Extracting Knowledge from Reactive Robot Behaviour

Agapito Ledezma
Universidad Carlos III de Madrid
Avenida de la Universidad, 30
28911, Leganés, (Madrid)
Spain
ledezma@inf.uc3m.es

Antonio Berlanga
Universidad Carlos III de Madrid
Avenida de la Universidad, 30
28911, Leganés, (Madrid)
Spain
aberlan@ia.uc3m.es

Ricardo Aler
Universidad Carlos III de Madrid
Avenida de la Universidad, 30
28911, Leganés, (Madrid)
Spain
aler@inf.uc3m.es

ABSTRACT

In previous work, we have presented an approach that allows to acquire a declarative representation of the behaviour of an agent, by observing what output it produces from its inputs. So is, we acquire a model of other agents. In that case, only domains coming from the UCI database were tried. In this paper, we test our approach to model the behavior of an actual agent: a simulated robot whose controller (a feed-forward neural network) was learned by using new coevolutionary techniques. In our previous work, we used C4.5 to model other agents. However, C4.5 cannot handle continuous classes, hence they have to be discretized. Besides reporting the results of applying C4.5+discretization to the robot domain, we extend our approach by using a machine learning technique able to use continuous outputs (M5). Finally, we compare the learned models to the neural net controller by allowing those models to control the robot directly with the models obtained from C4.5 and M5 in actual simulations.

Keywords

agent's models, robot behaviour

1. INTRODUCTION

In previous work, we have presented an approach that allows to acquire a declarative representation of the behaviour of an agent, by observing what output it produces from its inputs [2]. So is, we acquire a model of other agents. In that case, only domains coming from the UCI database were tried. In this paper, we test our approach to model the behavior of an actual agent: a simulated robot whose controller (a feed-forward neural network) was learned by using new coevolutionary techniques [3]. With this, we want to determine whether our approach is useful to model agents whose behavior is quite complex. Also, in a single run, the robot applies the controller many times. Therefore, even if the learned model follows closely the neural net outputs, it could happen that in a run small errors could mislead the robot controlled by the learned model. Here, we test for that possibility by letting the learned model control the robot in an actual run of the robot simulator.

In this paper we also test for more accurate modeling tools than previously. In our previous work, we used C4.5 to model other agents. However, C4.5 cannot handle continuous classes, hence they have to be discretized. Besides reporting the results of applying C4.5+discretization to the robot domain, we extend our approach by using a machine learning technique able to use continuous outputs (M5).

This paper is organized as follows. Section 2 gives background about our approach to automatically acquire agent models. Section 3 explains the experimental setup. Actual results are detailed in 4. We close the paper with some concluding remarks 5.

2. AUTOMATIC ACQUISITION OF MODELS

We will first present a description of a robot behavior in terms of its inputs and outputs. If we think in all possible inputs, we could represent them in terms of set of input parameters, such as their sensor readings at a given moment. Therefore, there is a clear analogy with a classification task in which each input parameter of the robot will be represented as an attribute that can have as many values as the corresponding input parameter. With respect to the output, since we have selected a set of tasks that have the characteristic of generating one-step decisions (solutions that do not require a set of steps, as in planning), we can think of them as atomic outputs. In terms of a classification task, this allows define a class for each possible output. Then, the task of modelling (generating a declarative representation of a robot behavior) has been translated into a classification task. In a previous paper we presented results only for discrete tasks in which the outputs belonged to a finite set of values[2]. Given that the outputs of the robot control task are wheel velocities, which are continuous values, two different approaches can be used: either discretize the output and use a typical symbolic classifier (like C4.5 [8]), or use a symbolic algorithm that is able to deal with continuous outputs (like regression trees [5, 10]). Here, we have followed both approaches.

Once we have a classification task, any classification technique c could be employed for solving the task: instance based learning [1], learning decision trees[9], learning rules[7, 8], or neural networks[11]. However, we would like to obtain a declarative representation, so this constrains the type of technique to be used to those that generate symbolic representations, such as decision trees, or rules. In the preliminary results presented here, we have used a neural network that controls the robot to be modelled, a technique that generates rules, such as C4.5[8], and a technique, M5, that generates regression trees[10]. These can be seen as rules whose then-part is a linear combination of the values of the input parameters.

