Reinforcement Learning of Multi-agent Communicative Acts

Shirley Hoet
Laboratoire d'Informatique de Paris 6
104 avenue du président Kennedy
75016 Paris,France
shirley.hoet@lip6.fr

Nicolas Sabouret
Laboratoire d'Informatique de Paris 6
104 avenue du président Kennedy
75016 Paris,France
Nicolas.Sabouret@lip6.fr

ABSTRACT

In this paper, we propose a reinforcement learning mechanism for multi-agent communication. Existing work underlines three main problems. First, agents need to construct appropriate communicative acts. Second, we have to implement the whole communication process as agent actions, so as to apply classical reinforcement learning algorithms. Last, the non-markovity of MAS raises several issues that have to be solved to guarantee the algorithm convergence. We propose a distributed method to deal with these three problems. This method is based upon introspective agents, that can reason about their own actions and data. We then prove that communicative acts can be seen as actions, according to the speech acts theory, using a Semi-Markovian model of the problem. Last, we show how communicative acts can solve non-markovity issues, using a limited memory of passed actions and observations.

Categories and Subject Descriptors

I.2.11 [**Dist A.I.**]: Intelligent Agent

General Terms

Keywords

Reinforcement Learning, Multi-Agent Systems, Agent communication

1. INTRODUCTION

Direct communication in multi-agent systems is an interaction method that consists in sending communicative acts and that is based upon agents' intention to communicate. Concretely, in current MAS, each agents know in advance the contents of the messages, the adequate recipients and when to send the message. However, in open and loosely coupled systems composed of heterogeneous agents, agents should be able to adapt its behaviour by learning to communicate in function of its needs, without any preliminary knowledge on what to communicate, when and to whom.

Much has been done in the field of mono-agent behaviour learning, in particular using reinforcement learning. Reinforcement learning is a method that allows an agent to learn the best possible action with respect to the situation, through trial and errors [17]. There exist several different reinforcement learning algorithm, the most famous being probably the Q-Learning [19], which relies upon a markovian decision process (MDP [14]) model of the agent's world.

Yet, as stated by the speech act's theory [10, 15] which is at the core of direct communication in MAS, a communication act can be seen as an action, since it has an effect on the recipient's beliefs. As a consequence, we claim that it is possible to use reinforcement learning to learn not only how to act, but also how to communicate.

However, this approach raises several difficulties when considering the MAS context. To start with, only few work studied the problem of communicative acts learning. For instance, [18, 11] used communication in their learning model to deal with hidden states, but their vision of communication is directed (the agent knows what to communicate and to whom). Learning a complete communicative act (in the sense of what, when and to whom) has not been studied in the frame of MDPs.

The second problem in MDPs is that actions are all considered as atomic: they perform on one single execution step and the effect is immediately available to the agent or the environment (depending on the learning model). On the contrary, communication is a typical example of variable-length action, especially since MAS are asynchronous systems. When an agent *requests* another agent to perform a given action, it has no guarantee that the message will be processed immediately, even once the received has agreed to perform the action. To address this issue, one solution could be to use Semi-Markovian Decision Processes (SMDPs) that permit to model MDPs with variable-length actions [14, 2]. One advantage of SMDPs, as showed by [3], is that classical reinforcement learning algorithms can be extended to SMDPs.

However, using SMDPs does not appear as sufficient, because the Markov property is no longer valid in asynchronous MAS. Although most work in multi-agent learning admit that this property is not mandatory for the learning algorithm to converge to a solution, they all propose several heuristic to "reduce" the non-markovity when the algorithm does not converge. One proposed solution is to model the problem using Partially Observable MDPs (POMDPs [4]), *i.e.* MDPs in which the agent has access to only part of the problem's state. POMDPs allows to take into account the fact that other agents can act on the environment by representing these actions as hidden states from the learning agent's point of view. By definition, reinforcement learning

Cite as: , Author1, Author2 and Author3, *Proc. of 8th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2009)*, Decker, Sichman, Sierra and Castelfranchi (eds.), May, 10–15, 2009, Budapest, Hungary, pp. XXX-XXX.

algorithms defined in the MDP frame do not converge on a POMDP [16]. However, [12, 6, 5] showed that it is possible to convert a POMDP into a MDP using memory. As will be discussed later, this solution has to be adapted in the context of communicative acts learning.

In this paper, we propose an algorithm to learn communicative acts that takes into account the partial observability and asynchronism of MAS using SMDPs and memory.

