Looking for Scalable Agents

Olivier Buffet BUFFET@LORIA.FR

Alain Dutech DUTECH@LORIA.FR

LORIA/INRIA, BP 239, 54506 Vandœuvre-lès-Nancy, FRANCE

Introduction

Reinforcement Learning intends to ease and possibly to perform automatically the design of systems such as software or robot agents. An important aspect is the ability of learning agents to adapt to their environment and to the task they have to accomplish. This kind of learning is unfortunately restrained by problems like combinatorial explosion of the state space that limits the number of sensors or objects an agent can reasonably deal with, especially in the case of Multi-Agent Systems.

Considering Markov Decision Processes, different solutions exist to overcome the difficulties related to large state spaces: hierarchical structures (Parr, 1998) or factored representations (Kearns & Koller, 1999) of the agent's behavior for example. Nevertheless, these tools require manual preparations before going through the learning step, and result in an agent designed for a specific task.

The work presented here intends to define agents able to be efficient in several complex situations, reusing prior knowledge (see also (Dixon et al., 2000)). The design is based on the use of many basic behaviors which the agent will have to manage, each behavior corresponding to a different motivation. For now, Reinforcement Learning is mainly employed for the "recording" of these basic behaviors. Nevertheless there is place for improvements of our framework through other uses of Reinforcement Learning.

Basic behaviors

For each motivation, a basic behavior b is learned as a stochastic policy mapping perceptions to actions (Dutech et al., 2001). And for a given behavior, perceptions only consider subsets of the set of all objects and are called **configurations**. Useful data concerning each behavior are stored in two tables:

- $\theta_b(c, a)$: gives the probability to choose action a while seeing configuration c, and
- $Q_b(c, a)$: is the expected discounted reward when choosing action a for the configuration c.

For a given behavior, both tables are learned through Reinforcement Learning. More precisely, a gradient descent is used to search for the stochastic policy π_{θ_b}, and afterwards the Q_b-table can be learned while the agent behaves according to the stochastic policy.

An agent confronted to a single configuration c of a behavior b will simply use $\theta_b(c, .)$ to make the choice of its next action. If this agent has to deal with more configurations, from possibly different behaviors, the θ-tables are not sufficient to weight the relative importance of the different decisions that can be taken. The Q-tables will thus give a measure of current motivations.

Scene decomposition

When choosing among its available actions, an agent has to analyse the present situation. This is done by searching in the scene which configurations are linked to behaviors, and are therefore of interest. If $\mathcal{P}$ is the perception, the useful configurations are in

$$\mathcal{C} = \{c \subset \mathcal{P} \mid \exists b \,;\, Q_b(c, .) \; and \; \theta_b(c, .) \; are \; well \; defined\}.$$[1]

A configuration may be related to different behaviors, leading to the use of

$$\mathcal{B}(c) = \{b \mid Q_b(c, .) \; and \; \theta_b(c, .) \; are \; well \; defined\}.$$[1]

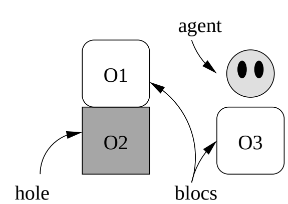


Figure 1. A not so simple scene
In the scene shown on this figure, the agent perceives objects O_1, O_2 and O_3. With two behaviors: *a*voiding holes (b_a) and *p*ushing blocs in holes (b_p), the agent has to manage three different $(behavior, configuration)$ pairs: $(b_a, \{O_2\})$, $(b_p, \{O_1, O_2\})$ and $(b_p, \{O_3, O_2\})$.

[1] By "well defined", we mean that the Q_b and θ_b tables are defined for configuration c. For a given behavior b, only a restricted subset of all possible configurations are of interest, and are therefore defined.

Basic behaviors combination

Once the scene is decomposed, the agent has to use its knowledge to decide its next action. This is done by calculating a weighted distribution $P(a) = \frac{1}{K}\sum_{c\in\mathcal{C}}\sum_{b\in\mathcal{B}(c)} w(b,c,a).P_b(c,a)$ for each action a, resulting in a probability distribution over actions. In this formula, the $w(b,c,a)$ are positive functions that can be defined in various ways, and K is a normalizing factor.

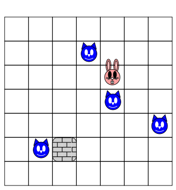
The basis of this computation is a heuristic lacking of theoretical support. Nevertheless, this part of the method is subject to discussion, as tools like fuzzy logic (Zadeh, 1973) or learned parameters could be efficiently used. For now our experiments only present different w functions that helped us analyze the problem.

Some experiments

One task we tried to deal with thanks to this method was for four predators to catch a prey while avoiding a wall. Two basic behaviors: *hunt* and *avoid* were already known (learned θ- and Q-tables) and had only to be adequately combined to obtain Θ. Five methods have been tried:

1. *hunt* behavior alone: $P(a) = P_{hunt}(a|c_h)$,
2. *avoid* behavior alone: $P(a) = P_{avoid}(a|c_a)$,
3. both behaviors with equal weights: $P(a) = \frac{1}{2}[P_{hunt}(a|c_h) + P_{avoid}(a|c_a)]$,
4. both behaviors weighted by $w(b,c) = \max_a\{abs(Q_b(c,a))\}$, and
5. *hunt* behavior with certain moves forbidden if $Q_{avoid}(c,a) < 0$.

Some results are shown on table 1 for this particular application.



method	*#captures*	*#hits*
hunt	802	349
avoid	1	0
3	91	65
4	790	220
5	834	0

Table 1. Results in a 7×7 grid-world
(Number of prey-captures and wall-hits for one predator in 10000 time steps.)

Results and Perspectives

The method was tried on two different problems: 1- predators catching a prey while avoiding a wall, and 2- agents having to merge pairs of blocs (with different possible fusions at the same time). These examples involve various forms of motivations (reaching a goal or avoiding an injury) and various relations between them (different levels of priority, or equal priority motivations). Whereas it illustrated the complexity of a behavior composed of many basic behaviors, the efficiency obtained through this method is encouraging.

The next step will be to learn parameters so as to better tune the relative importance of the different behaviors. But we also would like to dynamically learn hierarchies of basic behaviors, and automatize the creation of a behaviors' library.

References

Dixon, K., Malak, R., & Khosla, P. (2000). *Incorporating prior knowledge and previously learned information into reinforcement learning agents* (Technical Report). Carnegie Mellon University, Institute for Complex Engineered Systems.

Dutech, A., Buffet, O., & Charpillet, F. (2001). Multi-agent systems by incremental gradient reinforcement learning. *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence, IJCAI-01.*

Kearns, M., & Koller, D. (1999). Efficient reinforcement learning in factored mdps. *Proceedings of the 16th International Joint Conference on Artificial Intelligence, IJCAI'99, Stockholm.*

Parr, R. (1998). *Hierarchical control and learning for markov decision processes.* Doctoral dissertation, Computer Science, University of California at Berkeley.

Zadeh, L. A. (1973). Outline of a new approach to the analysis of complex systems and decision processes. *IEEE Transactions on Systems, Man and Cybernetics, SMC-3*, 28–44.