Learning Real Team Solutions

Cristina Versino and Luca Maria Gambardella

IDSIA, Corso Elvezia 36, CH-6900 Lugano, Switzerland
{cristina,luca}@idsia.ch, http://www.idsia.ch

Abstract. This paper presents "Ibots" (Integrating roBOTS), a computer experiment in group learning designed on an artificial mission. By this experiment, our aim is to understand how to use reinforcement learning to program automatically a team of robots with a shared mission. Moreover, we are interested in *learning real team solutions.* These are programs whose form strongly depends on the number of robots composing the team, on their individual skills and limitations, and on any other mission boundary condition which makes it worth to prefer "at a team level" certain solutions to others. The Ibots mission is specified implicitly by means of a single reinforcement signal which measures the team performance as a whole. This form of payoff leads to real team solutions. Benefits and drawbacks of using team reinforcement as opposed to individual robot reinforcement are discussed.

1 Introduction

The use of reinforcement learning [1] to produce self-programming robots is not new in the context of *single robot* missions, where a reinforcement signal directly evaluates the behavior of the only robot in charge of carrying out the task.

A Robot Credit Assignment Problem. The picture changes as *many robots* are acting at the same time, with little or perhaps no knowledge at all about teammate activities. In this scenario, if the reinforcement signal reflects the whole team performance, each single robot is faced with the problem of deciding to what extent its own behavior has contributed to the overall team's good or bad score: this is the *robot credit assignment problem*[1]. Because of the robot credit assignment problem, each robot has a *noisy* perception of the mission it is asked to accomplish. A single robot can behave identically many times (during different trials of the mission), and nevertheless, it may receive completely different payoffs. This occurs because it is not the only actor, and the reinforcement signal just *partially* depends on its actions.

Bypassing the Robot Credit Assignment Problem. Instead of addressing the robot credit assignment problem directly, one can bypass it by reformulating the team learning problem.

A *first* way is to enable *broadcast communication* between teammates. If a robot is *aware* of other robots' perceptions and actions, then it is in a position to

[1] In [2], this problem is called *inter-agent credit-assignment problem.*

make sense out of a global team payoff. Explicit communication makes the team equivalent to a "big robot" [3], whose perception and action are the union of perceptions and actions of all team members. Seen in this way, the team payoff is the measure of the big robot performance. The utility of communication has been proved experimentally for small teams of real robots both in a simplified hazardous waste cleanup mission [4], and in a box-pushing task [5, 6]. However, communication is not always possible technically, and it tends to become a bottleneck as the team size increases [7].

A *second* way of avoiding the robot credit assignment problem is to measure *each robot individual performance* instead of team performance. In [8, 9] this idea is applied to training a group of real robots in a foraging task. Pucks disseminated in the workspace have to be collected and delivered to a home area. Each robot in the team learns a personal policy through individual payoff. For example, a robot is rewarded whenever it grasps a puck or if it drops a puck at home. In this framework, a single robot is not interested in the performance of its teammates, because it addresses the mission in an individualistic sense. We see *two drawbacks* in this approach. *First:* its underlying assumption is that team performance indirectly increases because individual performance increases. However, if the robots do not learn the task at a *similar pace*, it cannot be guaranteed that each robot will learn and participate to the mission. If not all robots learn how to contribute to the mission, the team performance will be suboptimal. As an example, suppose that in the foraging task one robot in the team manages to learn (maybe just by chance) the individually optimal policy after a few trials. This "superrobot" will collect most of the pucks by itself, diminishing the learning opportunities of its teammates because pucks are a *limited, shared resource.* To improve this state of affairs, the superrobot should behave in a suboptimal way (by forgetting its optimal policy) so as to let the other robots take part in the task and learn. However, there is no reason for the superrobot to recede from its optimal policy, because it is designed to be the best possible individual. The superrobot phenomenon is not as unrealistic as it appears at first glance. It may arise, for instance, in *incremental learning* experiments where the team size is progressively increased: elder, experienced robots would tend to carry out the mission by themselves, limiting the possibility of novice robots to participate and learn. In [10], it is argued that *social rules* can be learnt by the robots to minimize resource competition and to direct their behavior away from individual greediness and towards global efficiency. The idea is interesting, but the experimental results reported in [10] are preliminary. The *second* drawback we find in the individualistic approach is summarized in the answer to the following question. Suppose that, in a team of homogeneous robots, all robots receive personal reinforcement signals generated by the same payoff function: where are the policies learnt by the robots expected to converge? In general, to the optimal policy for a robot carrying out the mission by itself. The robots will behave as "clones" of *a robot designed to work alone.*

Real Team Solutions. We feel this is not the spirit of group learning, which should be aimed at producing *efficient real team solutions. These are policies whose form is strongly influenced by the number of robots in the team, by each robot's skill and weakness, and by any other mission boundary condition which is relevant for preferring "at a team level" some solutions to others.* Real team solutions encourage the participation of all robots which are in a position to positively contribute to the mission.

Facing the Robot Credit Assignment Problem. *To obtain truly team solutions, one should use team payoffs at the price of dealing with the ambiguity posed by the robot credit assignment problem.* Multi-agent learning experiments based on team payoffs are illustrated in recent works [11, 12, 13, 14].

In [11] a team of simulated agents learns signaling behaviors to efficiently solve an object-gathering task in an unknown and changing environment. The reinforcement signal is based on the total time needed by the team to gather all the objects in the workspace. Experiments are carried out under several conditions: with teams of different size, with a variable number of objects, and with different object distributions; moreover, during learning the object distribution is occasionally modified to test the agents ability of traking environmental changes. Under these heterogeneous experimental conditions, each agent learns the appropriateness of exhibiting a given signaling behavior. For example: an agent perceiving an object is faced with the problem of deciding whether to activate its "object-signaling" behavior to attract other agents towards its position. The agent learns that exhibiting this signaling behavior is, in general, rewarding for the team when the objects are distributed in clusters: in this case, the detection of one object gives a high chance of finding other objects in the neighborhood, and these could be collected by the called agents. On the contrary, the same signaling behavior is inappropriate if the objects are uniformly distributed in the environment: the agent should refrain from calling other teammates whenever it finds an object. Finally, certain mission boundary conditions require the agents to acquire different signaling policies: they specialize into *signallers* and *harvesters.* The authors show by statistical analysis of the results that the team discovers by trial-and-error a near-to-optimal signaling policy given the specific mission conditions.

In [12] a team of Q-learning agents is engaged in the challenging real-world problem of elevator dispatching. Each agent is responsible for controlling one elevator car. Two different control architectures are tested. In the *parallel* architecture, the agents share a single neural network which models a common policy: this allows the agents to learn from each others experiences but forces them to use identical policies. In the *decentralized* architecture, the agents learn personal networks, which allow them to specialize their control policies. The team receives as global payoff the sum of squared wait times of passengers. Despite this noisy reinforcement signal and the inherent stochastic nature of the task, results obtained in simulation on both architectures surpass the best known heuristic elevator control algorithm. The authors expect an additional advantage of reinforcement learning over heuristic controllers in buildings with heterogeneous arrival rates at each floor, because Q-learning agents may adapt to each floor traffic profile.

