Distributed Reinforcement Learning in Multi-agent Decision Systems

J. Ignacio Giráldez and Daniel Borrajo

Universidad Carlos III de Madrid
c/ Butarque, 15
28911 Leganés, Madrid, Spain
{giraldez,dborrajo}@ia.uc3m.es

Abstract. Decision problems can be usually solved using systems that implement different paradigms. These systems may be integrated into a single distributed system, with the expectation of obtaining a group performance more satisfactory than individual performances. Such a distributed system is what we call a Multi Agent Decision System (MADES), a special kind of Multi Agent System, that integrates several heterogeneous autonomous decision systems (agents). A MADES must produce a single solution proposal for the problem instance it faces, despite the fact that its decision making is distributed, and every agent produces solution proposals according to its local view and to its idiosyncrasy. We present a *distributed reinforcement algorithm* for learning how to combine the decisions the agents make in a distributed way, into a single group decision (solution proposal).

Topics: Multi Agent Systems, Machine Learning, Distributed Artificial Intelligence

1 Introduction

Two kinds of learning techniques are usually found in Multi Agent Systems (MAS): *local learning* and *distributed learning* [4]. *Local learning* is carried out by an agent on its own, without the need of the participation of other agents. This implies that the only available view of the world comes from the agent's standpoint. *Distributed Learning* is carried out as a result of the joint action of several agents of the MAS. This means that *distributed learning* can not be accomplished as a result of the isolated action of one agent. In *distributed learning* tasks, agents may need to observe other agents, or use knowledge facilitated by other agents [5]. In these situations, the agent's view of the world may be qualitatively different from the view of the world that the agent perceives in *local learning*. Every agent observes the world from a different standpoint, and interprets this view according to its own insight. The concurrence of the agents individual actions gives rise to a group behaviour. Learning appropriate group behaviours is the most common distributed learning task (e.g. [2], [4]).

Some complex decision problems can be solved by several different monolithic decision systems, based on different machine learning or problem solving

paradigms. When the quality of the results obtained separately by these systems is not satisfactory, they can be united in a *Multi Agent DEcision System (MADES)*, with the expectation of obtaining a joint performance superior to the performance obtained using the monolithic systems in an isolated way. A MADES is a Multi Agent System built for decision making, where a single decision is output by the system as a group, although internally many decisions may be made locally by the component agents [1]. These decisions may be incompatible, and even contradictory. But in spite of this, a single decision has to be made by the system to solve the problem at hand. How to incorporate these individual local decisions, into a single group decision, poses a group behaviour learning problem for the MADES. In this work, we propose a distributed reinforcement learning algorithm to solve this problem. A simplified version of this algorithm was implemented, tested, and experimental results reported in [1]. Here we present the formalization of the full algorithm, and discuss the full range of its potential. We also formalize the concept of Multi Agent Decision System (MADES).

Section 2 summarizes the IAO architecture[1]. Section 3 presents an ordered stepwise procedure for the solution of problems using the IAO distributed reinforcement learning algorithm in MADES. Section 4 discusses the two ways in which this algorithm can influence behaviors in MADES. Sections 5 and 6 reproduce some results obtained with a simplified and restricted version of the algorithm presented here. We conclude with section 7 where we discuss important topics.

2 The Intelligent Agents Organization

The Intelligent Agents Organization (IAO) is a Multi Agent Decision System architecture aimed at solving complex decision problems [1]. Figure 1 shows a high level view of this architecture:

- One agent, known as the *referee*, is in charge of the overall system control. It broadcasts problem instance descriptions (service requests), and control signals to the rest of the team. It then receives the respective replies from the rest of the agents. These replies may be either advice, or problem solving proposals. The relationship among the referee and the rest of the agents can be regarded as a client-server relationship. The services the referee may request to an agent are either the solution proposal synthesis (only to worker agents), or an advice request (only to advisor agents). These service requests are scheduled in a way that maximizes parallelism (every agent runs on a different machine), so the MAS response time is minimized.
- A *worker agent* receives problem descriptions from either the referee, or another worker, and replies with solution proposals. Worker agents work in parallel on a solution proposal to the same problem instance, and are capable of autonomous decision making. Any of them could be the basis

[1] a more complete description of it can be found in [1]

