Intelligent Agent Querying

BA Thesis (*Afstudeerscriptie*)

written by

Tim Doolan
(born October 31st, 1987 in Amsterdam, Netherlands)

under the supervision of **Dr. Christos Dimitrakakis**, submitted in partial fulfillment of the requirements for the degree of

BA Kunstmatige Intelligentie

at the *Universiteit van Amsterdam.*

Abstract

Given agents which have learned the transition functions for a discrete MDP, we examen how another agent, with no information about the state space, can efficiently determine how to query the other agents for knowledge of the MDP. This includes determining when to stop querying, as maximize the reward minus the penalty for each query (the reward in the MDP will increased due to more knowledge as a results of more queries, but so will the penalty). We also develop a method that determines what agents and what parts of the state space to query and find approaches with promising results for all problems, but most still need some adaptation before they are applicable to real situations.

1 Introduction

Distributed sensor networks are used in many modern day applications, in which you try to reduce uncertainty regarding the value of a (hidden) variable by combining multiple sensor readings. The conventional approach to represent uncertainty is to use a Bayesian filter, in which each of the possible values of the hidden state is given a probability of being the true value. This is based on the input from the sensors, which can be updated over time. Ideally you would want to incorporate all sensor information to determine the probabilities. This can be done through a centralized architecture, where everything is calculated in one place. However, this can have a severe impact on complexity. If the state space is continuous, we will have to maintain the complete measurement history of every sensor in the central unit to calculate the probabilities. Secondly, the amount of communication required with the central unit can be very large. Both become a serious problem when a large amount of sensors is involved, thus this method can be unfeasible to implement in practice. In this case a decentralized architecture is a better alternative, where each sensor (agent) calculates their own probabilities and occasionally shares information with other sensors, thus making the network much more scalable (Rosencrantz et al., 2003).

In this thesis we will try to develop a method that determines what measurements will be communicated. The basic concept will be an agent queries another agent for some information; based on his current knowledge he will determine if he should query, if so what agent to query and for what information. Here we will cover one agent querying one agent (the single agent case) and one agent querying multiple agents (the multi agent case) in a discrete environment.

1.1 Related Work

There has been very little work done in this area. Up to now the only real method used has been asking for the most informative measurement about a random part of the environment from a random other agent (Rosencrantz et al., 2003). In this thesis we hope to find improvements for this somewhat naive method.

2 Intelligent Agent Querying

2.1 Problem

We consider an environment where the agents will act to reach a goal. The agents will share measurements (knowledge) of the environment; generally the more knowledge an agent has of the environment the more efficiently it reaches the goal. This allows us to quantify the value a set of measurements as the performance in an environment. We consider the problem in which the queried agents are inactive, meaning their knowledge of the environment does not change, the environment is modeled discrete instead of continuous, one agent asks for measurements and the agent is blank, with no knowledge of the environment beforehand, so we can best see the effects of the queries.

These kind of environments are usually modeled using a Markov Decision Process, which comprises of a set of states S, a set of actions A, a transition function T, conditioning the next state on the current state and action and a reward distribution R, conditioned on the states and actions. An MDP should always satisfy the Markov property $\mu(s_{t+1}|s_t, a_t) = \mu(s_{t+1}|s_t, a_t, s_{t-1}, a_{t-1}, ...)$, which states that the probability of ending up in a next state is only dependent on the current state and action taken and not on any previous states or actions.

In the case of an MDP uncertainties (that which we are trying to reduce) might be what the current state is or what the transition function for each state-action pair is, here we will consider the latter case. This type of uncertainty can be modeled using a Dirichlet distribution, which contains a probability for each possible transition $p(s_{t+1}|s_t, a_t)$, based on the number of times the transition has been observed to s_{t+1} given s_t and a_t and the number of times transitions to other states have been observed given s_t and a_t and the priors for the transition. The amount of times each transition from a state-action pair to some other state is observed are the Dirichlet parameters, from which the actual distribution for that state-action pair can ben calculated. These calculate distributions based on the current observations form the mean MDP; the average MDP given the current knowledge.

So, given agents which have learned the transition functions for a discrete MDP, we try to discover how another agent, with no information about the state space, can efficiently determine what agents and what parts of the state space to query, as to use a minimum amount of queries to gain sufficient knowledge of the state space.