This paper is organized as follows. In section 2, we briefly present the background knowledge on communicative acts and reinforcement learning algorithms. Section 3 presents our algorithm and section 4 discusses the perspectives of our work.

2. BACKGROUND

2.1 Agents Communications

In this paper, we will denote $\mathcal{A}$ the set of all agents. Each agent $Agt \in \mathcal{A}$ is characterized by a set of actions and data: $Agt = \langle A_{Agt}, D_{Agt}\rangle$.

The dataset D_{Agt} is a set of $\langle var, val\rangle$ pairs, where var is the name of the data and val its value. In our work, we distinguish three types of data:

- The agent's observations set: Ω_{Agt}. The variables in Ω represent the observations of the agent on the environment. The description of perception actions is out of the scope of this paper.
- The agent's beliefs set: B_{Agt}. These variables store information obtained by the agent through direct communication (as opposed to observations, which come from the agent's perception). The agent's beliefs can concern either other agent's state or the environment (for the elements it cannot observe directly).
- The agent's internal dataset Di_{Agt}. These variables are only known by the agent itself and cannot be "observed" by other agents.

The agent's state is characterized by D_{Agt}.

Each action $a \in A_{Agt}$ is a 3-tuple: $a = \langle name, \mathcal{P}, \mathbf{E}\rangle$ where:

- $name$ is a name of the action a. It is unique: each action a in Agt must have a different name.
- $\mathcal{P}$ is the set of action's preconditions, *i.e.* closed propositions built upon the agent's dataset D_{Agt}. These preconditions have to be true for the action to be performed by the agent. We will say that an action is *possible* when all its preconditions are evaluated to true. We will say it is *impossible* otherwise.

  We note $vars(p)$ the set of variables involved in the definition of a given precondition $p \in \mathcal{P}$. These variables will be used in our learning algorithm and protocols.
- $\mathbf{E}$ is the set of effects, *i.e.* closed positive and negative propositions built upon the agent's dataset D_{Agt} that will be true after the action's execution. We note $vars(e)$ the set of variables involved in the definition of a given effect $e \in \mathbf{E}$.

Given the agent's state s, we note $A_s \subset A$ the set of all possible actions in state s. In addition to these agent's actions, we will build a set of communicative acts, to be considered as additional possible actions by our learning algorithm.

2.2 Communicative Acts

In our model, a communicative act is a FIPA message [8, 7]. We did not address the problem of syntactic and semantic heterogeneity in this paper. For this reason, we assume that all agents use the same language and the same ontology. Moreover, to alleviate the notations, we will not represent the message's control tags (reply-with, conversation-id...). As a consequence a message is represented by a 4-tuple $\langle snd, rcv, p, c\rangle$ where $snd \in \mathcal{A}$ is the sender of message, $rcv \in \mathcal{A}$ is the receiver, p is the message's performative and c is a message's content, whose type depends on p. To comply with FIPA's standard, we use the following notation: $\langle snd, p(rcv, c)\rangle$

In this paper, we focused on communication for command and control, *i.e.* to query other agent's state or to request other agents to perform some action. To this purpose, we propose to use two different performatives : *query* and *request*.

Query

Messages of the form $m = \langle snd, query(rcv, v)\rangle$ allow agent snd to ask agent rcv the value of variable $v \in D_{rcv}$. It corresponds to the following FIPA request:

$$\langle snd, query - ref(rcv, Ref(x).value(rcv, v, x))\rangle$$

where the proposition $value(rcv, var, val)$ is true if and only if $(var, val) \in D_{rcv}$.

Agent rcv can reply with either:

- $\langle rcv, inform(snd, v = v_0)\rangle$ to assert that $(v, v_0) \in D_{rcv}$, as defined by FIPA.
- $\langle rcv, unknown(snd, v)\rangle$ to assert that variable v does not appear in D_rcv. It corresponds to the following FIPA message: $\langle rcv, not - understood(snd, \langle snd, m\rangle$ with m the initial message.

Request

Messages of the form $m = \langle snd, request(rcv, a)\rangle$ allow agent snd to ask agent rcv to perform action a, represented by its name[1]. Its semantics follows the Fipa's definition of the *request* performative. Thus, agent CV can reply with one of the following:

- $\langle CV, agree(Sand, a)\rangle$ to assert that CV accepts to perform a.
- $\langle CV, impossible(Sand, pe)\rangle$ to assert that rcv cannot execute a because preconditions pe are false. This corresponds to the following FIPA message: $\langle rcv, refuse(snd, \langle rcv, a\rangle, pe)\rangle$
- $\langle rcv, not - understood(snd, a)\rangle$ to assert that rcv cannot execute a because it does not know this action ($a \notin A_{snd}$). It corresponds to the following FIPA message: $\langle rcv, not - understood(snd, m)\rangle$ with m the initial message.

