Working Notes of the 5th Pacific Rim International Workshop on Multi-Agents (PRIMA2002), Tokyo, Japan, pp. 7-16, Aug. 2002.

RoboCup Agent Learning from Observations with Hierarchical Multiple Decision Trees

Ruck Thawonmas[1], Junichiro Hirayama[2], and Fumiaki Takeda[2]

[1] Department of Computer Science, Ritsumeikan University
1-1-1 Noji Higashi Kusatsu City, Shiga 525-8577, Japan
[2] Course of Information Systems Engineering, Kochi University of Technology
185 Miyanokuchi, Tosayamada-cho, Kami-gun, Kochi 782-8502, Japan
ruck@cs.ritsumei.ac.jp

Abstract. It is a dif£cult task to hand-code optimal condition-action rules for software agents. A solution to this is reinforcement learning. In reinforcement learning, agents acquire the condition-action rules by learning from their experiences. However, acquisition of complicated rules might take a great amount of learning time and learning might not converge. To solve these drawbacks, an approach called learning from observations has been proposed in which learning in an agent is performed by observing human actions in the same environment of that of the agent. In our earlier work, we have applied this learning approach to the RoboCup software agent domain and adopted C4.5 as a learning engine. In this paper, we discuss a novel learning methodology that exploits the hierarchy structure of classes using hierarchical multiple decision trees, each generated by C4.5. Simulation results con£rm the superiority of the hierarchical multiple version of decision tress over a single decision tree, in terms of both generalization ability and agent's performance.

Keywords: Autonomous Agents, Machine Learning, Learning from Observations, Decision Trees, C4.5, RoboCup

1 Introduction

It's a not-so-easy task to build software agents or multiagents (henceforth simply called agents) so that they could perform desired actions. To de£ne proper condition-action rules, one may choose to hand code them using if-then rules or to use machine learning techniques such as reinforcement learning [1]. Hand coding requires a set of rules that

must cope with various kinds of conditions. Though the learning ability of agents is not necessary, this approach requires special skills and much effort to write complete agent programs.

On the other hand, reinforcement learning can relatively easily make agents learn from their experiences a moderate number of condition-action rules. Compared to hand coding, this approach imposes the lesser burden on the agent builder. This approach is also useful for adding re£nements to other methods. However, in this approach, due to explosion of the number of pairs of actions and states, it might be dif£cult to learn complex rules or take a long learning time to reach them.

As an approach to solve the aforementioned problems, learning from observations [2,3] has been proposed and mainly applied to robotic applications. In this approach, learning in an agent is performed by observing the actions of a human expert in the same environment of that of the agent. The approach belongs to supervised learning paradigm. The task of the agent builder changes from directly coding the condition-action rules to training the agent, with selected pairs of conditions and actions of the human expert, by a supervised learning engine. Note that the human expert implies a person who has expertise in the £eld of interest, and is not necessarily the same person as the agent builder. Moreover, since agents are trained off line by the selected cases, unlike reinforcement learning in which the learning process is on-line and trial-and-error, the learning time can be reduced.

In our earlier work [4], we applied the learning-from-observations approach to the RoboCup software agent domain. As a supervised learning engine, C4.5 [5] was used there. We had later found that data classes have a hierarchical structure, and exploitation of this information might enhance the generalization ability, namely, the ability to cope with unknown data, as well as the performance of the agents in games.

In this paper, we propose a novel learning scheme that uses hierarchical multiple decision trees for learning from observations, and apply it to RoboCup software agents. In the proposed learning scheme, each decision tree in the hierarchy is generated by C4.5. The tree at the top of the hierarchy is for classi£cation of the main classes, and a successor tree is used for classi£cation of the subclasses of a corresponding main class.

The material presented in the remainder of this paper is organized as follows. In Section 2, we give a brief description of the RoboCup-agent-learning-from-observation system. Section 3 discusses the proposed learning scheme. We show experimental results and give discussions in Section 4, and provide some conclusions and possible future works in Section 5.

2 Learning-from-Observations System

In this section, we describe brie¤y the system for RoboCup agent learning from observations that we have proposed very recently. As shown in Fig. 1, this system consists of three subsystems, namely, the KUT-RP (standing for Kochi University of Technology Ritsumeikan University's Player) subsystem, the rules extraction subsystem, and the VH (standing for Virtual Human) subsystem (or agent).

