Learning by Exchanging Advice

Eugénio Oliveira

Luís Nunes

Abstract

The emergence of Multiagent systems brought new challenges to the field of Machine Learning, as it did to many others. One of the main challenges is to take advantage of the information available when several agents, possibly using different learning techniques, are dealing with similar problems, either in the same location (i.e. acting as a team) or in different ones. This work aims at studying the possible advantages and pitfalls of exchanging information during the learning process, leading to better adaptation. We will discuss the subject of when, how and to whom ask for advice, and present the results obtained in two experimental scenarios: the Pursuit (Predator-Prey) Domain and a Traffic Control simulation. Results show that exchange of information can improve the average performance of learning agents enabling them to escape from local maxima in some cases, although it may reduce the exploration of the space, preventing successful agents from finding better local maxima of the quality function.

1 Introduction

Decentralization and distribution of processes became an issue in *Artificial Intelligence*, as well as in several other areas of Computer Science, in the past decades. In response to the new challenges a new type of software entities was created and later labelled as

agents. These new entities are usually defined as being more autonomous, distributed and intelligent than previous software tools. Some problems require that agents adapt to new circumstances or behave "*intelligently*". Intelligence, as we perceive it, is strongly related to the capability of learning from previous experience and using stored knowledge to improve future behavior. Intelligent (or adaptive) behavior is becoming a competitive factor in today's software. One of the main challenges is to expand the Machine Learning (ML) paradigms from, the old single-agent perspective, to this new world. Currently, software agents inhabit dynamic environments. Often, they must provide answers even when they have only a partial and noisy view of the problem. The extension of learning to these new environments must overcome the difficulties of this new paradigm, but should also take advantage of its benefits. The fact that a multitude of agents populates the software environments, and in some cases they are learning to solve similar problems, leads to the current research issue: *"(How) can agents benefit from the exchange of information during the learning process?"*

In the following sections we will present and discuss a set of techniques for selection, exchange and incorporation of information from multiple sources. These techniques may provide a way to help a learning agent achieve its goal more efficiently than if it was learning only from the information generated by the environment. The main focus will be on *advice-exchange*, a technique introduced in Nunes and Oliveira (2002b) to exchange information within heterogeneous groups of learning agents that are solving similar problems. The heterogeneity constraint is one to which very little attention has been given until now. Results obtained in this research direction may point the way to more powerful learning paradigms.

The next section contains a brief review of related work. In section 3, the techniques that compose *advice-exchange* are explained and in the following section (4), the experimental setup is described. Section 5 presents the results and its discussion and finally, in section 6,

we have the conclusions and a brief word on the future work.

2 Communicating to improve learning: Historical notes and review

The work presented below touches several points such as: cooperation between learning agents; mixed use of different learning paradigms at several levels; learning trust relationships between agents. Several of these issues have been approached in the past by other authors. In the following sections the reader can find a summary of the main contributions in each of these subjects.

2.1 Early work on exchange of information during learning

The work on information exchange between QL-agents (agents that use *Q-Learning* (Watkins and Dayan, 1992) as a basis for their learning skills), started in the early nineties. (Whitehead, 1991) created a cooperative learning architecture labelled *Learning By Watching*. In this architecture the agent learns by watching its peers' behavior (which is equivalent to sharing series of state, action, quality triplets). The work presented in (Clouse and Utgoff, 1991) is reviewed and expanded in Clouse's Ph.D. thesis (Clouse, 1997). This important contribution reports the results of a strategy labeled *Ask for Help*, in which QL-agents learn by asking other agents of the same type for suggestions and perform the suggested actions. The work presented by Lin (1992) uses an expert trainer to teach lessons to a QL-agent that is starting its own training. Tan (1993) reports the results of sharing several types of information in the predator-prey problem. In these experiments QL-agents shared policies (internal solution parameters), episodes (series of state, action, quality triplets), and sensation (observed states). All these experiments showed improvements in the average performance of agents that exchange in-

formation.

2.2 Recent related work

The work on exchange of information between QL-agents continued in several fronts. The term *joint learning* was used by Berenji and Vengerov (2000) when referring to agents that update concurrently the same quality values. These authors presented a study of a "fuzzy" variant of QL-agents that cooperate during learning by updating a common table of Q-values. Several researchers studied applications of *Reinforcement Learning* (RL) (Sutton and Barto, 1987) variants to stochastic-games. The agents used in these domains are often referred to as *Joint-Action Learners* (JAL). JAL are QL-agents that learn, each on its own, the quality of joint actions. A joint action is composed of its own action plus the actions chosen by all the peers at a given time. This approach presupposes full observability of the actions done by all agents. The first references to this concept are in the work of Littman (1994) where agents of this type are used to solve (i.e. attain optimal equilibrium in) zero-sum games. This approach was labeled *minimax-Q*. Litmann (2001) presents a summary and comparison of the convergence properties of several types of JAL.

The research on trust relationships between agents was mainly developed by Sen's research group (Sen, 1996; Biswas *et al.*, 2000; Banerjee *et al.*, 2000). These, as well as other related papers, focus on several aspects of learning to trust/distrust other agents.

Many researchers have focused on the use of human advice, or preprogrammed teachers, to help QL-agents. These approaches range from using high-level languages to encode the advice, as in (Maclin and Shavlik, 1994), to direct observation of a human solving a problem and replication of this behavior (Nicolescu and Matarić, 2001).

One of the most interesting works in the subject is (Price, 2003), in

which QL-agents learn by *implicit imitation* of pre-trained expert-agents. This work has some very interesting characteristics: the student agent has no knowledge of the actions done by the expert, it can only observe its state transitions; there is no explicit communication between the expert and the student; the goals and actions of both agents can be different.

3 Advice Exchange

The problems considered in our research have several characteristics, they are partially-observable, non-static and distributed. A problem is called *partially-observable* if any given agent can observe only a part of the state, or have only a summarized view of the variables that may be important to the evolution of the environment's state. By *non-static* it is meant that, from an agent's point of view, the same action in the same state can have different outcomes at different times. This is a consequence of the environment's stochastic nature and the interaction between different agents. Distributedness is considered at two different levels: first, each problem is solved by a team of agents; second, several teams are working in similar problems in different locations. Members of the same team can interact either by communicating, or by the consequences of their actions in a local environment (we will refer to these as *partners*). Members of different teams only interact through explicit communication. The agent's objective is to maximize the average reward obtained in a given period. These periods will be called *epochs*. In this work we try to make as few assumptions as possible regarding the learning algorithms used by the agents. This will allow the use of heterogeneous groups, in which each team of agents uses different learning algorithms.

3.1 Exchanging information during learning

As mentioned above, the main question addressed here is: "(How) can communication between agents improve learning performance?" This question can be divided into several others:

- What information to exchange?
- How to integrate this information with the usual learning process?
- When should an agent request/accept information?
- How should an agent decide where to get/send information?