2.2 Environment

The agents will operate in a maze with a fixed start $(0, 0)$ and end state (n, m) and their goal is to get to the end state. The maze is an $n * m$ sized grid with walls in it. Each agent has 4 actions; up, down, left and right. If an agent takes the action to move to a neighboring grid cell and there is no wall, then he will be taken with a high probability to that next cell and with a small probability α will stay in the same cell. If there is a wall blocking that transition, the agent will always stay in the same cell. Every action gets a reward of -1, except transitions in the absorbing goal state, which get 0. The agent should try to minimize this negative reward. The value of the maze is the value of the starting state as calculated by value iteration over the mean MDP, unless specified otherwise. The maze is generated according to the specified size with randomized walls, while ensuring that a path to the goal state is still available (all grid edges are walled by default). The learning agents enter the maze i times following a random policy (the reason for this will be explained later) and store all their observed transitions as the dirichlet parameters. Then the querying agent will ask a particular agent about a particular state in the maze. The answer to this query will be the agents dirichlet parameters for all transitions from that state.

2.3 Methods

There are a number of issues that need to be tackled. Queries are thought of as valuable, because you will always gain some knowledge. So, if the queries are cost-free, there is no incentive to stop querying. Therefor there should be a penalty for each query, to make the problem non-trivial. No matter how we quantify this reward and penalty, there will always be a trade off between the two and a new problem arises: When do you stop querying, as to maximize reward - penalty? Given that we will not perform all possible queries to avoid penalties, we have to find a way to determine which agent and what part of the state space to query.

The problems of what state to query and when to stop querying can also be considered when querying one agent. For simplification we will start by trying to find a solution to these to problems in the single agent case, which will be discussed in section 3. In section 4 we will introduce multiple agents, and try to find out how to determine what agent to query.

3 Single agent case

3.1 Most valuable state

3.1.1 Highest estimated visits

Here we determine what state to query. We want to query the state which will provide us with the greatest increase in knowledge, which is the state that has been visited the most by the other agent, as more visits lead to more observations. We will now outline an approach to determine that state.

The blank agent starts by querying the starting state. Then for each state the estimated amount of visits is to the calculated.

$$E(n(s_{t+1})) = \sum_{i \in Q_t} \sum_{a \in A} \psi^{i_t,a}_{s_{t+1}}$$

Here $n(s_t)$ is the estimated times state s has been visited, $\psi^{s_t,a}_{s_{t+1}}$ is number of times that the transition from s_t, taking action a, to s_{t+1} has been seen (the Dirichlet parameters) and Q_t is the set of states that has been queried at time t (which means you are not taking all states into account, but you don't have information about all states and this should provide a good estimate). Now the state with the highest estimated visits is queried, again the estimates are calculated and the state with the highest estimate is queried, assuming of course it has not been queried already. This process repeats until it is no longer profitable to query the state with the highest estimate.

3.1.2 Results

We will now show the results of the state selection method versus a random state selection. We will calculate value after each query for both methods;

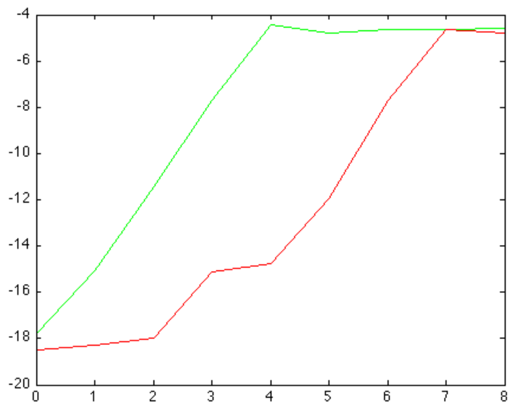

Figure 1: Maze type 1: X-axis = value, Y-axis = queries, Red = random, Green = highest estimated visits

it will be determined by entering the real maze 100 times using the current knowledge (updated with each new transition) and averaging the results. As is to be expected, the results are very dependent on the type of maze, however averaging over all mazes is computationally too expensive, so instead we will try to show the range of results.

In certain cases the results are very good, like the results shown in figure 1. You can clearly see the highest estimated visits methods outperforms random state selection by a large margin. This maze however (like all mazes with similar results) has a lot of dead states, meaning states that are completely walled off and can never be reached. If those are queried, there is of course no real increase in knowledge, as that knowledge will never be useful. Our highest estimated visits methods uses the observed transitions and seeing as no transitions are observed to the dead states, they will be queried last. In these type of cases, there is a definite benefit to be had from using a highest estimated visits method.