The role of the KUT-RP subsystem is to enable human players to participate simulated soccer games by operating agents called KUT-RP agents; usually one human

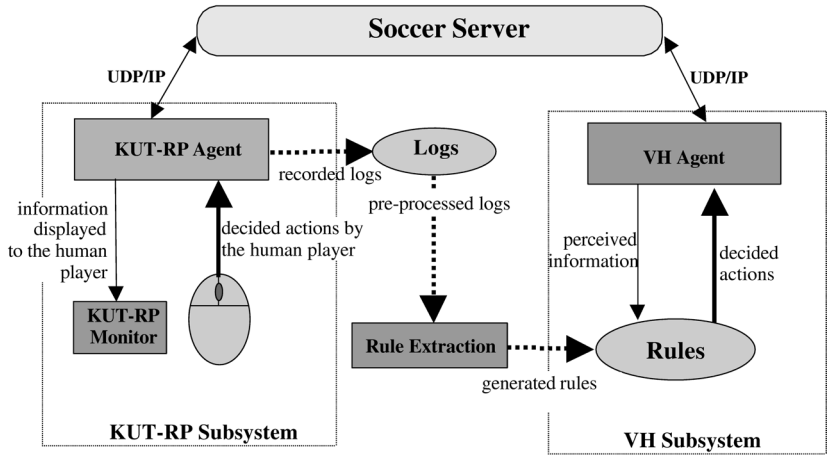


Fig. 1. Architecture of the Learning-from-Observations system.

player operates one KUT-RP agent. Our work on the KUT-RP subsystem was partly inspired by a system called OZ-RP [6]. The OZ-RP system highly depends on the model for describing the environment and internal states of their agents. It thus is not compatible with our Java-based agents that have been being developed for both education [7] and competitions since 1999[1].

The KUT-RP subsystem provides to the human player three composite commands described below as follows:

kick to speci£ed position that makes the KUT-RP agent kick the ball toward the speci£ed position,

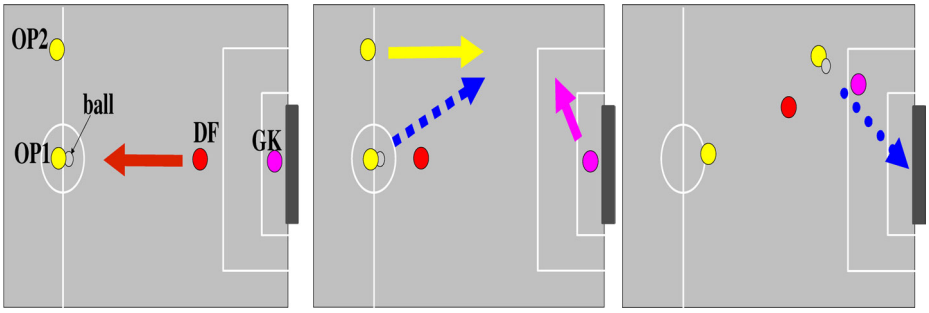
dash to speci£ed position that makes the KUT-RP agent dash to the speci£ed position, and

dribble to speci£ed position that makes the KUT-RP agent dribble the ball to the speci£ed position.

For each composite command, a speci£ed position is de£ned by the mouse-clicked position on the KUT-RP monitor. Logs, pairs of conditions and the actions decided by the human player, from the KUT-RP subsystem are used for extracting human decision-making behaviors.

The rules-extraction subsystem generates condition-action rules from the logs of the KUT-RP subsystem. Pre-processing of logs such as selection of particular logs or labeling of the action classes are performed before the rule generation module is invoked. In our previous work, the standard C4.5 is used for extracting the condition-action rules from those pre-processed logs. A new learning scheme for this is discussed in Section 3 in more detail.

[1] Our soccer simulation teams NoHoHoN and NoHoHoN G2 participated in the Japan Open 2000 and 2001, respectively, and got one win each.



(a) The First Part (b) The Middle Part (c) The Final Part

Fig. 2. The scenario of the learning tasks of the OP1 and OP2 agents.

The VH subsystem is an autonomous agent that performs its own actions according to the equipped condition-action rules. In addition to those rules automatically extracted from the rules-extraction subsystem, incorporation of hand-coded rules or rules obtained on-line by reinforcement learning might further improve the performance of the VH agents. Our reports on this issue will be given elsewhere.

3 Hierarchical Multiple Decision Trees

Let us consider a multi agent cooperative learning task[2] of a pair of offensive players. In this task, the objectives of the two offensive players, OP1 initially positioned near the center-circle and OP2 initially positioned near the upper side-line above OP1, are to cooperate with each other to get scores from the opponent team as many as possible. The opponent team also consists of two players, a defender (DF), whose role is to go to and take the ball, and a goalkeeper (GK), in which the goaltending technique discussed in [8] is adopted. The scenario of the task is as follows:

- DF moves toward OP1 who initially keeps the ball (see Fig. 2.a).
- After letting DF come close to a certain position, OP1 kicks the ball to a position near the opponent's upper-corner. At the same time, OP2 starts chasing the ball while competing with the DF (see Fig. 2.b). If OP2 starts too early then it will be off-side; if too slowly then the ball will be taken or intercepted by DF.
- After taking the ball, OP2 shoots the ball toward the opponent goal (see Fig. 2.c).