[1]In this paper, the name of an action encompasses not only the action's reference, but also the values of its parameters. For example: $take(a)$ and $take(b)$ are two different actions.

2.3 Q-Learning

Q-Learning [19] is a reinforcement learning algorithm that relies on the use of a function $Q : D * A \longrightarrow \mathbf{R}$ where $Q(s, a)$ denotes the expected reward for doing action a when in state s. The best action for state s is the action $a*$ where $a* = argmax_{a \in A} Q(s, a)$.

The Q-Learning algorithm iteratively builds the values of $Q(s, a)$. At every decisions epochs (*i.e.* every times the agent has to choose an action to perform), the agent chooses an action in A_s, performs this action, receives a reward $r(s, a)$ and observes his new state s'. It then updates the value of $Q(s, a)$ according to s' and $r(s, a)$ using the following formula:

$$Q(s, a) = (1 - \alpha_t)Q(s, a) + \alpha_t(r + \gamma \max_{a' \in A_{s'}} Q(s', a'))$$

where $\alpha_t \in [0, 1]$ is the current learning rate and γ is the discount factor (its value represents the agent's preferences: if the agent prefer instant reward, γ will be close to 0; if all rewards are equally important, γ will be close to 1).

The strategy used to select action to perform is the Boltzmann exploration [19, 1] where the probability of executing action a in state s is:

$$P(a_t = a | e_t = e) = \frac{e^{Q(s,a)/T_t}}{\sum_{b \in A} e^{Q(s,b)/T_t}}$$

where T_t is a temperature parameter that is decreased slowly according to time. When the temperature is high, the choice of an action a follows a uniform distribution. When it decreases, the select action a depends on the value of $Q(s, a)$ (an action a with a higher Q-value has more chance to be chosen).

On limitation of Q-Learning w.r.t. our problem is that it relies on a MDP model of the problem in which each action is performed within one single step. On the contrary, MAS are generally asynchronous. Communicative acts, for instance, take several steps to go.

2.4 Use of memory

The agent's memory stores its past actions and past observations. In MDPs, agents do not need any memory because of the Markov property:

$$\forall s' \in S \qquad P(s_{t+1} = s' | s_t, a_t) = P(s_{t+1} = s' | s_t, a_t, s_{t-1}, a_{t-1}, ..., s_1, a_1)$$

The agent's state depends only on the previous state and the performed action. In POMDP, state recognition techniques must use memory to distinguish two different states that correspond to the same observation from the agent's point of view [2]. Instead of defining the agent internal state by its observations only, the agent use its perception and its memory of past observations and actions. If the agent use enough memory, the problem can become markovian. However, the use of memory leads to a combinatorial explosion due to an increase of agent internal states. The method proposed by [12, 6, 5] is to find the minimal memory size for the learning algorithm to converge.

The general idea is to use a decision tree. The leaves of the tree represent the possible agent states, *i.e.* the information it must use to take its decisions. Each branch in the tree corresponds to an inserted distinction between two hidden states, using the memory. The first level represents distinctions made on the current observations, the next level represent distinctions on the immediately previous action, the third level adds new distinctions about previous observations, and so on... Figure 1 illustrates this process. The agent memory is characterized by the path from the root to the current observable state (leaf).

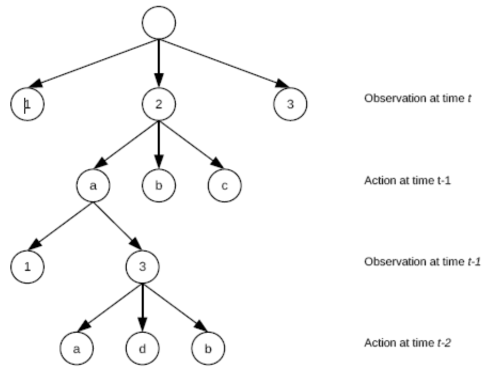

Figure 1: A decision tree : Perceptions are represented by integers and actions by letters.

The tree is constructed iteratively:

- In the beginning, the agent has no memory. It builds the first series of children (that correspond to the first-depth nodes) using the state S it observes.
- After $Niter$ decision epochs ($Niter$ being fixed experimentally), the N most ambiguous states are uncovered. An additional level of memory is added to the agent which enables it to memorize the last action it has performed. The agent's state is then characterized by $S + M_1$ and we start again the learning process. Note that M_1 is only used in the N ambiguous states (in other "unambiguous" states, the agent only uses S as its state).
- The process iterates. After k stages, the memory M_k used for ambiguous states contains k elements: S , action, S , action...