The other type of maze is a open maze, meaning there are no dead states. The results in for this type are shown in figure 2. Here you see that initially the random state selection performs slightly better, but after a number queries the highest estimated visits is clearly better. This can be explained by the fact that the highest estimated visits method only queries states next to states that have already been queried, this means that initially the knowledge will only increase for the area around the starting state (remember that the first state queried is always the starting state). The random state selection however, will increase knowledge over different parts of the maze and this might actually be more informative in some cases. After a number of queries a path to the goal state can be found using the acquired knowledge, which causes the value to increase dramatically. This generally happens quicker using the highest estimated visits

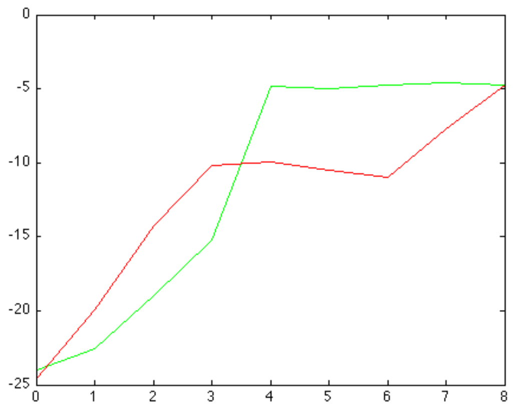

Figure 2: Maze type 2: X-axis = value, Y-axis = queries, Red = random, Green = highest estimated visits

(query 4 in this case), as its state selection is more directed. So we have shown that in a state space with states that are unreachable, our highest estimated visits method performs very well and would definitely be preferable over random state selection. If this is not the case and only a few queries are made, then random state selection can actually perform better. However the highest estimated visits method will quickly gain the knowledge to establish a path to the goal state, which results in a major increase in value and subsequently causes the method to outperform the random state selection.

3.2 Stopping problem

3.2.1 All possible knowledge increases

Here we determine when to stop querying. This can be considered a stopping problem, a very well studied problem. One of the most common approaches to these types of problems is to consider all possible outcomes and then decide if the average increase is worth it the penalty. However in this case all possible knowledge increases can be quite complex, as our priors for the transitions are evenly distributed over all states, thus all increases can be any number of permutations of distributions of parameters over all states. We can limit the total of these parameters to the estimated number of visits, calculated in the previous section, as to not make the number of permutations infinite, but still even in a small grid with a small number of estimated visits the number of permutations can become quite large. This approach of course requires the value of the maze to go up with increases in knowledge, which is not the case in this form, as shown in figure 3.

As knowledge increases, the value of the mean MDP decreases, because

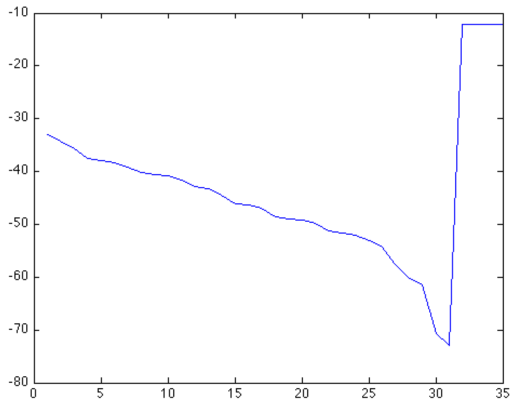

Figure 3: Uniform prior values: X-axis = value, Y-axis = queries

initially there is a $1/(n * m)$ probability of transitioning to the terminal state (due to uniform priors), but as transitions to other states become more likely, because of the knowledge increases, the odds of going to the terminal state are decreased. Once a path of known states to the terminal state has been established the value increases dramatically, far above that of the MDP without knowledge, because there is no real uncertainty in any of the transitions towards the goal.

3.2.2 Informed Prior

We will now work with informed priors; this means the priors are only distributed over a maximum of 5 states (up, down, left, right, self), and this eliminates the problem from the previous section. The transition probabilities to the goal state are no longer reducing with each knowledge increase, because most states have a 0 probability of going to the goal state. It also reduces the number of states that need to be considered when calculating the probabilities and thus limits the complexity of the approach. The value increasing with more knowledge, as opposed to the result from the uniform priors, is shown in figure 4.

However this will basically be distance reduction, as the unknown states will allow transitions to all neighboring states, thus the closer to the goal, the higher the value. Therefore this method is not able to handle detours required by random walls and is unsuitable.

3.2.3 Sample from Gamma

Now we will draw samples from the Dirichlet distribution, using an existing sample from Gamma method. The Dirichlet distribution is a specific two pa-