The targets of the learning are OP1 and OP2. To conduct learning from observations, we £rst use the KUT-RP subsystem to obtain logs of OP1 and OP2, and pre-process these

[2] To the best of our knowledge, all existing works on applications of learning from observations focused on learning of a single robot or agent. Apparently, this work represents the £rst attempt on multi agent cooperative learning with learning from observations.

Attribute Names	Attribute Types
myX	continuous
myY	continuous
$myBodyAngle$	continuous
$ballDirection$	continuous
$ballDistance$	continuous
$opponentDirection$	continuous
$opponentDistance$	continuous
$teammateDirection$	continuous
$teammateDistance$	continuous

Table 1. Attribute names and types of the data used for learning.

logs, Then for each position, we extract from the pre-processed logs the condition-action rules using the proposed learning scheme (to be described in Section 3), and £nally apply the rules to two VH agents, one playing the role of OP1 and the other of OP2.

The logs are pre-processed to obtain data whose input attributes, associated with their types, and output class labels are summarized in Table 1 and Table 2, respectively. In Table 1, the attributes myX and myY altogether represent the absolute coordinate of the corresponding agent in the soccer £eld. The attribute $myBodyAngle$ indicates the relative direction of the body of the agent toward the center of the opponent goal. The attributes $ballDirection$ and $ballDistance$ are the relative direction to the ball and the distance to the ball, respectively. Likewise, the attributes $opponentDirection$, $opponentDistance$, $teammateDirection$, and $teammateDistance$ are the relative direction and distance to the nearest opponent, and the nearest teammate, respectively.

The classes in Table 2 represent agent actions. The classes $kick$, $dash$, and $drib$ are each divided into 19 subclasses according to the targeted directions relative to the agent's body direction. Data labeled with these three classes are obtained by decomposing the three composite commands described in Section 2. The class $dash$ describes the action that makes the agent move into the speci£ed direction, while the class $drib$ represents the action that makes the agent dribble the ball into the relative direction. The class $Shoot$ is divided into 3 subclasses according to the targeted positions, namely, near the upper goal post, near the center of the goal, or near the lower goal post. The class $Wait$ is the action by which the agent remains in the same position in the £eld while keeps on watching at the ball. The class $Auto$ represents the action by which the agent £rst moves to the ball and then dribbles the ball toward the opponent goal. The action class $Keep$ is for the agent to move to the ball and keep the ball within the kickable distance.

The learning task for each agent boils down to a classi£cation problem with a relatively large number of class labels, i.e., 63 labels. This type of classi£cation problems

Class Labels				
kick-90	dash-90	drib-90	ShootTop	Wait
kick-80	dash-80	drib-80	ShootMid	Auto
kick-70	dash-70	drib-70	ShootBot	Keep
kick-60	dash-60	drib-60		
kick-50	dash-50	drib-50		
kick-40	dash-40	drib-40		
kick-30	dash-30	drib-30		
kick-20	dash-20	drib-20		
kick-10	dash-10	drib-10		
kick0	dash0	drib0		
kick10	dash10	drib10		
kick20	dash20	drib20		
kick30	dash30	drib30		
kick40	dash40	drib40		
kick50	dash50	drib50		
kick60	dash60	drib60		
kick70	dash70	drib70		
kick80	dash80	drib80		
kick90	dash90	drib90		

Table 2. List of classes and their subclasses.

imposes low generalization ability, an ability to classify unknown data not seen in the training set of data, on any single classi£er in use. However, from Table 2, it can be seen that labels do have a hierarchical structure.

Exploiting the hierarchical structure in the class labels, we propose a novel learning scheme that uses hierarchical multiple decision trees. Figure 3 shows the architecture of the hierarchical multiple decision trees to solve the aforementioned classi£cation problem. In this £gure, the decision tree at the top of the hierarchy is responsible for classi£cation of the parent 7 classes, i.e., $kick$, $dash$, $drib$, $Shoot$, $Wait$, $Auto$, and $Keep$. Then, four other decision trees are introduced at the next level for classi£cation of $kick$, $dash$, $drib$, and $Shoot$ subclasses. Under this learning scheme, the number of classes that a classi£er must cope with is reduced. Hence, higher generalization ability can be expected from each classi£er in the hierarachy. In addition, for a given unknown data, once the tree at the top layer correctly classi£es the parent class, misclassi£cation among subclasses in the lower layer may cause less catastrophe to the performance of the corresponding VH agent.

