

---

# Parallelization of the TD( $\lambda$ ) Learning Algorithm

---

**Odalric-Ambrym Maillard**

ODALRIC-AMBRYM.MAILLARD@ELEVES.BRETAGNE.ENS-CACHAN.FR

École Normale Supérieure de Cachan

**Rémi Coulom**

REMI.COULOM@UNIV-LILLE3.FR

**Philippe Preux**

PHILIPPE.PREUX@UNIV-LILLE3.FR

GRAPPA, Université Charles de Gaulle – Lille 3, France

## Abstract

Running the TD( $\lambda$ ) algorithm for complex simulated tasks may take several days of computation time. In order to reduce this time, it is possible to take advantage of parallel computation architectures. In this paper, we present a method to parallelize the TD( $\lambda$ ) algorithm. This method consists in running episodes in parallel, and summing the weight changes obtained by each processor at some synchronization points. We present experimental results on motor-control tasks of varying complexity, and a theoretical bound on parallelization error.

## 1. Introduction

Applying reinforcement learning to complex problems in simulation may require huge amounts of computation. Tesauro’s backgammon player (Tesauro, 2002) needed months of CPU time (1,500,000 games). Some of Coulom’s swimmers (Coulom, 2002) also took many days to learn.

In order to reduce these huge computation times, it seems worth trying to parallelize learning algorithms. Kretchmar (Kretchmar, 2002) proposed a framework for parallel reinforcement learning, where each of the agents running in parallel computes its own value function. The agents can share their experience by averaging their value functions. Kretchmar’s research was restricted to simple single-state problems, and tabular Q-learning. In this paper, these ideas are extended to multi-state episodic tasks, using the TD( $\lambda$ ) algorithm and generalizing function approximators.

The details of our algorithm are presented in Section 2. Section 3 presents the main empirical results of this work, both on computation time and speed of learning. Finally, Section 4 suggests possible avenues of research.

## 2. Experimental Setting

Three kinds of problems have been used to make the experiments, each one with the same TD( $\lambda$ ) algorithm. See Coulom’s thesis (Coulom, 2002) for more details. The pendulum problem is mainly used here since large amounts of data can be computed swiftly on it. The cart-pole problem is a little more complex, so it still enables us to have sufficient data rather quickly. The last one, the swimmer problem, is far more complex. Few experimentations have been carried out on it for it takes a lot of time to obtain results.

The parallelization acts by means of three-step cycles. The synchronization consists in broadcasting the same set of parameters say, the weights of the neural network, to all  $P$  processes. Then, each one learns independently during  $G$  episodes of duration  $T$  (progress step). Finally comes the gathering step, where all weight changes computed by each process are added to the initial weights, to produce a new starting point for the next cycle. It enables us to compute for each cycle a piece of learning of length  $PGT$  with  $P$  pieces of length  $GT$ .

Nevertheless, for this method to be successful, we assume that the weight changes between two cycles differs little from what we would have with the sequential method, and this does not hold if  $P$  or  $G$  is too big. The influence of these two parameters is discussed below.

## 3. Experiment Results

First, let’s focus on the computing time. Here we have used the MPI library with an Ethernet network on the swimmer problem, but we may hope for better results with an SMP architecture. We see that using more than an average seven processes is useless to speed up the computation. An other point of view shows that  $G$  has little effect on this time, so we can choose it

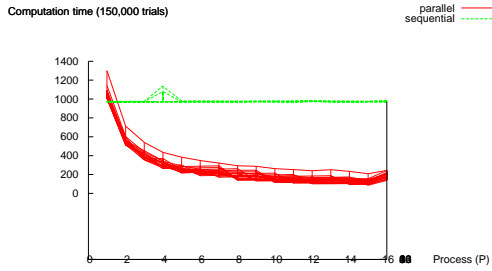


Figure 1. Computation time (swimmers)

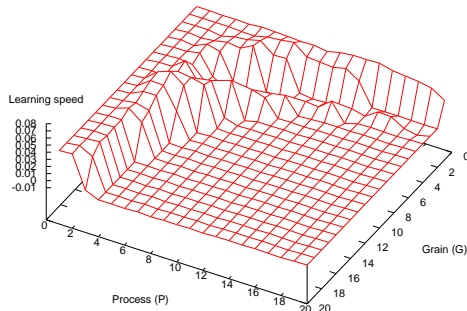


Figure 2. Learning speed (pendulum)

very small. Therefore, only small values of  $G$  and  $P$  are interesting.

In order to measure the influence on learning, one can use the average coefficient of the slope on the reward curve before stagnation appears. This is a quite good estimation of what happens. Figure 2 shows how this coefficient changes with  $P$  and  $G$ . It spots out that the learning speed does not go down gently, but stays rather unchanged before falling down to zero very fast. We observe a dissymmetrical hyperbolic profile parting the graph in two and so called the falling line. It is pushed away with more complex problems.

To try to understand where such a phenomenon comes from, we have searched a boundary to the approximation error made between the sequential and the parallel approach, and we have found this:

$$|error(kT)| \leq MT((k/PG) \cdot (PG(P-3)/2) - (k/G) \cdot G + 2k)$$

where  $k$  is the number of the current trial and  $M$  is some constant such that  $||\eta\mathcal{H}e|| \leq M$  (See (Maillard, 2005) for details of the proof).

Assuming that the error must be less than some value  $a$  to make learning possible provides no good adequation. But assuming that we should have  $|error(kT)| <$

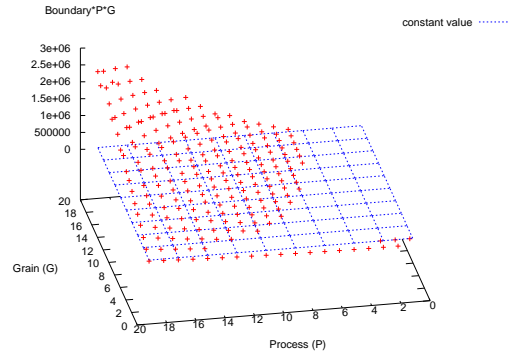


Figure 3.  $PG$  times the boundary of  $|error(kT)|$  (theory)

$a/PG$ , we observe the same dissymmetrical hyperbolic profile, which is encouraging (Figure 3).

## 4. Conclusion

This work has enabled us to know how to parallelize so as to benefit both from best speed and best quality of learning. While studying this problem, we have encountered some big issues, mainly:

- The choice of the method of confrontation, for there must be a compromise between ours and, for instance, choosing the best piece of trajectory out of  $P$ . The next step is trying to make the agent learn the best function.
- When the learning rate is reduced, the falling line moves away with  $P$  and  $G$ . Knowing more about this and what happens in the part where the agent keeps on learning should be very interesting.

## References

- Coulom, R. (2002). *Reinforcement learning using neural networks, with applications to motor control*. Doctoral dissertation, Institut National Polytechnique de Grenoble.
- Kretschmar, R. M. (2002). Parallel reinforcement learning. *The 6th World Conference on Systemics, Cybernetics, and Informatics (SCI2002)*.
- Maillard, O.-A. (2005). *Étude expérimentale de la parallélisation d'algorithmes d'apprentissage par renforcement* (Technical Report). Groupe Apprentissage par REinforcement (GARE), Université de Lille 3.
- Tesauro, G. (2002). Programming backgammon using self-teaching neural nets. *Artificial Intelligence*, 134, 181–199.