A Proposal for Inductive Learning Agent Using First-Order Logic

Tohgoroh Matsui, Nobuhiro Inuzuka, and Hirohisa Seki

Nagoya Institute of Technology
Gokiso-cho, Showa-ku, Nagoya 466-8555, Japan
{tohgoroh, inuzuka, seki}@ics.nitech.ac.jp

Abstract. In this paper, we propose new agent architecture which adapts its own behavior by avoiding actions which are predicted to be failure. We name this agent *inductive learning agent* (ILA). This agent consists of five parts: Observer, Planner, Checker, Actor, and Learner. These parts use first-order formalism and inductive logic programming (ILP) to acquire rules to predict. Adapting behavior of ILA works as follows: (1) Collect examples of actions. (2) Classify the examples. (3) Acquire prediction rules using ILP. (4) Behave under the prediction rules. We have implemented it in soccer using parts of the latest RoboCup competition champion CMUnited-99 and an ILP system Progol. We have confirmed that agents could acquire prediction rules and could adapt their behavior using the rules.

1 Introduction

Researchers have studied on machine learning in the toy world and tried to bring them to the real world. However, it gradually became clear that some of these technologies developed in the toy world are not useful in the real world. Recently, software agents for RoboCup (robot soccer world cup) software challenge [4] [7] has become an active area for study of the gaps between the real world and research. RoboCup requires many AI technologies such as design principles of autonomous agents, multi-agent collaboration, strategy acquisition, real-time reasoning and planning in the real world. Software agents that control players for RoboCup soccer are called soccer agents and are implemented as clients of Soccer Server [1] [13]. (See Fig. 1, an image of the virtual field which Soccer Server provides.)

Learning is one of the main capabilities of intelligent agents. Some researches using reinforcement learning or genetic programming have been succeeded in learning motor control or action selection [2] [9] [18]. However, this paper aims to learn symbolic knowledge on strategies inductively. Inductive learning is based on generalization of examples and enables agents to learn from their own experiences.

We propose new architecture of agent which uses inductive learning in this paper. The architecture deals with actions in large enough granules for being treated by human operators, not in low-levels such as motor control. Then we

Fig. 1. An image of the virtual field which Soccer Server provides.

can easily give knowledge to judge the success of actions, and accordingly, we expect to learn rules to predict results of their own actions. Since our agent uses first-order languages for descriptions of perceived information, they can use inductive logic programming (ILP) for learning prediction rules. Our agents can learn relational knowledge on action selection and combine various sorts of background knowledge with the perceived information by using ILP. Accordingly, agents may adapt their behavior in the strategy level.

In this paper, we also describe our implementation in soccer agents and experimental results. We used a program source code of the third RoboCup competition[1] champion simulator team CMUnited-99 [16] [17] for low-level implementation. We also used one of successful ILP system Progol [12] as the learning part. Finally, we discuss features of our architecture.

2 Inductive Learning Agent

We propose new architecture of *inductive learning agent* (ILA) which adapts its own behavior by predicting results in an uncertain environment.

2.1 ILA Architecture

An intelligent agent generally consists of three parts: *Observer*, *Planner*, and *Actor* [14][2], and our agent also includes:

[1] The third RoboCup competition was held in Stockholm, Sweden, 1999.
[2] Observer is called *sensor* and Actor is called *effector* in [14].

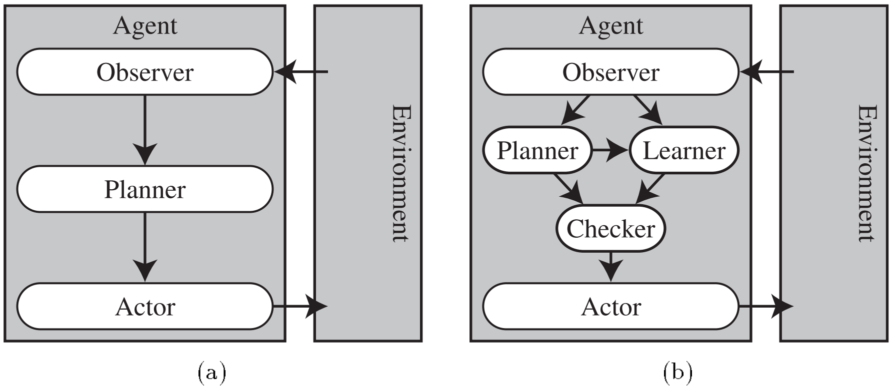


Fig. 2. Architectures of basic agent (a) and ILA (b).