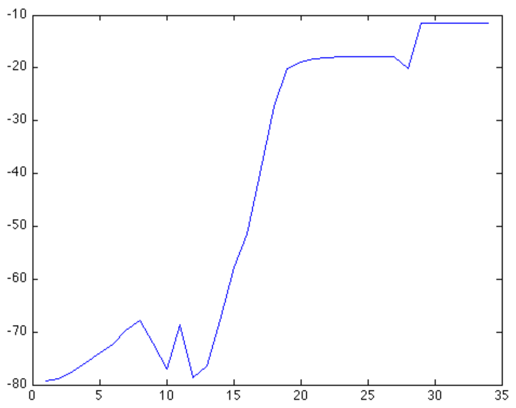

Figure 4: Informed prior values: X-axis = value, Y-axis = queries

rameter case of the Beta distribution, which is related to the Gamma distribution and because of that relation we can use a Gamma sampling method on the Dirichlet distribution. We will use the sample MDP as an estimate of the possible increases in value. Drawing samples from an MDP basically entails selecting transition probabilities to states for each state-action pare that sum to 1 and are distributed randomly according to the transitions you have already observed. This means the approximate relations between the transitions (according to the agents observations) are maintained, while introducing some randomness and the more observations there are for a transition, the less influence the randomness has. A good way to sample a random vector $x = (x_i, .., x_K)$ from a Dirichlet distribution with the parameters $\alpha = (\alpha_i, ..., \alpha_K)$ is to sample $\gamma_i = Gamma(\alpha_i, 1)$ for each parameter (observed transitions) α_i and then set

$$x_i = \frac{\gamma_i}{\sum_{j=1}^{K} \gamma_j}$$

Here $Gamma(\theta, k)$ is an existing gamma sampling method created by Ahrens and Dieter (Ahrens & Dieter, 1982). The benefit of drawing samples versus solving the mean MDP, is that in the mean MDP you are trying to find a policy that will perform best on average on all possible true MDPs, but when drawing a sample, you simple try to find a policy for one specific instantiation, which allows for much higher rewards. This produces much better results and more importantly, give a much better indication of potential value of the true MDP. Of course drawing only 1 sample, means you are very dependent on how the transitions where sampled, but you can obtain an average over many drawn samples. You should now draw enough samples to allow the average to converge, but not too many, as to prevent excessive computational cost. The simplest way to check if an average has converged is to draw another sample and see if the average changes. If it does not, it is either stable enough or the

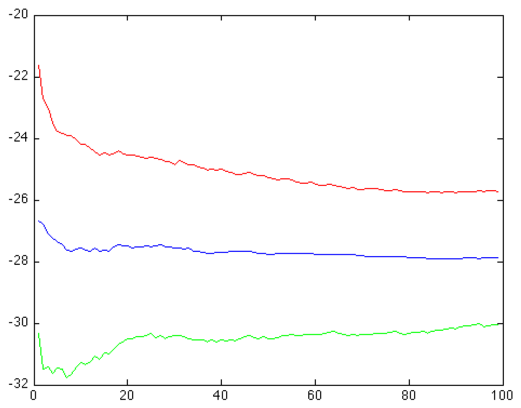

Figure 5: Convergence of samples with 0 queries: X-axis = number of samples taken (scale = 1:10), Y-axis = value, Red = top 10%, Blue = average, Green = bottom 10 %

sample just happens to have the same value as the current average. To filter out these latter cases, our current requirement is that the average does change with more than 0.01 for 20 times. Different levels of knowledge will require a different amount of samples to converge. To clearly show that samples from an MDP with more queries made (i.e. more knowledge), will vary less in value and thus converge quicker, we offer the following examples. Figure 5 has no knowledge of the MDP at all, thus the samples vary a lot, meaning it takes a long time to converge. The result in figure 6 hardly varies at all; this is because after 24 queries, a path is established to the goal state, meaning the starting state and goal state are connected via inter-connected states of which knowledge is available. After this the value of the sample will not converge quicker with more queries, nor will the value change (unless a shorter path is available). The sample and mean value will be almost the same at this point. Because of how much knowledge there is of the queried states (1000 entries in the maze), the samples will vary very little and be very close to the mean value. All unqueried states (which will vary in with each sample) will not on average provide a better path and thus the path in the sample MDP used is the same as in the mean MDP and the probabilities are the same, thus the value will be the same. The mean and sample value approaching each other is shown in figure 7. Figure 7 also clearly shows the potential influence of the penalty. We will consider a query non-profitable when:

$$MeanValue > SampleValue - Penalty$$

The higher the penalty, the quicker the querying will be stopped.

