Learning on BDI Multiagent Systems

Alejandro Guerra Hernández, Amal El Fallah-Seghrouchni, and Henry Soldano

Université Paris 13, Laboratoire d'Informatique de Paris Nord, UMR 7030 CNRS,
Institut Galilée, Av. Jean-Baptiste Clément, Villeteanuese 93430, France.
{agh,elfallah,soldano}@lipn.univ-paris13.fr

Abstract. We describe the design of BDI learning agents, using first-order induction of logical decision trees to learn when their plans are executable (plans' context). This design enable BDI agents to learn communicating in a Multiagent System (MAS). Learning in MAS has been motivated by the interest of MAS community on Machine Learning (ML) techniques to deal with the complexity inherent to such systems, and the possibility for the ML community to improve the understanding of learning principles, thanks to the extended view of ML dealing with MAS. Our research goes on these two directions considering i) the way a BDI agent should be designed; and ii) the way a BDI agent can build a set for inductive learning, and iii) the way a BDI agent can learn in a MAS.
Key words: Multiagent Systems, BDI agents, inductive learning.

1 Introduction

We are interested in MAS composed by intentional learning agents, e.g. BDI learning agents. We describe the design of a BDI architecture extended with learning competences considering that i) the behavior of BDI agents is explained in terms of propositional attitudes, e.g. beliefs, desires, and intentions, as proposed in practical rationality theories [2]; and ii) such agents are characterized as autonomous, reactive, proactive, and social systems [16].

Learning in MAS has been characterized as the intersection of MAS and ML [15]. Motivations for this are reciprocal: i) MAS community is interested on learning because it seems to be a way to deal with the complexity inherent to MAS; and ii) ML community considered that dealing with MAS will improve our understanding of learning principles. Our research goes on these two directions as explained here.

A *learning agent* [13] can be conceptually divided into four components: i) a learning component responsible for making improvements executing a learning algorithm; ii) a performance component responsible of taking actions; iii) a critic responsible for providing feedback; and iv) a problem generator responsible for suggesting actions that will lead to informative experiences.

The design of the *learning component*, and consequently the choice of a particular learning method, is usually affected by five major issues: i) which elements of the performance component are to be improved? ii) what representation is used for these elements? iii) what feedback is available? iv) what prior informations is available? and v) is it a centralized or decentralized learning case?

In the case of agents, their characterization as autonomous, reactive, proactive, and social systems, should be considered while answering these questions, and particularly when deciding what, when and how, the agent will learn.

In this paper we show how a BDI architecture inspired in dMARS [8] can be used to conceive intentional learning agents, able to learn in a MAS. Organization of the paper is as follows: Section 1 introduces the BDI terminology necessary to explain our approach. Section 2 describes the design of the BDI learning architecture, including the learning method –induction of first-order decision trees. Section 3 describes different settings for learning in a MAS using our architecture. Section 4 opens discussion.

2 BDI agency

BDI theories of agency are well known. Different aspects of intentionality and practical reasoning have been formally studied (for a roadmap see [14, 16]). This section sketches a BDI architecture inspired on dMARS [8], using a very simple scenario proposed by Charniak and McDermott [4]. This scenario (Fig. 1) is composed by a robot with two hands, situated in an environment where there are a board, a sander, a paint sprayer, and a vise. Different goals can be proposed to the robot, i.e. sand the board, or even get self painted! which introduces the case of incompatible goals, since once painted the robot is not operational (its state change from ok to painted) for a while. The robot has different options, e.g. plans, to achieve its goals. It is possible to introduce other robots in the environment to experiment social interactions. This scenario will be used in the exemples of the rest of the paper.

