Dealing with Non-Stationary Environments using Context Detection

Bruno C. da Silva BCS@INF.UFRGS.BR
Eduardo W. Basso EWBASSO@INF.UFRGS.BR
Ana L. C. Bazzan BAZZAN@INF.UFRGS.BR
Paulo M. Engel ENGEL@INF.UFRGS.BR
Instituto de Informática, UFRGS – Caixa Postal 15064 – CEP 91.501-970 – Porto Alegre, RS, Brazil

Abstract

In this paper we introduce RL-CD, a method for solving reinforcement learning problems in non-stationary environments. The method is based on a mechanism for creating, updating and selecting one among several partial models of the environment. The partial models are incrementally built according to the system's capability of making predictions regarding a given sequence of observations. We propose, formalize and show the efficiency of this method both in a simple non-stationary environment and in a noisy scenario. We show that RL-CD performs better than two standard reinforcement learning algorithms and that it has advantages over methods specifically designed to cope with non-stationarity. Finally, we present known limitations of the method and future works.

1. Introduction

When implementing learning algorithms, one often faces the difficult problem of dealing with environments whose dynamics might change due to some unknown or not directly perceivable cause. Non-stationary environments affect standard reinforcement learning (RL) methods in a way that forces them to continuously relearn the policy from scratch. In this paper we describe a method for complementing RL algorithms so that they perform well in a specific class of non-stationary environments.

The two usual approaches for reinforcement learning are called model-free and model-based. Model-free algorithms do not require that the agent have access to informations about how the environment works. They are usually able to compute good policies in relatively simple environments. For complex environments, however, a great engineering effort in designing smart state representations is needed. Model-based approaches, such as Prioritized Sweeping, can usually perform better than model-free systems and present a much lower convergence time, although at the cost of demanding a greater computational effort per iteration. For more details on these methods please refer to (Kaelbling et al., 1996).

It is important to emphasize that both of these approaches were designed to work in stationary environments. When dealing with non-stationary environments, they have to continually readapt themselves to the changing dynamics of the environment. This causes two problems: **1)** the time for relearning how to behave makes the performance drop during the readjustment phase; and **2)** the system, when learning a new optimal policy, forgets the old one, and consequently makes the relearning process necessary even for dynamics which have already been experienced.

The non-stationary environments that we are interested in in this paper consist on those whose behavior is given by one among several different stationary dynamics. We call each type of dynamics a *context* and assume that it can only be estimated by observing the transitions and rewards. We also assume that the maintenance of multiple models of the environment (and their respective policies) is a good solution to this learning problem. Partial models have been used for the purpose of dealing with non-stationarity by other authors, such as Choi et al. (2001) and Doya et al. (2002). However, their approaches require a fixed number of models, and thus implicitly assume that the approximate number of different environment dynamics is known *a priori*. Since this assumption is not always realistic, our idea is to overcome this restriction by incrementally building new models.

Appearing in *Proceedings of the 23rd International Conference on Machine Learning*, Pittsburgh, PA, 2006.

Our main hypothesis is that the use of multiple partial models makes the learning system capable of partitioning the knowledge into models, each of which automatically assuming for itself the responsibility for "understanding" one kind of environment behavior. Based on this hypothesis, we propose, formalize and show the efficiency of a method called **RL-CD**, or Reinforcement Learning with Context Detection, which performs well in non-stationary environments by continuously evaluating the prediction quality of each partial model. A brief discussion of how to measure prediction quality, in order to deal with non-stationary environments, can also be found in (Silva et al., 2006).

This paper is organized as follows. In section 2 we present some basic concepts about reinforcement learning. In section 3 we discuss how to measure the relative quality of each partial model and how to detect context changes. In section 4 we present three validation scenarios which empirically show that our method performs better than several other RL algorithms. Some concluding remarks, known limitations and future work are discussed in section 5.

2. Reinforcement Learning

Most RL problems are modeled as Markov Decision Processes (MDPs). Although RL algorithms are by no means restricted to MDPs, this theoretical framework provides an interesting way of studying learning methods and their properties. MDPs are used in order to model situations in which an agent has to decide how to act based on the observation of the current state.