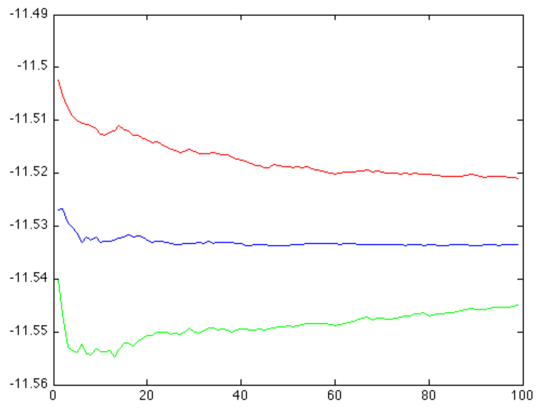

Figure 6: Convergence of samples with 24 queries: X-axis = number of samples taken (scale = 1:10), Y-axis = value, Red = top 10%, Blue = average, Green = bottom 10 %

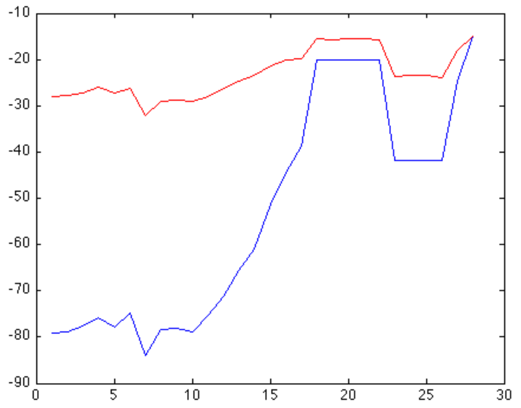

Figure 7: Sample and mean value converging: X-axis = value, Y-axis = number of queries, Red = sample value, Blue = mean value

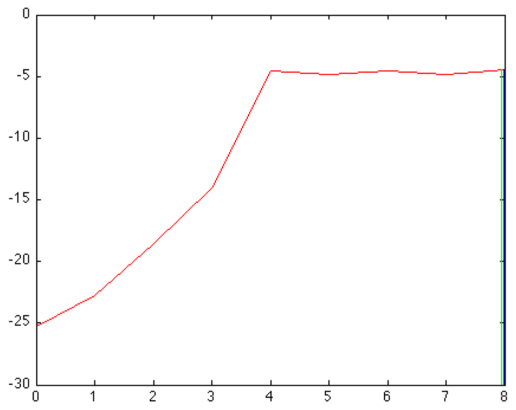

Figure 8: Penalty 0: X-axis = queries, Y-axis = value - querycost, Red = results, Blue = Oracle stoptime, Green = Sample from gamma stoptime.

3.2.4 Results

Here we will show the results for the stopping time, given different penalty sizes (state selection is the same for all methods; highest estimated visits). We will calculate value after each query; it will be determined by entering the real maze 100 times using the current knowledge (updated with each new transition), averaging the results and subtracting the penalty for the number of queries made. These results are then graphed and vertical lines will show the sample from gamma stop time and a comparison method stop time. The comparison method we will use is the optimal stopping time, calculated by a method we call oracle. What the oracle does is calculate the true value for each query, using the same method as the one that is used in the results and then selects the optimal stopping point. The results will be shown for different penalty sizes, as they should influence the stopping time. There are 4 types of results over the range of possible penalties, we will show and discuss them bellow.

In figure 8 the penalty is set to 0 (no penalty). Here the sample from gamma method stops at the same time the oracle does. This is a good results of course, but not very surprising; with no penalty, the problem is trivial. As there is no incentive to stop querying, both methods will make all the queries possible, resulting in the highest value when entering the maze.

In figure 9 the penalty is set to -3 (low/medium penalty). Again the sample from gamma method stops at the same time the oracle does. This is a really good result, because as the graph shows, this is no longer a trivial problem. Here the method does exactly what it is supposed to do.

In figure 10 the penalty is set to -6 (high penalty). With such a high penalty the sample from gamma method stops later than the oracle does. It stops a local optimum, but the oracle never even makes a query with this high penalty, which

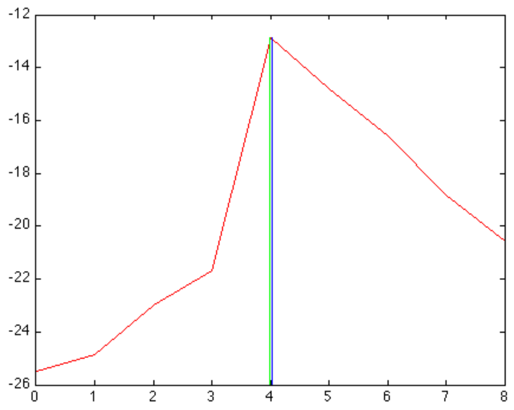

Figure 9: Penalty -3: X-axis = queries, Y-axis = value - querycost, Red = results, Blue = Oracle stoptime, Green = Sample from gamma stoptime.

leads to a global optimum. This can be explained by the fact that the difference between the sample and mean value is big enough relative to the penalty size, to say that it is profitable to make a query, but the method does not take into account the expected number of queries (i.e. penalties) required to make that improvement in knowledge.

