IADIS International Journal on Computer Science and Information Systems
Vol. 4, No. 3, pp. 82-98
ISSN: 1646-3692

USING "SOCIAL ACTIONS" AND RL-ALGORITHMS TO BUILD POLICIES IN DEC-POMDP

Thomas Vincent. *Equipe MAIA LORIA UHP NANCY 1 BP 239 54506 Vandoeuvre Lès Nancy, France.*
Vincent.Thomas@loria.fr

Akplogan Mahuna. *INRA, UR 875 BIA F-31326 Castanet-Tolosan, France.*
Mahuna.Akplogan@toulouse.inra.fr

ABSTRACT

Building individual behaviors to solve collective problems is a major stake whose applications are found in several domains. To do so, Dec-POMDP has been proposed as a formalism for describing multi-agent problems. However, solving a Dec-POMDP turned out to be a NEXP problem. In this study, we introduced the original concept of social action to get round the inherent complexity of Dec-POMDP and we proposed three decentralized reinforcement learning algorithms which approximate the optimal policy in Dec-POMDP. This article analyses the results obtained and argues that this new approach seems promising for automatic top-down collective behavior computation..

KEYWORDS

Multi-agent systems, Markov decision processes, reinforcement learning, interaction.

1. INTRODUCTION

Multi-agents systems are defined as groups of autonomous entities, called agents, which are set in a shared environment and can interact with this environment and with each other. These systems constitute a new way of addressing problems by focusing on decentralized control and have many applications from network, to distributed control of process (Van Parunak 1994). However, building distributed control among agents is still an open question due to the complexity of considering the interactions among agents.

This article focuses more precisely on cooperative multi-agent systems whose efficiency is assessed by a global utility function. The objective of the work is to propose new directions to control these systems while taking local and practical constraints into account: agents have only a local view of the system, they cannot directly assess the global utility function and can only have few local information exchanges.

These constraints lead us to investigate decentralized reinforcement learning approaches. The originality of this paper lies in the introduction of explicit interactions, called social actions, among agents. With the help of social actions, agents can consider the other agents and can reason on a more collective level. This work presents also several original algorithms based on reinforcement learning and two heuristics to take advantage of social actions and make the agents self-organize.

This article is organized as follows. Section 2 describes the Dec-POMDP formalism. It presents some of the existing approaches to build collective behaviour and their drawbacks. Section 3 proposes a new formalism based on Dec-POMDP with the addition of the social action concept. It describes decentralized learning algorithms based on social actions so that the agents can take the other agents into account. Section 4 presents two specific problems on which these algorithms have been applied and compares the results obtained by our algorithms to the results obtained by other decentralized learning approaches. Section 5 highlights the interests of the proposed approach and Section 6 concludes.

2. BACKGROUND

2.1 Dec-POMDP Formalism

The Dec-POMDP (Decentralized Partially Observable Markov Decision Process) formalism is a MDP extension that has been proposed by (Bernstein et al 2002) to formalize multi-agent decision problems. A Dec-POMDP for two agents is defined by a tuple $<S,A_1,A_2,T,R_1,R_2,G_1,G_2,O>$ where

- S is a finite set of states
- A_1, A_2 are finite sets of actions
- T is a transition probability table, $T(s, a_1, a_2, s')$ gives the probability to reach state s' from state s when agents take actions a_1 and a_2
- R is a reward function, $R(s,a_1,a_2)$ gives the immediate reward earned by the system when agents execute actions a_1 and a_2 from state s
- G_1 and G_2 are sets of observations
- O is the observation function giving for each agent its observation depending on the state.

A Dec-POMDP defines the possible actions for all agents as well as the dynamics of the process. The main characteristic of a Dec-POMDP is that its execution is decentralized: each agent can make decision only on the basis of local perception history.

Moreover, the transition matrix is a function of both actions of the agents. Because of that, an agent cannot predict the global effect of its action since this effect depends also on the action taken by the other agent. The actions of the agents can then only be interpreted as influences exerted by the agents to modify the trajectory of the process.

2.2 Solving Dec-POMDP Problem

The behaviour of an agent is characterized by a policy, defined as a function giving an action for each observation history. Solving a Dec-POMDP consists then in finding the local policies of the agents so that they maximize the cumulated reward received during their execution. This problem has been proven to be NEXP (Bernstein et al 2002) and several directions have been followed to reduce its inherent complexity, like taking advantage of the structure of some problems (Guestrin et al 2002 and Becker et al 2003) or using approximation techniques (Szer et al 2006). However, most of the existing works focus on centralized approaches in which the construction of behaviours is done by a single system with global knowledge.

This article has made the choice of a more practical point of view. Indeed, the issue of solving a Dec-POMDP is too complex to be tackled directly, even with a centralized approach. So, we preferred to search approximate solutions and to focus on decentralized adaptive solutions. In these solutions, each agent has to adapt itself to an initially unknown system and to the other agents. This leads to a recursive problem where learning the best action for an agent depends on what the other agents learn.

Nevertheless, we think that these approaches are promising since it is often difficult to have access to the whole system to centralize behaviour construction. In the next sections, we will first focus on how to build an adaptive agent by using a reinforcement learning approach called Q-learning, then consider how reinforcement learning behaves in multi-agent systems and what are the current propositions to solve the raised issues.