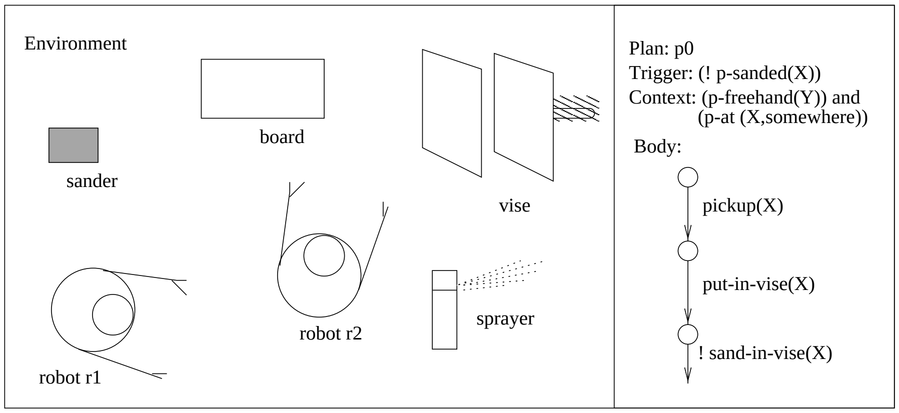

Fig. 1. An scenario for examples and a typical plan.

In general, a *BDI architecture* contains four key data structures: beliefs, desires or goals, intentions, and a plan library:

Beliefs represent information about the world. Each belief is represented as a ground literal of first-order logic. Two activities of the agent update its beliefs: i) the perception of the environment, and ii) the execution of intentions. The scenario show in Fig. 1 can be represented with the following beliefs: (p-at (sander,there)), (p-at (board,there)), (p-at (sprayer,there)), (p-state (r1,ok)), (p-handfree (r1,left)), (p-handfree (r1,right)).

Desires, or goals, correspond to the tasks allocated to the agent and are usually considered as logically consistent among them. Two kinds of desires are usually adopted i) to achieve a desire, expressed as a belief formula (a literal not necessarily grounded), i.e. `(! p-sand (board))`; and ii) to test a situation expressed as a disjunction and/or conjunction of belief formulae, i.e. `(? (AND (p-state (r1,ok)) (p-freehand (r1,X))))`.

Our architecture uses an *event queue* to process perceptions. Events are of four kinds: the acquisition or removal of a belief, i.e. `(add-bel (p-sand (board)))`; the reception of a message, `(told r2 (! (p-sand (board))))`; and the acquisition of a new (sub)goal.

Plans have several components. The invocation condition is a trigger event specifying the event a plan is supposed to deal with. The context specifies, as a situation formulae, the circumstances under which a plan execution may start. The body represents possible courses of action. It is a tree which nodes are labeled as states and arcs with actions or subgoals. The maintenance conditions describe the circumstances that must remain to continue the execution of the plan. Finally, a set of internal actions is specified for the cases of success and failure of the plan. Fig. 1 shows a simplified plan to sand an object in the vise, when the robot has a hand free, and the object is somwhere there. The last arc of the plan body is a subgoal that will be posted to the event queue, when executing the plan. Then other plans to deal with the event `(! (p-sandinvise(?x)))` will be considered, and so on.

An *intention* is implemented as a stack of plan instances. In response to an event, the agent must find a plan instance to deal with it. Two cases are possible: i) If the event considered is an external one, and emtpy stack is created and the plan instance selected is pushed on it, i.e. the event `(! (p-sand(board)))` triggers the plan `p0`, possibly among others. The substitution (board/X, left/Y) makes `p0` executable. So this substitution and `p0` form a new intention stack; ii) If the trigger event is an internal one, then the plan instance is pushed on an existing stack, i.e. the plan instance selected for the event `(! (p-sandinvise(?x)))` will be pushed on the stack genereated by `p0`.

A *BDI interpreter* implemented in Lisp, manipulates these structures, selecting appropriate plans based on beliefs and desires, structuring plans as intentions, and executing them.

3 BDI learning agents

The performance component of our BDI learning agents is the interpreter described in the previous section. All the elements of the interpreter are represented using first-order logic, and this discards learning methods using propositional representations. The feedback component is provided by the interpreter too, detecting failure and success in the execution of intentions. The problem generator has to collect examples of these executions.

Obviously, the elements of the performance component to improve are plans. The relationship between planing and learning has been studied [5] and also the relevance of the order in which plans are executed [3]. Following these works, we decided to extend the BDI architecture, enabling it to learn about the context of its plans, e.g. when plans are executable. Since plans are transformed in plan instances to be executed, context determine the way they are pushed in the structure of intentions. If an agent can learn such a context, there is an impact on reactivity (plans are executed under conditions learned from the environment) and autonomy (the agent itself determining such conditions). Context is represented as a conjunction and/or disjunction of belief formulae, so we decided to use decision trees as the target language for the learning method. Another consideration was that belief formulae are expressed in first-order logic.

