
R in fct. of episodes done
# of it. taken in fct. of episodes done
Reward
# iteration taken
# of episodes done
# of episodes completed



Q(sigma) - TypeBased - map#4
max qvalue
greedy policy
R in fct. of episodes done
# of it. taken in fct. of episodes done
Reward
# iteration taken
# of episodes done
# of episodes completed