In order to determine when to introduce a new distinction, [6, 5] propose to use the three following criteria:

Ambiguity of optimal action : A state s is more ambiguous when its optimal action isn't clearly defined, because the Q-values of two best actions $Q(s, a_1)$ and $Q(s, a_2)$ are very close.

Number of Q-Values updates : A state s is more ambiguous when its Q-Values $Q(s, .)$ are often updated. Indeed, a state which is often encountered could correspond to several hidden states of environment.

Convergence of Q-Values : A state s is more ambiguous when the convergence of the Q-value $Q(\omega, a)$ for ambiguous observations is slower than for non-ambiguous observations. In order to detect this property, [6] records for every actions a the last variations of Q-value $Q(s, a)$ and state s is considered more ambiguous when the variations of $Q(s, .)$ are important.

Every state is ranked according to the sum of its score w.r.t. each criterion. The N more ambiguous are then associated to an additional level of memory.

The algorithm stops when the size of the agent's memory reaches a threshold or when the gain in performance between the last two trees is not significant. The performance of a tree is the average reward obtained by the agent when using the policy induced by this tree.

This algorithm appears as a very promising method to deal with non-markovity issues. However, it requires that the agent knows the complete set of actions it can execute in every state s. In your problem the agent first has to construct its set of communicative acts to fulfil this need.

3. LEARNING COMMUNICATIVE ACTS

In this section we present our communicative acts learning algorithm. Our solution takes into account partial observability and asynchronism hypothesis of MAS. First of all, we show how the learning agent can determine the content of messages, using simple protocols. Then, we present our reinforcement learning process for selecting the agent's actions and communicative acts. We show how the agent can learn to wait for effects of its directives messages to occur. Last, we show how our algorithm can be adapted to deal with the non-markovity hypothesis by disambiguating the agent's observations.

3.1 Communicative acts construction

In this paper, we only consider messages with performatives *query* and *request*. The initial difficulty in learning communicative acts is to determine the content of messages, which cannot be given *a priori* in an open MAS. To this purpose, we propose to define two simple protocols of interaction which are based on two hypothesis. First, agents are fully cooperative, which means that they perform all requested actions (as long as they are possible) and that they reply correctly to all *query* messages (they don't lie). Second, our agents are introspective: they can reason about their own actions and data at runtime.

The general idea is that the agent explores the MAS in order to construct a maximum of communicative acts and to test the messages. We deal with the exploration/exploitation dilemma by using a parameter δ which decreases when exploration doesn't provide satisfying results. Note that this parameter is different from Boltzmann's temperature which determines the action to use in a given state. The parameter δ determines if the agent have to continue to construct actions or learn to use this constructed actions in a given state.

3.1.1 *Building* Request *messages*

To construct the content of messages *request*, we use a protocol *what-order* as illustrated in Figure 2.

The $what-order$ performative, used with an empty content, was proposed by [20, 13]. The $\langle snd, what{-}order(rcv, \emptyset)\rangle$ message corresponds to the following FIPA request:

$$\langle snd, query(rcv, Ref(x).possibleAction(x, rcv))\rangle$$

where possibleAction(x,rcv) means that rcv can perform action x. It allows snd to query the set of possible actions of rcv.

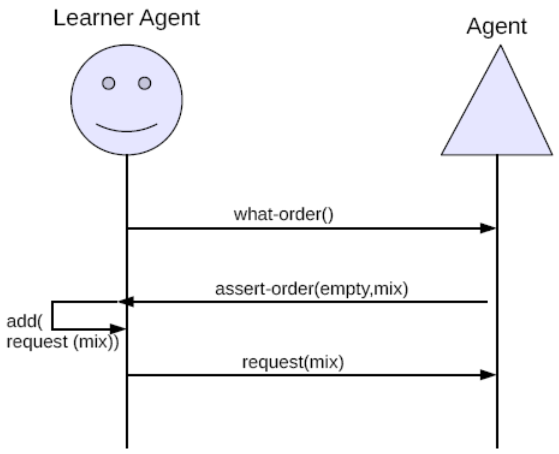

Figure 2: Protocol what-order

The expected answer is a message of the form:

$$m' = \langle rcv, assert - order(snd, \langle a_1, .., a_n\rangle)\rangle$$

with $\forall i \quad a_i$ the name of rcv's action. From this answer, the learning agent can build a set of potential communicative acts. $\forall a_i \in content(m')$, the message $\langle snd, request(rcv, a_i)\rangle$ is a valid communicative act. From the implementation point of view, the agent simply adds these acts to its set of actions A_s (s is the agent's state when it launches the *what-order* protocol).