3.1 First-Order Induction of Logical Decision Trees

Decision trees learning is a widely used and very successful method for inductive inference. As introduced in the ID3 algorithm [11] it approximates discrete value-target functions. Learned functions are represented as trees and instances as a fixed set of attribute-value pairs. Trees represent in general, a disjunction of conjunctions of constrains on the attribute values of the instances. Each path from the tree root to a leaf corresponds to a conjunction of attribute tests, and the tree itself is the disjunction of these conjunctions. Decision trees are inferred by growing them the root downward, greedily selecting the next best attribute for each new decision branch added to the tree.

A first-order representation for decision trees, known as *learning from interpretations* paradigm [12], is defined as follows. Given i) a set of classes C; ii) a set of classified examples E, where each example $e \in E$ is a set of facts; and iii) a background theory B. Find a hypothesis H, s.t. $\forall e \in E, H \wedge e \wedge B \models c$ and $H \wedge e \wedge B \not\models c'$, where c is the class of the example e, and $c' \in C - \{c\}$. Examples are partial interpretations that are completed taking the minimal Herbrand model of each example and the background theory B.

Tilde [12] is a learning from interpretations system, operating on logical decision trees. It uses the same heuristics that C4.5, a successor of ID3 (gain-ratio and post-pruning heuristics), but computations of the tests are based on the classical refinement operator under Θ-subsumption. We are using Tilde version 5.5.1, our BDI interpreter configure the learning set, as illustrated in the following subsection.

3.2 An exemple of our approach

Suppose that the agent in the scenario proposed in Fig. 1 is identified as `r1` and its event queue contains this trigger event: `(! (p-sand (board)))`, so that `r1` selects the plan **p0** as described above. Eventually, the execution of the intention composed by **p0** leads to a success or failure situation. Then, the agent can build models as these:

```
begin(model(1)).      begin(model(2)).      begin(model(3)).
success.              success.              failure.
p_state(r1,ok).       p_state(r1,ok).       p_state(r1,painted).
p_handfree(r1,left).  p_handfree(r1,right). p_handfree(r1,left).
p_at(board,there).    p_at(board,there).    p_handfree(r1,right).
plan(r1,p0).          plan(r1,p0).          p_at(board,there).
end(model(1)).        end(model(2)).        plan(r1,p0).
                                            end(model(3)).

begin(model(4)).      begin(model(5)).      begin(model(6))
failure.              success.              success.
p_state(r1,painted).  p_state(r1,ok).       p_state(r1,ok).
p_handfree(r1,right). p_handfree(r1,left).  p_handfree(r1,left).
p_at(board,there).    p_at(board,there).    p_at(board,there).
p_at(sander,vise).    plan(r1,p0).          plan(r1,p0).
plan(r1,p0).          end(model(5)).        end(model(6))
end(model(4)).
```

Models are labeled by the success or failure of the plan execution, in this case the plan **p0** of agent **r1**. The rest of the models are beliefs the agent had when executing **p0**. Models are recovered from a log file keeping the last 30 pairs beliefs - plan executed. We will test to delay the learning process attending for *n* executions after the first failure of the plan. Models are stored in a file with the name of the agent and the extension kb, for knowledge base, i.e. r1.kb

The agent can create a background theory file (r1.bg), containing information about the plan executed, i.e. the current context of the plan.

```
plan_context(Agent,Hand,Object) :- p_handfree(Agent,Hand),
                                   p_at(Object,there).
```

Finally, a language bias file (r1.s):

```
output_options([c45,lp]).
classes([succes,failure]).

rmode(1: p_state(-A,+S)).
rmode(1: #(30*2*S: p_state(A,S), p_state(A,S))).
...
```

The first lines are the default options for Tilde. The output file will containthe tree learned represented in C4.5 format, and a the equivalent logic progral (see below). The classes are default too, plans finishes with success or failure. The **rmode** instructions are used to define de predicates that will be considered in the tests by Tilde. For each precicate appearing in the knowledge base and the background theory, the agent will test i) the predicate with free variables as arguments (first rmode); and ii) the predicate with variables instantiated by the examples. The second rmode determines that the variable S in the predicate `p_state`, can be instantiated with 2 values using a maximum of 30 examples to do that. The values in this case are `ok` and `painted`.