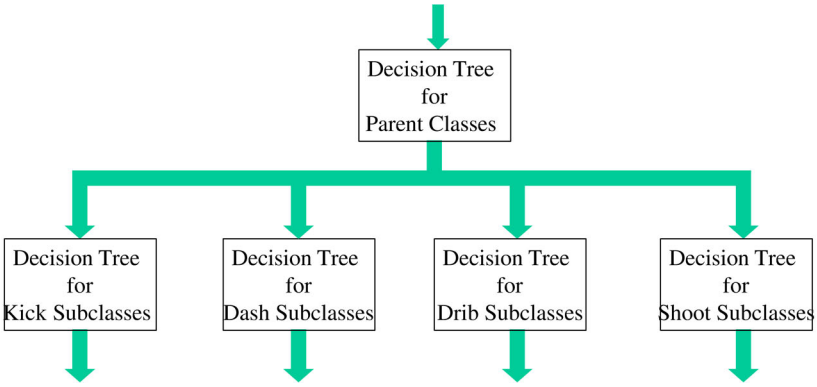


Fig. 3. Architecture of the proposed learning scheme using hierarchical multiple decision trees.

4 Experiments

In this section, we £rst describe an experiment to compare the generalization ability of the proposed learning scheme using hierarchical multiple decision trees and that of the learning scheme using a single decision tree. Then, we compare the performances of the VH agents to which the condition-action rules extracted by the former are applied, with the performances of the VH agents to which the rules from the latter are applied. To generate a decision tree in both schemes, C4.5 was used.

Two human players were asked to perform the cooperative learning task described in the previous section, each controlling one KUT-RP agent, until OP2 scores 20 goals (20 successful trials). The number of pre-processed logs for each successful trail is not £xed, but slightly varies. In total, the numbers of accumulated pre-processed logs of OP1 for 5, 10, 15, and 20 successful trials are 162, 492, 905, and 1310, respectively, and that of OP2 are 590, 1138, 1700, and 2254, respectively.

4.1 Generalization Ability

Figure 4 shows the recognition ratios of the correct sub-classes for unknown data of the two learning schemes over different numbers of successful trials for OP1 (Fig. 4.a) and OP2 (Fig. 4.b), respectively. For each number of successful trials and each player, the available pre-processed logs were divided equally into two portions, one for training the decision trees and the other for testing the recognition ratios of the decision trees when given unknown data. The training data set and the testing data set were then alternated to obtain the recognition ratios against the new testing data set. Recognition results shown in Fig. 4 are the averaged values of the recognition ratios for the former and latter testing data sets.

From these £gures, it can be seen that the generalization ability of the proposed learning scheme using hierarchical multiple decision trees is superior to that of the

Hierarchical Multiple Decision Trees	5	10	15	20
success	3	4	0	0
semi-success	9	2	3	1
failure	8	14	17	19
Single Decision Tree				
success	0	2	1	0
semi-success	0	2	1	0
failure	20	16	18	20

Table 3. The VH agents' performances, in terms of the numbers of goals shot by the OP2-VH agent out of 20 trails, when they are equipped with the rules extracted from the pre-processed logs at different numbers of successful trials from the KUT-RP subsystem.

single decision tree. In Fig. 4.b, however, the generalization abilities of both schemes decrease as the numbers of successful trials increase. We conjecture that this is due to ambiguity in decision making of the human player in charge. For example, the human player for OP2 came up with different, but right decisions for very similar (or even the same) conditions. Preprocessing techniques for cleansing logs or techniques for pruning decision trees might solve this problem. However, they are beyond the scope of this paper and will be discussed elsewhere.

4.2 Agents' Performances

We now de£ne three evaluation indices for each trial attempted by the two VH agents.

- **success:** the VH agent whose role is OP2 (henceforth called the OP2-VH agent) can receive the ball passed by the VH agent whose role is OP1 (henceforth called the OP1-VH agent), and can score a goal
- **semi-success:** the OP2-VH agent can receive the pass from the OP1-VH agent, but can not score a goal (for example, GK can block the shoot, etc.)
- **failure:** all other possible scenarios including being off-side

For either the OP1-VH agent or the OP2-VH agent, all the available pre-processed logs, at each number of successful trials, were used to generate decision trees and to extract rules out of these trees. The resulting rules were then applied to the corresponding VH agents.

Table 3 summarizes the results from the new experiment. As can be seen from this table, the performances of the VH agents whose rules were derived from the proposed learning scheme outperform that of the others.