2.3 Q-learning Algorithm

Q-learning is a reinforcement learning method (Watkins et al 1992) that can compute optimal policies in single agent problems. Therefore; it constitutes a solid basis for adaptive behaviour and will be used by our approach.

In Q-learning, the policy of the agent is parameterized by *Q-values(s,a)* which gives the expected cumulated reward when the agent does action *a* in state *s*. *Q-values* are updated online after each action of the agent to incorporate its whole experience. This update is made according to the following formula based on dynamic programming.

$$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha(r + \gamma \max_a Q(s',a))$$

After each action *a* leading from state *s* to state *s'*, the new expected cumulated reward from *s* corresponds to a combination (due to the alpha parameter) between the previous expectation from *s* and the new assessment made from the new experience. This assessment corresponds to the immediate reward *r* plus the discounted expected reward from arrival state *s'*.

This update has been proven to lead to the exact expectations if some conditions are met : the world need to be stationary, each couple *(s,a)* must be visited an infinite number of times and the alpha parameters must have certain properties depending on *t* (Watkins et al 1992). Thus, once these *Q-values* have been learnt, they can be used to produce the optimal behaviour of selfish agents put alone in a environment. The agent in state *s* has only to select the action *a* with highest *Q-values(s,a)*.

2.4 Multi agent Reinforcement Learning

However, when agents learn together, they have to face new issues like

- **conflicts** between collective and individual interests (Hardin 1968), explained by the fact that local individual maximizations does not lead to global maximization,
- **co-adaptation**, the fact that the best behaviour for an agent depends on the behaviour of the others agents,
- and **credit assignment issue**, the fact that an agent cannot determine with certainty which agent is responsible for a task advancement and for the earning of a reward (Weiss 1996).

(Busoniu et al 2008) presented a survey of multi-agents reinforcement learning approaches. Most of the approaches need agents that completely observe the system and there are still few works that focus on Dec-POMDP (where agents have only access to a local observation of the state). These works are presented in the following and the results of our algorithms will be compared to some of them.

- (Sen et al 94) proposed to use **independent learner agents** ruled by reinforcement learning to produce adaptive agents but this approach shows rapidly its limits due to the co-adaptation issue.
- (Schneider et al 1999) proposed **Distributed value function** where agents try to maximize a balanced sum of its own reward and the rewards of its neighbours. Thus, each agent can consider the satisfaction of the other agents while computing its behaviour. But the way reward is distributed is static and made by the conceiver before the learning phase.
- (Guestrin et al 2002) proposed **Coordination based methods** taking advantage of the structure of the problem, but the computation of policies requires constant communications.
- (Chades et al 2002) proposed **Co-evolution** to deal with co-adaptation issue. In this algorithm, each agent successively plans its actions while the policies of the other agents remain constant. But, this algorithm needs a global coordination mechanism and only leads to local optima.
- (Buffet et al 2007) proposed **incremental learning** where agents are confronted to more and more difficult situations to progressively learn how to improve their collective behaviour but this approach requires work from the conceiver to propose situations of incremental complexity.

In order to do decentralized learning, we propose to introduce interactions among agents. Each agent could then explicitly consider its relationships with the other agents and decide how it interacts with them.

Thus, the first contribution of this article is to add a new concept to the Dec-POMDP formalism in order to model interactions: the concept of social action. The second contribution is to introduce two heuristics to build collective behaviours on the basis of reinforcement learning and these social actions.

3. SOCIAL ACTIONS AND RL-ALGORITHMS

3.1 Formalization of "Socials Actions"

We first propose to represent interactions among agents as social actions. A **social action** is defined as a local joint action which is triggered by a specific agent and involves two agents. It can be formally defined by a couple of actions: an individual **original** action with an associated individual **symmetric** action. When agent *A* performs a social action, it executes the original action while agent *B* is forced to carry out the symmetric action. Thus, the execution of a social action by an agent constrains the actions of the other agent.

For instance, let us consider a social action called "exchanging resource X". This social action can be formally defined as the joint action {*PuttingDown(X), Taking(X)*}. When agent *A* performs this social action, it executes the original action *PuttingDown(X)* and agent *B* is forced to execute the symmetric action *Taking(X)*. The resource *X* is thus transferred from agent *A* to agent *B*.

Our algorithms will consider Dec-POMDPs with the addition of social actions as pairs of *{original action, symmetric action}*. The Dec-POMDP problem is then changed in this way:

- The set of possible actions for agent *i* is the set of individual actions from the Dec-POMDP plus the set of social actions. At each step, an agent has to execute an individual or a social action.
- The global reward can be broken down into additive individual rewards. These rewards are locally perceived by the agents. The collective goal is to optimize the discounted sum of global rewards over time.

Since the execution of a social action generates constraints for one agent, the execution of a social action must be negotiated by the two involved agents. The next section details how this negotiation is made.

3.2 Decision about a Social Action

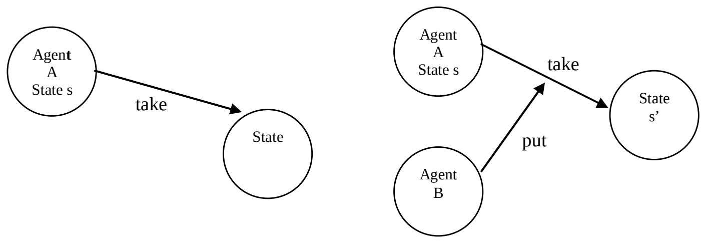
To correctly decide if the social action has an advantageous effect for the group, agents have to exchange information about perceptions, skills, past and future actions. In reinforcement learning, the *Q-values* integrate useful information regarding the task to be done, so we made the assumption that when *Q-values* are judiciously exchanged, they can convey enough relevant information for making this collective decision.