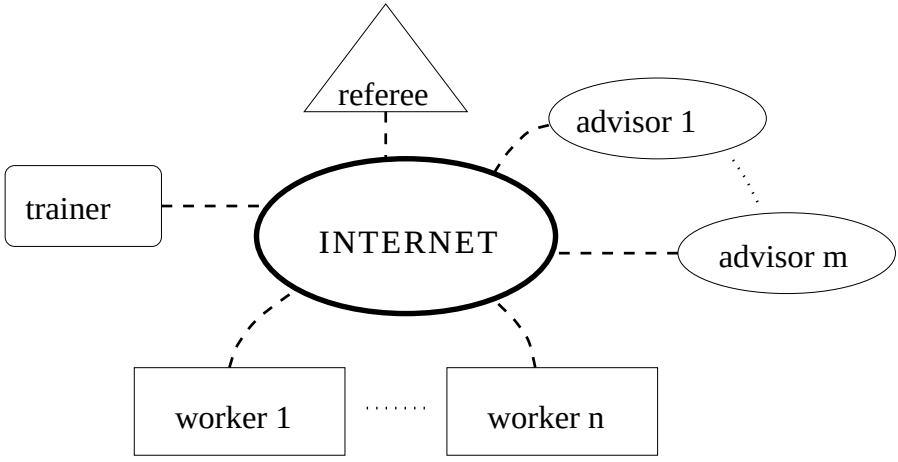

Fig. 1. A schematic view of the IAO Architecture.

of a monolithic system aimed to solve each problem. The only restriction imposed on workers is that they should adhere to the client-server protocol. Other than that, there is a total freedom for the system designer to use any agent he/she wishes as a worker. Due to this flexibility, the IAO model is widely applicable.

- Several agents play the role of *advisors*. They are contacted by, either the referee or a worker agent, and receive a problem instance as input. They reply to the requester with the identification of the worker agent that they expect to be the most competent of the group, in the solution of the aforementioned problem instance. The referee will use this advice, when one of the proposals the worker agents provide has to be selected. A worker agent may use this advice to ask the worker, that the advisor indicated, for its cooperation in the solution of the problem instance.
- A *trainer* agent produces problem instances that are used for training and testing. Problem generation can be made either randomly, or using an "ad hoc" scheme. The criteria for problem synthesis affects the success of the learning effort, as it is widely known.

3 The IAO Distributed Reinforcement Learning Algorithm

The purpose of this algorithm is to enable the agents to improve their collective behaviour themselves. It is applicable to any MADES that follow the role differentiation specified by the IAO model. Next, we describe the steps to be followed for problem solving with the IAO model.

3.1 Problem Modeling

The domain is modeled as a problem space:

$$World = (S_w, A)$$

where

- S_w is the state space, whose representation depends on the problem at hand.
- A is the action set, whose elements are all the possible actions that can be executed on states.

A *Multi Agent Decision System* is modeled as

$$MADES = (S_M, i, A_M, A_r, A_w, A_a, A_t, g)$$

where

- S_M is the set of states of the world as the MADES perceives them.
- i is the input function, that translates from the external to the internal state representation,

  $$i : S_W \rightarrow S_M$$

  In decision problems, i is usually the unit function, so S_M is usually equal to S_W.
- A_M is the set of actions that the MADES can execute on the world, which is usually equal to A.
- A_r is a single element set formed by an autonomous agent that controls the ordered execution of the system's decision making algorithm, and is known as the referee.
- A_w is the set of workers, autonomous agents that receive problem descriptions and reply with a proposal of the decision to be made to solve the problem. We use w to denote the number of workers in the MADES.
- A_a is the set of advisors, autonomous agents whose mission is to observe the worker agents, to learn their competencies, and to inform other agents about their findings.
- A_t is the set of trainer agents, whose mission is to sinthesize problem instances for the MADES to train on.
- g is the group behaviour, a control policy that specifies how to produce a single group decision from the agents decision proposals A_g^w,

  $$g : S_M \times A_g^w \rightarrow A_M$$

  An autonomous agent is modeled as

$$agent = (S_g, A_g, i_g, C_g, S'_g, S_I, b)$$

where

- S_g is the set of the possible inputs the agent may receive.