It is obvious that in order to use VH agents in practice, say, in RoboCup competitions, their performances need to be improved. We are currently improving the interface

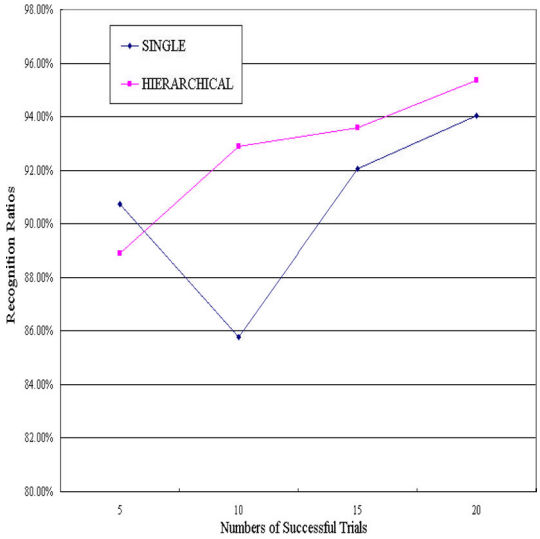
part and the precision of the composite commands in the KUT-RP subsystem. The improvement in the interface part will ensure that a human player can convey his or her decisions to the system more promptly. The latter improvement will ensure that the human decisions will be executed more precisely by the system.

5 Conclusions

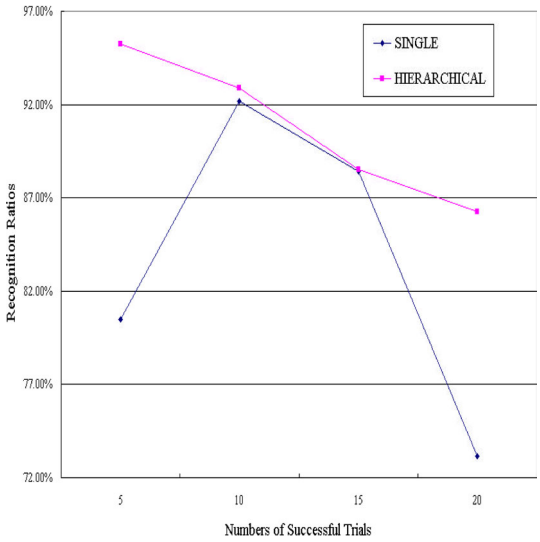
Applying human decision-making behaviors to software agents is an effective approach for implementing complicated condition-action rules. In this paper, we have proposed an effective learning scheme for dealing with learning (or classifying) tasks with a large number of action labels. In the proposed learning scheme, a hierarchy of multiple classi£ers is used rather than a single one. The experimental results given in the paper con£rmed the effectiveness of the proposed scheme, when C4.5 was used for generating decision trees, in terms of generalization ability as well as agent performance. Though we used C4.5 for each classi£er in the hierarchy, other powerful techniques such neural networks might also be considered. Use of hybrids of different classi£cation techniques is also possible and is an interesting future research topic.

References

1. R.S. Sutton and A.G. Barto, Reinforcement Learning (Adaptive Computation and Machine Learning), MIT Press, 1998.
2. Y. Kuniyoshi, M. Inaba, and H. Inoue, *Learning by watching: Extracting reusable task knowledge from visual observation of human performance*, IEEE Trans. Robotics and Automation, pp. 799-822, 1994.
3. D.C. Bentivegna and C. G. Atkeson, *Learning From Observation Using Primitives*, Presented at ICRA 2001 in Seoul, Korea, May 2001.
4. R. Thawonmas, J. Hirayama, and F. Takeda, *Learning from Human Decision-Making Behaviors - An Application to RoboCup Software Agents*, Proc. IEA/AIE 2002, Cairns, Australia, June 2002.
5. J.R. Quinlan, C4.5 Programs for Machine Learning, San Mateo: Morgan Kaufmann, 1993.
6. J. Nishino, et al., *Team OZ-RP: OZ by Real Players for RoboCup 2001, a system to beat replicants*, 2001 (under publication).
7. R. Thawonmas, *Problem Based Learning Education using RoboCup: a Case Study of the Effectiveness of Creative Sheets*, Abstract in the International Symposium on IT and Education (InSITE 2002), Kochi, Jan. 2002.
8. P. Stone, Layered Learning in Multiagent Systems: A Winning Approach to Robotic Soccer, MIT Press, 2000.



(a)



(b)

Fig. 4. The generalization abilities of the two learning schemes for OP1 (a) and OP2 (b).