An Accumulative Exploration Method for Reinforcement Learning

Edwin de Jong
Artificial Intelligence Laboratory
Vrije Universiteit Brussel
Pleinlaan 2, B-1050 Brussels
edwin@arti.vub.ac.be

Abstract

Agents in Multi Agent Systems can coordinate their actions by communicating. We investigate a minimal form of communication, where the signals that agents send represent evaluations of the behavior of the receiving agent. Learning to act according to these signals is a typical Reinforcement Learning problem. The backpropagation neural network has been used to predict rewards that will follow an action. The first results made clear that a mechanism for balancing between *exploitation* and *exploration* was needed. We introduce the *Exploration Buckets* algorithm, a method that favors both actions with high prediction errors and actions that have been ignored for some time. The algorithm's scope is not restricted to a single learning algorithm, and its main characterics are its insensibility to large (or even continuous) state spaces and its appropriateness for online learning; the Exploration/Exploitation balance does not depend on properties external to the system, such as time.

Introduction

When designing Multi Agent Systems (MAS), diminishing complex communication increases generality. Agents that use complex communication inherently depend on mutual knowledge. In previous work (De Jong, 1997), a framework for MAS design has been described in which a minimal form of communication is used; the only messages that agents send are numbers between 0 and 1 that represent evaluations of the recipient's behavior. These signals are meant to provide a means for agents to learn to act in unknown environments. To this end, an agent will have to learn to choose the actions which, depending on the current state, will cause other agents to send it high evaluations. Coordination signals are sent by agents that know their environment. These agents can be seen as teachers of a learning agent.

The two main tasks that have to be performed to create a MAS according to these ideas are defining coordination signals and implementing a learning agent. The first task cannot be seen separate from the task the MAS will be used for. In contrast, the second task should ideally depend as little as possible on the application domain. The task a learning agent is presented with is a typical example of a *Reinforcement Learning* (RL) problem, where the rewards correspond to the evaluation signals that will be received.

In RL, the learner chooses actions which affect the environment and yield rewards, which are high if the action is considered appropriate. Since these rewards do not contain information about the *direction* of the error in the action vector, RL refers to a class of *semi-supervised* learning problems. See (Kaelbling et al., 1996) for a review.

The scope of this design method is limited by the power of existing learning algorithms. As a testcase, it has been applied to the *Pursuit Problem*, first described by (Benda et al., 1988). The first task, defining appropriate coordination signals, has been accomplished satisfactorily, see (De Jong, 1997) for an account. Here, we describe the problem that had to be faced for solving the second task, i.e. implementing a reinforcement learning agent. The main contribution of this paper is a solution to the non-trivial problem of finding an adequate trade-off between *exploring* the environment to and *exploiting* the knowledge gained so far by selecting actions that yield high rewards. In a single-state environment, this problem reduces to the *k-armed bandit problem*, for which formally justified solutions exist; in the general multi-state, delayed-reinforcement case that we are concerned with here, no theoretically guaranteed methods can be applied (Kaelbling et al., 1996).

The Pursuit Problem

In our version of the pursuit problem, based on (Stephens and Merx, 1990) one prey and several predators are placed in a 30 x 30 grid. The prey is captured when the four orthogonal positions around it are occupied by predators. We use Stephens and Merx's efficiency measures. The prey selects its actions by randomly choosing between moving left, right, up or down, and staying at the current position, but does not consider any occupied positions. In the experiments described here, the minimum number of 4 predators is used. Three of these predators send coordination signals to their two neighbours, which represent evaluations of their coordination with these neighbours. The fourth predator is a learning agent. Since neighbours are determined by comparing angles with the prey, the learning agent will normally receive two coordination signals; one from both neighbours. In exceptional cases, when several predators have the same angle (but different distances) to the prey, the learning agent may receive a different number of signals. A learning agent interprets the average of the incoming signals as its reward. This setup contrast with the more commonly used environmental rewards, and provides a way for agents to put learning agents to their use. The coordination signals are computed as follows:

$$\begin{aligned}
signal &= Eval(A_2, A_1) + Eval(A_3, A_1) \\
\phi_{opt} &= \frac{2\pi}{\#predators} \\
Eval(A_p, A_q) &= 0.96 \cdot (1 - \frac{A_p.d + A_q.d}{2r\sqrt{2}}) + \\
0.04 \cdot (1 &- \frac{|\phi_{opt} - ((A_p.phi - A_q.phi) \bmod 2\pi)|}{E_{max}}) \\
E_{max} &= 2\pi - \phi_{opt}
\end{aligned}$$

where d is the distance to the prey, ϕ the angle between two predators and E_{max} the maximal error in this angle.