Therefore, the execution of a social action is negotiated on the basis of two heuristics involving *Q-values*. The aim of the first heuristic (section 3.2.1) is to assess the utility of social actions in order to compare them to other actions. The aim of the second heuristic (section 3.2.2) is to distribute the gain of a social action among the agents so that the agents might learn to trigger and to reproduce beneficial social action in the future.

3.2.1 Assessing the Utility of a Social Action

In a single-agent case, the agent chooses its action according to its utility. The utility of performing an action is defined by the sum of immediate reward and the discounted sum of future expected rewards from the arrival state *s’*. As explained in section 2.3, when the agent is alone in the environment, it can correctly learn its *Q-values* giving for each couple *(s,a)* the utility of the action *a* from state *s*. *V(s)* is defined as the utility associated to state *s* and equals to $\max_a Q(s,a)$. If *Q-values* are correctly learnt, when the agent takes action *a* from *s* to *s’*, the difference between the obtained cumulated reward by arriving in *s’* and the expectations from *s* corresponds to $(r_i + \gamma V_i(s')) - V_i(s)$ and is null by definition.

However, when the agent executes a social action, this social action involves another agent and the agent may arrive in a certain state *s’* which is not the state in which it would arrive if it was alone. The previous difference between its *Q-value* from start state *s* and its expected utility from arrival *s’* is not null anymore. This difference is the consequence of the influence of the other agent involved in the social action.



a) Individual action b) Social action involving agent B

Figure 1. Example of a difference between individual and social action.

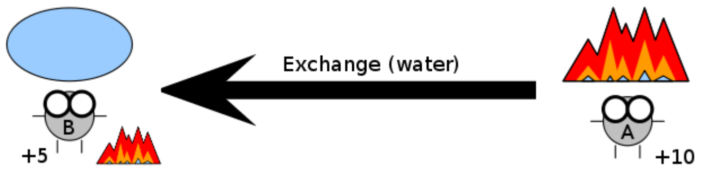
In case *a*, agent A can assess its individual action on the basis of its own individual Q-values

whereas in case *b*, it has to consider the action made by agent B leading to a new state.

This is illustrated by Fig.1. In this example, agent A can decide to take an individual ‘*take*’ action (case *a*) or a social ‘*exchange*’ action (case *b*). The individual ‘take’ action leads the agent to a new state but, this state is taken into account in its individual Q-values it has learnt alone when it was alone in the environment. However, the social action leads to a state *s’* due to the symmetric action exerted by agent B. The received reward does not correspond anymore to what agent A might have learnt while it was alone.

We defined the **gain of a social action** as this difference $(r_i + \gamma V_i(s')) - V_i(s)$. It represents the gain obtained by an agent when the social action is executed rather than any individual action. This gain determines the interest an agent has in using this specific social action.

In order to explain what is this gain, let us consider a firemen problem with two agents A and B whose goal is to extinguish a fire (Fig 2.). The agent B can receive a reward of 5 by extinguishing a small fire whereas agent A can receive a reward of 10 if it extinguishes the main fire. Both agents need water but only B has some. If B uses the “Exchange(water)” social action with A. B will give the water to A, A will take it and benefit from this acquisition. A can then expect a future gain of rewards of 10 whereas B can expect a loss of 5. Each agent can assess its gain by the difference:



$$gain_A = [r_A + \gamma V_A(s_A') - V_A(s_A)] = [0 + 1*10 - 0] = 10$$

$$gain_B = [r_B + \gamma V_B(s_B') - V_B(s_B)] = [0 + 1*0 - 5] = -5$$

Figure 2. Example of firemen problem

Once computed, this gain must be shared among the agents so that both agents are aware that this social action can be beneficial. This distribution is made through social rewards described in the next section.

3.2.2 Distributing the Social Rewards

The aim of social rewards is to stimulate other agents to reproduce a situation where a specific social action might be interesting. The idea is intuitive: when an agent has an interest in a specific social action, it will give a part of his gain to the other agent involved in this social action. This gain is transferred through a social reward so that the agents can add their social rewards to their immediate rewards and learn them by the same reinforcement learning method.

The way social reward is computed is the second heuristic of our algorithm. It consists in equally sharing the gains of a social action and is made according to the following equation (with lambda equal to 0.5):

$$r_{s,A} = -gain_A + \lambda(gain_A + gain_B),\ r_{s,B} = -gain_B + \lambda(gain_A + gain_B)$$

For the firemen problem, A will give a part of its gain to B. This social reward equals to 7.5 (5+0.5*(10-5)) in this case and will compensate the loss of reward of agent B for giving its water. On the contrary, this social reward will be deduced to agent A expected cumulated rewards which will still be positive.

3.3 Algorithms

Three slightly different algorithms have been proposed on the basis of these two heuristics. Each one of these algorithms tries to compute *Q-values* associated to individual and social actions by a decentralized learning. All these algorithms are constituted by two phases. In the first phase, each agent is put alone in the environment and builds an individual policy by using Q-learning algorithm. In the second phase, all the agents are put in the same environment. They can now interact and they use one of the following algorithms to assess and update the utilities of social actions and if needed, the utilities of individual actions.