Formally, a Markov model consists of a discrete set of environment states, $\mathcal{S}$, a discrete set of agent actions, $\mathcal{A}$, a reward function $R : \mathcal{S} \times \mathcal{A} \rightarrow \Re$ and a state transition function $T : \mathcal{S} \times \mathcal{A} \rightarrow \Pi(\mathcal{S})$, where $\Pi(\mathcal{S})$ is a probability distribution over $\mathcal{S}$. We write $T(s, a, s')$ as the probability of making a transition from s to s' taking action a.

An experience tuple $\langle s, a, s', r \rangle$ denotes the fact that the agent was in state s, performed action a and ended up in s', receiving reward r. The optimal value of a state, denoted $V^*(s)$, is the expected infinite discounted sum of rewards that the agent will gain if it starts in that state and follows the optimal policy. Given $V^*(s)$ for all states, we write the optimal policy π^* as the mapping from states to actions such that the future reward is maximized.

There exist simple iterative algorithms for determining the optimal policy, such as value iteration and policy iteration. Their limitation is mainly due to the fact that the agent usually does not have *a priori* estimates of T and R. Thus, in order to solve this limitation, the so-called model-free systems were created, which do not rely on T and R in order to converge to the optimal policy.

In this paper we compare the performance of our approach with two traditional RL algorithms: Q-Learning, a simple model-free algorithm, and Prioritized Sweeping (PS), a model-based algorithm. Since both of these methods are widely known by the machine learning community, we simply refer the interested reader to (Kaelbling et al., 1996). We also compare the performance of our approach with more specialized algorithms, which perform better in non-stationary environments: Multiple Model-Based Reinforcement Learning (Doya et al., 2002) and Compositional Q-Learning (Singh, 1992). These methods are shortly discussed in section 4.

3. RL with Context Detection

Before going into details of how RL-CD works, we first present an overview of how it creates models and detects context changes. First of all, we assume that the system contains several partial models, each one specialized in a different environment dynamics. The *quality* of a model is a value inversely proportional to its prediction error. At any given moment, only the model with the highest quality is activated. A context change is detected whenever the currently active model is replaced. If the quality of the best model is still worse than some *minimum quality*, a new one is created, which will learn both a dynamics predictor and the corresponding optimal policy.

The class of non-stationary environments that we are interested in is similar to the one studied by Hidden-Mode MDPs researchers (Choi et al., 2001). We assume that the following properties hold: **1)** environmental changes are confined to a small number of contexts, which are stationary environments with distinct dynamics; **2)** the current context cannot be directly observed, but can be estimated according to the types of transitions and rewards observed; **3)** environmental context changes are independent of the agent's actions; and **4)** context changes are relatively infrequent. These assumptions are considered plausible for a broad number of real applications (Choi et al., 2000), but in case they are not met scalability problems should be carefully considered and studied.

3.1. Learning Partial Models

RL-CD relies on a set of partial models for predicting the environment dynamics. A partial model m con-

tains functions which estimate transition probabilities (T_m) and rewards (R_m). Standard model-based RL methods such as Prioritized Sweeping and Dyna can be used to compute the locally optimal policy $\pi^*_m(s)$.

At each simulation step, given an experience tuple $\langle s, a, s', r\rangle$, we update the current partial model m by adjusting its model of transition, T_m, towards a maximum likelihood estimation, as follows:

$$\Delta T_m(\kappa) = \begin{cases} \dfrac{1 - T_m(s,a,\kappa)}{N_m(s,a)+1} & \kappa = s' \\ \\ \dfrac{0 - T_m(s,a,\kappa)}{N_m(s,a)+1} & \kappa \neq s' \end{cases} \qquad \forall \kappa \in \mathcal{S}$$

$$T_m(s,a,\kappa) = T_m(s,a,\kappa) + \Delta T_m(\kappa), \quad \forall \kappa \in \mathcal{S} \tag{1}$$

and the reward model, R_m, towards the moving average of all previous rewards, as follows:

$$\Delta R_m = \frac{r - R_m(s,a)}{N_m(s,a)+1}$$

$$R_m(s,a) = R_m(s,a) + \Delta R_m \tag{2}$$

The quantity $N_m(s,a)$ reflects the number of times, in model m, action a was executed in state s. We compute N_m considering only a truncated (finite) memory of past M experiences:

$$N_m(s,a) = min\Big(N_m(s,a)+1,\ M\Big)$$

A truncated value of N acts like an adjustment coefficient for T_m, causing transitions to be updated faster in the initial observations and slower as the agent experiments more. In case N is unbounded, equation (1) implements exactly a maximum likelihood estimation for the transition model, and equation (2) implements exactly a mobile average for the reward model.