In the following subsections we will address each of these questions and propose an approach to these problems.

3.1.1 What type of information?

The most obvious way to exchange the learned knowledge is to send a complete description of the solution to another agent (i.e. a complete set of parameters for its learning structure), but this has two major drawbacks: first, all agents would either have to be of the same type, or be able to use different learning structures, thus heterogeneity would be lost; second, the solution was constructed to fit the dynamics of the local situation of the advisor, so, even the smallest differences in the dynamics of the problem could render the solution useless and destroy what was learned by the advisee.

Advice-exchange Nunes and Oliveira (2002b) uses the information available in the environment, such as: states, actions and rewards. Using only these types of information, and statistics based on them, the agents can communicate: states that they experience more often; the quality of the actions they performed at a given state; the action they would choose for a state; sequences of actions that produced good results; etc. Different combinations of these types of information can provide a description of the environment, of the agents' policies and

the characteristics of the quality function that agents are trying to maximize. The most common type of communication is for an agent to send another its current state and receive as a reply the action the other agent advises for that particular situation. The answer to a query is computed using the parameters that have obtained a good score in a previous epoch.

3.1.2 How to integrate this information with the usual learning process?

The way to use exchanged data depends highly on the type information that is exchanged. It is difficult to envision a process that would be applicable to any type of information, learning algorithm and problem. Our approach simply points a general way and exemplifies its application. The extension to other learning algorithms would necessarily require adaptations.

The integration of advice with the knowledge gathered from acting in the environment is done in a different way for each type of agent. Nevertheless, it uses either imitation or a form of *supervised learning*. Most learning structures are able to integrate information given in this form and learning from supervision information is typically faster than from reinforcement.

When the advised action is different from the one the agent would have chosen it can either imitate the incoming action and learn from the result, or integrate the knowledge with its own hypothesis and then select an action. When an agent is rewarded this information can also be used in different ways, depending on how the action selection was performed. In some cases, just one presentation of the advice is not sufficient to achieve a reasonable effect and it is necessary to replay it. *Advice replay* consists in storing and replaying advice at specific times to improve the effectiveness of the procedure.

A more detailed explanation of how integration and imitation are merged with each of the learning algorithms can be found in section

4.3.

3.1.3 When should an agent request/accept information?

The answer to this question may depend on several factors: a) the comparison between the advisor and advisee's performances; b) the comparison of the experience each has in dealing with the current situation; c) how well defined is the response. The problem with a) and b) is that the environment is noisy and partially-observable. This makes it difficult to have a clear evaluation of performance and state. Consider, for instance, the case of controlling a traffic light at a crossing. The traffic volume is different depending on the time of day. At times there are no cars in the incoming lanes for a period of time, and on other occasions the traffic volume is over the limit of saturation. How can we compare the agent's policy in those two periods? What if all surrounding agents are, by a fault in their own policy, diverting all the traffic to one intersection. The agent at that intersection cannot cope with so much traffic no matter what policy it chooses. Should it be penalized by that?

Some of these matters can be dealt with by a careful choice of the state representation, quality-function and times of evaluation. Nevertheless, it is necessary to introduce mechanisms that provide several perspectives of the performance to overcome the noise generated by the dynamics of the environment. Using different statistics of performance, measured over different periods of time, compensates for some of these effects, specially in the above mentioned case a). Each agent calculates the following measures of performance:

- Average reward per epoch (R_n);
- Infinite discounted reward;
- Recent best reward;
- Short, mid and long-term average reward;

- Average reward evolution;
- Self-confidence;

The infinite discounted reward (idr_n) in epoch n is defined as:

$$idr_n = \alpha\, idr_{n-1} + (1-\alpha)R_n, \tag{1}$$

where $\alpha \in [0, 1[$ determines the balance between current and previous experience. Recent best reward (b_n) is:

$$b_n = \max(\beta b_{n-1}, R_n), \tag{2}$$

where $\beta \in [0.9, 1[$ is a decay parameter. The short, mid and long-term average rewards are based in R_n and calculated for fixed periods. Their calculation is, as follows:

$$RT_{n0+p} = \sum_{n=n0}^{n0+p} R_n/p, \tag{3}$$

where p represents the period which is different for short, mid and long-term measures. This allows the calculation of the average reward evolution e_n in a given epoch n, as:

$$e_n = \sum_{k=0}^{k=K} (RT_{n-(k+1)p} - RT_{n-kp}), \tag{4}$$

where K is the number of periods to use in the calculation (in these experiments $K = 3$).

An important measure in this case is the comparison of an agent's performance with that of others. This can help the agent to decide whether its own actions are becoming less adequate, or if the problem is in a more difficult state due to some external reason. To help in this respect we have introduced a parameter labelled *self-confidence*. Self-confidence is increased when the performance of a given agent

is good in comparison to that of its peers. Self-confidence is decreased when the performance is poor in comparison to others'. This increase or decrease is done by multiplying the parameter by a prespecified value that is either slightly above, or below 1.0.

In what concerns the comparison of experience it is necessary to know if the advisor has experienced a situation similar to the one it is evaluating. Some algorithms keep records of the states they evaluate while others do not. To normalize this situation all agents keep records of their experiences in certain epochs. The recorded epochs are: current, last and best. The epoch in which an agent saved the parameters it uses to give advice is also recorded. With this knowledge, an advisor can say how similar was the case in which it has used the same option and what was its reward. This will help the advisee to decide whether the situation is similar enough and, after accepting the advice, what is the difference between the announced reward and the one actually received.

To know how well defined an action is corresponds to calculating the (un)certainty in the choice of an action. This uncertainty measure can be represented differently depending on the learning algorithm used. When the agent produces a quality estimate for each of the available actions, if the variance of these qualities is low it may be interpreted as a high uncertainty coefficient, as was done in Clouse (1997).

Another important factor in the decision of when to request/accept information, concerns the agent's learning stage. At different stages, agents need different types and volumes of information. It is known that humans go through several stages when working in a team and learning from others. First they explore solutions individually until someone finds a way to solve the problem. Then others observe, in detail, how the problem was solved and try to mimic the solving behavior. As this phase progresses the "students" gain a growing degree of autonomy. They tend, first, to ask for a complete demonstration and afterwards to ask only for small pieces of information to

clear the doubts concerning a particular aspect of the solution. At this point, if there are several experts with different solutions, they may try to compose a new solution from parts of different approaches. After they have mastered the current solution the members of the group start acting in a more autonomous way and try, each individually, to improve on the previous solution, or find a better one. When one succeeds to find a solution that is proven better than the previous ones, the learning process of its peers may go back to one of the previous phases and some of the members of the group will start, once again, to learn this new solution by imitation of the new expert's behavior.