In [13] a general method for incremental self-improvement and multi-agent learning in unrestricted environments is presented. In one of the implementations, a recurrent neural net is applied to a non-Markovian maze task. Each connection in the net is viewed as an agent: the connection's weight represents the agent's policy. The net learns to guide an animat to a goal by using a team payoff whose value turns from 0 to 1 only once the animat hits the goal. Looking at the net's connections as if they were a team of agents is an unconventional point of view. But why not doing so? After all, a neural net is a good example of a set of partially independent agents (the net's weights, or the neurons themselves) which learn to act well "as a team". Following this view, the structural credit assignment problem in connectionist reinforcement learning can be regarded as being equivalent to the robot credit assignment problem described above.

Finally, this paper presents "Ibots" (Integrating roBOTS) [14], an experiment in group learning designed to understand how to use reinforcement learning to program automatically a team of robots with a shared mission. As in [11, 12, 13], Ibots learn through a reinforcement signal which measures the team performance as a whole. In this way, Ibots manage to learn real team solutions.

2 General Issues

Before describing the Ibots mission, we summarize the general issues addressed by the experiment, as well as the underlying assumptions and design choices.

Team Size. In the Ibots experiment, the same mission is handled with teams of *different size*. By this, our aim is to assess whether learnt policies change as the team size changes.

Team Composition. Ibots can be *homogeneous* or *heterogeneous*. When they are homogeneous, they all have same sensors and same actuators. When they are heterogeneous, they have different sensors and/or different actuators. When Ibots are heterogeneous, they have potentially different skills and weaknesses due to physical characteristics. The question is whether they can learn to specialize their behavior so as to emphasize skills and minimize the impact of weaknesses. The same question is addressed in [15].

Mission's Boundary Conditions. Ibots are confronted with *different boundary conditions* to the mission. We are interested in checking whether the strategy learnt by the team to accomplish the mission changes as the boundary conditions change. [11] also address this question by dealing with different object distributions in their object-gathering task.

Control Programs. Ibots run and learn control programs defining their behavior in the mission. The Ibots' control programs may be *public* or *private*. When they are public, all Ibots share the same program; when they are private, each Ibot works with a different, personal program.

Robots working with private programs instead of public programs is more general. If we choose the public program option, we assume that there exists a shared solution which works for every robot in the team. This is not the case when the robots are heterogeneous or when the mission requires the robots to specialize their behavior. If the robots are homogeneous and the mission does not require specialization, it makes sense to consider the public option, because it is faster to learn 1 shared program than n distinct programs, n being the number of robots in the team. Learning a shared program is faster for two reasons: first, the search space for the learning algorithm gets reduced, and, second, the robot credit assignment problem completely vanishes. Sometimes, even when the mission requires the robots to specialize their behaviors, it is possible to define the team learning problem in such a way that the public program option is still valid. As an example, we guess that if the agents in [11] were designed to learn directly the distribution of object-signaling agents over the whole team (instead of the individual agent's tendency to exhibit the object-signaling behavior), it would be easier for the team to learn the correct proportion between signaling and non-signaling agents. Finally, we stress that learning 1 shared program in a team of n robots is not the same as learning 1 program with a single robot and then cloning it n times. *A shared program is a real team solution, while cloned programs are not.* The Ibots experiment illustrates this point clearly (see "Public Control Programs", in Section 6.2).

In our opinion, the possibility of learning a single shared program instead of several private programs has been overlooked by most collective robotics works. This is curious if one considers the recent trend of designing robot teams inspired to Swarms [16]. Robot Swarms have shown that an interesting group behavior may emerge from the interaction between robots running a public control program. Odd enough, when learning techniques are used to derive automatically a team behavior, people usually neglect the possibility of learning a public program.

The only work we are aware of where both public and private policies have been considered, is the recent paper on elevator dispatching by [12]. Their "parallel" and "decentralized" architectures are equivalent to our "public" and "private" policies, respectively. In this application, however, it remains unclear which are the benefits of using the decentralized architecture in a problem where the elevator cars to be controlled are homogeneous.

Communication. Ibots do *not communicate*. By this we mean that, when Ibots work with private control programs, they do not tell each other which these programs are. In this restricted sense, there is no communication. By this choice we depart from works in collective robotics where robots are aware of teammates activities through explicit communication [5, 4, 15, 6].

Sharing Limited Resources. In collective robotics, robots usually share physical resources (the workspace, objects to gather, etc).

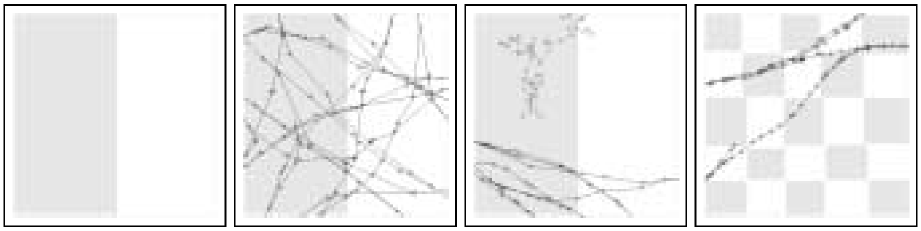

Fig. 1. (From left to right) (a) The squared arena and the "half-full" *Region*: $I = 0.49$. The arena side is 500 unit long. (b) Trajectory of one Ibot running a solitary trial with $N_{max} = 100$ and $Prog = (30°, 200)$. Crosses indicate sampled points. The result is $N_{in} = 52$, $\hat{I} = 0.52$. (c) Trajectories of two Ibots running a shared trial with $N_{max} = 100$, starting from a scattered configuration and running private control programs: $Prog^1 = (180°, 50)$ (gray trace), $Prog^2 = (20°, 150)$ (black trace). The result is $N^1_{in} = 55$, $N^2_{in} = 30$, $\hat{I} = 0.85$. Ibot 1 took $N^1_{sam} = 61$ samples, Ibot 2 took $N^2_{sam} = 39$. (d) Trajectory of one Ibot running a trial on the "chessboard" *Region*: $I = 0.49$. $N_{max} = 100$ and $Prog = (10°, 40)$. The result is $N_{in} = 45$, $\hat{I} = 0.45$.

In our experiment, Ibots share a limited, non-physical resource. However, there is no a priori rule which decides how this resource should be shared. We thought the Ibots should be more or less resource-*greedy* depending on their control programs. Thus, while the Ibots learn their control programs, they implicitly learn to share the resource in a way which is convenient for the team. In particular, when they run *public programs*, they exhibit the same greediness, and, on the average, each Ibot takes the same amount of the total resource. This is not the case when the Ibots run *private programs*, where each Ibot has a different propensity in consuming the resource. This fact has a great impact on the team performance in the mission.

Learning through Reinforcement. Ibots learn to accomplish the mission through *trial-and-error*. At each trial, each Ibot receives a *reinforcement signal* which measures the *team performance*. Ibots are confronted with their performance as a group because we want them to learn a team strategy. This view is shared with [11, 12, 13], but it is different from that pursuited in [8, 9], where learning is driven by individual performance.

3 Ibots

The *mission* for the Ibots is to guess the integral I ($0 \leq I \leq 1$) of an arbitrary gray *Region* drawn on the white ground of a squared arena (Fig. 1(a),(d)).

How are robots turned into Ibots? For the sake of clarity, let us first consider the case of a team composed of one Ibot.

3.1 One Ibot

The Ibot's dowry is a *control program Prog* which lets it explore the arena while sampling the ground color. By activating *Prog*, the Ibot performs a *trial* run (Fig. 1 (b),(d)).