3.2. Detecting Context Changes

In order to detect context changes, the system must be able to evaluate how well the current partial model can predict the environment. Thus, a *quality signal* is computed for each partial model. The quality of a model is proportional to a *confidence value*, which reflects the number of times the agent tried an action in a state. Given a model m and an experience tuple $\langle s, a, s', r\rangle$, we calculate the confidence, $c_m(s,a)$, and the instantaneous quality, e_m, as follows:

$$c_m(s,a) = \frac{N_m(s,a)}{M} \tag{3}$$

$$e_m = c_m(s,a)\Big(\Omega\ e^R_m + (1-\Omega)\ e^T_m\Big) \tag{4}$$

The instantaneous quality, e_m, is a linear combination of the quality of reward prediction, e^R_m, and the quality of transition prediction, e^T_m. These values are linearly interpolated by Ω, which specifies the relative importance of rewards and transitions for the the model's quality.

$$e^R_m = 1 - 2\ (Z_R(\Delta R_m)^2) \tag{5}$$

$$e^T_m = 1 - 2\ (Z_T \sum_{\kappa \in \mathcal{S}} \Delta T_m(\kappa)^2) \tag{6}$$

Moreover, Z_R and Z_T are normalization factors corresponding to $(R_{max} - R_{min})^{-1}$ and $\frac{1}{2}(N(s,a)+1)^2$, respectively; R_{max} and R_{min} are the maximum and minimum values for rewards. Notice that (5) and (6) rescale values from the range $[0,1]$ to $[+1,-1]$, where $+1$ is the best prediction quality and -1 is the worst.

In case $\Omega = 0$, the instantaneous quality, e_m, can be related to the Bayesian *a posteriori* probability of m, given an experience tuple. Also, the confidence value, c_m, can be seen as the *a priori* probability of selecting m. The major difference between our method and a Bayesian approach is that we do not normalize the values by the marginal probability of the observation. Non-normalized values give us absolute measurements of quality, which are necessary in order to decide when a new model must be created.

Once the instantaneous quality of the model has been computed, a trace of quality E_m for each partial model is updated:

$$E_m = E_m + \rho\Big(e_m - E_m\Big) \tag{7}$$

where ρ is the adjustment coefficient for the quality.

The quality E_m is updated after each iteration for every model m, but only the active one is corrected according to equations (1) and (2). Whenever some model m becomes better than the current, m_{cur}, the system detects a context change and activates m. A *minimum quality threshold*, E_{min}, is used to specify

how much a partial model can be adjusted. When there is no model with quality higher than E_{min}, a new one is created. RL-CD starts with only one model and then incrementally creates new ones as they become necessary. Note that, since the models are probabilistic, the observations in a context don't need to repeat exactly in order to select the most appropriate model.

We formalize, in algorithm 1, the Reinforcement Learning with Context Detection (RL-CD) method.

Algorithm 1 RL-CD algorithm

Let *newmodel*() be a routine which creates and initializes a new partial model.
Let m_{cur} be the currently active partial model.
Let $\mathcal{M}$ be the set of all available models.
1: $m_{cur} \leftarrow newmodel()$
2: $\mathcal{M} \leftarrow \{m_{cur}\}$
3: $s \leftarrow s_0$, where s_0 is any starting state
4: **loop**
5: Let a be the action indicated by $\pi_{m_{cur}}(s)$
6: Observe next state s' and reward r
7: **for all** $m \in \mathcal{M}$ **do**
8: Update E_m according to (7)
9: **end for**
10: $m_{cur} \leftarrow \arg\max_m (E_m)$
11: **if** $E_{m_{cur}} < E_{min}$ **then**
12: $m_{cur} \leftarrow newmodel()$
13: $\mathcal{M} \leftarrow \mathcal{M} \cup \{m_{cur}\}$
14: **end if**
15: Update $T_{m_{cur}}$ according to (1)
16: Update $R_{m_{cur}}$ according to (2)
17: $N_m(s,a) \leftarrow \min(N_m(s,a)+1, M)$
18: $s \leftarrow s'$
19: **end loop**