This process was transposed to the domain of software learning agents, by the introduction of four different learning stages:

1. Exploration: Agents do not exchange information with their peers and learn to choose their actions using only the reward information provided by the environment. This stage ends when the learning performance of the agent stabilizes.

2. Novice: At this stage an agent will ask for advice frequently (in this case for all the actions of an epoch) and keeps the same advisor for long periods. Agents in this stage have a considerably lower performance than the best of their peer's. This phase is ended when the performance stabilizes or when the advisee reaches a level of performance that is very similar to that of its advisor.

3. Stable: When an agent reaches a performance level that is close enough to that of its advisor it will only request advice when the choice of the next action is unclear, or when another peer reports a higher performance in acting from a given state. A Stable agent may choose different advisors for each new situation.

4. Expert: Expert agents are those with the best performances for a particular role. They may request advice when it appears to be

much better than its own options, but most of the time they are engaged in individual exploration.

Agents evaluate the conditions to switch from one stage to the other at the end of each training epoch. An agent can move one level down in each epoch or as many levels up as it can by meeting the appropriate conditions. The conditions concern the comparison of current or best performances with the best that was achieved by other agents, for example: an agent will transit from Novice to a Stable stage if $idr_{i,n} > \lambda\, idr_{k,n}$, where, $idr_{i,n}$, is the infinite discounted reward for agent i at epoch n, k is the index of the agent with best performance and $\lambda \in [0, 1]$ a discount parameter. To change learning stages an agent is also required to have a certain self-confidence level. This condition prevents agents from changing stages due to a result acquired in exceptional conditions and provides a more stable evaluation. Different learning stages may also imply different learning parameterizations. For example, when an agent enters a Novice stage its reinforcement learning rates are lowered and the supervised ones are increased, so that it learns more from advice than from the environment feedback. The definition of learning stages to set learning rates and other parameters is reported in Dorigo and Colombetti (1994). The transitions from the Exploration and Novice stages also include another type of evaluation. An agent will leave these stages when the, short, mid and long-term average reward evolutions are lower than a certain threshold. This will guaranty that the learning process is stabilized when the agent is promoted.

In the experiments reported below agents have used two different ways to decide whether or not to get advice. The first, used in the Predator Prey experiment, is based on the uncertainty of the selected action. The second is based on the comparison of the reward the advisor announces for a given state with the one expected by the advisee. In the first case an agent would request advice if the best n options were all within a given vicinity of the best. In QL-agents the

estimated quality of each option is directly available when selecting an action, so this calculation presents no problems. In agents that use other algorithms the output is interpreted as a classification since the chosen action is the one with highest output. An action was considered to be unclear when the best n options were within a given vicinity of the best. In the second case an agent would ask for advice if:

$$tolerance * r_j(s) > r_i(s), \tag{5}$$

where $r_j(s)$ is the advisor's announced reward for state s, and $r_i(s)$ is the advisee's expected reward. The tolerance parameter used in the experiments was 1.0 for Stable agents and 0.9 for Experts.

3.1.4 Where to get information?

After having decided that it needs advice, the agent must choose the best source of information. In previous experiments, reported in (Nunes and Oliveira, 2002a, 2003), it was clear that the best advisor was not always the agent with best performance. This can happen for several reasons, but the most common of these is that advisor's and advisee's local environments have different dynamics and respond differently to the same actions in the same state. In this case an agent needs to learn which of its partners it can trust, which it should not and also which has the most successful solution for a given state or class of states. The combination of these types of information can help in the decision of which advisor to choose.

The decision of which peer an agent i should request advice to can take two forms depending on whether the agent needs advice for a full epoch (Novice agents) or for a particular situation. In the first case, the agent will select an agent k among its peers, where:

$$k = argmax_j(trust_{ij}\ ar_j\ rl_i), \tag{6}$$

where $trust_{ij}$ represents the level of trust agent i has in peer j, ar_j is the average reward of the advice given by peer j and rl_i is the *role*

discount. $trust_{ij}$ is calculated by:

$$trust_{n,ij} = \alpha trust_{n-1,ij} + (1-\alpha)1/na \sum_{a} r_{j,a}/r_{i,a}, \qquad (7)$$

where $r_{j,a}$ is the reward announced by advisor j for advised action a and $r_{i,a}$ is the actual reward received by the advisee i. na is the number of advised actions in epoch n. This update is done at the end of each epoch. The average reward of advice given by peer j is initialized with the score achieved in the epoch where it as saved the advice parameters and updated with the advisee's reward. Whenever the reward in a given epoch is higher than ar_j the parameters of the current epoch are saved and ar_j is set to the current average reward. The role discount is a constant that has a value of 1.0 for all peers that have the same role and a lower value for advisors with different roles (in these experiments the role discount was 0.7). In previous experiments the role discount was calculated in run-time. To do this each agent used an *unsupervised learning* algorithm to keep track of the classes of states it experienced. The role discount was calculated as a normalized measure of similarity between the class representatives of the requesting agent and all its peers. This proved to have reasonably accurate results and was abandoned only because of its computational cost. By accurate we mean that the role discount for agents with the same role was higher than for others and in most cases close to 1.0.

When an agent needs advice for a particular state (s), the choice of advisor is:

$$k = argmax_j(trust_{ij}\ r_j(s)\ rl_j), \qquad (8)$$

where $r_j(s)$ is the estimated reward agent j expects for state s, based on the data stored in the epoch were advice parameters were saved.

Table 1 shows a summary of the *advice-exchange* procedure. This section stated the main questions concerning exchange of information during learning and sketched the most important components of

advice-exchange along with the reasons why they were integrated in the procedure. Details that are algorithm or problem-dependent will be presented in the following sections.

4 Experiments

In the following sections we will describe two experiments in which we have tested advice-exchange. The first was the pursuit (Predator-Prey) problem, usually considered a toy-problem although some versions are in fact difficult to solve with most learning algorithms. The second was a simulation of an environment in which agents control the traffic in a grid of roads by setting the color and timing of the traffic lights at intersections. This simulation is based on real data, graciously made available by the Lisbon City Hall's Traffic Control Department. The data contains the number of cars that passed in several of the most congested avenues of the city at different days and times during a continuous 15 days period in June 2002. The raw data contains car counts in 5 minutes intervals.

4.1 Predator-Prey

This problem was first introduced in (Benda *et al.*, 1985), although the version presented here has been inspired on several variations presented in (Tan, 1993) and (Haynes *et al.*, 1995). One of the grid-worlds in which experiments were performed (the 10x10 version) is depicted in figure 1.