3.1.2 *Building* Query *messages*

We construct the content of *query* messages by using the content of failures in *request* messages. When the receiver cannot perform the requested action, it replies with an *impossible* message (as presented in section 2), which contains the set of failed preconditions of the requested action. Indeed, each failed precondition p contains a set of variables $vars(p)$ which can be used to build new *query* messages: $\forall p \in m', \forall d \in vars(p)$, the message $\langle snd, query(rcv, d)\rangle$ is a possible communicative acts (the agent can add it to its set of actions A_s). The use of message of performative *impossible* is illustrated in Figure 3.

3.1.3 *Exploration algorithm*

To every decisions epochs, a variable *Rand* is randomly generated:

- If *Rand* $< \delta_s$ with s the agent's state, the agent will use a protocol *what-order* to determine new communicative acts[2]. The value of parameter δ is immediately corrected according to the reply of the message: If the agent receives a message *assert-order* containing at least a new action, then the value of δ is increased : $\delta = \delta * 2$ (note that since δ represents a probability, we limit its upper value to 1). Otherwise, δ is decreased : $\delta = \delta * 0.8$.
- If *Rand* $\geq \delta_s$, the agent choose an action to perform among the set A_s of possible actions in state s, using the Boltzmann's temperature.

[2]The receiver of the message *what-order* is chosen according to a uniform distribution.

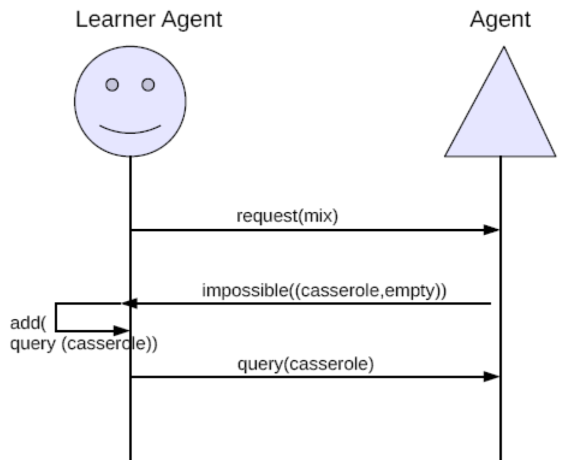

Figure 3: Use of message of performative impossible

3.2 Learning mechanism

For the agent to learn when to use its actions (own actions and constructed communicative acts), we use a classical Q-Learning algorithm. However we this approach raises three problems.

First, agents are asynchronous in a MAS: times of reply to communicative acts could be variable. Therefore the agent's actions have different times of execution. And yet MDP and POMDP apply to action which take one time step which is permanent and identical for every agent's action. That is why, we use the SMDP which enables to model MDP where actions are performed for variable times and resolves with adapted Q-Learning.

Second, we have to anticipate the absence of answer (either due to message loss or to the an agents' autonomous decision not to answer). To avoid agents to wait for an answer indefinitely, we use a time-out limit, beyond which the agent will consider its message as lost.

Third, the reception of an *agree* message inform the agent that its peer agrees to perform the requested action, but it doesn't mean that the execution of action is finished. In the FIPA-Request protocol, this issues is solved by the use of *confirm* messages to indicate that an action is finished[9]. However, this approach has several limits. On the one hand, this solution is requires additional message sending (which can be extremely costly). On the other hand, it is hardly compatible with learning algorithms, which require the agent to wait for the result of an action to try and learn another one. Using *confirm* messages in our learning algorithm would mean that, in addition to the possible loss of this confirmation, the agent could not do anything else until the requested action is actually performed, which is totally in contradiction with the MAS paradigm. For these reasons, we proposed another mechanism, based on learning to wait for the actual effects of *request* communicative acts to apply.

3.2.1 Using SMDPs

Semi-Markov decision processes (SMDPs) are MDPs in which actions take more than one time step[2, 14] An SMDP is a 5-tuple $\langle S, A, p, r\rangle$ where:

- S is a set of states.
- A is a set of actions.
- $P(\tau, s'|s, a)$ is a probabilistic transition function mapping $S \times A \times S \times \mathbf{N} \longrightarrow [0,1]$ where $\mathbf{N}$ is a natural number specifying action duration
- $r : S * A \longrightarrow \mathbb{R}$ is a reward function.