The learning task is the following:

- Inputs:
  - set of attributes that model the input parameters of robot r_1
  - for each attribute, the set of values that its corresponding input parameter can have (in this case, continuous variables)
  - set of possible outputs in the case of discrete classes, and continuous in the case of continuous classes
  - set of training instances
  - a classification technique c
- Output: a declarative classifier that provides the same (or approximate) output as the robot r would provide given the same input instance

The general framework is described in figure 1 which shows the interrelation among the robot r_1, the modeler r_2 that tries to learn and reason about a model of r_1, the classification technique c used for modelling its behavior, and the obtained classifier m (model of r_1). This classifier m should model the behavior of robot r_1, in such a way that if one presents the same set of input patterns (sensory data) to both r_1 and m the error between the output provided by r_1 and m should be minimal.

3. EXPERIMENTAL SETUP

This section describes the experimental sequence needed to obtain a symbolic model m from an agent r_1 (which is considered as a black box) than can be used by an agent r_2. To do so, we have carried out three phases: a robot training phase for r_1, a training phase for obtaining a model m of r_1, to be used by r_2 and the test of model m in a robot simulator. During robot training (r_1), Uniform Coevolution was used to obtain a neural network that controls the movement of the robot based on a Braitenberg scheme [4]. The robot moves in a bi-dimensional world, where there are obstacles with different shapes. Its goal is to move to a goal position in an efficient way. The schema of the robot is shown in Figure 2. It has five sensors. Three of them (the proximity sensors) inform the robot how close obstacles are. The next two, measure how far the target location is, and what the angle to that location is. The robot has two wheels that can move at different speeds v_1 and v_2, so that the robot can turn. However, for experimental purpose, the speed of wheel v_1 is fixed, therefore the neural network can only control wheel v_2. The aim of the neural network learned by Uniform Coevolution is to control r_1, that is, to map its sensors into v_2 speed. We refer the reader to[3] for details about Uniform Coevolution.

Once the controller for r_1 has been learned, the knowledge that tries to model the behaviour of r_1 is obtained by two different techniques: a rule modeler (C4.5, discretized outputs) and a regression tree modeler (M5, continuous outputs). The detailed steps for training r_2 are as follows:

1. The robot r_1, being controlled by the (co)evolved neural network is run several times. At every instant, the readings of the five sensors (inputs) and the wheel velocities (outputs) are logged to produce a trace of the reactive behaviour of the robot. From this trace it is straightforward to obtain a set of examples T so that r_2 can learn and model r_1.
2. Let T be the whole set of available examples from r_1 inputs and outputs. Each example $t_i \in T$ is made of two parts: an n-dimensional vector representing the attributes $r(t_i)$ and a value $c(t_i)$ representing the class it belongs to.
3. The Set T is used to train r_2, which generates modeling knowledge. As the class part of the examples in T are r_1 outputs, the rules should model the behaviour of r_1 neural network. That is, r_2 knowledge should predict the output of r_1, no matter whether it is the right or the wrong output. Of course, we are also interested in modelling r_1's mistakes, if any.

Now, we have to determine how closely r_2 knowledge models r_1 behaviour. To do so, we carry out ten-fold cross-validation. In C4.5, the closeness of the performance of both r_1 and r_2 is measured as the number of examples in which the predictions of r_2 and r_1 differ (for the same sensory input). In the experiment that uses M5 to model r_1 behavior we use the correlation coefficient to measure the model error. Once the model m have been generated, we test it in a robot simulator, SimDAI [12], and we compared the performance of our model against the robot controlled by the neural network in terms of time consumed and distance covered.

4. RESULTS

For experimental evaluation of the approach presented in this paper we have used a feed-forward neural network learned by co-evolution as robot r_1 (the robot to be modeled) and the C4.5 and M5 classifiers as r_2 (the system that acquires the model). In this section we present the obtained results.