The problem faced by the predator consists in learning to catch a prey in a grid world. A predator is said to have caught the prey if at the end of a given turn it occupies the same position as the prey. The grid-worlds (arenas) are spherical (although they are usually represented in their planar form) and each contains two agents (predators) and one prey. The state of the environment consists on the position of the prey relative to the agent, i.e. if an agent is at position (2,4)

Table 1. Summary of the *advice-exchange* algorithm.

While not train finished
- Novice : Select best advisor k (eq. 6)
- While not epoch finished
  1. Get state s for evaluation
  2. Should get advice ?
     - Exploration: No, Novice: Yes
     - Stable/Expert: eq. 5? Yes
     - Otherwise, No (go to 3)
     - 2.1 Stable/Expert: Select best advisor (k): (eq. 8)
     - 2.2 Send agent k the current state s and request advice
       - Agent k: Load advice parameters,
       - Agent k: Evaluate state s,
       - Agent k: Produce an advised action (a) for state s
       - Agent k: Return a to advisee.
       - If integrating advice before acting
         - Supervised learning: learn(s, a)
  3. Evaluate state s and produce action (a')
  4. If imitating and advised, use action a, otherwise a'
  5. Receive reward (r)
     - 5.1 If advised
       - If imitating, learn from the result of a':
       - EA: If reward $>$ expected reward
         - backpropagation(s,a')
       - QL: Regular update based on s, a, r
         - and state after action s' performed
     - 5.2 If not advised or $a = a'$ learn from reward
       - (using regular QL or EA updates)
- End epoch loop: Update performance statistics and *trust*.

End train loop

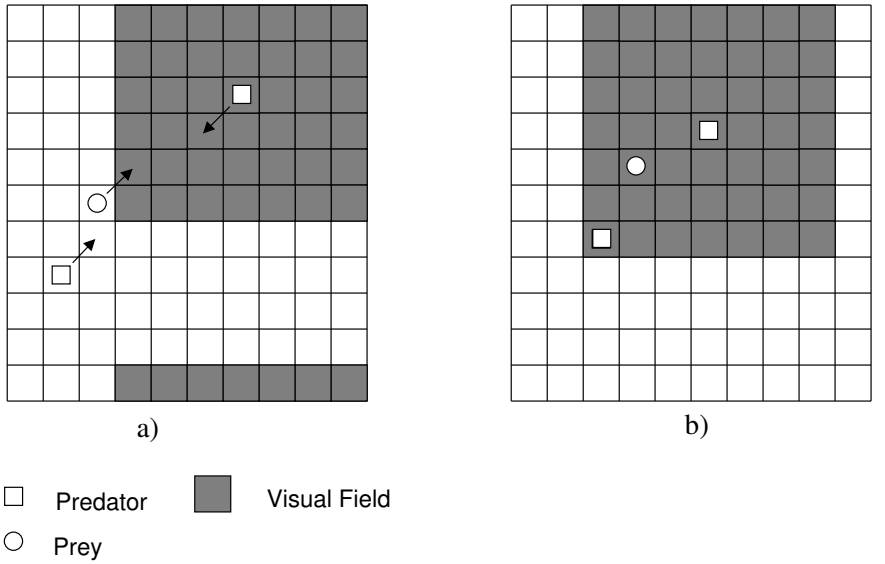

Figure 1. Predator Prey environment at time t (a), and $t + 1$ (b). Arrows represent movements.

and the prey is at (3,6), the state would be (1,2) (see the predator in the bottom-left corner of figure 1 a)). The spherical shape of the grid is taken into account when computing the relative position of the prey as well as when moving. In the experiments reported below an agent's state also contains the position of the prey relative to the partner. Previously published results, using only the prey's position relative to the deciding agent (Nunes and Oliveira, 2003) lead to similar conclusions.

The accuracy of the state representation received by the predator depends on its visual range. The predator perceives the correct position of the prey up to a limit defined by the visual range parameter (3 in this case). If the prey is at a distance greater than the visual range its relative position is disturbed (by Gaussian noise with null average). The further the prey is from the limit of the visual range, the higher is the random noise added to the prey's position.

Each predator has to choose between nine possible actions in each turn, i.e. it chooses either to move in one of the eight possible directions (four orthogonal, and four diagonal), or to remain in its current position. Each predator moves one step in one direction in each turn. Each agent outputs a vector of nine real numbers and the index of the highest output defines the action to be performed.

The prey moves before the predators. To decide in which direction to move the prey detects the presence the closest predator (closest in the sense of being at the shortest Euclidean distance) and moves in the opposite direction, as depicted in figure 1. If there is more than one predator at the same distance one of the predators is picked randomly and the prey moves away from it regardless if it is approaching the other predator. The prey moves only nine out of each ten turns. Any two units can occupy the same cell simultaneously at any time.

In each turn predators are given a reward that as an individual and a global component weighted by $(1 - \beta)$ and β, respectively. The value of β in these experiments was 0.25. The individual component is based on their distance to the prey (d). This reward is equal to 1.0, if the predator has captured the prey, or $0.1(1.0 - d/d_{max})$ otherwise. The constant d_{max} represents the maximum distance between any two positions in the grid world. The global component is the partner's reward. After a successful catch the predator that caught the prey is randomly relocated in the grid-world.

This version of the predator-prey problem does not involve either explicit cooperation or competition between the predators, although cooperative behavior has emerged in some experiments. The same effect was observed using only the position of the prey as state description. Agents use either *Q-Learning* (QL-agents) or *Evolutionary Algorithms* (Holland, 1975; Koza, 1992) (EA-agents) to learn this task. Another type of agents, labelled Heuristic (H-agents), perform a fixed (hand-coded) policy that always tries to reduce the distance to the prey as much as possible. Heuristic agents do not request advice,

but they can reply to advice requests. H-agent's policy would be optimal if there was only one agent in each arena considering that an agent will not keep records of the previous movements of the prey.

The scenarios used for these experiments are the following:

1. *Individual*: Four arenas, each with two predators and one prey. In each arena all predators use the same learning algorithm and they do not exchange advice. Two arenas have EA-agents, the other two have QL-agents.
2. *Social Heterogeneous*: Same as previous, except that agents may request advice to any of its seven peers in the same or other arenas.
3. *Social Heuristic*: Same as previous but with an extra arena where two H-agents are performing the same task and may also be chosen as advisors.
4. *Social Homogeneous*: Same as the *Social Heterogeneous* scenario, except that all agents use the same learning algorithm (either QL or EA).

For each of the above scenarios 11 trials were made (x4, or x8, agents of each type), each with different random seeds. Each trial ran for 9000 epochs and each epoch has 150 turns. For each trial there is a corresponding test which runs for 1000 epochs without learning or exchange of advice. Each agent in the beginning of the test loads the parameters saved during training when it achieved the best score. Partnerships were kept unchanged by this procedure. Agents do not exchange any information except for the one necessary to perform *advice-exchange*.