The Q-Learning algorithms can be applied with the following formula[3]:

$$\begin{aligned} Q(s_t, a) \quad &= \quad (1-\alpha_t) * Q(s_t, a) + \alpha_t * [\gamma^k \max_{a' \in A'(s_{t+k})} Q(s_{t+k}, a') \\ &\quad + r_{t+1} + \gamma r_{t+2} + .. + \gamma^{k-1} r_{t+k}] \end{aligned}$$

where:

- k is the number times step necessary for the execution of action.
- $A'(s_{t+k})$ is the set of action at times $t+k$
- r_t corresponds to given reward at times t
- The parameters α_t and γ are the same parameters that classical Q-Learning

3.2.2 Time-out

In order to solve the "no-reply" problem, we used a time-out. If the agent does not receive an answer to its communicative act after this time-out, it stops this action and it doesn't modify the Q-Value of this message, because it assumes that its message has been lost.

At each decision epoch, the agent choose an action a to execute and the counter k, which stores the number of steps since the agent has started this action (initially, $k = 0$). At every time step, k is increased by 1 until:

- $k > TLIM$. In this case, the agent stops its action.
- Action a is finished (an answer was received). The agent then modify the Q-value $Q(s, a)$ (where s was the agent's state when the action a was launched).

3.2.3 Learning to wait

When the agent receives a message with performative *agree*, this does not mean that the requested action is over, but simply that the requested agent agrees to perform it, at some time in the future. However, our learning agent cannot use the FIPA solution [9] based on the $confirm$ performative for several reasons. First, all these $confirm$ messages might be costly. Second, it means, again, to use a timeout to deal with the no-reply issue. Third, in most situation, if the interesting effects of the action occur before the $confirm$ message has been received (either because the action is complex and involves several sub-effects, only some of them being interesting for our agent, or because another agent changed the environment's state in between), our learning wastes a lot of time waiting for a useless *confirm* message. On the contrary, it could have used its observations and launched an other action.

For these reasons, we propose to use another solution: an action $wait = \langle wait, \emptyset, \emptyset\rangle$ has been added in the set of possible actions of the agent, which enables the agent to wait for the effects of requested actions. Thus, our learning agent will not only learn to act and communicate, but also to wait, through the SMDP-adapted Q-Learning algorithm (see the section 3.2.1). Note that the *wait* action is not very costly since the agent doesn't do anything. Furthermore the use of this *wait* actions enables to the agent to launch other actions (instead of waiting), so as to have the system perform actions in parallel whenever possible.

3.3 Using memory

As was explained in section 2, the Markov property is not valid in a MAS. On the one hand because agents cannot always observe completely their environment. On the other hand because the other agents act on environment too. In this paper we are interested in single learning agent. Therefore the set of actions performed by the other agents could be model as environment's states which are hidden to the learning agent. That is why we can model our problem with a POMDP.

To ease the convergence of the Q-learning algorithm on our POMDP, our idea it to use the *query* messages to obtain information about the environment and other agents' data. This information is stocked as beliefs in our model. However, since each belief can become false during the learning process, our agent has to learn how long it must to remember a given belief.

To do this, we choose to use the memory-based learning algorithm which was introduced in section 2.4. The general principle is as follows: every belief $b \in B$ which is received after a *query* sending is added to the agent state for the next decision epoch only and discarded in the immediately following state. However, since the agent state includes its observations and its beliefs, this knowledge is not lost. Actually, instead of learning how long the agent has to store a belief, the algorithm presented in 2.4 enables the agent to learn the number of times step that passed since the agent has received a given belief, which is even more informational.

Another important motivation to choose the use of memory enables the agent to remember its own past actions. Using this information, the agent can learn, for instance, what communicative act *query* is more relevant after a given communicative act *request*.

Furthermore the use of memory will allow to avoid blocking situations due to use of wait action. If it needs to wait for in the state s when learning agent have just launched an action a, it doesn't seems relevant to wait for in the same state s when learning agent have just wait for at the last decision epoch. Without memory, agent could learn to wait in state s and so could be blocked in this state if this isn't changed by an other agent or by effects of previous action. To the contrary with memory the learning agent could learn to wait after launching of request message and not after launching a wait action. As was explained , a node where the best action is a *wait* action, is a node very ambiguous, that is why we consider this information to choice the most ambiguous nodes.