A trial starts from a random location in the arena. It is a sequence of N_{max} *elementary movements* separated by stops. Whenever the Ibot stops, it samples the ground color. A color reading equal to gray gives evidence for the sample to be "inside *Region*", while a white reading is interpreted as "outside *Region*". At the end of the trial, the Ibot returns the number N_{in} of samples it counted inside *Region*. $\hat{I} = N_{in}/N_{max}$ is its estimate of I at this trial. $E = |I - \hat{I}|$ is the error in the estimate.

How are elementary movements generated? The Ibot control program *Prog* depends stochastically on two parameters $(\alpha_{prog}, \delta_{prog})$ which remain fixed during a trial. α_{prog} is used to generate a rotation instruction for the Ibot, while δ_{prog} induces a translation. The semantics of α_{prog} and δ_{prog} is as follows. First, a number α is drawn from a uniform distribution in $[-\alpha_{prog}, +\alpha_{prog}]$. This is interpreted by the Ibot as: "Rotate α degrees.". Second, a number δ is drawn from a uniform distribution in $[0, \delta_{prog}]$. The interpretation for δ is: "Translate δ units in your current heading direction, calling the *bumping rule* if necessary.". The bumping rule is called when the Ibot meets the arena border before having covered the whole distance δ. In this case, the Ibot rotates 180° and covers the remaining distance *minus* 1: bumping against the arena border consumes 1 unit of translation to prevent the Ibot from bumping forever without consuming δ, a rare case which may occur when the Ibot lies in a corner. The bumping rule can be called recursively. We decided the bumping rule should just reverse the Ibot's travelling direction to affect minimally its natural motion angle which already depends on the parameter α_{prog}.

In our computer experiment, a rotation instruction is executed by the Ibot in one time unit whatever the rotation angle α; while the execution time of a translation instruction is directly proportional to the distance δ.

Finally, both program parameters α_{prog} and δ_{prog} take values in a finite range: $0 \leq \alpha_{prog} \leq \alpha_{max}$ and $0 \leq \delta_{prog} \leq \delta_{max}$.

3.2 A Team of Ibots

Each Ibot i $(i = 1, \ldots, N_{ibots})$ being equipped with a control program $Prog^i = (\alpha^i_{prog}, \delta^i_{prog})$, there are several ways of generalizing from the single Ibot case to the team case (Fig. 1 (c)). We have considered both the cases where the $Prog^i$s may be *public* or *private*. Public means that all instances of $Prog^i$s are constrained to be the same *Prog*, while private means that each $Prog^i$ may be different.

The Ibots activate the $Prog^i$s in parallel to run a team trial. At the beginning of the trial, the Ibot locations are chosen at random in the arena, and these are either *clustered* or *scattered*. In the clustered configuration, all Ibots have the same initial position and orientation; in the scattered configuration, they have different positions and orientations. During a trial, the Ibots are granted a total of N_{max} elementary movements to collect N_{max} samples of the ground color *overall*: *samples are the team limited and shared resource.* Whenever an Ibot stops to take a sample, it is allowed to do so only if less than N_{max} samples have been taken by the team so far. Otherwise, the Ibot gives up the sampling, and the team trial is terminated. Notice that when the Ibots work with private $Prog^i$s, each Ibot i will collect a different number of samples N^i_{sam}: Ibots with small δ^i_{prog}s (travelling for shorter distances) will take on the average more samples than Ibots with larger δ^i_{prog}s. At the end of the trial, each Ibot i returns the number N^i_{in} of samples it counted inside *Region*. These contributions are summed in $N_{in} = \sum_i N^i_{in}$, leading to the team integral estimate $\hat{I} = N_{in}/N_{max}$, the error being $E = |I - \hat{I}|$.

Finally, Ibots are *immaterial*, they do not collide when their trajectories intersect.

4 Programmed Ibots vs. Learning Ibots

How can a team of Ibots learn to provide good estimates $\hat{I}$ of the integral I? *The goal of group learning is to find control programs $Prog^i$s leading to estimates $\hat{I}$ close to I.* As the behavior of each program depends on its parameters α^i_{prog} and δ^i_{prog}, *the target of learning is to discover "good" pairs* $(\alpha^i_{prog}, \delta^i_{prog})$. Notice that we know a *general* solution, namely:

$$\forall i : \begin{cases} \alpha^i_{prog} = 180^\circ \\ \delta^i_{prog} = \text{“length of arena diagonal”} \end{cases} \tag{1}$$

With this choice of parameters, an Ibot may reach *any* position in the arena starting from *any* other position and orientation in just *one* elementary movement. Knowing the Ibot's current position and orientation, its next position remains highly unpredictable. An observer would describe the points sampled by this Ibot as being *uniformly distributed* in the arena. This brings us to the hypothesis of the Monte Carlo method [17] for integration. This states that, by drawing N_{max} points from a uniform distribution in the arena, the error E in the estimate of the integral is probabilistically bounded by N_{max}:

$$P\left\{E \leq \frac{1}{\sqrt{N_{max}}}\right\} \geq 0.9999 \tag{2}$$

For example, by drawing 100 points, we are almost guaranteed that the error in the estimate will not exceed 0.1, whatever the *Region*'s integral and shape (remember that $0 \leq I \leq 1$). Given N_{max}, this result quantifies the *admissible error* for the Ibots mission. We call the control programs defined by Eq. 1 the "programmer solution", because it reflects, in our opinion, the way a programmer would address this robot programming task: by looking for a *general* solution, which will work for any *Region*, whatever the number of Ibots in the team, independently of their starting configuration in the arena. Though appealing, we are not interested in this *a priori* solution. Rather, we are looking for *real team solutions* established through experience. These should depend on the specific *Region*, on the number of Ibots, on their initial configuration, and on their specific skills when these latter are no longer homogeneous.

To see why this "adaptation to circumstances" makes sense, consider a team of 100 Ibots started in a scattered configuration, i.e. each Ibot's initial position is drawn at random in the arena. In this situation, it would be *unnatural* to see the Ibots running the programmer's solution, as the integral is perfectly guessable by the more economic "staying in place" program:

$$\forall i : \begin{cases} \alpha^i_{prog} & = 0° \\ \delta^i_{prog} & = 0 \end{cases}$$

How can real team solutions be derived? When dealing with private control programs, exaustive search in the multi-dimensional program space of the Ibots is unfeasible. Suppose the Ibot's *continuous* program space is discretized so that each Ibot selects its control program only among a set of N_{progs} different control programs. In a team of size N_{ibots}, the number of possible distinct program selections for the team is [18]:

$$\binom{N_{ibots} + N_{progs} - 1}{N_{ibots}} = \frac{(N_{ibots} + N_{progs} - 1)!}{N_{ibots}!(N_{progs} - 1)!}$$

Moreover, each program selection should be tested several times to be sure about its quality, because "bad" control programs have a stochastic performance. As an alternative to exaustive search, true team solutions can be derived through learning: by measuring the Ibots team performance as a whole.

5 Learning to Be Good Ibots

"Good" $Prog^i$s are programs which lead *repeatedly to admissible* estimates of I. Admissible estimates are defined with respect to N_{max} by Eq. 2. Moreover, estimates must be repeatable because we are interested in programs with a *stable* performance.