Two sets of experiments were made differing only in the dimension of the grid. In the first set the grid was 10x10, and in the second 20x20 positions wide.

In this problem advice was incorporated and not imitated. Also, trust was computed as follows: $trust_{ij}$ was initialized to 1.0 and multiplied by a factor slightly higher than 1.0 when advisor j provided advice in epochs where the average reward was higher than the infinite discounted reward. Similarly, it was multiplied by a factor lower than 1.0 when j provided advice and the average reward was not better than the infinite discounted reward. Trust in a given peer was also influenced by other agents. When an agent issued a message of "trust agent X" the receiving agent updated the trust on agent X in the same way as if it had been an advisor for a successful epoch, thus, increasing the trust on this advisor. An agent would issue a message of "trust agent X" to its partner when it had a successful epoch based on the information communicated by a single advisor. In this case "X" would be the advisor's partner. This policy replaced the static role-assignment procedure and allowed agents synchronize their advice requests so that each member of a group would request advice to a different member of the advisor-group. In situations where agents acquire different roles during learning even when they start out in exactly the same circumstances, the run-time definition of roles is important for the success of advice-exchange.

4.2 Traffic Control

The Traffic Control (TC) environment consists of several detached *locations*, each containing one (TC1) or two (TC2) connected crossings. Each location in the TC2 scenario contains 12 lanes over an area of, roughly, 300 x 300m. An environment contains 3 locations, each controlled by a team of agents (QL-agents, EA-agents or H-agents as in the previous experiment). The number of cars generated in each time interval (of 5 minutes) is taken from a set of real data collected in some of the most busy avenues of Lisbon. The TC environment uses real data to calculate, at each time, the number of cars to be inserted. At each 60 seconds interval 1/5 of the cars that must pass through the crossing during the 5 minutes period is introduced

simulating a fixed policy traffic-light outside the scope. The insertion of cars is done at different times in each set of adjacent lanes.

This experiment presents several difficulties. Firstly, to find the appropriate timing to end training epochs. Agents must have enough epochs to learn and simultaneously each epoch must be long enough to have a fair estimate of an agent's performance. An optimal situation would be to test an agent's policy for at least one full day, but computational constraints do not allow this. The compromise solution was one simulated hour for each epoch. To diminish the differences in training epochs the data used is restricted to the traffic between 8 a.m. and 8 p.m in regular working days (i.e. holidays and weekends are discarded). All agents start each epoch with no cars in the system.

Another difficulty is that too many mistakes induce heavy traffic jams. In these circumstances some agents are not able to learn because they have no data (when the cars are jammed in a previous crossing for long periods), and huge traffic jams make the simulation too slow to provide results within a reasonable time-frame. To allow the experiments to run with the necessary speed car movements were simplified. Cars do not exchange lanes, nor turn at crossings and the only limitations to their forward movement are the current speed, the position of the car directly in front and the existence of a closed traffic-light ahead. The movement of cars is calculated as proposed in (Nagel and Shreckenberg, 1992) to simulate realistic drivers. The simulation step is of one second and each epoch consists of one hour of simulated-time, i.e. 3600 turns. Agents are asked for a new decision every 20 seconds. In this problem agents imitate the advised action and learn from the result. Training is done for 10000 epochs and test lasts 500 epochs.

It is important, at this point, to state that we do not propose that these techniques are adequate solutions to a real-life traffic control problem. Even though some lessons may be taken that apply to this

problem, as well as to others, the objective is not to supply an architecture to solve any specific problem. The choice of this particular test-bed for evaluation is because it has all of the characteristics of the environments these techniques are aimed at. Although the tests are based on real data, the constraints put on the simulation that make it possible to run several hundreds of times faster that real-time, do not allow us, at this moment, to extrapolate the conclusions taken to a real-life scenario.

The quality of the state is composed by a local and a global component, weighted by a factor β (in this case equal to 0.75). The quality $q_{m,i,t}$ for agent i at location m and time t is calculated by:

$$q_{m,i,t} = \beta p_i(C_l/ast_{i,t}) + (1-\beta)p_m(C_g/ast_{m,t}), \tag{9}$$

where C_l is the patience threshold of drivers for one crossing (10 seconds in this case), $ast_{i,t}$ is the average stopped time of cars currently situated in the incoming lanes of the crossing controlled by agent i. If $ast_{i,t}$ is smaller than C_l it takes the value of C_l so that the first component is always in the interval $[0, 1]$. p_i is a penalty value that is 1.0 if all incoming lanes have an occupation-rate smaller than 1.0, and is multiplied by a fixed value (0.75 in this case) for every full incoming lane. The calculation of the global component is similar, although it uses a different patience threshold (20 seconds) and $atc_{m,t}$ is the average stopped time for all cars in location m. This value includes the time cars may had to wait before entering the scenario. The global penalty value is computed across all lanes in the agent's location.

In this experiment there are two scenarios (*Individual and Social*). They are equal in all respects except that in the social scenario agents are able to communicate.

Agents observe a state (s) composed of 10 variables (s_i), where $s_i \in [0, 1]$, for all $i \in [1, 10]$:

- 1–4: Normalized occupation rate of each of the four incoming

lanes, i.e. number of cars present over the number of cars required to fill the smallest lane in the location.

- 5–8: Incoming traffic from a given direction (0 for no traffic, 1 otherwise). It is considered that there is incoming traffic if a car has entered the lane in the last 10s or if the lane is full.
- 9: Current color of the agent's traffic-light (0 for red, 1 for green).
- 10: Time since the last change in the traffic-light, normalized at a value of 180 seconds.

Each agent must choose one of two actions. Either set the North-South lanes to green or red (the East-West lanes will automatically switch to the opposite color). An action is requested every 20 simulated seconds. Yellow times (of 5 seconds) are introduced automatically when the light changes from green to red.

Heuristic agents set to green the traffic-light on the lanes with the maximum occupation rate. To prevent quick oscillation of the traffic-light the lanes that have green light have their occupation-rate multiplied by a value larger that 1.0 (1.3 in this case).

4.3 Learning Algorithms

This section contains a summarized description of the learning algorithms used in the experiments. This description is focused on the variations from their standard form that were introduced in this work. *Q-Learning* and *Evolutionary Algorithms* will be given more attention due to their importance in these experiments. For details on the standard versions of these algorithms the reader should consult the referred bibliography or (Nunes and Oliveira, 2004).

Backpropagation (BP) (Rumelhart *et al.*, 1986) is a well known *supervised learning* algorithm, commonly used with networks of

differentiable non-linear units connected by weighted links (Artificial Neural Networks, ANN). In this work we use classical, online, *backpropagation* to integrate the information received by *advice-exchange* in agents whose main learning algorithm is also based on ANN and to perform state-to-quality mapping in Connectionist Q-Learning.