In all the experiments we do ten-fold cross-validation.

4.1 Generating rules

In this domain, the learning task is the prediction among eleven classes, based on 5 attributes (sensors data). The number of instances are 976 corresponding to six simulations of r_1. The classes have been obtained discretizing the r_1 outputs (one class was assigned to velocity of -1.0, another

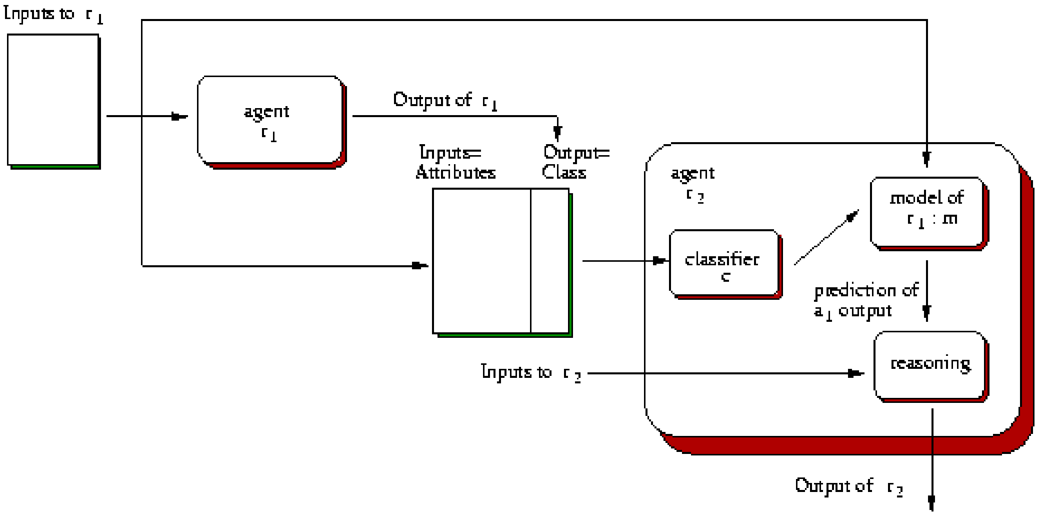

Figure 1: Architecture of the modelling of robots behavior.

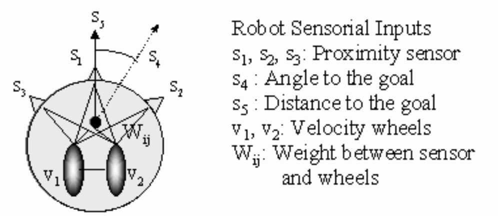

Figure 2: Robot description

class to velocity of 1.0 and other to null velocity, the rest of class was assigned to velocities intervals of 0.25). Results are shown in Table 1. The first column shows the 10-fold cross-validation modeling accuracy of pure C4.5, whose output is a decision tree. The second column shows the accuracy for C4.5 when it generates a set of rules. In short, the model m generated by C4.5-RULES is able to guess the output of the neural net 89 times out of 100, which is a quite good result.

	Hits/c4.5	**Hits/c4.5Rules**
Best result	89.70%	89.80 %
Average	84.44%	84.02 %
Deviation	3.8624%	4.2424 %

Table 1: Results using C4.5.

After the model m was generated, we implemented its rules in a robot simulator, SimDAI, to test it. In total there were 50 trials (robot motion) with r_1 and model m. Each trial begin from different places in a bi-dimensional world (Figure 3) and consist in reaching the goal in an efficient way.

As we can see in Figure 4, the covered distance of r_1 and the model m are almost the same.