The Ibots learn good control programs $Prog^i$s by reinforcement through a sequence of trials. Each time instant t corresponds to a team trial. Let $Prog^i(t) = (\alpha^i_{prog}(t), \delta^i_{prog}(t))$ be Ibot i control program at time t. The following 4 steps are repeated forever.

1. *Each Ibot "i" independently generates a new, tentative control program* $New^i(t) = (\alpha^i_{new}(t), \delta^i_{new}(t))$ *by slightly modifying* $Prog^i(t)$.

   Given:

$$\alpha^i_{temp} = \alpha^i_{prog}(t) + \alpha^i_{rand}(t) \cdot \alpha_{step}$$
$$\delta^i_{temp} = \delta^i_{prog}(t) + \delta^i_{rand}(t) \cdot \delta_{step}$$

it is:

$$\alpha^i_{new}(t) = \begin{cases} 0^\circ & \text{if } \alpha^i_{temp} < 0^\circ \\ \alpha_{max} & \text{if } \alpha^i_{temp} > \alpha_{max} \\ \alpha^i_{temp} & \text{otherwise} \end{cases}$$

$$\delta^i_{new}(t) = \begin{cases} 0 & \text{if } \delta^i_{temp} < 0 \\ \delta_{max} & \text{if } \delta^i_{temp} > \delta_{max} \\ \delta^i_{temp} & \text{otherwise} \end{cases}$$

where:

- $\alpha^i_{rand}(t)$ and $\delta^i_{rand}(t)$ are uniform random numbers in $[-\rho(t), +\rho(t)]$ (see below for the definition of $\rho(t)$);
- α_{step} and δ_{step} are constants.

2. *The Ibots collectively carry out a trial with the newly generated programs $New^i(t)s$.*

   At the beginning of the trial, the Ibots are positioned at a random configuration (which may be clustered or scattered). At the end of the trial, each Ibot returns $N^i_{in}(t)$. The team integral estimate is $\hat{I}(t) = N_{in}(t)/N_{max}$, with $N_{in} = \sum_i N^i_{in}$. The error in the estimate $E(t)$ is:

$$E(t) = \frac{|I - \hat{I}(t)|}{\max(I, 1 - I)}$$

3. *The team reinforcement signal $R(t)$ is computed and communicated to each Ibot.*

   $R(t)$ is defined as the difference between the team errors in two successive trials:

$$R(t) = E(t-1) - E(t) \tag{3}$$

   $E(t-1)$ can be thought of as a naïve predictor of $E(t)$.

4. *Each Ibot "i" independently computes a modification for its control program and updates it.*

   The modification is:

$$\Delta\alpha^i_{prog}(t) = \epsilon \cdot R(t) \cdot (\alpha^i_{new}(t) - \alpha^i_{new}(t-1)) \tag{4}$$

$$\Delta\delta^i_{prog}(t) = \epsilon \cdot R(t) \cdot (\delta^i_{new}(t) - \delta^i_{new}(t-1)) \tag{5}$$

where ϵ, a real parameter, is the learning rate.
Given:

$$\alpha^i_{temp}(t+1) = \alpha^i_{prog}(t) + \Delta\alpha^i_{prog}(t)$$
$$\delta^i_{temp}(t+1) = \delta^i_{prog}(t) + \Delta\delta^i_{prog}(t)$$

the updated control programs is:

$$\alpha^i_{prog}(t+1) = \begin{cases} 0^\circ & \text{if } \alpha^i_{temp}(t+1) < 0^\circ \\ \alpha_{max} & \text{if } \alpha^i_{temp}(t+1) > \alpha_{max} \\ \alpha^i_{temp}(t+1) & \text{otherwise} \end{cases}$$
$$\delta^i_{prog}(t+1) = \begin{cases} 0 & \text{if } \delta^i_{temp}(t+1) < 0 \\ \delta_{max} & \text{if } \delta^i_{temp}(t+1) > \delta_{max} \\ \delta^i_{temp}(t+1) & \text{otherwise} \end{cases}$$

The description of the algorithm is completed by the following remarks and definitions.

– When the Ibots work with public control programs, only one Ibot generates the tentative program $New^i(t)$ at step 1; then, $New^i(t)$ is communicated to the other team members.
– About the error measure $E(t)$: by dividing $|I - \hat{I}(t)|$ by $\max(I, 1-I)$, $E(t)$ varies between 0 and 1, no matter what the value of I. As a consequence, the reinforcement signal also varies in a fixed interval, namely $[-1, +1]$.
– $E(-1) = E(0)$. At trial 0, we assume that the expected error $E(-1)$ is equal to the measured error $E(0)$. This implies $R(0) = 0$, so no change is made to $Prog(0)$.
– $\rho(t) = max(|R(t-1)|, \rho_c)$. The amount of variation in the new control programs is proportional to the absolute value of the reinforcement signal at previous trial. This is to enhance the tendency of escaping from programs with unpredictable performance, and, viceversa, to favor the convergence towards programs with stable performance. ρ_c is a positive constant which maintains a minimal level of exploration in program space when $R(t-1) = 0$.

We conclude this section by placing this experiment in the reinforcement learning panorama. We have set the stage for a *nonassociative, immediate* reward learning experiment. This experiment qualifies as nonassociative, because there is no perception-action mapping to be learned: the only input to the learners is the reinforcement signal. Moreover, since this is fed to the Ibots as soon as the trial is finished and the trial is the team's atomic action, this is immediate reinforcement learning. Equation 3 establishes that the payoff is positive if the $New^i(t)$s provided a better estimate of the integral than the $New^i(t-1)$s, negative if the estimate was worse, zero if it was equal (equally good or equally bad). Thus $R(t)$ decides the direction of change in the control programs. As an example[2], if $\alpha^i_{new}(t)$ is greater than $\alpha^i_{new}(t-1)$, $\alpha^i_{prog}(t+1)$ will increase or decrease with respect to $\alpha^i_{prog}(t)$ depending on whether $R(t)$ was positive or negative (Eq. 4). On the contrary, if $\alpha^i_{new}(t)$ is less than $\alpha^i_{new}(t-1)$, $\alpha^i_{prog}(t+1)$ will increase if $R(t)$ is negative, decrease if $R(t)$ is positive. In all cases the direction of change in the control programs is meant to increase the probability of those $Prog^i$s which proved to be better in the integral estimate process. Moreover, $Prog^i$s remain unchanged when $R = 0$. This learning method is a non-associative version of the basic, connectionist reinforcement learning algorithm proposed in [19]. Overall the learning algorithm is expected to guide the team towards admissible and stable control programs.

6 Experiments

On the "half-full" *Region* ($I = 0.49$) of Fig. 1(a), we have run repeated learning experiments with Ibots' teams of increasing size ($N_{ibots} = 1, \ldots, 14$), with public or private $Prog^i$s, and starting from clustered or scattered configurations. We have also considered the situation where the Ibots' skills are no longer homogeneous because of differences in sensing and acting capabilities. In all experiments we have set: $N_{max} = 100$ (the corresponding admissible error being 0.1), $\alpha_{max} = 180°$ and $\delta_{max} = 500$ (length of arena side), $\alpha_{step} = \alpha_{max}/10$ and $\delta_{step} = \delta_{max}/10$, $\rho_c = 0.1$, and, for each Ibot i, $Prog^i(0) = (\alpha^i_{prog}(0), \delta^i_{prog}(0)) = (0°, 0)$: this "staying in place" program was chosen *to bias* the Ibots towards "economic" programs, i.e. programs requiring small translations. To examine the form of the learnt programs, a learning experiment was stopped either after having obtained 10 consecutive admissible estimates, or after a fixed number of trials, depending on which of these two events occured first. The former stopping criterion was introduced for convenience, for not devoting too much time to experiments which did converge rapidly.