- A_g is the set of the actions the agent can execute. In MADES, the typical actions are decision making and adaption.
- i_g is the agent's input function,

$$i_g : S_M \rightarrow S_g$$

- C_g is the set of the communication's acts that the agent can perform. In MADES, they are usually either the request of a decision making service to another agent (i.e. a consultation), or the reply to a decision request with the results.
- ${S'}_g$ is the set of all the data the agent receives from another agents (e.g. a classification petition).
- S_I is the set of the agent's internal states.
- b is the agent's behaviour function,

$$b : S_g \times {S'}_g \times S_I \rightarrow A_g$$

3.2 Partitioning of the State Space

If an advisor based on reinforcement learning is used for the estimation of which worker (say $worker_i$) is the most competent of the group for the solution of a given problem instance ($problem_i$), then you need to store a huge reinforcement table of the form $(problem_i, worker_i)$. The storage of this table is unaffordable in sufficiently complex problems. If all the problem instances with some qualitative similitude, were handled best by the same worker, then we might build reinforcement tables associating the appropriate worker to the set containing this collection of similar problem instances. Now the reinforcement table has much fewer entries, and it is of the following form $(set_i, worker_i)$. In order to obtain a set description of the state space we partition it as it is explained next.

The state space S is partitioned in s subsets: $S = \bigcup_{i=1}^{s} S_i$ such that $S_i \bigcap S_j = \emptyset$ when $i \neq j$. Similar problem instances should be contained within the same subset, because the algorithm works assuming the following hypothesis: if one worker is more competent for the solution of a given problem than the rest of the workers, then it will also be more competent than its peers in the solution of problem instances similar to the aforementioned (i.e. in the solution of problem instances that lie in the same subset of the partition). Unsupervised classification methods may be used for this task. The problem lies in finding and adequate distance function in S: a good function requires that the person that builds it have deep knowledge about the problem domain, and about which are the most predictive features in the problem representation.
The indexed characteristic function locates the subset of the partition that contains a given problem instance:

$$c : S \rightarrow N / x \epsilon S, c(x) = i \Rightarrow x \epsilon S_i$$

A cumulative reinforcement table $T_i = [r_1 \ldots r_w]$, where w is the number of workers, is assigned to every S_i. The meaning of the ith entry of T_i is the cumulative reinforcement received by the ith worker as a result of its past decisions. The reinforcement tables T_i, form the rows of the cumulative reinforcement matrix

$$M = \begin{bmatrix} r_{1,1} \ldots r_{1,w} \\ \vdots \quad\quad \vdots \\ r_{s,1} \ldots r_{s,w} \end{bmatrix}$$

where $r_{i,j}$ is the cumulative reinforcement received by agent j, as a result of its intervention in the solution of problems that belong to S_i.

3.3 Problem Solving Trace

Some problems require that the decision system make a series of decisions before the problem can be considered as solved (as in robotics problems or games). In these problems, the "world" evolves following a series of states,

$$Trace = [s_1 \ldots s_n]$$

as a result of a series of actions/decisions taken by the MADES,

$$SystemTrace = [D_1 \ldots D_n]$$

The MADES makes decision D_i based on the local decisions made by the agents at time i: $d_{i,1} \ldots d_{i,n_a}$ where n_a is the number of autonomous agents in the MADES.

3.4 Distributed Credit Assignment

When the outcome of the problem solving episode is finally known, the decision system is reinforced according to the desirability of this outcome. Since the decisions were made by a MADES, in a distributed fashion, the credit assigned will also have to be distributed among the agents that participated in the decision. Let us consider that the world is in the state s_i, and the agents propose decisions/actions $d_{i,1} \ldots d_{i,n_a}$, and the resulting group's decision is D_i. The distributed credit assignment algorithm distributes credit in a twofold way: on the one hand, credit is distributed in time (because early distant states influence less the outcome than recent states), on the other hand, credit is also distributed "spatially" among the agents that take part in the distributed decision making procedure, according to their opposition/support to the decision that the MADES finally produced. Thus, the distributed credit assignment function depends on the iteration i, the desirability of the final outcome o, and the participation of the agents that compose the MADES (which is represented by the local assesments/decisions they forward $d_{i,1} \ldots d_{i,n_a}$),

$$dca : N \times \Re \times A_g \ldots A_g \rightarrow \Re^w$$

the output of the distributed credit assignment function dca is used as the reinforcement vector at time step i,

$$\boldsymbol{RV}(i) = dca(i, o, d_{i,1} \ldots d_{i,n_a})$$

so we get a series of reinforcement vectors, $\boldsymbol{RV}(1) \ldots \boldsymbol{RV}(n)$.