The following pruned tree and logic program are obtained running Tilde on this learning set:

```
p_state(A,painted) ?
+--yes: failure [2 / 2]
+--no:  succes [4 / 4]

n1 :- p_state(A,painted).
class(failure) :- p_state(A,painted).
class(succes) :-  not n1.
```

Fractions of the form $[i/j]$ indicate that there where i examples in the class, and that j where well classified. This example used 6 models and the time of induction 0.03 seconds, running on a Sun Ultra 4, with SunOS 5.6. The agent can read this file output (r1.ptree) and add the predicate `(not (p-state (?A,painted)))` to the context of `p0`. In this way, the learning process is coupled together with the BDI interpreter. Particularly, learning is executed in response of events and based on information the agent itself processes. The result of the learning process is incorporated by the agent in the BDI architecture. For a more detailed explanation of the learning set see [7].

4 Learning in MAS

We have observed that the challenges of MAS to ML are indicative of a hierarchy of MAS of increasing complexity, which could be useful to adopt a incremental approach to MAS learning. Levels are as follows: i) Agents learning from the interactions with their environment without direct interactions with other agents. This level corresponds roughly to the BDI learning agent presented in previous section; ii) In the second level, direct interaction among agents using exchange of messages is considered; iii) The third level consider agents learning from the obsevation of other agents actions. For the moment, we are working on the second level of this hierarchy.

Communication is very important when learning in a MAS, but in order to exploit it, the design of the agent should determine when, how, and with what purpose should an individual agent communicate [1].

Our agents are intended to communicate after the execution of a learning process. Two situations determine the *purpose of communication*: i) the trigger event considered has not been modified, e.g. the agent asks for help; and ii) the trigger event has been successfully modified, e.g. the agent offers help. The subject of this help is one or more elements of the learning set to other agent. In this way agents are proactive towards learning.

The concept of competence is used to address communications. *Competence* is defined as the set of all the trigger events an agent can deal with (the union of the trigger events of all the agent's plan library). Two kinds of communications are possible: i) the agent broadcast its message composed by the plan instance considered (observe that it includes the trigger event), and the rest of the agents accept the message if the trigger event is on their competences; ii) Competence is used to build a directory for each agent, associating with each event in the competence of the agent, the agents dealing with the same event.

Competence and plans determine what to communicate. If two agents have the same plan for the same event, they can be engaged in a process of *distributed data gathering*, e.g. they can share the examples they have collected. In this case agents are involved in collecting data, but each agent learns locally. We have test this in our scenario, forcing one agent to paint itself, so that the models 3 and 4, in the previous section, are models provided by an agent `r2`.

We are still testing the case where agents have the same competence, e.g. they can deal with the same event, but have different plans to do it. Intuitively, in this case they have to exchange plans too, but our results are not conclusive yet.

5 Conclusions

We have used a BDI architecture to explain how intentional agents can learn in a MAS. Using this architecture, we explained different considerations about choosing a learning method for an agent. Once we adopted First-Order Induction of Logical Decision Trees as the learning method, we explained how to integrate it to the BDI architecture. Different sets for learning were there introduced, at the mono agent level and the multi agent one. These cases exemplify different uses of communication among agents while learning, and how the BDI agents can focus its messages.

Perhaps the decision of doing our own implementation of a BDI interpreter needs some justification. Even when we knew that different implementations for BDI architecture existed, i.e. PRS, dMARS, !JAM (See [16] for an overview), we decided to implement our own BDI interpreter because we had access only to formal specifications of them, ignoring if we could modify or extend them accordingly to our needs. The specification of dMARS we used [8] suggested immediately the use of Lisp. Tilde was provided by Blokeel.

Some works in the same direction that ours include: Olivia and co-authors [10] present a Case-Based BDI framework applied to intelligent search on the web. Their interpreter is not properly a BDI interpreter, being closer to a Case-Based system. We preferred to force the agents to integrate what they learn in the BDI components of the architecture, so that they continue operating under BDI principles. Grecu and Brown [6] have a similar position about the way agents learn, but their agents are not intentional and used propositional representations. Jacobs and co-authors [9] presents the use of Inductive Logic Programming for the validation of MAS systems. This lead us to Tilde.