In figure 11 the penalty is set to -12 (very high penalty). Here our sample from gamma method stops at the same time the oracle does again. This is a good result, because the method is now capable of picking up the fact that no query will ever be profitable with such a high penalty.

In general our method performs very well, it only selects sub optimal stopping points in the high penalty interval ('high' is of course relative to the reward and maze size, the exact interval is not known). It does then still select a local optimum and when the method would consider the number of queries required, this problem would probably be fixed.

4 Multi-agent case

4.1 Agent selection

4.1.1 Soft-max

Ideally you would use a soft-max type selection, which is a very well studied algorithm, for the agent selection, based on the amount of knowledge they have. It would still allow for some exploring, as it has some randomness, but it picks the more knowledgeable agents more often. There are some problems with this method though; we need a way to define when one agent is more knowledgeable than another. Query results return observations and the amount of observa-

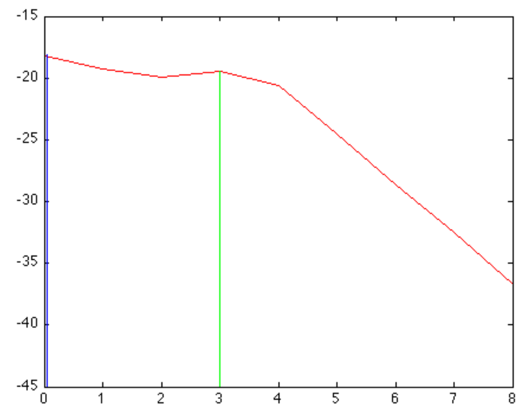

Figure 10: Penalty -6: X-axis = queries, Y-axis = value - querycost, Red = results, Blue = Oracle stoptime, Green = Sample from gamma stoptime.

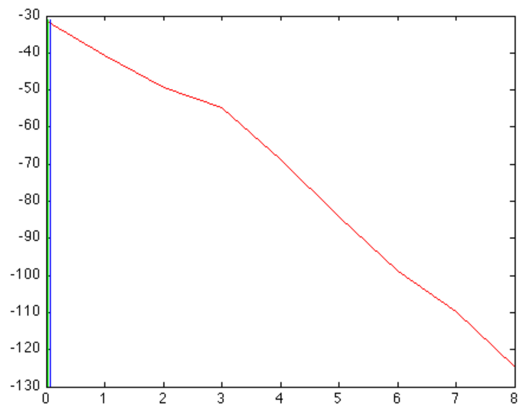

Figure 11: Penalty -12: X-axis = queries, Y-axis = value - querycost, Red = results, Blue = Oracle stoptime, Green = Sample from gamma stoptime.

tions is equal to the number of visits. Seeing how we want to maximize the number of observations we get, we want to query the agent who has visited the state the most. However simply comparing the querying results is not going to work, because all queries are of different states and relative visits to different states might vary due to policy or location in the maze. The only proper way to compare the knowledge of agents is to compare the number of entries in the maze, however you do not have that information. The only way to get that information, is to query the starting state, but querying the starting state for each of the considered agents, is simply not profitable, as it is unlikely to give us more information after the first query and we would like each query to significantly contribute to our knowledge. Thus we need a method that allows us to define the expected reward in a way that does not require such explicit knowledge information.

4.1.2 Follow the perturbed leader

Here we will not try to predict the expected reward from each agent, but rather define a loss for each query made. The loss will be defined as follows:

$$l_t = \frac{1}{1 + c_t}$$

where c is the total number of observations received from the query. If a query is made and no information is returned, the loss is 1, if perfect information is returned (i.e. $c = \infty$) the loss is 0. This loss is added to the total loss for that agent L_t^n; these updates are only performed when the agent is queried. Z_t^n is the random component added to allow for some exploration, it is bounded between $\{0, m * t\}$, where t is the number of queries made and m the number of agents. The agent to be queried is traditionally defined by:

$$V_t^n = L_t^n + Z_t^n$$

where the agent with the minimum value for V_t^n is selected at time t. Meaning you (with some randomness) select the agent which has returned the least amount of loss. This method is generally known as follow the perturbed leader. It requires the loss to be bounded between 0 and 1, which it currently is. However there are a large number of observation each agent makes, leading to very small loss updates each time. So l_t is multiplied by 10000, in which case it still hardly ever is above 1, and produces far less random results, as the influence of the loss becomes greater relative to Z_t^n.

4.1.3 Results

We will now show the results from the follow the perturbed leader method compared to a random agent selection method. We will calculate value after each query; it will be determined by entering the real maze 100 times using the current knowledge (updated with each new transition) and averaging the results. The state selection and stopping time are the same for both methods.