Q-Learning (QL) (Watkins and Dayan, 1992) is the most commonly used learning algorithm in the class of RL. Its most simple form, one-level Q-Learning, is based on a table that stores the estimated quality $Q(s, a)$ of performing action a at state s for all state-action pairs. When a reward r_t is received, at time t, the value of $Q(s, a)$ is updated as follows:

$$Q_{t+1}(s_t, a) = (1 - \alpha)Q_t(s_t, a) + \alpha(r_t + \beta Q_{max}(s_{t+1})), \qquad (10)$$

where s_{t+1} is the state of the environment after performing action a at state s_t, α is the learning rate and $\beta \in]0, 1[$ a discount factor applied to the estimated quality of the next state ($Q_{max}(s_t)$), which is given by:

$$Q_{max}(s_t) = \max_a(Q(s_t, a)), \qquad (11)$$

for all possible actions a when the system is in state s_t.

To incorporate the advice from other agents before acting, a form of *supervised learning* is employed in the update of the quality values. When an agent is advised to use action a as response to state s it will sum a positive value (b_{up}) to $Q(s, a)$ and a negative value (b_{down}) to all other actions available at state s (in the current experiments $b_{up} \approx na|b_{down}|$, where na is the number of actions available for the current state). A similar technique, labelled *Biasing-Binary* (Whitehead, 1991), uses the same absolute quantity both for positive and negative feedback. When incorporation of the knowledge is done after the reward is received, i.e. the agent is imitating and learning from the result, this process is not necessary because QL can learn from the reward even when the action was not its own choice.

When the state and action spaces are continuous, or too large, the implementation of Q-Learning with quality tables is not feasible. In these cases a discretization of the state-space may be necessary, but even this strategy cannot cope with some problems. In the most difficult problems, in which a discretization would either be too coarse or need tables that would be too large to store and too time-consuming to access, the usual solution is to use connectionist Q-Learning (QConn) (Lin, 1992). In this approach the table that stores a mapping of state-action pairs to their estimated quality is replaced by a set of ANN (one for each action) trained with *backpropagation*. Instead of the update described in equation 10 the ANN that corresponds to the executed action is trained using the state (s) as the input example and $r_t + \beta Q_{max}(s_{t+1})$ as the desired response. In the Predator-Prey problem we used standard QL and in the Traffic-Control problem QConn.

Evolutionary Algorithms (EA) (Holland, 1975; Koza, 1992) are a well known learning technique, with biological inspiration. The solution parameters are interpreted as a specimen (or phenotype), and its performance in a given problem as its fitness. After the evaluation of all the specimens the ones with best fitness are selected for breeding. The selected specimens are then mutated and crossed-over to generate a new population, usually of the same size as the previous one.

The variant of EA used in these experiments is based on the work presented in (Glickman and Sycara, 1999), and its main characteristics are:

- The genotype is the set of real-valued matrixes that correspond to the weights an ANN of fixed size.
- Each specimen is evaluated during a certain number of epochs (3 in this case). In the first epoch of evaluation *advice-exchange* is inhibited.

- The selection strategy is elitist (keeps a number of the best specimens in the next generation).
- Mutation is done by disturbing all the values of the parameters with random noise with null average and normal distribution. The variance of the mutation rate is slowly decayed during training.
- The crossover strategy consists on choosing two parents from the selected pool and copying each node of the ANN (along with the weights of all incoming connections) from a randomly chosen parent.

Each agent contains a population of specimens. To incorporate information given by advice an agent will use backpropagation with the advised action as desired response. The selection process will work on the specimens after they have been changed by the *backpropagation* algorithm, which can be interpreted as *Lamarckian learning*. Advice may be replayed at the end of each learning epoch. To do this advice is stored after being used. When the storage-space is filled, incoming advice will replace the oldest advice stored. Only Novice agents may replay advice and this is done when the last epoch was more successful than previous ones. When an agent changes advisor, stored-advice is cleared. Advice-replay allows to incorporate knowledge gathered by advice more rapidly, but it's use must be carefully considered because it is computationally heavy and it may reduce the diversity of the specimens. Other options that may allow a more effective use of advice are being studied.

Since backpropagation of the advised action implies that this is the best action for the given state it should only be done when the agent verifies that this is the case. When advice is integrated before acting the agent must trust the advisor and believe that the advised action is indeed the best response for its state. When an agent is imitating, i.e. incorporation is done after acting, the agent can verify if the action did produce a reward that is similar or higher than what it would ex-

Table 2. Test results in experiment 10x10 for each scenario-algorithm pair.

Scenario	Alg	Best Team	Percent	Avg
Social Heu.	Heu.	0.3477	-	0.1730(+/-0.0025)
Social Heu.	QL	0.4009	-0.3%	0.1942(+/-0.0074)
Social Heu.	EA	0.4044	+28%	0.1883(+/-0.0116)
Social Het.	QL	0.4018	-0.1%	0.1885(+/-0.0232)
Social Het.	EA	0.4061	+29%	0.1892(+/-0.0148)
Social Hom.	QL	0.4022	-0.0%	0.1843(+/-0.0375)
Social Hom.	EA	0.3263	+3.4%	0.1466(+/-0.0234)
Individual	QL	0.4023	-	0.1908(+/-0.0237)
Individual	EA	0.3156	-	0.1429(+/-0.0184)

pect for that state. If the action proves to be bad advice and produces a relatively low reward, the EA-agent will not incorporate the advice.

5 Results and Discussion

In this section the results of the experiments described above will be presented and discussed.

5.1 Predator-Prey

The results presented in tables 2 and 3 refer to the average performance achieved in test by the best team (labelled *Best Team*) and the average individual performance of all agents (labelled *Avg*) in several different cases. The team performances are calculated by adding the rewards of both agents in a team at each epoch. The average performance is accompanied by the correspondent standard deviation. The column labelled *Percent* contains the percentual difference between the result on its left and the same result for individual agents of the same type (QL or EA). The measures were taken for each experiment

Table 3. Test results in experiment 20x20 for each scenario-algorithm pair.

Scenario	Alg	Best Team	Percent	Avg
Social Heu.	Heu.	0.2444	-	0.1217(+/-0.0006)
Social Heu.	QL	0.2387	+19%	0.1177(+/-0.0013)
Social Heu.	EA	0.2412	+3.4%	0.1179(+/-0.0018)
Social Het.	QL	0.2299	+15%	0.1110(+/-0.0020)
Social Het.	EA	0.2292	-1.7%	0.1122(+/-0.0019)
Social Hom.	QL	0.2188	+9.22%	0.0874(+/-0.0097)
Social Hom.	EA	0.2281	-6.2%	0.1118(+/-0.0018)
Individual	QL	0.2004	-	0.0860(+/-0.0049)
Individual	EA	0.2332	-	0.1071(+/-0.0097)

(10x10 and 20x20), scenario (Individual, Social Homogeneous, Social Heterogeneous and Social Heuristic) and learning algorithm (QL and EA).