Experimental results are very promising. Up to now, we have deal with agents that have contexts already defined, and modify them when plans are not executed correctly. We will experiment the case of agents learning contexts without previous specifications. Also, as mentioned, more experiments are needed at the multi agent level.

6 Acknowledgments

Discussion with David Kinny and Pablo Noriega, had been really helpful in our research. The first author is supported by Mexican scholarships from Conacyt, contract 70354, and Promep, contract UVER-53.

References

1. Bradzil, P., et.al.: Learning in Distributed Systems and Multi-Agent Environments. In: Kodratoff (ed.): Machine Learning - EWSL-91, European Working Session on Learning. Lecture Notes in Computer Science, Vol. 482. Springer-Verlag, Berlin Heidelberg New York (1991)
2. Bratman, M.: Intention, Plans, and Practical Reasoning. Harvard University Press, Cambridge MA., USA (1987)
3. Cayrol, M., Regnier, P., Vidal, V.: LGCP: une amélioration de graph-plan para relâchement de contraintes entre actions simultanées. In: 12ème Congrès Francophone AFRIF-AFIA de Reconnaissance des Formes et Intelligence Artificiel. Paris, France (2000)
4. Charniak, E., McDermott D.: Introduction to Artificial Intelligence. Addison-Wesley, USA (1985)
5. García, F.: Apprentissage et Planification. In: Proceedings of JICAA'97 USA (1997) 15–26
6. Grecu, D.L., Brown, D.C.: Guiding Agent Learning in Design. In: Tomiyama, T., Mantyla, M. (eds.): Proceedings of the Third IFIP Working Group 5.2 Workshop on Knowledge Intesive CAD. Tokio, Japan (1998) 237–250
7. Guerra-Hernández, A., El Fallah-Seghrouchni, A., Soldano, H.: BDI Multiagent Learning based on First-Order Induction of Logical Decision Trees. In: Zhong, N., Liu, J., Ohsuga, S., Bradshaw, J. (eds.): Intelligent Agent Technology, research and developement. Proceedings of the 2nd Asia-Pacific Conference on IAT. World Scientific, New Jersey London Sigapore Hong Kong (2001) 160–169
8. D'Inverno, M., Kinny, D., Luck, M., Wooldridge M.: A Formal Specification of dMARS. In: Intelligent Agents IV. Lecture Notes in Artificial Intelligence, Vol. 1365. Springer-Verlag, Berlin Heidelberg New York (1997) 155–176
9. Jacobs, N., et.al.: Using ILP-Systems for Verification and Validation of Multi-Agent Systems. In: Lavrac, N., Dzeroski,S. (eds.): Inductive Logic Programming. Springer-Verlag, Berlin Heidelberg New York (1997)
10. Olivia, C., et.al.: Case-Based BDI agents: An Effective Approach to Intelligent Search on the WWW. In: AAAI Symposium on Intelligent Agents. Stanford University, Stanford CA., USA (1999)

11. Quinlan, J.R.: Induction of Decision Trees. Machine Learning bt1:81–106 (1986)
12. De Raedt, L., Blockeel, H.: Top-Down Induction of Logical Decision Trees. Technical Report, Department of Computer Science, Katholieke Universiteit Leuven, Belgium (1997)
13. Russell, S.J., Norvig, P.: Artificial Intelligence, a modern approach. Prentice-Hall, New Jersey NJ, USA (1995)
14. Singh, M., Rao, A.S., Georgeff, M.P.: Formal Methods in DAI: Logic-based representations and reasoning. In: Weiss, G. (ed.): Multiagent Systems, a modern approach to Distributed Artificial Intelligence. MIT Press, Cambridge MA., USA (1999)
15. Weiss, G., Sen, S.: Adaptation and Learning in Multiagent Systems. Lecture Notes in Artificial Intelligence, Vol. 1042. Springer-Verlag, Berlin Heidelberg New York (1996)
16. Wooldridge, M.: Reasoning about Rational Agents. MIT Press, Cambridge MA., USA (2000)