Finding a Suitable Reinforcement Learning Algorithm

A common algorithm for reinforcement learning problems is Q-learning (Watkins, 1989). This technique found its way into the domain of multi agent systems, see e.g. (Sen and Sekaran, 1995), (Sandholm and Crites, 1995). Q-learning involves storing valuations for each possible action in every state. This is only feasible in problems with a small number of possible states and actions. For the problem at hand, the inputs consist of the polar coordinates of all predators relative to the prey, and the cartesian coordinates of the prey. Although the number of possible actions is only 5 (moving in the compass directions and staying), the number of combinations of state and action already amounts to $5 \cdot \frac{900!}{895!} / = 2.9 \cdot 10^{15}$ [1] To avoid this combinatorial explosion, several methods for *generalization* exist. In (Sutton, 1996), a function approximator is applied to discretize a continuous state space, so that algorithms based on storing information about combinations of actions and states can be used all the same. Another approach to generalization is to use neural networks that interpolate between states situations (and possibly, but not in our case, actions). Of the many neural network architectures that are described in the literature, quite a lot use binary-valued rewards. Since the succes of a predator that bases its actions on the coordination signals depends quite critically on small differences in these values, binary rewards are not sufficient here. A possible solution to this would be to train the network on differences between subsequent rewards; increases can then be used as good rewards (1), and decreases as bad ones (0). This scheme is not investigated here, since its assumptions (i.e. a good action will always result in an increase of the reward) make it less generally applicable.

As a learning algorithm, we use the common backpropagation network, see (Rumelhart et al., 1987). The inputs are the polar positions of all predators (including the learning agent) and the cartesian position of the prey, and a bias input neuron. The network has one hidden layer of 20 neurons. The output layer contains 5 neurons, each representing an action. Each output neuron should learn to predict the reward its corresponding action would return on execution. Only the neuron corresponding to the action that was executed changes its weights to learn the received reward. The learning agent was designed to be used in the abovementioned framework for coordination in multi agent systems. The goals of this framework include that the learning agent should in principle be independent from the application domain. As a consequence, the intervals within wich the values of the inputs may vary are unknown during the construction of the network. The problem of scaling the input values is handled by computing the mean and variance of each input as if its values were normally distributed. Inputs within $[\mu - 2 \cdot \sigma..\mu + 2 \cdot \sigma]$ are linearly scaled and translated to fit within [0..1]; inputs outside this interval are replaced by the scaled and translated value of their nearest bound.

[1]This is the number of actions (4 directions + resting) multiplied by the number of ways in which the 5 recognizably different agents (the agent, the prey and the 3 other predators) can be placed on the 30 x 30 grid.

The Exploration Buckets Algorithm

One thing all RL algorithms have in common is the need for a strategy that defines when to explore and when to exploit. The motivation for an agent to learn is to be able to exploit the knowledge it gradually builds up to choose actions that yield high rewards. But when no attention is paid to exploring actions with lower expectated rewards, underrated actions will never be recognized as such. The task of an Exploration / Exploition algorithm is to find a good balance between these. According to (Thrun, 1992), exploration mechanisms can be divided into *undirected* and *directed* strategies. Within the class of directed strategies, which are generally more successful than undirected ones such as random or stochastic exploration, three types are encountered: stategies based on counters, on recency and on errors. We are interested in finding an exploration mechanism that possesses the following properties:

- It can be used for online learning
  (Thrun, 1992) describes several algorithms that distinguish between the *learning phase* and the *performing phase*. As (Sutton, 1993) notes, in order to obtain *learning* systems instead of merely *learned* ones, online learning is required. This refers to learning that is done during the operational phase of a system. Online learning agents that have to operate in a changing environment should never stop learning, and should therefore avoid to distinguish between learning and operating phases. (Wilson, 1996) gives a systematic account of the variables on which the balance between exploration and exploitation may depend. Similarly, dependency of this balance on the running time is to be avoided when designing an online learning agent. This excludes the popular exploration mechanisms that depend on the slowly decaying influence of randomness , such as Gullapalli's SRV unit (Gullapalli, 1990), which implements exploration by adding random numbers from a zero-mean Gaussian distribution to a neuron's output and gradually decreasing the variance of this distribution over time. A further requirement for an online learning agent is limited computational complexity.

- It is not restricted to small problems
  As we saw before, even an apparently simple domain such as the pursuit problem may lead to forbiddingly large search memory requirements when all combinations of states and actions are seen as separate. Exploration mechanisms that depend on storing values for each such combination therefore cannot be used without substantial changes.