Algorithm using memory

The learning agent A_l is an agent as presented in the section 2.1 with a memory $\mathcal{M} = \{a_t, e_t, a_{t-1}, e_{t-1}, ..\}$ where $a_t \in A, e_t \in E$. $A_l = \langle Agt, \mathcal{M} \rangle$. As McCallum in [12] we use a tree which represents states that the learning agent uses to learn. Every node $s \in N$ of this tree is a 7-tuple

$$\langle Di, A, Q, \Delta Q, UpD, \delta, \mathcal{F} \rangle$$

- Di is a distinction. It is equal to the value of the branch which go in the node N. $Di \in \{E, A\}$. $E = \langle \Omega, B \rangle$ is the set of agent's state and A is a set of actions.
- A_s is the set of actions that the learning agent can perform when it is in the state s. this set contains communicative acts and an action *wait*.
- Q_s is the set of Q-Value for the state s. $Q_s = \{q_a\}$ where $a \in A_s$
- $\Delta Q_s = \{\Delta q_a | \Delta q_a(t+1) = |q_a^{t+1} - q_a^t|$ with $a \in A_s$ and $q_a^t, q_a^{t+1} \in Q_s\}$.
- UpD is the number of update of this node e, *i.e.* the number of times where this node has been visited (and updated) during the learning process.
- δ is the parameter of exploration. It defines the probability that the learning agent chooses to perform a *what-order* protocol.
- $\mathcal{F}$ is a set of node's children such $\mathcal{F} \subset N$

At the end of the learning stage, the N most ambiguous nodes are detected. A node is most ambiguous than an other node if its best action is the *wait* action. In the equality case, our algorithm use the heuristic of Dutech and al.[6, 5] to decide between two nodes.

- The ambiguity rate of node s is defined by the following function:

$$\begin{aligned} ambiguous(s) &= wait(s) \\ &+\frac{1}{3}(rank_s(UpD_s) + \frac{1}{|\Delta Q_s|} \\ &+(rank_s(\frac{1}{|\Delta Q_s|}\sum_{a \in A_s})\Delta q_a) \\ &+rankD_s(q_{bestAction_s} - q_{bestAction2_s})) \end{aligned}$$

  Where $rank_s(x)$ returns the position of node s, if we order all the nodes in increasing order according to x. $rankD_N(x)$ returns the position of node s, if we order all the nodes in decreasing order according to x. $bestAction_s$ (respectively $bestAction2_s$) returns the best action (respectively the second best action) for the node s and $wait(s)$ returns 0 if the *wait* action is the best action for the node s or a constant value M otherwise (in our implementation , $M = 10000$ which makes this criterion predominant over all others)

- The N most ambiguous are theirs which have the smallest ambiguity rate: $\{Nd | leaf(Nd, tree), rank(ambiguous(Nd))$ $N\}$

As proposed by the initial algorithm [6, 5], we stop our learning process either because the maximal size of the agent's memory has been reached, or because the performances of tree cannot be improve adding memory. More formally, our algorithm stop when :

$$\begin{aligned} arrest() = \quad & 1 \text{ If } t > Mlim \text{ Or} \\ & |\frac{1}{N}\sum_1^N(R(tree_t)) - \frac{1}{N}\sum_1^N(R(tree_{t-1}))| < \epsilon \\ & 0 \text{ Else.} \end{aligned}$$

where $R(tree_t)$ is the given reward by the learning agent when it uses the tree $tree_t$ and $Mlim$ the maximal size of the agent's memory.

3.4 Implementation and ongoing evaluation

Our reinforcement learning algorithm has been implemented on the VDL[3]multi-agent platform, which offers adequate introspection capabilities for our model [13]. We programmed a simple example of asynchronous MAS , based on the classical "maze learning" problem. We chose this example because it can be easily tuned to different levels of complexity, which allows us to test all the component of our model (what-order protocol, learning of wait actions, memory...).

Currently, our algorithm converges for a learning agent controlling a robot asynchronously (the agent commands are not immediately proceeded by the robot) in 8*8 map with 1 treasure (success) and 25% deadly holes (failure). To perform this task , the learning agent uses wait actions and request messages built after the robot's capacities (i.e. movement commands). The obtained results are very promising: the robot (piloted by learning agent) find the treasure 95% of the time when using 2 levels of memory.

We are currently completing our experimental model in order to introduce 1) the construction of query acts and 2) their use when the learning agent can only partially observes the environment[4]. Our first aim is to validate our results by comparing our solution with "classical" learning algorithm that do not make use of communication such as [6, 5]. By studying the convergence of the algorithm, the cost of communication and the gain in the resulting policy, we intend to demonstrate that learning communicative acts greatly improves the quality of the learning process. This part is still under realisation.