Observer *Observer* works as sensor and describes the present state S. Observation at time t is denoted by O_t, which is described in a language $\mathcal{L}_O$. A description includes all of perceivable information by the agent, such as sensor information and also a time tag.

Planner *Planner* creates a plan from S and a goal. A plan is a series $A_1; A_2; \ldots; A_n (A_i \in \mathcal{A})$ of actions for achieving the goal. The actions are taken from a set $\mathcal{A}$ of available actions for the agent. *Planner* also describes the action as a description $\underline{A_t}$ in a language $\mathcal{L}_A$. It also includes a unique action id aid_t for each action and situation.

Actor An agent acts upon the environment through *Actor*.

In a deterministic environment, the agent is guaranteed to reach a goal when the agent finished all actions under the plan. Under uncertainty, like football games, there is no guarantee. There may be no plan which certainly achieves a goal[3]. Under the circumstances, learning process can help to solve the uncertainty.

Fig. 2 shows an overview of basic agent and ILA architecture. Learner and Checker are added to the basic agent. An ILA adapts its own behavior by predicting the results using rules acquired by inductive learning.

Learner *Learner* acquires prediction rules from results of past behavior using inductive learning.

Checker *Checker* predicts whether a selected action will succeed or not using prediction rules. When an agent forecasts that a selected action will fail, the agent does not take the action and changes its plan to the next plan. If all of the plans are rejected, the agent behaves under the first plan.

Observer and Actor are the same as those of the basic agent. Although Planner of the basic agent makes only one plan, that of ILA should make more than one

[3] Many researches about planning under uncertainty have been reported, such as probabilistic planning [5] [10] and reasoning about plans [8].

plan because Checker may reject the plan. Planner should create more than one plan and choose one from them according to Checker's suggestion. An ILA is given a teacher P in addition to those given to a basic agent.

Since first-order logic is a general-purpose representation language that is based on an ontological commitment to the existence of objects and relations in the world [15], it has been used as the language of agents. We use a first-order language for both $\mathcal{L}_O$ and $\mathcal{L}_A$. By this formalization of languages, we can make Learner use ILP which is an inductive learning framework based on first-order logic.

2.2 Acquisition of Prediction Rules

Since our agent uses inductive learning, that is based on generalization of examples, agent needs examples in order to construct prediction rules. An ILA agent is given a teacher P for classifying examples and uses id's of its own actions as examples. Let $O_0, \ldots, O_T$ and $\underline{A}_0, \ldots, \underline{A}_T$ denote the observations and the action descriptions experienced by an agent during a period $[0, T]$, and let $\mathcal{I}$ denote all of action id's appeared in some action description $\underline{A}_t$ $(t \in [0, T])$. First we construct the following two sets as examples:

$$\mathcal{E}^+ = \{aid_t \in \mathcal{I} \,|\, B \wedge \boldsymbol{O}_T \wedge \underline{\boldsymbol{A}}_T \wedge P \models Succeeded(aid_t)\}\,, \tag{1}$$

$$\mathcal{E}^- = \{aid_t \in \mathcal{I} \,|\, B \wedge \boldsymbol{O}_T \wedge \underline{\boldsymbol{A}}_T \wedge P \not\models Succeeded(aid_t)\}\,, \tag{2}$$

where $\boldsymbol{O}_T = O_0 \wedge \ldots \wedge O_T$, $\underline{\boldsymbol{A}}_T = \underline{A}_0 \wedge \ldots \wedge \underline{A}_T$ and $Succeeded(x)$ represents "x is the id of a succeeded action." $\mathcal{E}^+$ is a set of id's of succeeded actions, and $\mathcal{E}^-$ is a set of failed ones. B is background knowledge, which includes useful knowledge such as knowledge about the field and regulations to play. P is a first-order theory of teacher knowledge which judges $Succeeded(x)$ to be true or not with B, $\boldsymbol{O}_T$, and $\underline{\boldsymbol{A}}_T$.