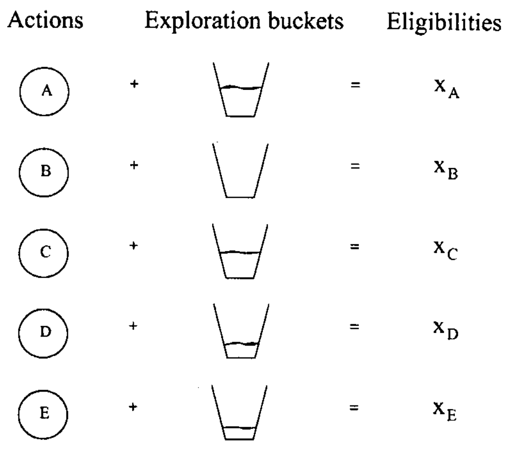

Figure 1: Exploration buckets

Many approaches to exploration feature some of these requirements, but none was encountered that satisfies all. (Sutton, 1990)' introduces exploration bonuses that keep a memory for each state **x** and each action **a** of "the number of time steps that have elapsed since **a** was tried in **x** in a real experience". This interesting recency-based strategy is restricted to problems with small numbers of states and actions. In (Cohn, 1994), an algorithm based on Federov's Theory of Optimal Experiment Design, see (Federov, 1972), is described. This algorithm can be very useful for choosing expensive experiments, such as drilling oil wells, but will generally be too slow for online learning. In (Dayan and Sejnowski, 1996), an estimation of the Bayesian balance between exploration and exploitation is computed. This method is limited to systems that keep a model of state transitions. We introduce the Exploration Buckets algorithm, which meets all of the above requirements. The exploration bucket of action k is updated as follows, assuming action i was chosen:

$$\begin{aligned}
&for\ k\ =\ i:\\
B^k_{t+1} &= 0\\
E^k_{t+1} &= |predicted\ reward\ -\ actual\ reward|\\
&for\ k\ \neq\ i:\\
B^k_{t+1} &= B^k_t + \alpha\cdot\frac{(0.9\cdot E^k\ +\ 0.1\cdot\frac{\Sigma^n_{i=1}E^i}{n})}{n}\\
E^k_{t+1} &= E^k_t
\end{aligned}$$

where B^k is the content of the exploration bucket of action k, and E^k the error between real and predicted reward when action k was last executed. The exploration bucket of the chosen action is emptied. Its content is not transferred to other buckets. The buckets of the other actions are increased by a value proportional to the error in the prediction of the reward when that action was last selected. The average previous error of all actions, multiplied by a small factor, is added to this to ensure exploration of all actions when the general performance is decreasing due to a changed environment. In all experiments reported on here the influence of the exploration bonuses, α, was 1. The eligibility of an action is the sum of its predicted reward and its exploration bucket. The action with the highest eligibility is selected for execution. The resulting behaviour is that exploration is stimulated by these two aspects:

- the last time the action was taken, the prediction of the corresponding reward was not very precise (error-based)
- the action has not been tried for a long time (recency-based)

The effect of accumulating exploration bonuses is that even an action with small bonuses will eventually be selected. In environments where the rewards are highly stochastic, it can be foreseen the algorithm will keep exploring when no more benefit is to be gained. Since the exploration mechanism avoids storing information (such as the unpredictability of rewards) about particular states, a solution to this is not obvious. An important property of this exploration policy is that it does not depend on quantities external to the system, such as time.

Results

Qualitatively, the results of the learning algorithm can be described as follows. At the beginning, all weights of the network are random; some actions are consequently estimated to yield high returns, and others low returns. Since these predictions are initially random, the high errors result in high exploration bonuses and hence much exploration. As a result of this exploration, the predictions soon converge to a situation where all actions yield predictions approaching the average reward. Then the agent learns to distinguish more subtle differences in the environment, resulting in more accurate predictions. Finally, enough detail is reached to distinguish between the rewards of the different actions in a situation. This is particular to the definition of the rewards. Since the evaluation of the learning agent's behavior is done by continually judging the current state, not its action, the rewards are strongly correlated to the previous reward, and only implicitly depend on the chosen action. The very small differences in returns that are the effect of

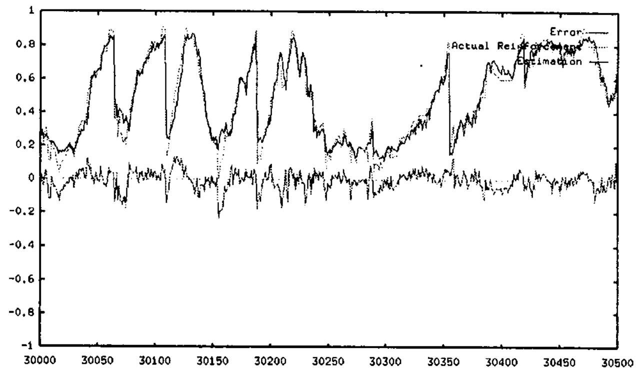

Figure 2: Upper two lines: network's prediction and real reward as a function of the number of moves. Lower line: error. The network has learned to predict the rewards it will receive quite accurately.