The *newmodel*() routine, used in algorithm 1, creates a new partial model. This routine initializes the partial model by: **1)** setting its quality trace E_m to zero; **2)** setting the values $R_m(s,a)$ and $N_m(s,a)$ to zero for all $s \in \mathcal{S}$ and $a \in \mathcal{A}$; and **3)** initializing the transition model as follows:

$$T_m(s,a,\kappa) \leftarrow \frac{1}{|\mathcal{S}|} \quad \forall s \in \mathcal{S}, \forall a \in \mathcal{A}, \forall \kappa \in \mathcal{S}$$

The values of parameters M, ρ, Ω and E_{min} must be tuned according to the problem. Small values of ρ are appropriate for noisy environments; higher values of M define systems which require more experiences in order to gain confidence regarding its predictions; in general applications, Ω might be set to 0.5; the minimum quality E_{min} should be set to negative small values, since a zero quality means that the system has not yet learned anything. Formal support for these statements is being developed but detailed discussions are not presented due to lack of space.

4. Empirical Results

In this section we present three validation scenarios which evaluate how well our system performs in non-stationary environments. The following experiments use Prioritized Sweeping to build the policy for each partial model. However, any other reinforcement learning method could have been used, since our mechanism is only responsible for detecting the context changes and deciding when to create new models. Even model-free methods can be used, but in that case a representation of the transitions and expected rewards for each partial model has to be explicitly computed.

4.1. Ball Catching

Our first validation scenario consists in a non-stationary environment built as follows:

- The environment is a toroidal discrete grid of 15x15 cells;
- The agent is a cat whose goal is to catch a moving ball;
- The moving ball starts in a random column and row;
- The ball can take one of following behaviors: **1)** moves to the left; **2)** moves to the right; **3)** moves down; or **4)** moves up.

The non-stationary factor of the simulation is related to the fact that the transition rules for the movement of the ball change over time. In the current scenario, Ω is set to 0 since the non-stationarity affects only the transition model.

The first experiment performed aims at testing the relative efficiency of the classic model-free and model-based algorithms: Q-Learning (QL) and Prioritized Sweeping with Finite Memory (PS-M)[1]. The agent is trained for one of the ball behaviors at a time, until the algorithm converges to the optimal policy. After that, we change the environment dynamics by modifying the ball behavior and measure how fast each algorithm converges. It can be seen in figure 1 that,

[1] We verified that standard PS requires exponential time to converge in non-stationary environments, and thus we designed and used **PS-M**, a modified PS with truncated memory instead of maximum likelihood estimation.

as the context changes, the time needed to recalculate the optimal policies, both in the model-free and the model-based approaches, is much superior to the convergence time of RL-CD. Although PS-M performs better than Q-Learning, it still has to reestimate T after every context change. Our method, on the other hand, takes only a few steps until realizing that the context has changed. After that, it automatically selects the most appropriate partial model for the new dynamics.

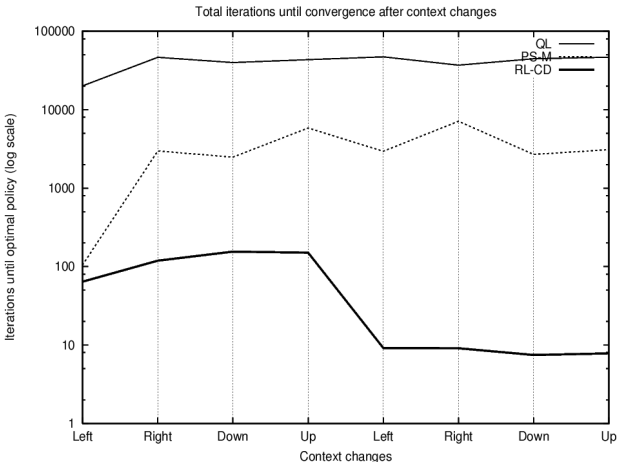

Figure 1. A comparison of convergence times for Q-Learning, PS-M and RL-CD (Ball Catching).