QL and EA-agents have quite different behaviors in these two scenarios. In the 10x10 experiment (table 2) QL-agents are clearly better than EA-agents and H-agents. This is achieved by learning joint strategies in which predators push the prey towards each other. In the 20x20 experiment (table 3) joint strategies are harder to develop and maintain, thus the best result is achieved by H-agents, and the best learning agents are EA-agents. The difference in performance of EA versus QL-agents, in these two experiments, was one of the main reasons for the choice of these variants of the problem. Analyzing the results we can observe that:

- Agents show similar or better team performances in Social scenarios (with the exception of EA-agents in Social Homogeneous scenarios, in experiment 20x20).
- For the agents that show lower individual performances in each experiment (EA-agents in 10x10 and QL-agents in 20x20), improvements on the team performance in Social Heterogeneous

and Heuristic environments are clear (ranging from 15% to 29%).

- The best average performances of agents occur in *Social Heuristic* scenarios, even when H-agents show relatively poor performances (as in experiment 10x10).

- Cooperation in Homogeneous environments leads to slightly better results for the agents that have lower individual performances in each scenario.

- The experiments do not allow for a firm conclusion on the comparison of the average results of the best agents in Individual versus Social Heterogeneous scenarios. In experiment 10x10 the average performance of the best agents (QL-agents) shows a slight decrease in Social Heterogeneous scenarios. In experiment 20x20 the average performance of the best agents (EA-agents) shows a slight increase in the same test. Both of these changes are negligible considering the standard deviations of the results.

In summary, the average performance of the best agents is maintained and the agents with lower individual performance learn from their more successful peers, although they seldom surpass their performances. The fact, observed in some experiments, that agents can surpass the performance of their advisors and become Experts themselves, is not as frequent as expected. Nevertheless, it is interesting to point out that QL-agents do achieve better average scores in the Social Heuristic scenario of experiment 10x10 than in any other, even when other agents have lower scores. The hypothesis that we pose for this behavior is that H-agents lead QL-agents to a quick stabilization of the most obvious choices. This can grant QL-agents enough time to explore more thoroughly the possible options to their advisors' individualistic strategy.

5.2 Traffic Control

Table 4. Test results in experiment TC1 for each scenario-algorithm pair.

Scenario	Alg	Avg	%	Std. Dev.	Best	%
Individual	HE	0.3424	89%	+/-0.0008	0.34	88%
Individual	QL	0.3832	100%	+/-0.0034	0.39	99%
Individual	EA	0.3059	79%	+/-0.0459	0.37	94%
Social	HE	0.3420	89%	+/-0.0009	0.34	87%
Social	QL	0.3814	99%	+/-0.0059	0.39	99%
Social	EA	0.3652	95%	+/-0.0166	0.39	100%

The results presented in tables 4 and 5 refer to the scenarios TC1 and TC2, respectively. The column labelled "Avg" contains the average results for each algorithm-scenario pair. Under the label "Std. Dev" are the standard deviations of each of the values for average performance. The column labelled "Best" shows the average test result of the best agent. The columns labeled % show the percentage relative to the best result in each column, the first % column refers to the average results, and the second to the best results. Table 6, as the same structure and refers to the global part of the reward of each team.

In the single-crossing tests (TC1, table 4), the average results for QL agents is similar in social and individual trials, and EA-agents have a 16% performance increase. The best results for EA-agents have a slight improvement. As in the previous experiment, the agents that have lower performances profit more from communication than others. From the standard deviations we can also observe that the behavior of EA-agents is not only better in average but also more stable.

In the twin-crossing tests (TC2, table 5), the average results of the best agents (QL) is, again, similar in individual and trials, but EA-agents show a 7% improvement in performance. EA-agents seem able to reach scores higher than QL-agents as can be seen in the Best

Table 5. Test results in experiment TC2 for each scenario-algorithm pair.

Scenario	Alg	Avg	%	Std. Dev.	Best	%
Individual	HE	0.2708	93%	+/-0.0013	0.27	68%
Individual	QL	0.2852 98%	+/-0.0093	0.30	75%	
Individual	EA	0.2638	91%	+/-0.0347	0.37	93%
Social	HE	0.2710	93%	+/-0.0011	0.27	68%
Social	QL	0.2895	100%	+/-0.0111	0.31	76%
Social	EA	0.2859	98%	+/-0.0352	0.40	100%

results column. It is interesting to notice that the same percentual increase in social trials also applies to the best results of EA-agents. The best EA-agent's performance is increased even when none of their peers is able to reach scores at the same level. This seems to indicate that, in some cases, communication can lead a agents to behave better is social domains than the best they are able to do in any individual trial. Nevertheless, in average we can only say that the agents with lower individual performance (EA-agents) can learn to behave as the best (with only 2% difference).

If we measure only the global component of the reward we reach similar conclusions. EA-agents improve this component by 22%, in average. QL-agents do not fully take advantage of this rise, but they also show a small improvement (4%).

Table 6. Global reward results in experiment TC2.

Scenario	Team	Avg	%	Std. Dev.	Best	%
Individual	HEU	0.1041	94%	+/-0.0011	0.11	59%
Individual	QL	0.1024	92%	+/-0.0089	0.12	64%
Individual	EA	0.0865	78%	+/-0.0247	0.16	89%
Social	HEU	0.1044	94%	+/-0.0007	0.11	59%
Social	QL	0.1066	96%	+/-0.0100	0.13	71%
Social	EA	0.1106	100%	+/-0.0274	0.18	100%

These tests point in the same direction as the previous ones, advice exchange does improve the average performance of the agents with lower scores, although it may prevent exploration. As is the previous cases not all agents reach the same performance levels and some advisees do not reach the level of performance of the advisors.

6 Conclusions and Future Work

Advice-exchange has evolved, from a simple form of sharing experiences with other agents, to the draft of an architecture that enables cooperation between heterogeneous groups. Along the way, problems such as: disturbances in the learning process (caused by conflicting advice), learning of joint strategies from more successful groups, spurious results that caused agents to trust the "wrong" advisor, and many others, have been addressed by introducing new concepts and tools. The initial objective was to create a process by which agents could improve their learning skills through communication with other agents. The results presented here show that low performance agents can improve their performance by communicating with others.

One of the goals of this work was to create a learning system that could generate better behaviors than individual learning. So far this was only detected in a few specific cases and it is not the rule. In fact, communication can even hurt the learning performance by leading all agents into an area with a local minimum early in the trial.