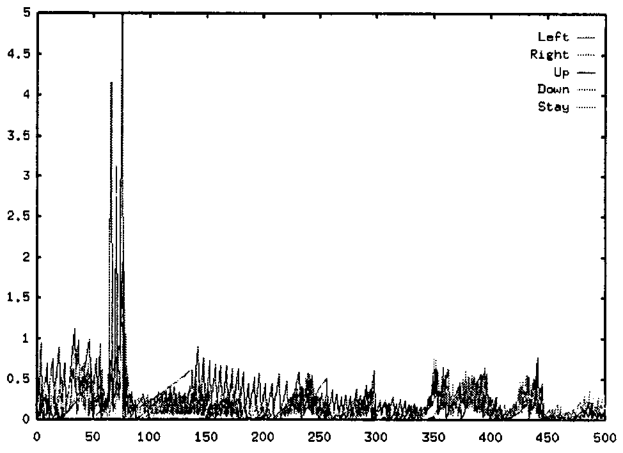

Figure 3: Exploration bonuses of the 5 actions peak when the network makes large errors in its predictions (see figure 4). The oscillating exploration value between 150 and 300 solves the low initial predictions of 'moving left'. The high prediction errors quickly fill this action's exploration bucket, thus causing the system to select the action frequently and learn its real value. Later on (not shown), the influence of exploration drops.

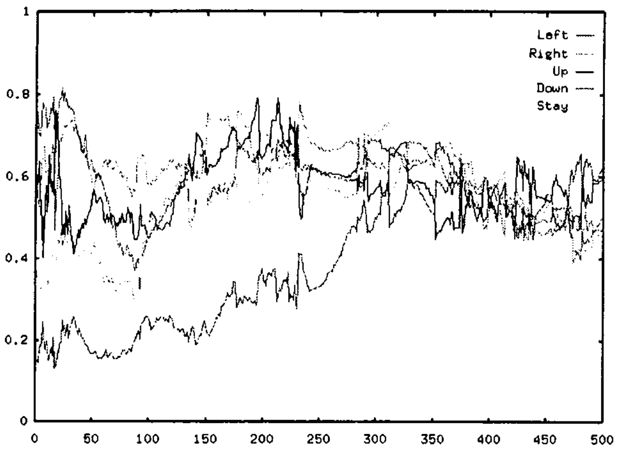

Figure 4: Exploration sees to it that the initial low predictions of action 'move left' are quickly increased.

this make this learning problem a rather difficult one. These first experiences with the Exploration Buckets algorithm show that it performed better than *Roulette* exploration, see (Goldberg, 1989), which was used as a comparison. With Roulette exploration, the outputs of the network are used as relative probabilities to choose the corresponding action. Using a delayed rewards algorithm, see (Sutton, 1988), could improve this. The network using the Exploration Buckets algorithm that has been presented can thus successfully be used for finding a balance between exploration and exploitation for reinforcement learning problems with direct rewards.

Conclusions

We investigated the possibilities for exploration in a particular a reinforcement learning problem. The *Exploration Buckets* algorithm was introduced. The algorithm was tested by applying it in a network for an agent that has to learn to become a predator in the Pursuit Problem. The results were positive, and have been compared to the 'Roulette' strategy. A limitation is its inappropriateness for highly stochastic environments. Attractive properties of the method are:

- The balance between exploration and exploitation does not change as a function of properties external to the system, such as time. This is essential when designing agents that have to operate in changing environments.
- The time complexity is low. The space complexity is linear in the number of actions, and independent of the number of possible states, also resulting in a low complexity for the problem at hand.