- In **SimpleIncomeExchange**, the agents do not learn social action. However, they can use a social action during exploitation. The utility of a social action is then assessed online when it is needed.

$$\pi_i(s_i) \leftarrow \arg\max_b \begin{cases} Q_i(s_i, b) & \text{si } b \in A_i - SocAct_i \\ r_i + \gamma V_i(s_i') + \boldsymbol{gain_j} & \text{si } b \in SocAct_i \end{cases}$$

- In **IncomeExchange1**, the agents learn simultaneously the utilities of individual and social actions by exchanging social rewards. However, only the utility of social actions is updated according to the utilities of individual actions. In other words, agents cannot consider executing several social actions.

$$Q_{soc_i}(s_i, a_i) \leftarrow (1-\alpha)Q_{soc_i}(s_i, a_i) + \alpha(r_i + \boldsymbol{r_{s,i}} + \gamma \max_b \boldsymbol{Q_i(s'_i, b)})$$

$$\pi_i(s_i) \leftarrow \arg\max_b \begin{cases} Q_i(s_i, b) & \text{si } b \in A_i - SocAct_i \\ Q_{soc_i}(s_i, b) & \text{si } b \in SocAct_i \end{cases}$$

- In **IncomeExchange2**, the agents learn simultaneously the utilities of individual and social actions. But the utilities of social **AND** individual actions are assessed according to the social rewards and utilities of future individual **AND** social actions. Agents can now consider executing sequentially several social actions and they estimate the reciprocal influence of social and individual actions.

$$Q_{soc_i}(s_i, a_i) \leftarrow (1-\alpha)Q_{soc_i}(s_i, a_i) + \alpha(r_i + \boldsymbol{r_{s,i}} + \gamma \max_b \boldsymbol{Q_{soc_i}(s'_i, b)})$$

$$\pi_i(s_i) \leftarrow \arg\max_b Q_{soc_i}(s_i, b)$$

All these algorithms are decentralized: the agents learn simultaneously their *Q-values* without needing a global coordination mechanism. These algorithms only require exchanges of local information during the execution of social actions to locally update the utilities of social actions.

Once this learning is done, each agent uses its *Q-values* to decide the best individual or social action. Even if the *Q-Values* do not correspond to the exact expected cumulated rewards as in single-agent Q-learning, we hope they can nevertheless give a good policy.

To asses that, the next section presents the experimental studies on which we carried out our algorithms.

4. EXPERIMENTAL EVALUATION

4.1 Cage of Meal Problem

Two agents (Fig 3) are locked up in a room and must satisfy their needs. Their levels of hunger and thirst increase at each step but the agents can choose to consume a resource in order to reduce the pain related to their state of dissatisfaction. The resources are meals and drinks and the agents must spend energy to access them. Agent 1 spends more energy for fetching a drink than a meal, whereas Agent 2 has complementary skills. The two agents have a common zone where they could store and get resources.

The goal of this problem is to build the local policies of agents, so that they can have both meals and drinks without losing too much energy.

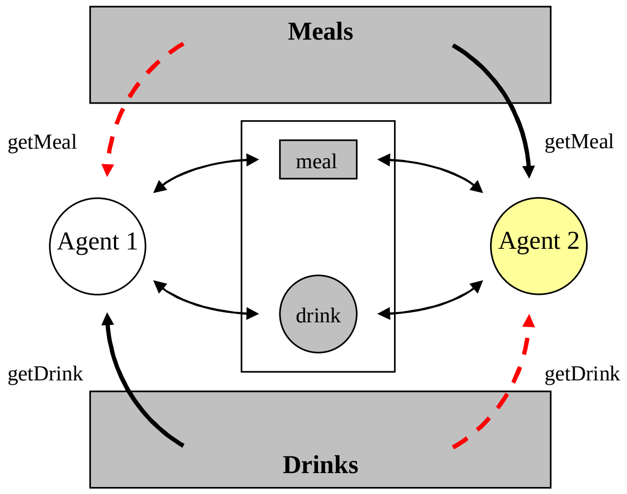


Figure 3. Cage of meal problem

This problem defines a cooperation problem in which each agent has a partial control on the system.

State: The state of each agent is characterized by 4 variables which are: *Hunger* (an integer in [0;3]*)*, *Thirst* (an integer in [0;3]), *HasMeal* (a Boolean true/false) and *HasDrink* (a Boolean true/false).

The global state *S* of the corresponding Dec-POMDP is constituted by the state of Agent 1, the state of Agent 2 and the presence or absence of meals/drinks in the common zone.

Observation: Each agent can only have access to its variables and the presence/absence of meals and drinks in the common zone. Thus, the evolution of the world seen by an agent is neither stationary nor deterministic: meals can appear on the common zone due to the action of the other agent.

Action and transition: Each agent can choose one of the following actions: *getMeal*, *getDrink*, *eat*, *drink*, *putDrink*, *takeDrink*, *putMeal*, *getMeal*. When an agent drinks (eats) and has the corresponding resource, its thirst (hunger) is reset to 0. *getMeal* and *getDrink* are always successful: when an agent executes one of these actions, it receives the resource but looses energy. *takeDrink* and *putDrink* consist in taking or putting a drink on the common zone and are always successful if the common zone contains the corresponding resource. Finally, at each time step, the hunger and the thirst of the agents increase.