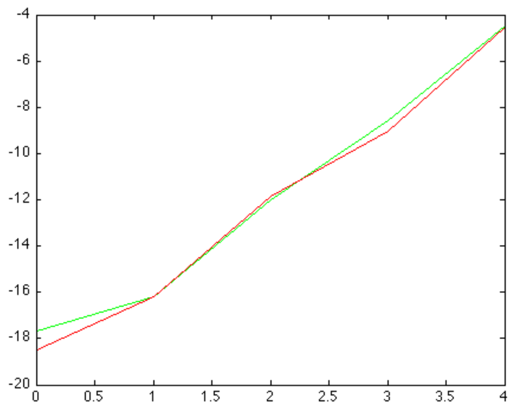

Figure 12: Agent selection value: X-axis = queries, Y-axis = value, Red = random, Blue = follow the leader

As you can see the results for the two methods shown in figure 12 are hardly any different, there is only a marginal benefit for the perturbed leader algorithm. This result is really surprising, especially if you consider that there were 2 learning agents available for querying; the first had entered the maze once and the second 10000 times. The random querying agent selected the first learning agent for 3 out of 4 the queries, but it performed almost exactly the same as the follow the leader querying agent, which selected first learning agent only once. However this result can be explained; due to the fact that there is little randomness in our maze, this one trip actually provides enough information. The agent has only observed a couple of transitions, but has gone from start to goal, and this is enough to establish the correct tendencies in the probabilities (only because there is little randomness). This allows for optimal policy calculation by the querying agent, based on a couple of observations in each state.

This result then does not mean that the agent selection is faulty, but that agent selection is not very relevant with this problem. This of course can be solved by using a harder problem (more randomness, more states). To actually demonstrate the result of the agent selection, we will try to show that learning agents that have visited the maze more, will be selected more by the querying agent. The amount of entries is selected randomly between 0 and 10000 for 4 agents. Figure 13 shows the number of maze entries (knowledge) of 4 different agents and figure 14 shows the number of times those same agents where queried. The results seem to be quite good, but this visual correlation proves very little. A possible improvement would be even less randomness in the selection, as there is a big difference between the entries of the agents, and consequently some agents will never be the best to select. For example, in figure 13 agent 0 knows very little, but is still queried 4 times.

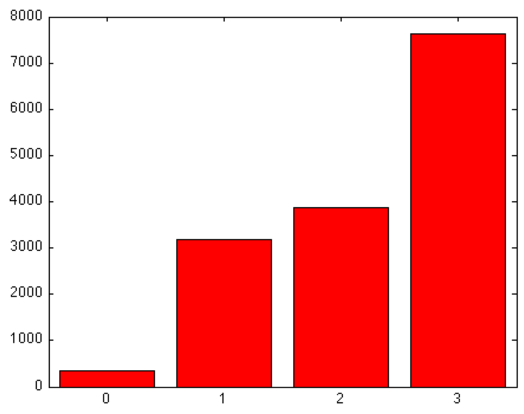

Figure 13: Agent knowledge: X-axis = agent, Y-axis = number of entries into the maze (i.e. knowledge)

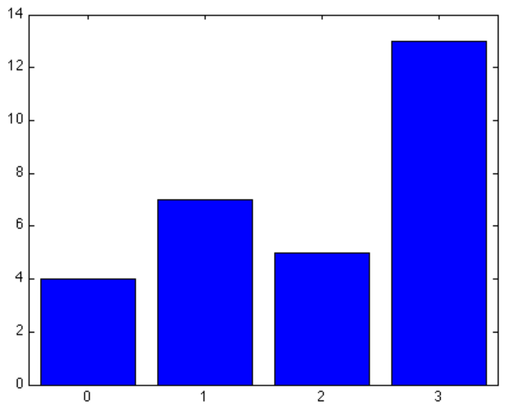

Figure 14: Agent selection: X-axis = agent, Y-axis = number of times selected

4.2 Combined state/agent selection

4.2.1 Best agent for best state

One of the problems with the follow the perturbed leader is that it does not take the location of the state that is being considered for querying, relative to the location of states that that agent was queried for earlier, into account. If an agent knows a lot about a state (has visited it a lot), then he probably knows a lot about neighboring states, especially if there is a known transition with a high probability. We will try to exploit that fact in this approach and try to estimate the number of visits to a specific state by an agent.