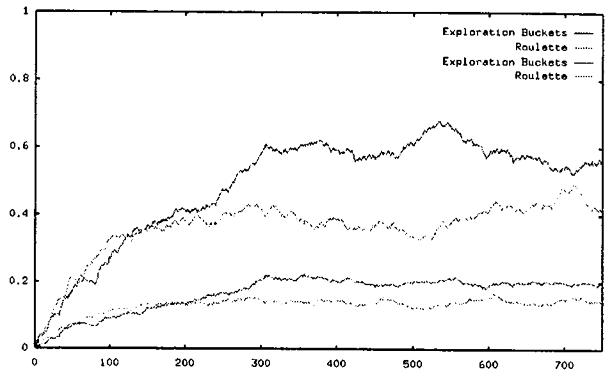

Figure 5: Resulting efficiencies of the pursuit problems. The topmost and the lower middle line are the efficiencies of the Exploration Buckets network, the other two lines are the same 2 efficiencies for Roulette exploration.

- It assures that actions with bad reward predictions are selected frequently so they can as to improve their predictions; meanwhile it has a minor influence on exploitation once the agent has learned its environment; these are results of the error-dependence of the method. The use of buckets that retain exploration bonuses when actions are not selected yields the recency-based character of the algorithm. In future work, we plan to combine the exploration buckets algorithm with other learning methods and compare the resulting system to other explore/exploit strategies.

Acknowledgments

The author wants to thank Walter van de Velde for valuable suggestions and Bart de Boer for reading and commenting on an earlier version. This research was partially funded by EU Telematics Information Engineering projects MAGICA and GEOMED. GEOMED (IE 2037) stands for 'Geographical mediation systems'. MAGICA (IE 2069) stands for 'Multi-Media Agent-based Interactive Catalogs'. Additional funding has been provided by the OZR contract for research on *models for coordination and competition in multi agent systems.*

References

Benda, M., Jagannathan, V., and Dodhiawalla, R. (1988). On optimal cooperation of knowledge sources. Technical Report BCS-G2010-28, Boeing AI Center.

Cohn, D. A. (1994). Neural network exploration using optimal experiment design. Technical Report AI Memo 1491 and CBCL Paper 99, MIT AI Lab and Center for Biological and Computational Learning Department of Brain and Cognitive Sciences.

Dayan, P. and Sejnowski, T. J. (1996). Exploration bonuses and dual control. *Machine Learning*, 25.

De Jong, E. D. (1997). Multi-agent coordination by communication of evaluations. In *Proceedings of Modeling Autonomous Agents in a Multi Agent World MAAMAW '97.*

Federov, V. (1972). *Theory of Optimal Experiments.* Academic Press, New York.

Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison-Wesley.

Gullapalli, V. (1990). A stochastic reinforcement algorithm for learning real-valued functions. *Neural Netw.*, 3:671–692.

Kaelbling, L. P., Littman, M. L., and Moore, A. W. (1996). Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4.

Rumelhart, D., McClelland, J., and et. al. (1987). *Parallel distributed processing; explorations in the microstructure of cognition*, volume 1-2. MIT Press.

Sandholm, T. W. and Crites, R. H. (1995). On multiagent q-learning in a semi-competitive domain. In G. Weiss, S. S., editor, *Adaption and Learning in multi-agent systems*, pages 191–205, Berlin, Heidelberg. Springer Verlag.

Sen, S. and Sekaran, M. (1995). Multiagent coordination with learning classifier systems. In G. Weiss, S. S., editor, *Adaption and Learning in multi-agent systems*, pages 218–233, Berlin, Heidelberg. Springer Verlag.

Stephens, L. M. and Merx, M. B. (1990). The effect of agent control strategy on the performance of a dai pursuit problem. In *Proceedings of the 1990 Distributed AI Workshop.*

Sutton, R. S. (1988). Learning to predict by the methods of temporal differences. Technical Report TR87-509.1, GTE Laboratories.

Sutton, R. S. (1990). Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *Proceedings of the Seventh Int. Conf. on Machince Learning, pp. 314-321*, pages 216–224. Morgan Kaufmann.

Sutton, R. S. (1993). Online learning with random representations. In *Proceedings of the Tenth Int. Conf. on Machince Learning, pp. 314-321*, pages 314–321. Morgan Kaufmann.

Sutton, R. S. (1996). Generalization in reinforcement learning: Successful examples using sparse coarse coding. *Advances in Neural Information Processing Systems*, 8:1038–1044.

Thrun, S. (1992). *The role of exploration in learning control.* Van Nostrand Reinhold.

Watkins, C. (1989). Learning from delayed rewards. *Ph.D. Thesis.*

Wilson, S. W. (1996). Explore / exploit strategies in autonomy. In et.al., P. M., editor, *Proceedings of the fourth international conference on simulation of adaptive behavior. From animals to animats 4.* The MIT Press.