Hence our inductive learning problem is to find a hypothesis H satisfying:

$$B \wedge \boldsymbol{O}_t \wedge \underline{\boldsymbol{A}}_t \wedge H \models Succeeded(aid_t) \qquad (\forall aid_t \in \mathcal{E}^+)\,, \tag{3}$$

$$B \wedge \boldsymbol{O}_t \wedge \underline{\boldsymbol{A}}_t \wedge H \not\models Succeeded(aid_t) \qquad (\forall aid_t \in \mathcal{E}^-)\,. \tag{4}$$

These formulae say that the hypothesis H correctly classify atoms $Succeeded(aid_t)$ using observations and action descriptions before the time t. Although P may not be able to classify $succeeded(aid_t)$ without information after the time t, H can do it. Accordingly, our agents can predict the result of their actions. While $\boldsymbol{O}_t$ and $\underline{\boldsymbol{A}}_t$ depend on the time t when the example aid_t happened, we can re-formalize them as follows:

$$B' \wedge \boldsymbol{O}_T \wedge \underline{\boldsymbol{A}}_T \wedge H \models Succeeded(aid_t) \qquad (\forall aid_t \in \mathcal{E}^+)\,, \tag{5}$$

$$B' \wedge \boldsymbol{O}_T \wedge \underline{\boldsymbol{A}}_T \wedge H \not\models Succeeded(aid_t) \qquad (\forall aid_t \in \mathcal{E}^-)\,, \tag{6}$$

where B' is a subset of B, the subset which tells nothing about the future observations and actions. We also restrict H to be described in first-order logic

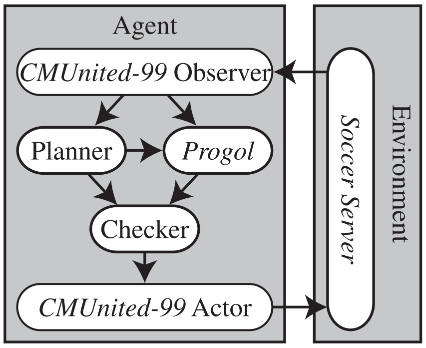


Fig. 3. An overview of ILSA implementation.

with predicate symbols in B' and $Succeeded$. This restriction constructs the hypothesis space. Then, if we define $B' \wedge \boldsymbol{O}_T \wedge \underline{\boldsymbol{A}}_T$ as a new background knowledge $\boldsymbol{B}$, this learning problem is a typical problem of ILP. That is, an ILP problem is to find a hypothesis as a logic program H which satisfies $B \wedge H \models E^+$ and $B \wedge H \not\models E^-$ from positive and negative examples, E^+ and E^-, of a target predicate, and background knowledge B.

2.3 Adaptive Behavior Using Prediction Rules

There are four phases for ILA to adapt its own behavior: (i) collecting examples, (ii) classifying examples, (iii) acquiring prediction rules using ILP, and (iv) acting applying the prediction rules.

First, an ILA acts during a period $[0, T]$. Secondly, the agent creates two sets $\mathcal{E}^+$ and $\mathcal{E}^-$ as Equations (1) and (2). Then, the agent acquires a hypothesis H which satisfies Formulae (5) and (6). In the final phase, the agent takes actions using the hypothesis H. Consider that the agent is going to take an action A'_t at time t' $(t' > T)$. The agent predicts the result of $A_{t'}$ and takes it if it is predicted to be success, that is, when the following formula is satisfied:

$$B' \wedge \boldsymbol{O}_{t'} \wedge \underline{\boldsymbol{A}}_{t'} \wedge H \models Succeeded(aid_{t'}).$$

Otherwise, the agent avoids taking the action and changes to another action under the next plan. When all actions are predicted to be failure, the agent takes action under the first plan.

3 An Implementation in Soccer Agents

We have implemented an inductive learning agent for RoboCup [4] [7] in order to examine our architecture. We call our agent *inductive learning soccer agent* (ILSA). Fig. 3 shows an overview of the ILSA implementation. For demonstrative experiments, we used a program source code of the third RoboCup competition champion simulator team CMUnited-99 [16] [17]. We also used one of successful ILP system Progol [12] version 4.4 as our learning engine. A good point of ILP is that it allows to use the representation power of first-order logic in background knowledge. It has been evidenced acquiring good prediction rules in many fields, and consequently, we expect it to work in the agent learning.

We have prepared a set of actions based on the CMUnited-99 source code. Table 1 shows these actions. We have also prepared background knowledge for Progol shown in Table 2. Most of them are functions used in the CMUnited-99 source code.

`get_ball`	approaching to the ball in order to get it.
`kick_to_goal`	kicking the ball into the goal.
`hold_ball`	keeping the ball near the agent.
`dribble`	dribbling the ball towards the goal.
`pass_ball`	kicking the ball towards a team-mate player.
`receive_pass`	moving to a proper position to receive the ball.
`face_neck_and_body_to_ball`	turning neck and body to the ball.