Rule 13:		
sensor5	$>$	0.940451
	class =	null [79.4%]
Rule 1:		
sensor1	$\leq$	0.08 AND
sensor3	$\leq$	0.426667 AND
sensor4	$\leq$	-0.863739 AND
sensor5	$\leq$	0.298303
	class =	very-high [89.9%]
Rule 73:		
sensor1	$>$	0.346667 AND
sensor2	$\leq$	0.16 AND
sensor5	$\leq$	0.141961 AND
	class =	very-low [98.1%]
Rule 80:		
sensor1	$>$	0.573333
	class =	super-low [98.0%]
Rule 69:		
sensor1	$>$	0.28 AND
sensor4	$>$	-0.039041
	class =	super-low [97.1%] ...

Table 2: Samples of Rules generating by C4.5.

In the case of the time consumed (Figure 5) we can see that r_1 does not reach the goal only in one trial (time $>$ 2000) and our model m does not reach the goal twice. Nonetheless, the time of the remainder trials are almost the same.

4.2 Generating regression trees with M5.

Like in the first experiment, we have used instances from six r_1 simulations. The total number of instances is 976 with continuous classes (the class is the velocity of one wheel of r_1). We applied the M5 algorithm to generated a regression tree. 10-fold cross-validation results can be seen in Table 3. The regression tree predicts almost perfectly r_1 neural net-

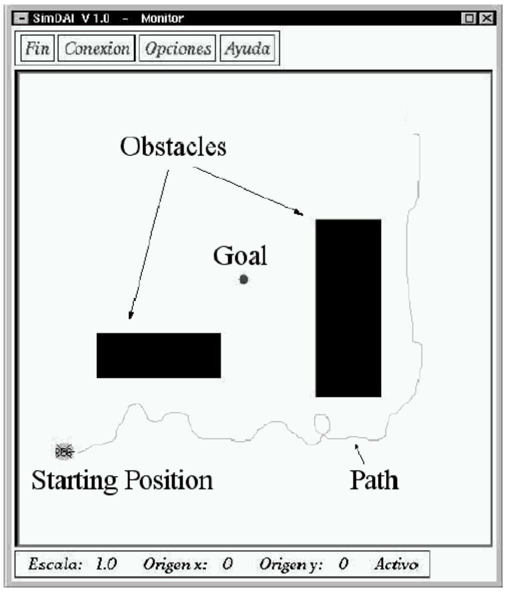

Figure 3: Bi-dimensional world.

work.

Correlation Coefficient	0.9954
Mean Absolute Error	0.0342
Root Mean Squared Error	0.0641

Table 3: Results of Ten-Folds Cross-Validation.

The regression tree obtained is shown in Table 4. Each rule from a regression tree corresponds to a Linear Model (table 5) that estimate the class value (velocity of wheel v_2).

As we can see in table 3 we obtained a high correlation coefficient that indicate that the model r_2 and r_1 are similar.

5. CONCLUSIONS

Results show that our approach is quite good at modelling simple neural agents evolved by means of coevolution in a goal-seeking obstacle-avoiding robot domain. In this paper we have used two machine learning techniques to obtain the modelling knowledge: C4.5 (that cannot use continuous outputs) and M5 (which can). As expected, results show that M5 obtains better results than C4.5, although the knowledge obtained is not so easily understandable. Using the modelling knowledge to control the robot in actual simulations confirm our results.

In the future we want to extend our approach with the following ideas:

- Instead of using an ad-hoc discretization, we intend to use a discretization method that has been successfully applied in the reinforcement learning field [6].
- Testing our approach with other obstacle configurations.

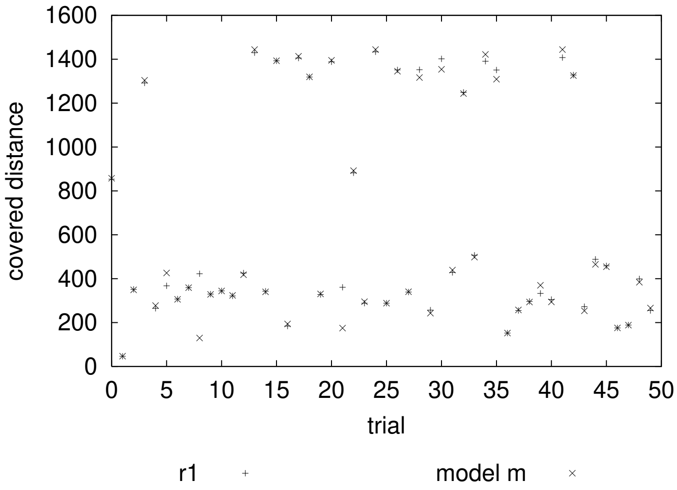

Figure 4: Covered distance.