6.1 One Ibot

The single Ibot experiment is a point of reference for comparing results obtained with teams of Ibots. It requires to discover a pair $(\alpha_{prog}, \delta_{prog})$ which produces admissible and stable integral estimates. As the program space searched by the learning algorithm is bidimensional, one can explore it in a systematic way to test the quality of a significant number of programs. Thus, before starting the actual learning experiments, we have run background trials with all combinations of α_{prog} and δ_{prog}, with α_{prog} ranging in $\{0°, 10°, \ldots, 180°\}$, and δ_{prog} ranging in $\{0, 25, \ldots, 500\}$. Each combination program was tested on N_{trials} different trials to have a sample of integral estimates $\{\hat{I}_k, k = 1, \ldots, N_{trials}\}$. Then, for each program we computed the *mean of errors* $\mu_{prog}(E)$ and the *variance of errors* $\sigma^2_{prog}(E)$:

[2] All the following observations made on α^i_{new} hold for δ^i_{new} too, see Eq. 5.

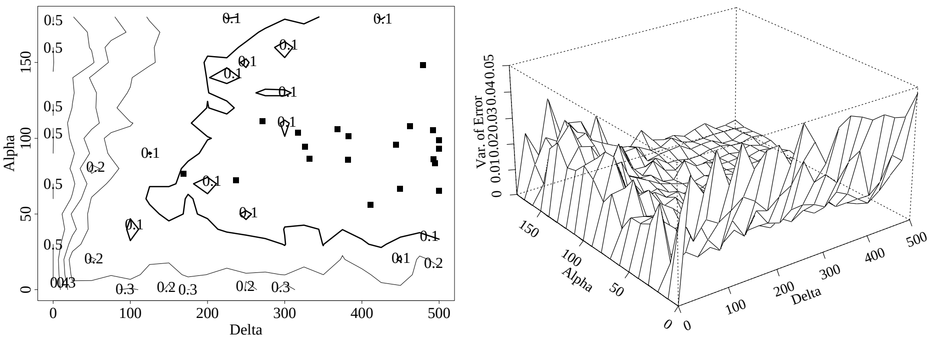

Fig. 2. One Ibot on the "half-full" *Region*. (Left) Contour plot of $\mu_{prog}(E)$ and convergence points for 20 learning experiments. (Right) Plot of $\sigma^2_{prog}(E)$.

$$\mu_{prog}(E) = \frac{1}{N_{trials}} \sum_k |I - \hat{I}_k| = \frac{1}{N_{trials}} \sum_k E_k$$

$$\sigma^2_{prog}(E) = \frac{1}{N_{trials}} \sum_k (E_k - \mu_{prog}(E))^2$$

$\mu_{prog}(E)$ and $\sigma^2_{prog}(E)$ describe the suitability of the corresponding program for the integration task: $\mu_{prog}(E)$ gives an indication on the accuracy of the estimates, while $\sigma^2_{prog}(E)$ measures their stability.

Figure 2 shows the plots in program space of $\mu_{prog}(E)$ (left) and of $\sigma^2_{prog}(E)$ (right): the program space axes are indexed by δ_{prog} and α_{prog}. On the $\mu_{prog}(E)$ plot we have highlighted the contour lines of level 0.1: these lines identify the space of admissible programs. Notice that these programs are also stable (see $\sigma^2_{prog}(E)$ plot). Most of them are distant from the "programmer solution" $(180°, 700)$: admissible and stable solutions start at $(50°, 200)$. Observe also that large values of α_{prog} require large values for δ_{prog} to be admissible, because small values of δ_{prog} would confine the Ibot's motion to a localized area (as an example, see the gray trace of Ibot 1 in Fig. 1 (c)). Finally, from the $\sigma^2_{prog}(E)$ plot we remark that not only admissible programs are stable. For example, all "staying in place" programs ($\delta_{prog} = 0$) have a very predictable performance. This makes the learning task more difficult as the Ibot's initial program $Prog(0) = (0°, 0)$ acts as a local minimum with respect to the stability criterion.

The results of 20 different learning experiments for the single Ibot have been overlaid on the left plot of Fig. 2. The convergence point of each experiment is indicated by a square. Learning stopped in all cases before the limit of 1000 trials. All experiments ended inside the space of admissible and stable programs.

For the sake of comparison, Fig. 3 shows the plots of $\mu_{prog}(E)$ (left) and of $\sigma^2_{prog}(E)$ (right) for one Ibot running on the "chessboard" *Region* ($I = 0.49$,

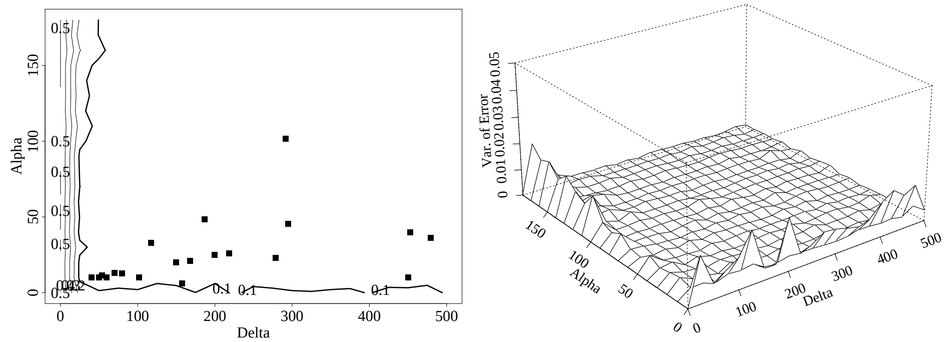

Fig. 3. One Ibot on the "chessboard" *Region*. (Left) Contour plot of $\mu_{prog}(E)$ and convergence points for 20 learning experiments. (Right) Plot of $\sigma^2_{prog}(E)$.

Fig. 1 (d)). The space of admissible programs is considerably larger than the one for the "half-full" *Region*, because the integral of the "chessboard" *Region* can be predicted even by a localized motion. Hence, the Ibot can afford travelling shorter distances to do a good job (Fig. 1 (d)).

6.2 Teams of Ibots

The form of admissible and stable programs completely changes for teams of Ibots.

Public Control Programs. The first case study we have addressed is that of Ibots equipped with public control programs. As in the single Ibot case, the program space is bidimensional, so we first ran background trials (with no learning) for teams of increasing size, both for the clustered and the scattered configuration. The results for the largest team of 14 Ibots are shown in Fig. 4. The left plot is the contour plot of $\mu_{prog}(E)$ for the clustered configuration, the right plot shows $\mu_{prog}(E)$ for the scattered configuration. On the left plot, the bold line delimits the space of admissible programs; for the scattered configuration, all programs result to be admissible (all contour lines are below level 0.1).

By comparing these results with those of Fig. 2 (left), one observes how the single Ibot's solution space "shrinks" or "expands" depending on whether the Ibots are started in the clustered or in the scattered way. Why?