3.5 Reinforcement Tables Update

The cumulative reinforcement tables $T_j, j = 1 \ldots s$, are updated with the proper reinforcement vectors:

$$T_{c(s_i)} = T_{c(s_i)} + \boldsymbol{RV}(i)$$

4 The Use of Distributed Reinforcement Learning in a MADES

The distributed reinforcement learning algorithm we present can improve the group's behaviour in two ways:

1. It is a way to learn the workers competencies. This means that by comparing the cumulative rewards of the workers for a given problem, the advisors can determine which worker is expected to handle that problem best. This results in a group behaviour's improvement.
2. Workers can evolve in a way that maximizes their future rewards, which produces a "local" adaptation.

5 An Instance of a MADES

We have chosen checkers endgames as the problem domain for our experiments. These endgames contain 8 pieces at most, and the majority pose a considerable difficulty for human players [3]. We have solved the problem following the steps outlined in section 3. The details are shown next.

The world is modeled as

$$World = (S, A)$$

where

- S is composed by all the legal checkers situations with eight pieces at most. We use the following notation, $s_j = (c_1 \ldots c_{32})$ such that $c_i \epsilon \{w, p, b, m, e\}$ where w denotes a white man, p a white king, b a black man, m a black king and e an empty space.
- A is composed of a single type of action, the move of a piece from one box of the board to another, observing the rules of checkers. We denote it like this: $mov(x_1, y_1, x_2, y_2, Capture, CaptureList)$ which represents an action that moves the piece at location (x_1, y_1) to (x_2, y_2), removing from the board all pieces whose locations are specified in $CaptureList$ when $Capture$ is bounded to yes.

The *Multi Agent Decision System* is modeled as

$$\begin{aligned} MADES = (&S, 1, \{observe, move\}, \{Ref\}, \\ &\{alphaAG, bayesAG, backpropAG, c4.5AG, hybridAG\}, \\ &\{reinf, rote\}, \{ctrainer\}, g) \end{aligned}$$

where

- Since the input function is the unit function, no preprocessing is done upon the observed state.
- $Ref = (S, \{move\}, 1, \{ask, listen\}, \{A_w \bigcup S^w\}, \{\sigma_i, i = 1 \ldots 6\}, b_{ref})$ which means that the input the referee receives is the aforementioned set of the legal checkers situations (S). The action it can execute is a legal move on the checkers board. Again, the input function is the unit function. The only communication acts it can perform are asking for something to another agent, and listening to its reply. The internal σ_i states are described in the following control algorithm that specifies the behaviour b_{ref} of the referee. The referee is at internal state σ_i when the step i of the algorithm is being executed.
  1. get current state description (the current board's situation)
  2. broadcast it to the rest of the agents
  3. receive their replies (the move they would execute on that situation)
  4. choose one of the proposals the agents forwarded
  5. execute that action (i.e. make a move on the board)
  6. if a final state has not yet been reached, then repeat control cycle
- $A_W = \{alphaAG, bayesAG, backpropAG, c4.5AG, hybridAG\}$ is the set of workers. *alphaAG* is a simple alpha-beta searcher. *hybridAG* is an alpha-beta searcher, that consults the *reinf* advisor at leaf nodes, and receives from it the identification of the worker that is expected to solve best that situation. Then, *hybridAG* contacts that worker and requests its collaboration to evaluate the node. If that collaboration is not possible, the node is evaluated locally. The rest of the workers are classifiers that make decisions based on what they have learnt during their training. The specification of all the workers is quite similar. The input set is the same for all the workers: the set of legal checkers situations. The input function is different for every worker, because they do not share internal representations. The behavior function provides a proposal according to decision making formalism of the agent (feed-forward neural network, decision tree or whatever).
- $A_a = \{reinf, rote\}$ The *reinf* advisor learns the competencies of the workers, and represents them in reinforcement tables. The reinforcement a worker obtains for the solution of a certain class of problems is used by other agents as an indication of how good the worker is at the task. The *rote* advisor keeps track of who is the worker that best solves a given problem by means of rote learning. Since it cannot generalize, it is not useful in problems with huge state spaces like this one, so we discontinued its use after some testing.
- $A_a = \{asses, reply, ask, listen\}$ An agent can produce a decision as result of an assesment (compute next move), can communicate the decision back to the requester, or it may ask another agent for some service (decide on which of the workers is the competent to solve this checkers situation, or recommend a move to perform next, or evaluate a checkers situation)
- $A_t = \{trainer\}$ is a special agent built to produce checkers problems of tunable difficulty.
- The group behaviour is the result of a voting mechanism. Every worker votes for a move (the workers supported by the advisors get extra votes), and the winning move is the one the MADES outputs.