It is also important to measure the average time the cat takes to catch the ball *while* the policy is being learned in a context. In order to implement this experiment, we ran the learning algorithms and changed the ball behavior 8 times. For every algorithm, we performed 5 batches of 100 episodes, each episode corresponding to the cat being placed in a random place and running after the ball. We measured the average time of a batch as the average number of iterations needed to catch the ball during all episodes. The results are shown in figure 2.

In figure 2, vertical stripes are used to indicate different contexts. Each time the ball behavior changes, the average steps per episode grows (peaks in the graph), indicating that the algorithms take some time to relearn how to behave. During the first 4 contexts, our method is actually very similar to PS-M. However, as soon as the contexts begin to repeat, RL-CD is capable of acting optimally all the time, while the other algorithms present periods of relearning and suboptimal acting.

Finally, figure 3 presents the trace of quality E_m for each model m. Notice that our method creates 4 models, each corresponding to a different environment dynamics. Also notice that during the first contexts not all models were available, indicating that they were created on-demand. The horizontal line at -0.1 corresponds to the minimum quality E_{min}. At every moment, the system selects the model with highest quality, and creates a new one if all models are worse than E_{min}. Just like in the previous experiment, we use stripes to indicate the current context of the environment.

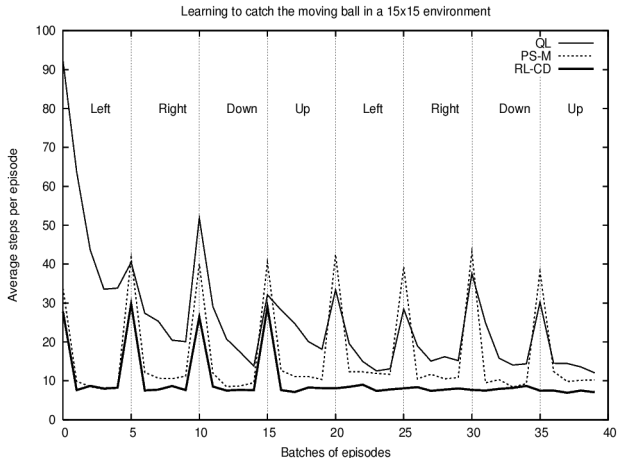

Figure 2. A comparison of performance for Q-Learning, PS-M and RL-CD (Ball Catching).

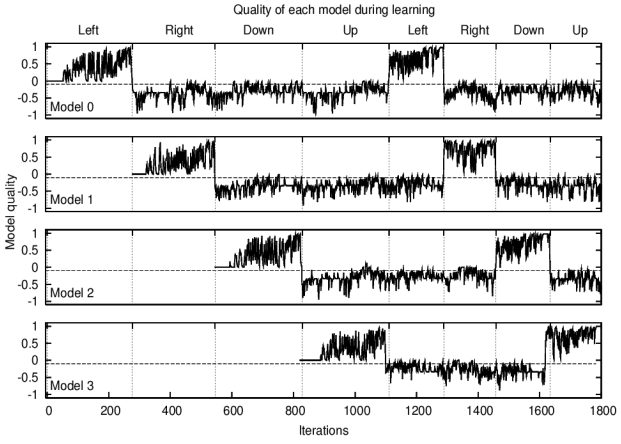

Figure 3. Trace of the quality E_m for all models (Ball Catching).

We explicitly change the environment dynamics each time the cat caught the ball for the 25th time. The fact that the contexts last less as time goes by, as can be seen in figure 3, indicates that the cat is able to progressively finish the 25 catches faster. In other words,

the system is able to promptly switch to the policy of the model which best describes the new dynamics.

4.2. Traffic Lights Control

Our second validation scenario consists of a traffic network which is a 3x3 Manhattan-like grid, with a traffic light in each junction. Figure 4 depicts a graph representing the traffic network, where the 9 nodes correspond to traffic lights and the 24 edges are directed (one-way) links.