First, consider the clustered configuration starting condition. As the Ibots begin a trial from the same position and with the same orientation, they have to disperse in the arena in order to explore it. Moreover, the number of samples they are allowed to take as individuals decreases as the team size increases, because the samples budget N_{max} remains the same whatever the team size. As a consequence, in a large team, each Ibot is granted fewer samples and fewer

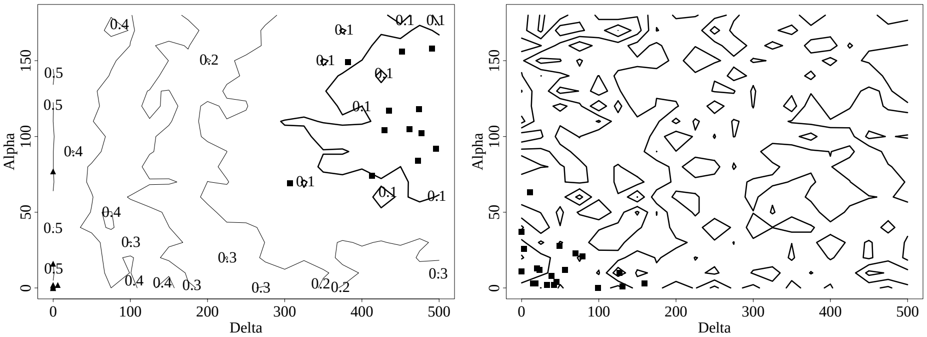

Fig. 4. A team of 14 Ibots with public control programs on the "half-full" *Region*. Contour plot of $\mu_{prog}(E)$ and convergence points for 20 learning experiments for clustered configurations (left) and for scattered configurations (right).

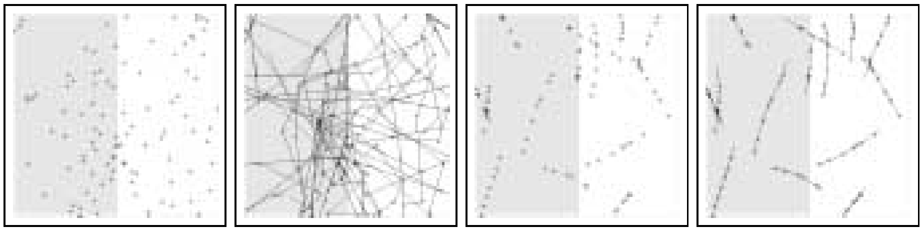

Fig. 5. A team of 14 Ibots running trials with learnt public control programs. (From left to right) (a) Samples and (b) trajectories for a team started in the clustered configuration and running the control program: $Prog = (100°, 450)$. (c) Samples and (d) trajectories for a team started in the scattered configuration and running the control program: $Prog = (10°, 50)$.

elementary movements to disperse in the arena. Given this constraint, the only way of achieving dispersion in few movements is through control programs with large variability both in translation and in rotation. *In conclusion, most of the solutions which are valid for the single Ibot would not work for this team.*

Second, consider the opposite case where the Ibots start a trial from scattered positions. As they are already uniformly distributed in the arena, any kind of motion program would lead to admissible estimates. In principle, the solution which would profit the most from this favorable start would be to perform a very localized motion around each Ibot's initial position. *We stress that this team solution would not be admissible for the single Ibot.*

Figure 4 also reports the convergence points of 20 learning experiments for both initial configuration types. Experiments which converged to admissible and stable programs within the time limit of 2000 trials are represented by squares,

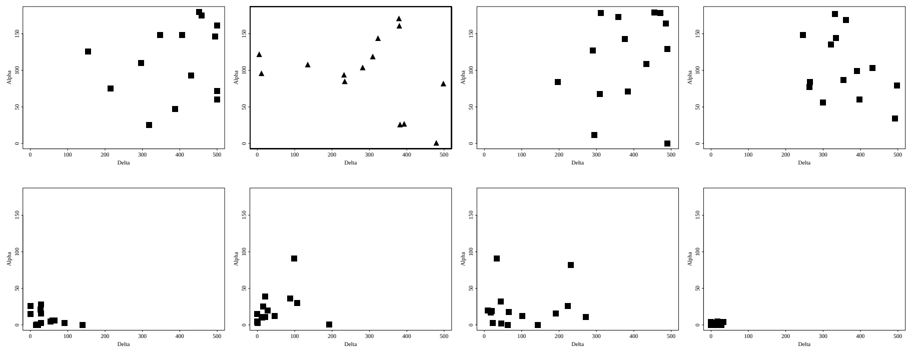

Fig. 6. A team of 14 Ibots with private control programs on the "half-full" *Region*. Convergence points for 4 learning experiments for clustered configurations (first row) and for scattered configurations (second row). For each experiment, the 14 private programs learnt by the team members are shown. In each plot, the vertical axis is indexed by α_{prog}, the horizontal axis by δ_{prog}.

while non-converging experiments are represented by triangles. Not surprisingly, all the experiments for the scattered configuration converged very rapidly (right). On the contrary, for the clustered configuration, not all experiments managed to converge to admissible programs within the predefined time limit (left). This is due to the fact that the Ibots initial program is $Prog^i(0) = (0°, 0)$, a bad but very stable program. As a matter of fact, the stability of this program becomes stronger as the number of Ibots grows. Therefore, it may take a considerable amount of time for the team to get away from this inconvenient initial program.

Finally, figure 5 shows trajectories and samples taken by a clustered ((a) and (b)) and a scattered team ((c) and (d)) of 14 Ibots running trials with learnt public programs.

Private Control Programs. Figure 6 shows a representative set of learning experiments performed with a team of 14 Ibots working with private control programs and started from clustered (plots on the first row) or scattered configurations (plots on the second row). Each plot shows the programs learnt by the team within the time limit of 20000 trials; programs which did not converge have been represented by triangles (second plot on the first row).

Essentialy, these solutions are similar to those obtained with public control programs. Within the same category of initial configuration, Ibots learn the same typology of programs: clustered Ibots need large variability in angle and translation, while scattered Ibots don't. This uniformity in the shape of the solutions is not surprising, because the integration task does not require differentiation in behavior as long as the Ibots have homogeneous skills.

From the point of view of learning, the main difference between dealing with a single public program or with many private programs is the robot credit assign-

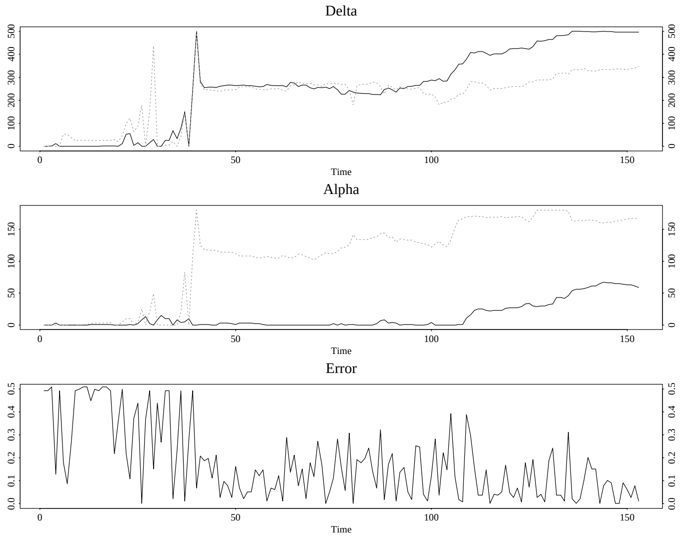

Fig. 7. Robot credit assignment problem for a team of two Ibots: Ibot 1 (dashed line) and Ibot 2 (solid line). Time evolution of δ^i_{prog}s (first plot), of α^i_{prog}s (second plot), and of the error E (third plot).