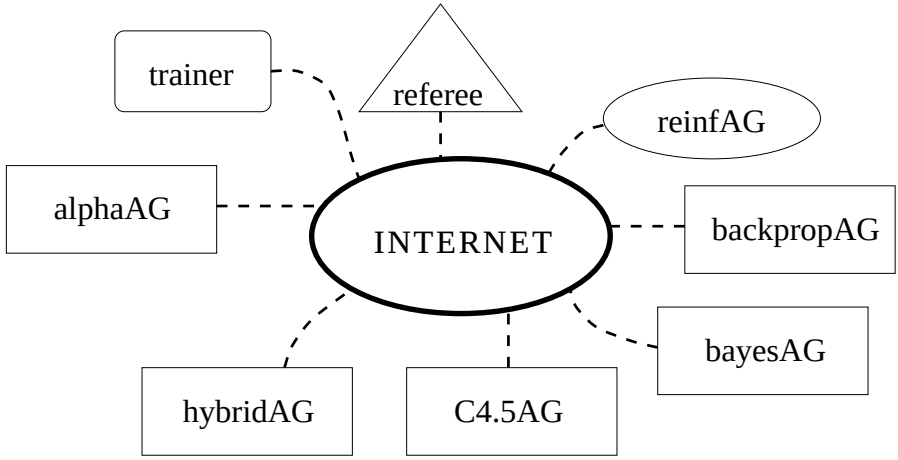

Fig. 2. Architecture of our Multi Agent DEcision System.

5.1 Partitioning of the State Space

The space of legal checkers situations is partitioned according to the following criteria:

- number of white men
- number of black men
- number of white kings
- number of black kings
- existence of capture opportunities for white
- existence of capture opportunities for black
- existence of crowning opportunities for white
- existence of crowning opportunities for black

The indexed characteristic function $c(x)$ counts these 8 indexes to locate the subset of the partition where x lies.

As an example, we show the cumulative reinforcement table for the subset whose indexes are (1, 0, 2, 1, yes, no, no, yes):

$$T_{(1,0,2,1,yes,no,no,yes)} = \\ [backprop(0.3333), bayes(-0.2053), c4\_5(-0.0625), alpha(-0.5357), hybrid(0)]$$

If the *reinf* advisor were consulted about who is the expected most competent worker, to make decisions when the current state belongs to the aforementioned subset, the advisor would reply with the *backpropAG* identifier, since this is the worker with highest cumulative reward.

5.2 Problem Solving Trace

A trace is kept of the series of successive checkers situations, along with the identification of the workers that supported the decision (action) that leads to the next state. As an example a part of a sample trace is shown:

```
credit(proposal(e, m, e, e, e, e, e, e, b, e, p, e, e, e, w, e, e,
e, e, e, e, e, w, e, e, e, p, b, e, e, e, e), [alpha]).
credit(proposal(e, m, e, e, e, e, e, e, b, w, p, e, e, e, e, e, e,
e, e, e, e, e, w, e, e, e, p, e, e, e, e, m), [c4_5, alpha]).
credit(proposal(e, m, e, e, e, e, e, e, e, w, e, e, e, b, p, e, e,
e, e, e, e, e, w, e, e, e, p, e, e, e, e, m), [c4_5, bayes]).
credit(proposal(e, m, e, e, e, e, e, e, e, w, e, e, e, e, p, e, b,
e, e, e, e, e, w, p, e, e, e, e, e, e, e, m), [alpha]).
credit(proposal(e, m, e, e, e, e, e, e, e, w, e, e, e, e, p, e, e,
e, w, e, b, e, e, p, e, e, e, e, e, e, e, m), [backprop]).
```