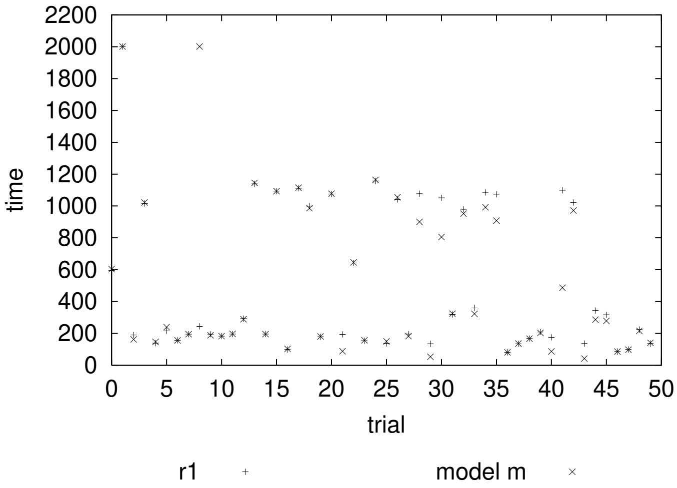

Figure 5: Time

- Make some agents learn models of other agents so that they can cooperate together.
- Currently, the acquisition of models of other agents is off-line. However, it would be very interesting that an agent could learn about other agents by observing them on-line.

It is important to remark that this method will only be applied successfully to reactive agents. If the agent to model is not reactive (i.e. its output depends on something else, like memory, besides the sensors), models would be quite inaccurate. However, it would be interesting to see how the performance of models degrades as the non-reactiveness increases.

$Sensor1$	$Sensor2$	$Sensor3$	$Sensor4$	$Sensor5$	Model
$\leq$ 0.0333	-	$\leq$ 0.233	$\leq$ -0.841	$\leq$0.29	LM1
$\leq$ 0.0333	-	$\leq$ 0.233	$>$ -0.841 and $\leq$ -0.743	$\leq$0.29	LM2
$\leq$ 0.0333	-	$\leq$ 0.233	$>$ -0.743	$\leq$0.29	LM3
$\leq$ 0.0333	-	$\leq$ 0.233	$\leq$-0.59	$>$0.29 and $\leq$0.761	LM4
$\leq$ 0.0333	-	$\leq$ 0.233	$\leq$-0.59	$>$0.761 and $\leq$0.975	LM5
$\leq$ 0.0333	-	$\leq$ 0.233	$\leq$-0.59	$>$0.975	LM6
$\leq$ 0.0333	-	$\leq$ 0.233	$>$-0.59 and $\leq$ -0.453	-	LM7
$\leq$ 0.0333	-	$\leq$ 0.233	$>$-0.453	-	LM8
$\leq$ 0.0333	-	$>$ 0.233	$\leq$-0.161	-	LM9
$>$ 0.0333 and $\leq$0.22	-	-	$\leq$-0.161	-	LM10
$>$0.22 and $\leq$0.587	-	$\leq$0.213	$\leq$-0.161	-	LM11
$>$0.22 and $\leq$0.587	-	$>$0.213	$\leq$-0.161	-	LM12
$>$0.587	-	-	$\leq$-0.161	-	LM13
$\leq$0.193	-	-	$>$-0.161 and $\leq$0.134	-	LM14
$\leq$0.193	-	-	$>$0.134 and $\leq$0.711	-	LM15
$\leq$0.193	-	-	$>$0.711	-	LM16
$>$0.193	-	-	$>$-0.161	-	LM16

Table 4: Rules from regression tree.