ment problem, which arises when the Ibots learn personal programs. Figure 7 illustrates the robot credit assignment problem for a team of two Ibots. The first and the second time plot present the evolution of the Ibots' δ^i_{prog}s and α^i_{prog}s parameters, respectively; the third time plot shows the error E in the integral estimate produced by the team. Observe that, around time 30, Ibot 1 (dashed line) has already acquired an admissible program ($Prog^1 = (49°, 437)$), but this does not appear at level of team performance because Ibot 2 (solid line) is still locked to the initial "staying in place" program. Consider also that Ibot 2's contribution to the team integral weighs more than Ibot 1's contribution, because Ibot 2 takes more samples: "bad Ibots count more". This is also the cause for the non-converging experiment of Fig. 6, where a minority of Ibots translating for short distances damage the team performance. To improve this state of affairs, Ibot 1 (Fig. 7), first backtracks from its admissible program, then relearns at a similar pace with Ibot 2.

Private Control Programs for Heterogeneous Ibots. As a last experiment, we wanted to make the learnt control programs more specialized. A way of achieving this is by differentiating the Ibots' individual skills. Figure 8 refers to a learning experiment with a team of two heterogeneous Ibots. Ibot 1 (dashed line) translates four times as fast as Ibot 2 (solid line). Moreover, Ibot 2 is *blind*: its ground color sensor reads "white" whatever its position in the arena.

The strategy discovered by the team to provide admissible and stable esti-

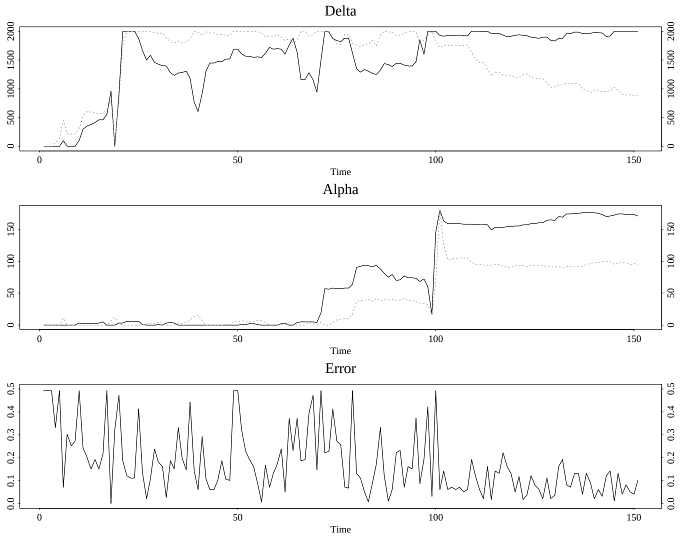

Fig. 8. A team of two heterogeneous Ibots: Ibot 1 (dashed line) moves 4 times faster than Ibot 2 (solid line), which is blind. Time evolution of δ^i_{prog}s (first plot), of α^i_{prog}s (second plot), and of the error E (third plot).

mates is clear from the δ^i_{prog}s and α^i_{prog}s plots of Fig. 8. The blind Ibot minimizes its catastrophic contribution to the team integral estimates by travelling long distances ($\delta^2_{prog} = 2000$, having set $\delta_{max} = 2000$ for this particular experiment). Observe that the error E stabilizes to low values only when the difference between δ^1_{prog} and δ^2_{prog} is sufficiently large. Still, Ibot 1 can afford a parameter of $\delta^2_{prog} = 800$ because it moves very fast.

Table 1 reports more programs learnt by this team in 5 repeated experiments. Ibot 1 always travels for shorter distances than Ibot 2. In all experiments, the balance between δ^1_{prog} and δ^2_{prog} is such that Ibot 1 consistently manages to collect at least 85 samples out of the 100 available to the team.

Finally, figure 9 shows the behavior of $\mu_{prog}(E)$ in program space of Ibot 1, when Ibot 2 works with a fixed control program. On the left plot, Ibot 2 is running $Prog^2 = (0°, 2000)$. This gives Ibot 1 the possibility of choosing its admissible program in a rather large set of programs: it can easily run up to $\delta^1_{prog} = 1000$ because its speed is four times the speed of the teammate. However, if the blind Ibot reduces its translations to $\delta^2_{prog} = 1000$ (right plot), the fast Ibot is forced to fit its control program to a reduced space of admissible programs.

7 Conclusions

The overall objective of the Ibots experiment was to understand how to use reinforcement learning to program automatically a team of robots with a common

δ^1_{prog}	α^1_{prog}	δ^2_{prog}	α^2_{prog}	$\mu(N^1_{sam})$	$\sigma(N^1_{sam})$
565	95	1528	179	85.5	1.5
876	95	2000	171	85.9	1.7
361	176	1549	2	88.9	1.3
537	165	1646	102	86.9	2.0
422	75	1061	66	84.1	0.5

Table 1. (Columns 1–4) Programs learnt by the heterogeneous team in 5 repeated experiments (one experiment per row). (Columns 5 and 6) Mean and standard deviation of the number of samples taken by Ibot 1 over 10 trials.

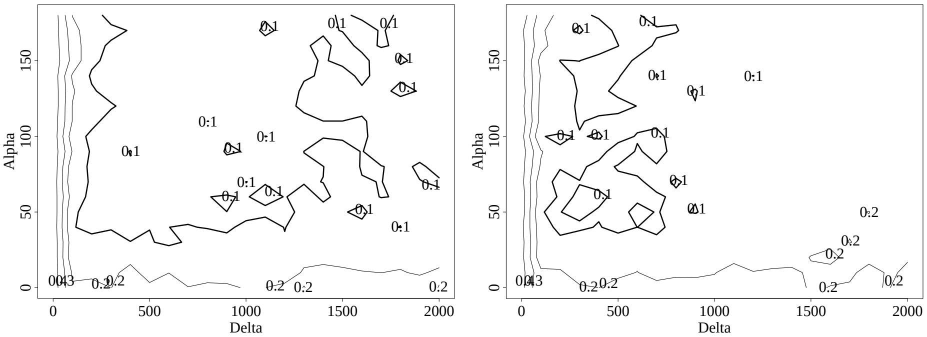

Fig. 9. Contour plots of $\mu_{prog}(E)$ in program space of Ibot 1 when Ibot's 2 program is: (Left) $Prog^2 = (0°, 2000)$ or (right) $Prog^2 = (0°, 1000)$.

mission. In addition, we wanted to derive *real team solutions*.

The "integration" mission of the Ibots is an artificial task for robots. However, the mission could also be interpreted as an exploration task, where the Ibots learn patterns of movement to reliably collect evidences about the region extension. Interestingly, it has been pointed to us [20] that the Ibots resemble networks of patrolling ants engaged in the task of monitoring events occurring throughout their territory [21].