Social actions: Two social actions have been identified: *exchangeDrink={putDrink, takeDrink}* and *exchangeMeal={putMeal, takeMeal}*. When they are chosen, the agent which executes the social action drops the resource on the common zone while the other agent takes it.

Reward: The reward function of the system is the sum of the individual rewards of the two agents. The reward of an agent is the sum of the rewards associated to its hunger, to its thirst and to its actions (energy loss for *getDrink*/*getMeal*). The table 1 presents the reward function. *X* and *Y* correspond to global parameters, each value of *(X,Y)* defines a new problem.

Table 1. Rewards function

Agents	Rewards of hunger				Rewards of thirst				getDrink	getMeal
Hunger/Thirst	0	1	2	3	0	1	2	3		
Agent 1	0	-1	-2	X	0	-1	-2	X	Y	-1
Agent 2	0	-1	-2	X	0	-1	-2	X	-1	Y

4.2 Comparative Analysis

We first used a centralized MMDP Value Iteration algorithm proposed by (Boutilier 1999) to build the optimal policies for different configurations *(X,Y)*. This study enabled us to fix two interesting configurations of the reward function to assess the quality of our solutions:

1. **For X=-50,Y=-50, the problem corresponds to an egoistic situation** where the optimal policy consists in not exchanging any resource
2. **For X=-50, Y=-5**, **the problem corresponds to an altruistic situation** where the optimal policy consists in exchanging both resources and the optimal individual policies must be synchronized.

For each configuration, our algorithms have been tested. Our results were compared to the results obtained with other decentralized approaches: *Q-learning with global reward*, *Q-learning with local rewards* (Sen et al 1994), *Eco-evolution* (Chades et al 2002) and *Distributed value function* (Schneider et al 1999).

More precisely, each learning algorithm has been executed during several episodes (1Ep. = 1000 Iterations) with a sigmoid-ε-greedy policy. After that, we exploited the built policies for 1000 iterations. The execution of these policies produces cyclic collective behaviour. Tables 2 and 3 present the results obtained for 50 experiments. They present means of sizes, sums of rewards and the number of exchanges for the obtained cycle. They also present the mean of the sum of the rewards for the 1000 iterations.

4.2.1 Egoistic Configuration

For the egoistic configuration, the optimal policy does not need any exchange and the learning task is quite easy because there is no useful interaction among agents. The results of the various approaches are presented in Table 2 and showed that all the approaches managed to build egoistic policies without any exchange.

Table 2. Comparison for egoistic configuration, the first line presents the optimal policy found by Value iteration. Columns respectively represent Algorithm, learning duration, number of exchange, size, sum of rewards and its mean by cycle, discount factor, mean of sum of rewards for 1000 iterations and policy found.

Algorithms	Nb.it	Characteristics of cycle				$\overline{\sum_{t=0}^{1000} \gamma^t R_t}$		Policy
		Nb.exch	Size	R_{cycle}	$\overline{R_{cycle}}$	γ		
VI (Boutilier 99)	175	0	6	-422	-70.33	0.99	-6887.46	*Egoistic*
Q-learning (G)	600Ep.	0	16.8	-1736.3	-102.5	0.99	-10174.29	-
Q-learning (Sen et al 94)	600Ep.	0	6.24	-440.45	-70.44	0.99	-6898.83	*Egoistic*
Eco-evol. (Chades et al 02)	700	0	6	-422	-70.33	0.99	-6887.49	*Egoistic*
DVF (Schneider et al 99)	600Ep.	0	5.63	-602.37	-106.89	0.99	-10430.77	-
Smpl-IncEx	400Ep.	0	6	-422.0	-70.33	0.99	-6887.47	*Egoistic*
IncEx-1	450Ep.	0	6	-422.0	-70.33	0.99	-6887.47	*Egoistic*
IncEx-2	500Ep.	0	6	-422.0	-70.33	0.99	-6887.47	*Egoistic*

Nevertheless, experiments showed first that even if the task is easy, a direct **decentralized Q-learning approach with a global reward** failed to produce an efficient behaviour. Indeed, this algorithm faced the problem of *credit assignment* and cannot correctly assign the earned rewards to the correct actions.

Also, **Distributed value function** cannot correctly approximate the optimal policy. This is because the method uses the satisfaction of the neighbours to update the agents' Q-value and is not precise enough if the dynamics of communicated Q-values is too fluctuating.

Finally, the results obtained by **our algorithms**, approximate very well the optimal policy. These results showed that the heuristics proposed do not lead to the execution of a social action if it is useless.

4.2.2 Altruistic Configuration

For the altruistic configuration, results showed that none of the existing algorithms managed to build an altruistic policy. (see Table 3).

Eco-evolution cannot lead to the optimal policy because, the two agents need to simultaneously change their behaviour in order to obtain the best policy and eco-evolution requires a policy to be fixed.

For **direct Q-learning with global reward**, agents are confronted to credit assignment issue like previously. When agents have access to **local rewards**, they can solve the credit assignment issue but do not take the satisfaction of the other agent into account and their policy can only converge to selfish ones.