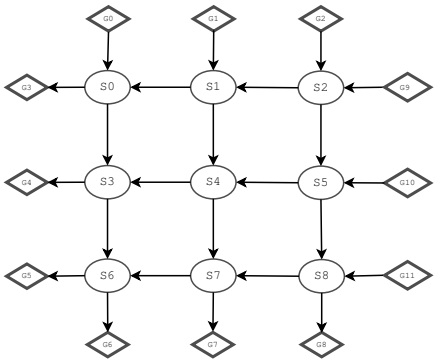

Figure 4. A Network of 9 Intersections.

Each link has capacity for 50 vehicles. Vehicles are inserted by *sources* and removed by *sinks*, depicted as diamonds in figure 4. The exact number of vehicles inserted by the sources is given by a Gaussian distribution. If a vehicle has to be inserted but there is no space available in the link, it waits in an external queue until the insertion is possible. The vehicles do not change directions during the simulation, and upon arriving at the sinks they are immediately removed.

We modeled the problem in a way that each traffic light is controlled by one agent, each agent making only local decisions. Although decisions are local, we assess how well the mechanism is performing by measuring global performance values. As a measure of effectiveness for the control system, we measure the total number of stopped vehicles.

After discretizing the length of queues, the occupation of each link is considered to be either *empty*, *regular* or *full*. The state of an agent is given by the occupation of the links arriving in its correspondent traffic light. The reward for each agent is given by the negative summed square of incoming link's queues.

Traffic lights normally have a set of signal plans used for different traffic conditions and/or time of the day. We consider here only three plans: signal plan **1** gives equal green times for both directions (North-South, East-West); signal plan **2** gives priority to the vertical direction; and signal plan **3** gives priority to the horizontal direction. The agent's action consists of selecting one of the three signal plans at each simulation step.

In order to model the non-stationarity of the traffic scenario, we assume 3 traffic patterns with different car insertion distributions:

- *Low*: low insertion rate in the both North and East sources, allowing the traffic network to perform relatively well even if the policies are not optimal (i.e., the network is undersaturated);
- *Vertical*: high insertion rate in the North sources (G0, G1, and G2), and average insertion rate in the East (G9, G10, and G11);
- *Horizontal*: high insertion rate in the East sources (G9, G10, and G11), and average insertion rate in the East (G0, G1, and G2).

The Gaussian distributions in the contexts *Vertical* and *Horizontal* are such that the traffic network gets saturated if the policies are not optimal. Simultaneous high insertion rates in both directions are not used because no optimal action is possible, and the network would inevitably saturate in few steps, thus making the scenario a stationary environment with all links at maximum occupation.

Notice that this scenario is much harder than Ball Catching, since it is probabilistic and it has three different causes for non-stationarity: **1)** explicit changes in insertion rates; **2)** poor discretization; and **3)** neighbor traffic lights actions, which are hard (or impossible) to foresee.

In figure 5 we compare RL-CD performance with two standard RL methods, namely Q-Learning and Prioritized Sweeping. We modify the traffic pattern every 200 time steps, which corresponds to nearly 3 hours of real traffic flow. Moreover, we measure the performance as the total number of stopped cars in all links (including external queues). This means that, the lower the value in the figure, the better the performance.

Since Q-Learning is model-free, it is less prone to wrong bias caused by non-stationarity. However, for this same reason it is not able to build useful models of the relevant attributes of the dynamics. Prioritized Sweeping, on the other hand, tries to build a single model for the environment and ends up with a model which mixes properties of different traffic patterns. For this reason, it can at most calculate a policy which is a

