Incremental Reinforcement Learning for Designing Multi-Agent Systems

Olivier Buffet
LORIA, BP 239
54506 Vandoeuvre-lès-Nancy
France
buffet@loria.fr

Alain Dutech
LORIA, BP 239
54506 Vandoeuvre-lès-Nancy
France
dutech@loria.fr

François Charpillet
LORIA, BP 239
54506 Vandoeuvre-lès-Nancy
France
charp@loria.fr

1. INTRODUCTION

Complex systems seem more easily described and controlled if seen as Multi-Agent Systems (MAS): constellation of satellites, multi-robots applications... The design of a MAS is usually built "by hand", with the need of repeated simulations to tune the system. Here, we propose to build the system by having each agent locally learn its own behavior.

The use of Reinforcement Learning (RL) in the context of MAS suffers from several limitations which can make the learning task nearly impossible :

- **combinatorial explosion.** The computationnal burden of RL algorithms grows exponentially with the number of states and actions in the system.
- **hidden global state.** Usual situated agents can only rely on an imperfect, local and partial perception of their environment.
- **credit assignment problem.** When a positive reward is given by the environment, it is not always evident to credit positively the "good" actions that led to this reward.

Our answer to these problems is to use a **decentralized adapted incremental learning algorithm** based on a classical RL technique.

2. OUR FRAMEWORK

2.1 The agents

The agents we decided to work with are very simple reactive ones. Among many possible choices, our agents can be caracterized as:

- **situated with local perception**
- **with identical capabilities**
- **possibly heterogeneous** (different roles may appear)
- **cooperative**
- **non-communicating**

AGENTS'01 May 28-June 1, 2001, Montréal, Quebec, Canada.

2.2 RL and MDP

Q-learning (see [2]) is a classical RL algorithm for which convergence has been proven in the case of stationnary Markov Decision Processes (MDP).

A MDP is defined as a $< \mathcal{S}, \mathcal{A}, T, r >$ tuple, $\mathcal{S}$ being a finite set of states and $\mathcal{A}$ a finite set of actions. When the system is in given state s, an action a being chosen, the probability for the system to end in state s' is given by $T(s, a, s')$. After each transition, the environment generates a reward given by $r(s, a)$. The problem is then to find the optimal mapping $\pi(s, a)$ between states and actions so as to maximize the reward received over time, usually expressed as a utility function $V(s) = \sum_{t=0}^{\infty} \gamma^t (s_t | s_0 = s)$. Such a mapping is called a policy.

Whereas a centralized MAS may be considered as a big MDP, we work with decentralized systems where each agent is independent from the other as far as decision and learning are concerned. It corresponds to the NEXP-complete problem of solving DEC-POMDPs (see [1]). Thus we face two major difficulties :

1. **Non-stationary transitions.** By considering other agents as part of the environment, and because this other agents have evolving behaviors (they learn it), transitions are non-stationary.
2. **Partial observability.** Our agents only have a partial (local) view of the system's state.

2.3 Incremental Reinforcement Learning

To converger, *Q-learning* requires the knowledge of the actual state. As we only have access to observations of the states, we use a modified version of *Q-learning* where observations are assimilated to states and policies are stochastics.

We thus have reduced the problems complexity, but are now subject to perception aliasing. Furthermore, having the agents learn a task requiring coordination remains difficult. We propose to help them to incrementaly learn their policies:

- **progressive task**: learning begins with a very simple version of the task to be executed. Then, as learning progresses, the task is made harder usually by giving more freedom of action to the agents.
- **number of agents**: learning starts with a small number of agents. Then, more agents are added, with initial policies taken from the original agents and then refined through learning.

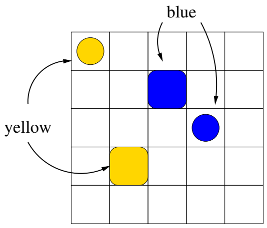

Figure 1: the bloc merging problem
Agents (circles) have to merge blocs (squares).