The main goal of **Distributed Value function** was to adapt the local reward Q-learning approach by making each agent consider the satisfaction of the other agent. Each agent maximizes a balanced sum of its *Q-values* and the *Q-values* of the other agents. However, this approach gives bad results here, because as Schneider pointed out, it is implicitly based on the mean of the *Q-values* of the other agent, but this information hides the real state of the other agent and is too fluctuant to be informative enough.

Table 3. Comparison for altruistic configuration, the first line presents the optimal policy found by Value iteration. Columns respectively represent Algorithm, learning duration, number of exchange, size, sum of rewards and it mean by cycle, discount factor, mean of sum of rewards for 1000 it and policy found.

Algorithms	Nb.it	Characteristics of cycle				$\overline{\sum_{t=0}^{1000} \gamma^t R_t}$		Policy
		Nb.exch	Size	R_{cycle}	$\overline{R_{cycle}}$	γ		
VI (Boutilier 99)	175	2	10	-112	-11.2	0.99	-1105.74	Altruistic
Q-learning (G)	600Ep.	0	6.55	-87.45	-13.17	0.99	-1299.32	-
Q-learning (Sen et al 94)	600Ep.	0	2.2	-26.53	-12.04	0.99	-1179.87	Egoistic*
Eco-evol. (Chades et al 02)	700	0	2	-24.0	-12.0	0.99	-1176.15	Egoistic*
DVF (Schneider et al 99)	600Ep.	-	5.65	-74.51	-12.81	0.99	-1265.42	Egoistic*
Smpl-IncEx	400Ep.	2	8	-94.0	-11.75	0.99	-1154.73	Altruistic
IncEx-1	450Ep.	1	4	-47.0	-11.75	0.99	-1154.59	Altruistic
IncEx-2	500Ep.	1	4.31	-51.16	-11.87	0.99	-1166.38	Altruistic

On the contrary, in the same context, **our algorithms** offer a better approximation of the optimal policy. For **SimpleIncomeExchange**, there are two exchanges of resources in one behavioural cycle. This is because the agents implicitly exchange their observations during the exchange of gain. Thus, they have more information on the state of the system and their decision is consequently more rational.

For **IncomeExchange 1** and **IncomeExchange 2,** there is one exchange in a behavioural cycle. Indeed, agents are not sure of the behaviour and the state of the other agent when they are learning. So, they preferred to keep the resource rather than to give it to the other agent whose behaviour and rewards are uncertain. Both algorithms managed nevertheless to make the agents cooperate by sharing a resource (on the two available).

4.3 Cage of Meal with Three Agents

4.3.1 Presentation of the Cage of Meals Problem with a Mediator

Now that we have shown that our algorithms can lead to cooperative policies, they have been tested on a variation of the cage of meals problem in order to highlight their various efficiencies. From now on, the task to be solved involves three agents and several possible interactions.

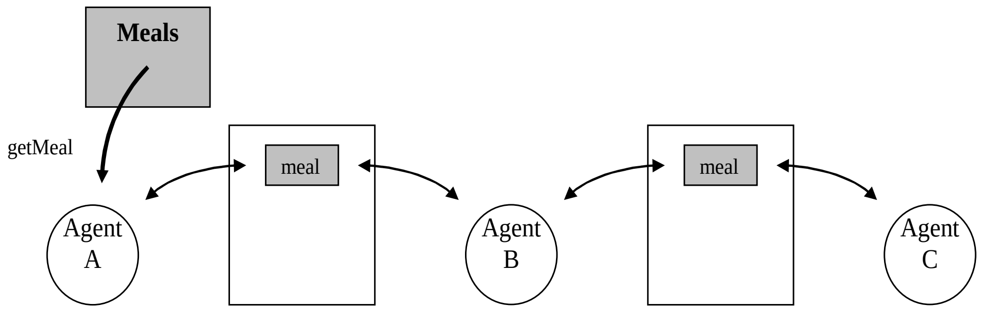


Figure 4. The cage of meals problem with a mediator (agent B)

In this collective task (Fig. 4), the two agents at the extremities (agents A and C) suffer both from an increasing hunger. The agent on the left (agent A) has a direct access to meals at low cost, whereas the agent on the right (agent C) cannot access directly to meals. Between them, a third agent (agent B), the mediator, does not feel any hunger but participates to the task since it can help agent C to access food.

Agents A and B can exchange food in the same manner as in the classical Cage of meal problem through a common zone and two social actions: an exchange from agent A to B *ExchangeABFood = {PutFoodRight,GetFoodLeft}* and an exchange from agent B to agent A. Agent B and C can also exchange food through another zone and two other social actions. Thus, the only way for agent C to access food requires a sequence of two social actions: agent A has first to exchange food with agent B and in a second time, agent B has to exchange this food with agent C.

Hence, the objective of this problem is to investigate how the proposed algorithms behave when the task must be distributed among more agents than the two agents involved in only one interaction.

More formally, the problem can be described as follows:

State: the state of the system is characterized by the *Hunger* variables of agent A and agent C (an integer in [0;3]*)* and the *HasMeal* variables (a Boolean true/false) of agents A, B and C.

Observation: each agent can observe its own state and the presence/absence of meals in the closest common zone (agent B can observe both).

Action: each agent can choose an action between *getMeal* and *eat* but the agent A is the only agent to find food when its action is *getMeal*.

Social action: four social actions are possible depending on the direction of the exchange and the agents implied: *ExchangeABFood*, *ExchangeBAFood* , *ExchangeBCFood and ExchangeCBFood*. Of course, each agent can only perform one of the social actions in which it is involved.