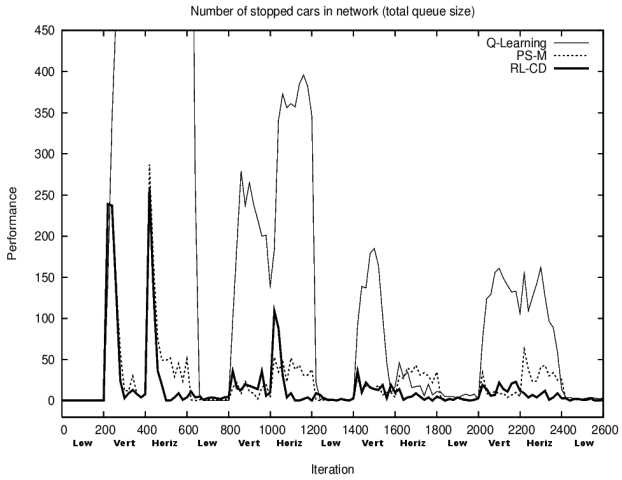

Figure 5. A comparison of performance for Q-Learning, PS-M and RL-CD (Traffic Control).

compromise for several different (and sometimes opposite) traffic patterns. RL-CD performs better than PS-M specially in one of the three contexts. This occurs because the model built by PS-M consists in a mixture of all contexts, and the time between contexts is not enough for it to relearn the new dynamics. Moreover, RL-CD is clearly much more stable than Q-Learning since it does not need to continuously readapt all of its state–value estimations.

4.3. Ball Catching Revisited

As shown in subsections 4.1 and 4.2, approaches which try to deal with non-stationarity by relearning, such as PS-M, require higher convergence times due to the need for rediscovering policies.

Alternative methods, such as those based on multiple models, have been proposed. Due to their similarities to RL-CD, we highlight the works of Doya et al. (2002) and Choi et al. (2001), respectively called MMRL and HM-MDP. HM-MDPs (Hidden Mode Markov Decision Processes) are used to model non-stationary environments by considering the non-stationarity as an effect of a hidden attribute, which is named *mode*. This approach assumes that the transitions between modes are rare, and models each mode as an independent MDP. The objetive is to learn a model of transitions for the set of modes. Since the goal of HM-MDP is only to model the non-stationary environment, no comparison is performed with RL-CD.

The MMRL (Multiple Model-based Reinforcement Learning) algorithm deals with non-stationarity by using multiple partial models of the environment. The idea is to decompose a task into multiple domains in which the environment is predictable. A *responsibility signal* is computed for each partial model in a way that the models with better predictions are assigned with higher responsibility values. The main differences from MMRL to our approach are: 1) the former makes use of special measures of spatial locality and temporal continuity; 2) MMRL does not decide actions based only on the best model, but averages the actions proposed by each model and weights them by the model's responsibility; 3) MMRL assumes that the number of models (i.e. environment dynamics modes) is known *a priori*; and 4) MMRL requires fewer parameters than RL-CD since it knows *a priori* the number of models.

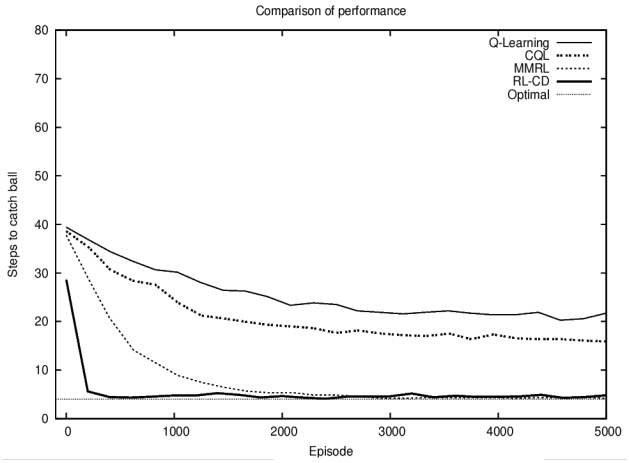

Figure 6. A comparison of performance for Q-Learning, Compositional Q-L, MMRL and RL-CD (Ball Catching Revisited).

In (Doya et al., 2002), MMRL is compared with a modified version of Compositional Q-Learning (CQ-L) in a 7x7 ball-catching task similar to the one presented in subsection 4.1. CQ-L is a multiple modular Q-Learning in which the probability of module selection is given by its capability of predicting the state–value function. CQ-L, therefore, measures errors in the state–values, while MMRL and RL-CD measure errors regarding expected transitions and rewards. Doya et al. (2002) modified Compositional Q-Learning, allowing it to perform without requiring that environment changes be explicitly signaled to the algorithm. In the following comparison, we use this modified version of CQ-L.