5.3 Distributed Credit Assignment

The aforementioned problem solving trace, is used to determine which workers agree with the main variation at every node of the game tree. The tree is traversed from leaves to root, assigning credit to the workers at every node. At a node at depth $CurrentDepth$, the workers receive the following reinforcement:

$$Reinf(worker) = Result \cdot Agree \cdot \frac{CurrentDepth}{NMoves}$$

where $NMoves$ is the total amount of moves executed by the MADES in the endgame. $Result$ is equal to $+1$ if the MADES won the game, and -1 otherwise. $Agree$ is equal to $+1$ if $worker$ agreed with the main variation, and 0 otherwise (no credit assigned). The discounting mechanism is implemented in the $CurrentDepth$ variable, agents are assigned less credit in the final outcome when their decisions are made at shallow nodes. This formula computes the entries of the reinforcement vector. The indexed characteristic function is used to locate the subset the current node belongs to. Then the reinforcement table associated to this subset is fetched, and updated with the reinforcement vector. This algorithm is repeated for every node along the main variation.

6 Results

The MADES played test games against every one of its workers, with the following results:

opponent	MADES advantage
c4.5AG	21%
backpropAG	17.5%
bayesAG	17%
hybridAG	2%
alphaAG	2%

To explain how the MADES advantage percentage is computed, let's use the test against *alphaAG* as a reference. A 2% advantage of the MADES over *alphaAG* means that the MADES wins 4% more test endgames than it loses. So, its real advantage is 4% / 2 = 2%, because if the opponent (alphaAG) won 2% more endgames its score would be incremented in 2% and the MADES' score would be diminished by 2%, so both would be even. This can be mathematically expressed by this formula:

$$s = \frac{[w(G) - l(G)] \times 100}{2 \times G}$$

where s is the score percentage, G is the number of played games, $w()$ is the number of won games, and $l()$ is the number of lost games.

These results show that the MADES beats any of its members, and this justifies integrating the workers into the MADES. They have been obtained taking advantage of only one of the adaptation opportunities that the distributed reinforcement learning algorithm provides: reinforcement tables have been used for competencies' learning, but have not been used for the workers' "local" adaptation (which is now underway).

Some previous results are also worth mentioning. The MADES' score has evolved from -20% (when it had only played 300 training games) to +2.03% (after 10832 training games). This improvement has been mainly due to the learning of the workers competencies. But to a lesser extent it has been due to worker replacements. The flexibility of the IAO model allows the replacement of a worker by another, and the adaption of the rest of the system to the new MADES composition, thanks to the adaptive behaviour of the advisors. We replaced two workers during the MADES' lifetime, the first replacement improved the MADES' score 1.7 points, and the second 1.17 points.

7 Discussion

Distributed reinforcement learning can play a double role in Multi Agent Decision Systems. On the one hand, control information can be learnt in the form of a competencies map, that is a map of the state space where it is specified which worker is expected to handle best every kind of problem instance. Since it may be known which worker is the most trustworthy for the solution of a problem instance, the distributed decision making procedure takes this worker's proposal with a special consideration. So the group's decision making procedure is adapted following the predictions of the competencies map.

On the other hand, the cumulative reinforcement the worker agents receive provides an indication that can be used for local adaption, which will eventually be noticeable in the group's behaviour, and in how the advisors characterize them. The use of a different reinforcement table for every subset of the partition, provides an indication of how the adaption should be directed in a more detailed way than in other reinforcement learning proposals.

References

1. J. I. Giráldez and D. Borrajo. A distributed model for the combination of heterogeneous knowledge based agents. In *J. Hunt, R. Milnes (eds.) "Research and Development in Expert Systems XIV",* December 1997. Proc. of ES97, ppl75-184, SGES Publications. 149
2. Thomas Haynes and Sandip Sen. Evolving behaviorial strategies in predators and prey. In *G. Weiss, S. Sen (eds.), Adaption and Learning in Multi Agent Systems, Proc. of the IJCAI 95 Workshop, Springer,* 1996. 148
3. Jonathan Schaeffer, Robert Lake, Paul Lu, and Martin Bryant. Chinook, the world man-machine checkers champion. *AI Magazine,* 17(1):21-29, Spring 1996. 154
4. G. Weiss. Learning to coordinate actions in multi-agent systems, august 1993. Proc. IJCAI, Chambery, France, volume 1, pp311-316. 148
5. G. Weiss. Some studies in distributed machine learning and organizational design. Technical Report FKI-189-94, Institut für Informatik, Technische Universität München,1994. 148