Rewards: the reward is still defined as the sum of individual rewards. These rewards depend on the hunger of agent A and agent C and the fact that agent A had made a *getMeal* action (as illustrated in table 4).

Table 4. Rewards for the cage of meals with mediator problem.
X means that the action has no significant result and no reward.

Agents	Rewards of hunger				getMeal
Hunger/Thirst	0	1	2	3	
Agent A	0	-1	-2	-5	-1
Agent B	0	0	0	0	X
Agent C	0	-1	-2	-5	X

4.3.2 Analysis of the Results of our Algorithms

As in section 4.2, we have first computed the optimal policy by a centralized MMDP Value Iteration algorithm (Boutilier 1999) to determine the optimal collective policy. This policy corresponds to a cooperative behaviour, where

- agent A fetches food and sometimes gives it to agent B or sometimes consumes it;
- agent B gets food from A whenever possible and gives it to C;
- agent C gets food from B when it is possible and consumes it.

In this behaviour, both agents are eating and the food is equally distributed among agent A and agent C. When the hunger of agent A equals to 1, it gives its food to B whereas when this hunger equals to 3, agent A consumes its food. Thus, agent C can eat and the hunger of both agents remain low.

The results obtained with our three algorithms are summed up in Table 5. For each algorithm, a first individual learning phase is simulated with an individual value iteration (it converges after 100 iterations). Then 300 episodes of 1000 steps are performed to learn social Q-values (even if they stabilize faster than that). During these steps, the alpha parameter is set constant to 0.7.

Table 5. Results obtained on the cage of meals with mediator problem.

Algorithms	Nb.it	Characteristics of cycle				$\overline{\sum_{t=0}^{1000} \gamma^t R_t}$		Policy
		Nb.exch	Size	R_{cycle}	$\overline{R_{cycle}}$	γ		
VI (Boutilier 99)	350It	2	4	-18	-4.5	0.99	-446.21	Altruistic
Smpl-IncEx	350It	0	2	-12	-6	0.99	-588.07	Egoistic
IncEx-1	350It+300Ep	0	2	-12	-6	0.99	-588.07	Egoistic
IncEx-2	350It+300Ep	2	4	-18	-4.5	0.99	-446.21	Altruistic

For the **SimpleIncomeExchange** algorithm, the behaviour of agent A is selfish: it gets food but keeps it for him. The hunger of agent C increases to reach a maximum of 3 and reduces the global reward of the system. The algorithm converged to this behaviour because the utilities of exchanges are assessed online only on the basis of previously learnt individual *Q-values*. However, in individual Q-learning, without considering any interaction, agent B cannot earn anything from having a meal and all its individual Q-values correspond to 0 whether it has a meal or not. Moreover, agent A has always a selfish interest in keeping its food to reduce its hunger. Thus, the utility of exchanging a food from A to B is always negative since it lead to the loss of a meal for the agent A without any advantage for B. Due to that, the needs of agent C are hidden by the presence of agent B.

For **IncomeExchange 1** algorithm, the behaviour of agent A is also selfish and the optimal behaviour is not found. This case is similar to the previous one. The *Q-values* of social actions are learnt and not only assessed but they are only based on action *Q-values*. Agent A assesses the utility of an exchange only by considering the direct benefit of the food exchange with agent B. The fact that agent B can also exchange with C is taken into account into agent B's social *Q-values* but not into its action *Q-values*. Thus, in this context, the exchange between C and B cannot be taken into account to assess the utility of exchanging food between A and B.

At last, **IncomeExchange 2** algorithm leads to the optimal policy. In this case, the update of social Q-values depends on the action and on the social actions *Q-values*. This makes the agents consider the next social actions that can occur in the system while assessing any action. Moreover, the exchange of social reward propagates individual rewards among interacting agents. Thus, agent A considers that agent B can make a future exchange with agent C and that it can have an interest in the future.

The next section details more deeply how this learning is made.

4.3.3 Analysis of IncomeExchange2 Algorithm Results

Even if all the learning processes of **IncomeExchange2 algorithm** are made simultaneously, these learnings can be understood as follows:

- In a first time, agent C learns that it can earn reward by consuming a meal whenever it manages to access to one due to exploration. By experimenting *eat* action, its *Q-values* for the states in which it possesses a meal increase.
- Once it is done, agent B can learn that exchanging a meal with C has an interest. Indeed, the utility of these exchanges involve the *Q-values* of agent C corresponding to states where it has a meal. The utility of an exchange with C is hence positive. Thus, exchanges between agent B and C will occur more regularly. In return, agent C will give to agent B a part of this social utility through a social reward. This is translated as an increase of the agent B's Q-values of the states corresponding to the possession of a meal. From now on, Agent B will try to reach one of those states.
- Finally, the utility of exchanges from A to B increases since the Q-values of B have previously increased. Agent A will learn that exchanging with B can have an interest when the utility of this exchange is higher than the utility of its *eat* action.

It must be noted that the **IncomeExchange2 algorithm** manages to learn the correct directions of exchanges and their transitivity. Moreover, the exchanges with B are synchronised since the utility of an exchange is directly linked with the individual rewards defining the problem. Thus, when agent A really needs food, the competition between individual action and social action leads him to keep it, whereas when agent A is not hungry, the way the social reward is distributed makes agent A to exchange with agent B. This algorithm constitutes a response to the credit assignment problem by correctly distributing the individual rewards in the global system through the possible interactions.