First we will find set of reachable, non-queried states from our current knowledge, which we will call R_t. Reachable means there that particular state and the starting state are connected via inter-connected states of which knowledge is available (i.e. there is a path to that state). Then for all states about which the agent was queried, we will call this set O_t^n, we determine the optimal path to every element in R_t, using value iteration (using knowledge of transitions from all agents). If any state in the path is an element in O_t^n, we will discard the path, because you have more accurate estimate of the amount of visits than can be provided by the path. Then for all remaining paths, we will multiply the number of visits to O_t^n by all transition probabilities in the path, as to get an estimate for the number of visits to $R_{t,i}^n$ from $O_{t,i}^n$. Then we sum over all path estimates from O_t^n to $R_{t,i}^n$, to get a total estimate of the number of visits. We can use this, to not only select the best state for that agent (with the highest estimated number of visits), but repeat this for all agents and selecting the best state/agent combination.

This method does have some assumptions, like the fact that the agent was trying to get to $R_{t,i}^n$, when in fact the policy might dictate otherwise. But this upper bound is used for all states and does not show any negative effects. Also there is no randomized element, and thus no exploring among agents, but as shown with the follow the leader method, this might lead to better results.

4.2.2 Results

We will not show the results for the state selection, as the method is in principle very similar to highest estimated visits and thus the results are also very similar. However the agent selection here is very different and the results for it are shown below.

We will use the same method of comparison used for the follow the leader method. To demonstrate the result of the agent selection, we will try to show that learning agents that have visited the maze more, will be selected more by the querying agent. The amount of entries is selected randomly between 0 and 10000 for 4 agents. Figure 15 shows the number of maze entries (knowledge) of 4 different agents and figure 16 shows the number of times those same agents where queried. The results here look very good, as the agent that have visited the maze the most are queried the most by far. The other two agents are queried only once for exploration, after which they are never queried again. The lack

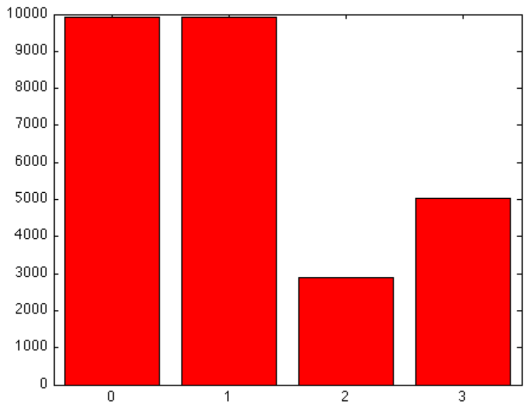

Figure 15: Agent knowledge: X-axis = agent, Y-axis = number of entries into the maze (i.e. knowledge)

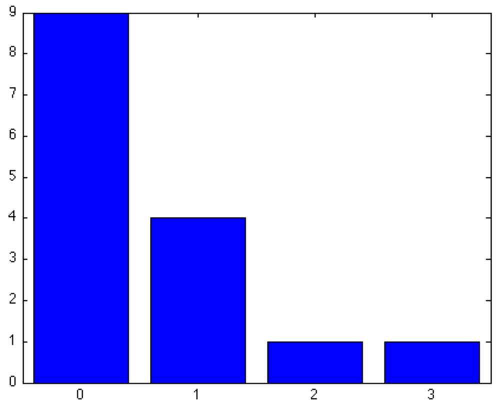

Figure 16: Agent selection: X-axis = agent, Y-axis = number of times selected

of randomness in this method seems to improve performance relative to the old agent selection method, showing that the assumption of exploration being necessary only once, was a correct one. This is probably because the amount of visits to states for an agent are so closely related; if an agent has visited one state many times, he must have visited other states many times in order to get there.

5 Conclusions

We have shown that we can produce a good solution for all described problems. However some solutions only showed significant improvement under specific conditions: The stopping problem solution can not yet deal with high penalties, the state selection provides the best results if there are states that are never visited and the agent selection requires a more complex problem to be tested thoroughly. Despite these shortcomings, all methods still perform reasonably when these conditions are not met and overall performance is good. Thus the results are definitely encouraging, but the methods still require some work and expansions before they are truly applicable (see Future Work).

6 Future work

Future work will consist of addressing the problems with the approaches described in the conclusions sections, most significantly allowing the sample from gamma method to deal with high penalties and testing in more complex environments.

In order for these method to be applicable in real situations, they should expanded to be capable of handling continues environments (using particle filters instead of Dirichlet distributions), learning agents who's knowledge is non-stationary (they continue to move in the environment) and multiple agents querying each other, instead of an explicit distinction between learning agents and querying agents.

7 References

M. Rosencrantz, G. Gordon, and S. Thrun. (2003) Decentralized sensor fusion with distributed particle filters. *In Proc. of UAI.*

J.H. Ahrens, and U. Dieter (1982) Generating gamma variates by a modified rejection technique. *Communications of the ACM* 25, 47-54.