3. EXPERIMENTING WITH INCREMENTAL LEARNING

The task chosen involves agents (either yellow or blue) in a grid world whose goal is to push yellow cubes against blue ones[1]. When two agents coordinate their movements to attain this goal -pushing *together* a pair of cubes- both cubes temporarily disappear. Simultaneously, agents responsible for this fusion receive a positive reward.

Our agents just have four possible actions: moving North, East, South and West. Agents can push other agents and other cubes, which makes the consequences of their actions stochastic, depending on which constraints will be considered first.

An agent's perceptions[2], as shown on figure 3, are :

- `dir(oa)` : direction of nearest agent from other color,
- `dir(cy)` : direction of nearest yellow cube,
- `dir(cb)` : direction of nearest blue cube,
- `near(cy)` : is there a yellow cube next to the agent,
- `near(cb)` : is there a blue cube next to the agent.

4. RESULTS

4.1 The 2-agent and 2-cube case ($2a2c$)

As said before, we first help our agents by having them learn increasingly harder tasks. To be more precise, we define a *try* as a sequence of n steps[3] beginning in a given situation. The intention is to put agents in a situation from which they can easily learn how to behave. So this *try* must be repeated sufficiently (N times) to be useful. This succession of *tries* will be called an *experiment* for our agents. The trainer has to define a sequence of progressive *experiments* to help learning.

Figure 2 shows the efficiency of this help: high quality policies are obtained much faster than in the classical helpless situation.

4.2 More agents and more cubes

Thanks to local perceptions, agents are adapted to environments with various numbers of agents and cubes. Therefore, policies learned in the simple $2a2c$ case may be reused with growing populations. Experiments show that this method leads to better results than when learning from scratch. On figure 3 are presented the results when learning

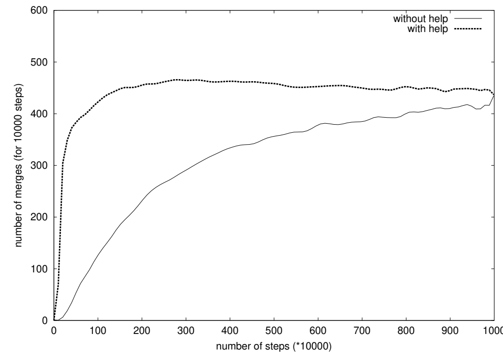

Figure 2: $2a2c$: incremental vs. raw learning

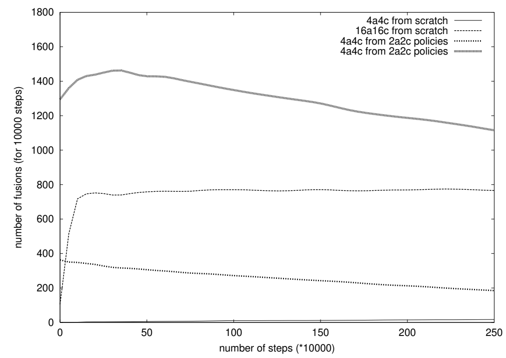

Figure 3: more agents and more cubes

either from scratch or $2a2c$-policies with more agents and cubes[4].

The solutions cannot be optimal. More improvements (short-term memory, communication) would be necessary to solve recurrent problems linked to the agents' lack of focus.

5. CONCLUSION

The results obtained with this incremental learning are encouraging. Nevertheless, several ameliorations could be conducted: to use another learning algorithm such as a gradient descent to obtain better policies, and to generate the help given to agents (by first looking for situations near rewarded state-action pairs).

6. ACKNOWLEDGMENTS

We are particularly thankful to Bruno Scherrer, Iadine Chadès and Vincent Chevrier for numerous discussions and their invaluable help in writing this paper.

7. REFERENCES

[1] D. Bernstein, S. Zilberstein, and N. Immerman. The complexity of decentralized control of markov decision processes. In *Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence, Stanford, California*, 2000.

[2] C. Watkins. *Learning from delayed rewards.* PhD thesis, King's College of Cambridge, UK., 1989.

[1] Colors of agents and cubes are not related in this problem.
[2] (directions are discretized)
[3] In a *step*, all agents make one move simultaneously.

[4] The decreases are due to the inadequation of *Q-learning*.