Thus, IncomeExchange2 algorithm manages in this small task to adequately take into account the several interactions that can occur in the system. The agents by considering their social Q-values consider the other agents with which they can interact even if they do not know precisely their state and if they do not directly interact with them.

5. DISCUSSION

Our learning algorithms are at their firsts steps and face many limitations: they were only applied on some specific tasks with a few different configurations. Moreover, they are constituted by few heuristics and their applications are limited due to the absence of proof convergence. Finally, the social actions need to be predetermined by the conceiver as couples of existing individual actions before applying any algorithm.

Nevertheless, despite these current limitations, the approach seems promising. First of all, we must stress the problem we wanted to tackle is very difficult. It consists to make agents learn in a decentralized way their behaviour by considering the other agent. The results showed this algorithm can distinguish two global situations characterized by a global reward function. The agents can then adapt themselves to these different situations whereas they have only access to a local perception of the problem through local rewards.

Secondly, this algorithm proposes a new way to share a task among the agents. This sharing is done with the help of social rewards which are automatically computed and integrated by the agents during execution. This reward sharing is only based on the problem to solve and seem to be quite generic. It can constitute a first response to credit assignment problem.

Ultimately, on a very simple task, this algorithm seems to perform better than the existing approaches that have been tested. On the specific cage of meal problem, it manages to take interactions among agents into account better than distributed value function by being more precise in the Q-values exchanged, better than Eco-evolution by authorizing simultaneous behaviours learning, and better than decentralized Q-learning by solving credit assignment issue. In particular, the IncomeExchange2 algorithm gave good result when it was applied in a more complex problem where several agents must interact through several interactions.

Thus, we speculate this approach can be a promising research direction even if, up to now, it is only based on experimental works. It seems that this approach can be a first step to intelligently distribute individual rewards and to build adaptive agents that can consider the other ones.

6. CONCLUSION

In this article, we introduced a formalism to make interactions explicit in Dec-POMDPs by using social actions and we proposed three algorithms based on heuristics to take into account the other agents while respecting locality constraints. The results have shown that each agent can automatically adapt itself to the global context of the system in a specific task.

One of the many remaining questions is how to correctly share the gain of a social action among agents. We plan to investigate more deeply this question by addressing simpler problems and analyzing how planning can be made on the same basis of what we have proposed. Another interesting approach would be to take inspiration from works done by (Guestrin et al) to see if our approach can take advantage of MDP factorizing to determine what are the correct social actions to consider.

REFERENCES

Bernstein, D. and Givan, R. and Immerman, N. and Zilberstein, S., 2002, The Complexity of Decentralized Control of Markov Decision Processes, *Mathematics of Operations Research*, Vol 27, pp. 819-840

Becker, R. and Zilberstein, S. and Lesser, V. and Goldman, C., 2003, Transition-independent decentralized markov decision processes, *AAMAS '03: Proc of the int joint conf on Autonomous agents and multiagent systems*, pp. 41-48

Boutilier, C., 1999, Sequential optimality and coordination in multiagent systems, *IJCAI99 Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence,* San Fransisco, USA, pp 478-485

Busoniu, L. and Babuska, R. and De Schutter, B., 2008, A Comprehensive Survey of Multiagent Reinforcement Learning. *IEEE Transactions on Systems, Man, and Cybernetics,* Part C 38(2), pp. 156-172

Buffet, O. and Dutech, A. and Charpillet, F., 2007, Shaping multi-agent systems with gradient reinforcement learning, *autonomous Agents Multi Agent System*, vol 15, N.2, pp 197-220

Chades, I. and Scherrer, B. and Charpillet, F., 2002, A heuristic approach for solving decentralized-POMDP: assessment on the pursuit problem, *Proceedings of the 2002 ACM symposium on Applied computing,* Madrid, Spain, pp 57-62.

Guestrin, C. and Lagoudakis, M. and Parr, R., 2002, Coordinated Reinforcement Learning, *ICML-2002 The Nineteenth International Conference on Machine Learning,* Sydney, Australia, pp. 227-234

Hardin, G., 1968, The tragedy of the commons, *Sciences,* pp 1243-1248.

Schneider, J. and Wong, W. and Moore, A. and Riedmiller, M., 1999, Distributed Value Functions, *Proceedings of the 16th International Conference on Machine Learning*, pp. 371-378

Sen, S. and Sekaran, M. and Hale, J., 1994, Learning to Coordinate without Sharing Information, *Proceedings of the Twelfth National Conference on Artificial Intelligence*, Seattle, WA, pp. 426-431

Szer, D. and Charpillet, F., 2006, Point-based Dynamic Programming for DEC-POMDPs, *21st National Conference on Artificial Intelligence AAAI'2006*, Boston, USA, pp 1233-1238

Parunak V., 1994, Application of distributed intelligence in industry, *Foundations of distributed artificial intelligence*, chap 4

Watkins, C. J. C. H. and Dayan, P., 1992, Technical note: Q-learning, *Machine learning*, vol 8, pp 279-292

Weiss G., 1996, Adaptation and learning in multi-agent systems: Some remarks and a bibliography, *Lecture Notes in Computer Science,* Volume 1042/1996, pp. 1-21.