In figure 6 we compare the performances of RL-CD, MMRL, QL and CQ-L in Doya's 7x7 ball-catching experiment. Notice that this ball-catching scenario is slightly different than one used in subsection 4.1. This is because we now compare our method under exactly

the same conditions and parameters as the ones proposed in (Doya et al., 2002), since these are the ones which yield the best results for MMRL. In this experiment, the ball direction changes after each episode, and an episode lasts until the cat catches the ball or 100 steps. Since RL-CD is not able to detect context changes when the contexts are too brief, performance of RL-CD was computed changing the context after each 5 episodes. In the very beginning of learning, we verified that all partial models of MMRL were equally responsible for the entire domain. As the agent learns, the models became more specialized in smaller domains, allowing each model to better predict some type of dynamics. Since in RL-CD only one partial model is active at a time, its partial models specialize faster than in MMRL. The downside of this is that RL-CD's selection mechanism is more susceptible to noise.

5. Conclusions

In this paper we have formalized a method called RL-CD for solving reinforcement learning problems in non-stationary environments. Our method was validated in deterministic and in noisy non-stationary environments. We have empirically shown that RL-CD performs better than standard RL algorithms and that it has advantages over methods specifically designed to cope with non-stationarity.

A formal analysis of RL-CD parameters is being developed in order to help to tune them in scenarios which differ significantly from the class of non-stationary environments we assume. We also plan to develop a more detailed study regarding the trade-off between memory requirements and model quality in highly non-stationary and noisy environments, since RL-CD might fail in case of scenarios with overlapping contexts. One possible solution for dealing with contexts which are not readily separable by the dynamics is to consider the model's spatial locality when computing the quality, allowing models to specialize in only a subspace of the state space. We have yet to study the effect on RL-CD of scenarios in which the time between context changes is very short. Experimental results show that RL-CD performs worse in these cases.

This research is still in its early stages. Experimental results so far point to the fact that RL-CD is a good alternative to classic RL algorithms and also to more sophisticated approaches, when dealing with non-stationary environments. We expect to further improve RL-CD by extending it with capabilities for solving some of the downsides above mentioned, but the results so far are undoubtedly promising.

Acknowledgments

Authors are partially supported by CNPq.

References

Choi, S. P. M., Yeung, D.-Y., & Zhang, N. L. (2000). An environment model for nonstationary reinforcement learning. *Advances in Neural Information Processing Systems 12* (pp. 994–1000).

Choi, S. P. M., Yeung, D.-Y., & Zhang, N. L. (2001). Hidden-mode markov decision processes for nonstationary sequential decision making. *Sequence Learning - Paradigms, Algorithms, and Applications* (pp. 264–287). London, UK: Springer-Verlag.

Doya, K., Samejima, K., Katagiri, K., & Kawato, M. (2002). Multiple model-based reinforcement learning. *Neural Computation*, *14*, 1347–1369.

Kaelbling, L. P., Littman, M., & Moore, A. (1996). Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, *4*, 237–285.

Moore, A. W., & Atkeson, C. G. (1993). Prioritized sweeping: Reinforcement learning with less data and less time. *Machine Learning*, *13*, 103–130.

Silva, B. C., Basso, E. W., Bazzan, A. L., Engel, P. M., & Perotto, F. S. (2006). Improving reinforcement learning with context detection. *Fifth International Joint Conference on Autonomous Agents and Multi Agent Systems - (AAMAS 2006) – to appear*. Hakodate, Japan.

Singh, S., James, M. R., & Rudary, M. R. (2004). Predictive state representations: a new theory for modeling dynamical systems. *AUAI '04: Proceedings of the 20th conference on Uncertainty in Artificial Intelligence* (pp. 512–519). Arlington, Virginia, United States: AUAI Press.

Singh, S. P. (1992). Reinforcement learning with a hierarchy of abstract models. *Proceedings of the Tenth National Conference on Artificial Intelligence (AAAI)* (pp. 202–207).

Sutton, R. S., Precup, D., & Singh, S. P. (1999). Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, *112*, 181–211.