The learning scenario for the Ibots is applicable to other missions because it relies on weak assumptions. A *team reinforcement signal* evaluates the behavior of the group as a whole; a single Ibot has no direct way of assessing its own performance, as distinct from the performance of its teammates. A *limited, common resource* constrains the Ibots, and there is no a priori rule to decide how this resource should be shared. When working with private control programs, the Ibots are *unaware* of teammate programs; during learning, each Ibot changes its own program independently, and has no information on how teammates are changing theirs.

As a general conclusion, experiments have demonstrated how different mission conditions require completely different control programs, and that a simple reinforcement learning procedure can find them. The key issue is: to optimize team performance instead of individual performance.

As far as the specific Ibots experiment is concerned, we cannot claim that the "general pattern" of the solutions discovered through learning were completely unexpected. However, as robot programmers, we have only a limited intuition for program parameters tailored to specific mission conditions (i.e. for a specific *Region*, for a given team size, for a specific team configuration, or for a particular set of robot skills). Sometimes, we are able to specify "a priori" programs which work for every possible mission condition (like those of Eq. 1); but, in certain mission contexts, these general solutions look unnatural. To write "ad hoc" programs for robots, a programmer will usually need to learn by trial-and-error himself: therefore, why not consider letting the robots do this, i.e learn by trial-and-error on their own [22]? *Second*: in general, a program which is admissible for a single Ibot is not ammissible for a team of Ibots, and viceversa. Thus, we cannot simply find a solution for one Ibot and clone it n times, n being the number of team members. The form of the solution to a problem changes as the number of "problem solvers" changes. Moreover, the robots become aware of this fact only if they are confronted with their performance as a team. On the contrary, a group of robots learning from individual payoffs would ignore opportunities which become evident only if the task is considered at a team level. *Third*, the space of admissible programs strongly depends on the number of Ibots involved in the mission and on their initial configuration in the arena. The admissibility space "shrinks" (and learning requires more time) when the Ibots are started in the clustered configuration and the team size grows. The increased difficulty is due to the fact that the number of samples is shared. On the contrary, the admissibility space "enlarges" (and learning requires less time) when the Ibots are started in the scattered configuration and the team size grows. The mission becomes easier in this case because, by initially distributing the Ibots at random in the arena, we bring them close to the problem solution; a team of individualistic robots would not be aware of this opportunity. *Fourth*, when the Ibots work with private control programs, the robot credit assignment problem arises, resulting in longer learning time. Interestingly, the robot credit assignment problem forces the Ibots to learn admissible programs at a similar pace, to prevent "slow" learners from jeopardizing the team mission. *Fifth*, the robot credit assignment problem vanishes when the Ibots learn a shared policy, and the learnt policy is still a real team solution. The possibility of learning a single public program instead of several private programs should be not overlooked in missions where specialization of the robot behavior is not required, because the time necessary for the team to learn a public program is much shorter. Finally, *sixth*, the Ibots with their heterogeneous acting and sensing capabilities manage to specialize their control programs so as to take advantage of their skills and to minimize the impact of their weaknesses.

Acknowledgements

Cristina Versino is supported by project No. 2129-042413.94/1 of the Fonds National de la Recherche Scientifique, Berne, Suisse.

References

1. A.G. Barto, R.S. Sutton, and C.J.C.H. Watkins. Learning and sequential decision making. Technical Report COINS-89-95, Dept. of Computer and Information Science, University of Massachusets, Amherst, 1989.
2. G. Weiß. Adaptation and learning in multi-agent systems: Some remarks and a bibliography. In G. Weiss and S. Sen, editors, *Adaptation and Learning in Multi-Agent Systems*, volume 1042, pages 1–21. Springer-Verlag, Lecture Notes in Artificial Intelligence, 1996.
3. M.J. Matarić, 1995. Personal Communication.
4. L.E. Parker. The effect of action recognition and robot awareness in cooperative robotic teams. In *IROS95, IEEE/RSJ International Conference on Intelligent Robots and Systems, Pittsburgh, PA, August*, volume 1, pages 212–219, 1995.
5. L.E. Parker. ALLIANCE: An architecture for fault tolerant, cooperative control of heterogeneous mobile robots. In *IROS94, IEEE/RSJ International Conference on Intelligent Robots and Systems, Munich, Germany, September*, pages 776–783, 1994.
6. M.J. Matarić, M. Nilsson, and K.T. Simsarian. Cooperative multi-robot box-pushing. In *IROS95, IEEE/RSJ International Conference on Intelligent Robots and Systems, Pittsburgh, PA, August*, 1995.
7. C.R. Kube and H. Zhang. Collective robotics: From social insects to robots. *Adaptive Behavior*, 2(2):189–218, 1994.
8. M.J. Matarić. Interaction and intelligent behavior. Technical Report AI-TR-1495, MIT Artificial Intelligence Lab, Boston, 1994.
9. M.J. Matarić. Learning in multi-robot systems. In G. Weiss and S. Sen, editors, *Adaptation and Learning in Multi-Agent Systems*, volume 1042, pages 152–163. Springer-Verlag, Lecture Notes in Artificial Intelligence, 1996.
10. M.J. Matarić. Learning to behave socially. In Meyer J.A. and S. Wilson, editors, *From Animals to Animats: International Conference on Simulation of Adaptive Behavior*, pages 453–462, Cambridge, MA, 1994. MIT Press.
11. A. Murciano and J. del R. Millán. Learning signaling behaviors and specialization in cooperative agents. *Adaptive Behavior*, 5(1):5–28, 1997.
12. R.H. Crites and A.G. Barto. Improving elevator performance using reinforcement learning. In D.S. Touretzky, M.C. Mozer, and M.E. Hasselmo, editors, *Advances in Neural Information Processing Systems 8*, pages 1017–1023, Cambridge MA, 1996. MIT Press.
13. J. Schmidhuber. A general method for incremental self-improvement and multi-agent learning in unrestricted environments. In X. Yao, editor, *Evolutionary Computation: Theory and Applications*. Scientific Publ. Co., Singapore, 1996.
14. C. Versino and L.M. Gambardella. Ibots: Learning real team solutions. In *ICMAS96 Workshop on Learning, Interaction and Organizations in Multiagent Environments*, Kyoto, Japan, December 1996.

15. L.E. Parker. L-ALLIANCE: A mechanism for adaptive action selection in heterogeneous multi-robot teams. Technical Report ORNL/TM-13000, Oak Ridge National Laboratory, Tennessee, November 1995.
16. J. C. Deneubourg, S. Goss, N. Franks, A. Sendova-Franks, C. Detrain, and L. Chrétien. The dynamics of collective sorting: Robot-like ants and ant-like robots. In Meyer J.A. and S. Wilson, editors, *From Animals to Animats: International Conference on Simulation of Adaptive Behavior*, pages 356–363. MIT Press, 1991.
17. J.M. Hammersley and D.C. Handscomb. *Monte Carlo Methods.* Methnen & Co., London, 1964.
18. A. Tucker. *Applied Combinatorics.* John Wiley & Sons, New York, 1995.
19. A.G. Barto, R.S. Sutton, and P.S. Brouwer. Associative search network: A reinforcement learning associative memory. *Biological Cybernetics*, 40:201–211, 1981.
20. A. Martinoli, 1996. Personal Communication.
21. F.R. Adler and D.M. Gordon. Information collection and spread by networks of patrolling ants. *The American Naturalist*, 140(3):373–400, 1992.
22. L.P. Kaelbling. *Learning in Embedded Systems.* MIT Press, Cambridge, MA, 1993.