4. CONCLUSION

In this paper, we tackled the problem of reinforcement learning of communicative acts in a multi-agent context. To this purpose, we outlined three key issues in relation: the construction of the communicative act itself, its assimilation to classical actions (so as to apply classical learning algorithms) and the management of non-markovity aspects due to the MAS context. In order to build the communicative acts, we proposed a solution that relies on the agent's introspection capabilities and its capacity to reason about its own actions. The resulting acts are seen as variable-time actions in a Semi-Markovian Decision Process. Our final algorithm deals with non-markovity by using both the communicative acts it builds and a memory of the agent's past actions and observations.

Our approach aims at improving reinforcement learning in multi-agent systems. However, it raises several research perspectives. First, in order to avoid combinatory explosion in space and time, we have to assume that all learnt actions were useful, as most current work do. In a more realistic frame, this hypothesis is no longer valid. It would be interesting to define a mechanism that allows the agent to only store actions that are useful to its problem, especially for communicative actions that can be numerous and particularly inappropriate.

[3]`http://www-poleia.lip6.fr:8180/ sabouret/demos/index.html`

[4]For instance, when adding a moving monster in the maze, our agent could learn to query the monster about its position

Another research direction is to define a better heuristic for the agent's memory. The heuristic we used, directly taken from [6, 5], which has very good results in zero-communication MAS. Using information provided by communication (*e.g.* the number of failed communicative acts for each state of the agent), we think this heuristic can be greatly improved. Moreover, it could be interesting to define a mechanism to compute which type of information has to be stored (actions or observations), depending on the current state.

In longer term, we would like to extend our research to the whole range of communicative acts, so as to benefit from the richness of Agent Communication Languages. Learning how to communicate is the next step to adaptive open MAS.

5. REFERENCES

[1] B. Barto, A. Real-time learning and control using asynchronous dynamic programming. Technical report, Department of Computer Science, University of Massachusetts, Amherst, 1991.

[2] D. P. Bertsekas. *Dynamic Programming and Optimal Control.* Athena Scientific, Belmont , Massachusetts, 1995.

[3] S. J. Bradtke and M. O. Duff. Reinforcement learning methods for continuous-time markov decision problems. In *Advances in Neural Information Processing Systems 7*, pages 393–400. MIT Press, 1995.

[4] A. R. Cassandra, L. P. Kaelbling, and M. L. Littman. Acting optimally in partially observable stochastic domains. In *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94)*, volume 2, pages 1023–1028, Seattle, Washington, USA, 1994. AAAI Press/MIT Press.

[5] A. Dutech. Solving pomdps using selected past events. In *ECAI*, pages 281–285, 2000.

[6] A. Dutech and M. Samuelides. Un algorithme d'apprentissage par renforcement pour les processus décisionnels de Markov partiellement observés : apprendre une extension sélective du passé. 2003.

[7] FIPA.org. Fipa communicative act library specification. 1997.

[8] FIPA.org. Fipa acl message structure specification. 2002.

[9] FIPA.org. Fipa request interaction protocol specification. 2003.

[10] J. Austin. How to do things with words. *Oxford University Press, Oxford England*, 1962.

[11] M. J. Matari'c. Using communication to reduce locality in distributed multi-agent learning. *Journal of Experimental and Theoretical Artificial Intelligence*, 10:357–369, 1998.

[12] A. K. Mccallum. *Reinforcement learning with selective perception and hidden state.* PhD thesis, 1996. Supervisor-Dana Ballard.

[13] Nicolas Sabouret. A Model of Requests about actions for active components in the semantic web. 2002.

[14] M. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming.* John Wiley & Sons, Inc, New York, Etats-Unis, 1994.

[15] J. R. Searle. Speech acts: An essay in the philosophy of language. 1985.

[16] S. P. Singh, T. Jaakkola, and M. I. Jordan. Learning without state-estimation in partially observable markovian decision processes. In *International Conference on Machine Learning*, pages 284–292, 1994.

[17] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press.

[18] M. Tan. Multi-agent reinforcement learning: Independent vs. cooperative agents. In *Proceedings of the Tenth International Conference on Machine Learning*, pages 330–337. Morgan Kaufmann, 1993.

[19] C. J. C. H. Watkins. *Learning from delayed rewards*. PhD thesis, Cambridge University, Cambridge, United Kingdom, 1989.

[20] N. S. Y. Charif. An Agent Interaction Protocol for Ambient Intelligence,. In *2nd International Conference on Intelligent Environments (IE'06)*, 2007.