An important factor to consider is the delay introduced by advice-exchange. If we exclude advice-replay the delay introduced by the process is negligible. Advice-replay may cause considerable delays depending on the number of stored advice. Another important delay factor is communication between processes in different physical locations. In this simulation that factor cannot be measured since all agents are part of the same process. The communication necessary

to allow advice-exchange is simple in structure and reasonably low in average, although it tends to be bursty because all agents tend to ask for advice at the same times (when another agent improves its performance making their current solution obsolete). To give a reasonable idea of the learning-time we can say that the Traffic Control simulation, with advice-exchange, is running several hundred times faster than real-time and more than 70% of the computation-time is used in the movement of vehicles and not in the learning procedures.

The objectives of future work are mainly to refine the process into an architecture that promotes better team behaviors. The identification of the type of problem an agent is facing and the adequate control of learning parameters in run-time, based on this information, seems to be a key point to improve performance. Other improvements, based on combination of advice from different sources, rendering unsolicited advice and the use of other learning algorithms, are also scheduled for future work. In some situations it is important to have a global view of a situation, the introduction of team supervisors that learn from a summarized view of the state and may influence the decisions of agents can improve the cooperation capabilities of a team.

The road to new learning systems that interact with complex environments and communicate is still long. Above we have sketched what we believe are some of the fundamental characteristics of agents that can learn by communicating with others as well as from the environment. As was mentioned during the description, the implementation of some of these concepts must be adapted to the type of agents or the environment, but, by creating a set of techniques that promotes communication and effective integration of the exchanged information in the learning process, we can give agents the possibility to interact with others and profit from this interaction.

Acknowledgments

Our thanks to CML (Lisbon City Hall Traffic Control Department) that graciously made available the necessary data to implement the Traffic Control environment, and also to Luís Botelho and Pedro Figueiredo.

References

Banerjee, B., Mukherjee, R., and Sen, S. (2000). Learning mutual trust. In *Working Notes of AGENTS-00 Workshop on Deception,Fraud and Trust in Agent Societies*, pages 9–14.

Benda, M., Jagannathan, V., and Dodhiawalla, R. (1985). On optimal cooperation of knowledge resources. Technical Report BCS G-2012-28, Boeing AI Center, Boeing Computer Services, Bellevue, WA.

Berenji, H. R. and Vengerov, D. (2000). Advantages of cooperation between reinforcement learning agents in difficult stochastic problems. In *Proc. of the Nineth IEEE International Conf. on Fuzzy Systems (FUZZ-IEEE '00)*.

Biswas, A., Sen, S., and Debnath, S. (2000). Limiting deception in groups of social agents. *Applied Artificial Intelligence Journal*, **14**(8), 785–797. Special Issue on Deception, Fraud and Trust in Agent's Societies.

Clouse, J. A. (1997). *On integrating apprentice learning and reinforcement learning*. Ph.D. thesis, University of Massachusetts, Department of Computer Science.

Clouse, J. A. and Utgoff, P. E. (1991). Two kinds of training information for evaluation function learning. In *Proc. of AAAI'91*. Anaheim.

Dorigo, M. and Colombetti, M. (1994). Robot shaping: Developing autonomous agents through learning. *Artificial Intelligence*, **71**(2), 321–370.

Glickman, M. and Sycara, K. (1999). Evolution of goal-directed behavior using limited information in a complex environment. In *Proc. of the Genetic and Evolutionary Computation Conf., (GECCO-99)*.

Haynes, T., Wainwright, R., Sen, S., and Schoenfeld, D. (1995). Strongly typed genetic programming in evolving cooperation strategies. In *Proc. of the Sixth International Conf. on Genetic Algorithms*, pages 271–278. Pittsburgh, Pennsylvania, July.

Holland, J. H. (1975). *Adaptation in Natural and Artificial Systems*. University of Michigan Press.

Koza, J. R. (1992). *Genetic programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge MA.

Lin, L.-J. (1992). Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine Learning*, **8**, 293–321.

Litmann, M. L. (2001). Value-function reinforcement learnnig in markov games. *Journal of Cognitive Research*, **2**, 55–66.

Littman, M. L. (1994). Markov games as a framework for multi-agent reinforcement learning. In *Proc. of the Eleventh International Conf. on Machine Learning*, pages 157–163.

Maclin, R. and Shavlik, J. (1994). Incorporating advice into agents that learn from reinforcement. In *Proc. of the Twelfth National Conf. on Artificial Intelligence (AAAI-94)*. AAAI Press.

Nagel, K. and Shreckenberg, M. (1992). A cellular automaton model for freeway traffic. *J. Phisique I*, **2**(12), 2221–2229.

Nicolescu, M. and Matarić, M. J. (2001). Learning and interacting in human-robot domains. *IEEE Transactions on systems, Man and Cybernetics, special issue on Socially Intelligent Agents - The Human In The Loop*.

Nunes, L. and Oliveira, E. (2002a). Advice-exchange in heterogeneous groups of learning agents. Technical Report 1-11/02, FEUP/LIACC/NIAD&R. unpublished.

Nunes, L. and Oliveira, E. (2002b). On learning by exchanging advice. In *Proc. of the First Symposium on Adaptive Agents and Multi-Agent Systems (AISB'02)*. Imperial College, London.

Nunes, L. and Oliveira, E. (2003). Advice exchange between evolutionary algorithms and reinforcement learning agents: Experimental results in the pursuit domain. In *Proc. of the Second Symposium on Adaptive Agents and Multi-Agent Systems (AAMAS/AISB'03)*.

Nunes, L. and Oliveira, E. (2004). Exchange of information during learning. Technical Report 3-02/04, FEUP/LIACC. unpublished.

Price, B. (2003). *Accelerating Reinforcement Learning with Imitation*. Ph.D. thesis, University of British Columbia.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning internal representations by error propagation. *Parallel Distributed Processing: Exploration in the Microstructure of Cognition*, **1**, 318–362.

Sen, S. (1996). Reciprocity: a foundational principle for promoting cooperative behavior among self-interested agents. In *Proc. of the Second International Conf. on Multiagent Systems*, pages 322–329. AAAI Press, Menlo Park, CA.

Sutton, R. S. and Barto, A. G. (1987). A temporal-difference model of classical conditioning. Technical Report TR87-509.2, GTE Labs.

Tan, M. (1993). Multi-agent reinforcement learning: Independent vs. cooperative agents. In *Proc. of the Tenth International Conf. on Machine Learning*, pages 330–337.

Watkins, C. J. C. H. and Dayan, P. D. (1992). Technical note: Q-learning. *Machine Learning*, **8**(3), 279–292.

Whitehead, S. D. (1991). A complexity analisys of cooperative mechanisms in reinforcement learning. *Proc. of the 9th National Conf. on Artificial Inteligence (AAAI-91)*, pages 607–613.