Model	Prediction	Factor	$Sensor1$	$Sensor2$	$Sensor3$	$Sensor4$	$Sensor5$
LM1:	class =	0.36	-0.0936	+0.0711	-0.895	-0.723	- 0.276
LM2:	class =	0.358	-0.0936	+0.129	-1.25	-0.668	-0.35
LM3:	class =	0.0201	-0.0936	+0.0288	-1.2	-1.15	-0.423
LM4:	class =	0.201	-0.0936	+0.0751	-1.29	-0.969	-0.531
LM5:	class =	0.886	-0.0936	+0.0365	-1.11	-0.427	-0.859
LM6:	class =	0.933	-0.0936	+0.0365	-1.11	-0.427	-0.943
LM7:	class =	0.0814	-0.0936	+0.0257	-0.312	-1	-0.134
LM8:	class =	-0.0041	-0.0936	+0.0202	-1.22	-1.2	-0.453
LM9:	class =	-0.00956	-0.0936	+0.0155	-1.14	-1.12	-0.166
LM10:	class =	-0.073	-2.6	+0.0441	-0.964	-1.03	-0.264
LM11:	class =	-0.149	-2.23	+0.0065	-0.407	-0.895	-0.0413
LM12:	class =	-0.427	-1.33	+0.0065	-0.424	-0.488	-0.0413
LM13:	class =	-0.719	-0.62	+0.0065	-0.226	-0.305	-0.0413
LM14:	class =	-0.0313	-2.87	+0.0897	-1.14	-1.13	-0.33
LM15:	class =	-0.0435	-2.25	+0.0879	-1.11	-1.11	-0.329
LM16:	class =	-0.408	-0.757	+0.0364	-0.38	-0.652	
LM17:	class =	-0.839	-0.126	+0.013	-0.129	-0.135	

Table 5: Linear Models.

6. REFERENCES

[1] D. W. Aha, D. Kibler, and M. K. Albert. Instance-based learning algorithms. *Machine Learning*, 6(1):37–66, jan 1991.

[2] R. Aler, D. Borrajo, I. Galvan, and A. Ledezma. Learning models of other agents. In *Proceedings of the Agents-00/ECML-00 Workshop on Learning Agents*, pages 1–5, Barcelona, Spain, June 2000.

[3] A. Berlanga, A. Sanchis, P. Isasi, and J. M. Molina. A general coevolution method to generalize autonomous robot navigation behavior. In *Proceedings of the Congress on Evolutionary Computation*, pages 769–776, La Jolla, San Diego (CA) USA, July 2000. IEEE Press.

[4] V. Braitenberg. *Vehicles: experiments on synthetic psychology.* MIT Press, Massachusets, 1984.

[5] L. Breiman, J. Friedman, K. Olshen, and C. Stone. *Classification and Regression Tress.* Wadsworth & Brooks, Monterey, CA (USA), 1984.

[6] F. Fernández, D. Borrajo, and V. Matellán. Vqql: A model to generalize in reinforcement learning. In S. Biundo and M. Fox, editors, *Preprints of the Fifth European Conference on Planning, ECP'99*, pages 385–386, September 1999.

[7] R. S. Michalski. A theory and methodology of inductive learning. *Artificial Intelligence*, 20, 1983.

[8] J. Quinlan. *C4.5: Programs for Machine Learning.* Morgan Kaufmann, San Mateo, CA, 1993.

[9] J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, 1986.

[10] J. R. Quinlan. Combining instance-based and model-based learning. In *Proceedings of the Tenth*

International Conference on Machine Learning, pages 236–243, Amherst, MA, June 1993. Morgan Kaufmann.

[11] D. Rummelhart, J. McClelland, and the PDP Research Group. *Parallel Distributed Processing Foundations*. The MIT Press, Cambridge, MA, 1986.

[12] L. Sommaruga, I. Merino, V. Matellán, and J. Molina. A distributed simulator for intelligent autonomous robots. In *In Proccedings of Fourth International Symposium on Intelligent Robotic Systems*, pages 393–399, 1996.