Table 1. An overview of actions.

`my_body_ang(+time,-angle)`	`uniform_number(+pid,#int)`
`my_neck_rel_ang(+time,-angle)`	`player_distance(+pid,-dist)`
`my_speed(+time,-veloc)`	`player_angle_from_body(+pid,-angle)`
`my_stamina(+time,-stamina)`	`player_angle_from_neck(+pid,-angle)`
`ball_distance(+time,-dist)`	`player_speed(+pid,-veloc)`
`ball_angle_from_body(+time,-angle)`	`ball_kickable_for_player(+pid)`
`ball_angle_from_neck(+time,-angle)`	`gteq/2`
`ball_speed(+time,-veloc)`	`lteq/2`
`ball_moving(+time)`	`=/2`
`ball_kickable(+time)`	`sq_distance(+pid,+pid,-dist)`
`ball_catchable(+time)`	`acted_time(+aid,-time)`
`goal_distance(+time,-dist)`	`selected_action(+time,-aid)`
`goal_angle_from_body(+time,-angle)`	`action_type(+aid,-action)`
`goal_angle_from_neck(+time,-angle)`	`previous_time(+time,-time)`
`known_player(+time,-pid)`	`*next_time(+time,-time)`
`opponent(+pid)`	`*albf/3`
`teammate(+pid)`	

Table 2. A list of predicates in background knowledge for Progol. `+`, `-`, and `#` are mode declarations, which indicate *input* variable, *output* variable, and *constants*, respectively. Predicates with $*$ are used in only B, but not in B'.

We have examined two types of simple agents: a *shooter* agent and a *passer* agent. Public version of the CMUnited-99 source code includes parts of Observer and Actor, but does not include that of Planner. Accordingly, we have prepared simple Planner for each agent. Fig. 4 shows outlines of Planner algorithms for shooter and passer. A shooter agent tries to kick the ball into the goal anywhere at any time if the agent can kick the ball. A passer agent tries to pass the ball to a team-mate player when the agent can kick the ball and knows the team-mate position. In our simple Planner, a plan consists of only an action. The agents have their target actions, `kick_to_goal` for shooter and `pass_ball` for passer. For our experiment, we let them acquire prediction rules only about the target actions.

```
Planner-for-shooter:
begin
  initialize plan[]
  if BallPositionValid then
   if BallKickable then
    plan[1] := kick_to_goal
    plan[2] := dribble
   else
    plan[1] := get_ball
   endif
  else
   plan[1] :=
     face_neck_and_body_to_ball
  endif
end
```

```
Planner-for-passer:
begin
  initialize plan[]
  if BallPositionValid then
   if BallKickable then
    if TeammatePositionValid then
     plan[1] := pass_ball
     plan[2] := hold_ball
    else
     plan[1] := hold_ball
    endif
   else
    if isNearestToBall then
     plan[1] := get_ball
    else
     plan[1] := receive_pass
    endif
   endif
  else
   plan[1] :=
     face_neck_and_body_to_ball
  endif
end
```

Fig. 4. Outlines of Planner algorithms for shooter and passer.

4 Experiments

We have experimented in order to examine ILSA. We have prepared a teacher for each action and the details are described in the following subsections.

4.1 Shooter agent

First we defined three states as follows:

State a: The ball is in the goal.
State b: There is more than one kickable player.
State c: The ball goes out of the field.

Then we gave a shooter the following teacher knowledge P:

$$P = \left\{ \begin{array}{l} \texttt{succeeded(X) :- action\_type(X,kick\_to\_goal), acted\_time(X,Y),} \\ \qquad\qquad\qquad \texttt{can\_kick\_to\_goal(Y).} \\ \texttt{can\_kick\_to\_goal(X) :- albf(X,a,b), albf(X,a,c).} \end{array} \right\},$$

where `albf(x,y,z)` means "after the agent had taken an action `x`, the agent observed a state `y` at least before a state `z`," where `y` and `z` take a state from states a, b, and c.

We made a shooter agent collect examples for 50 minutes which corresponds to five games[4]. We dropped the ball various places and the agent tried to kick it into the goal. During the period, the agent collected 221 examples of actions and classified them into 80 successes and 141 fails using P. The rate of success was 36.2%. Although the agent usually acquires prediction rules from the whole examples, we divided into 10 groups in order to evaluate our induction process with 10 fold cross validation. Table 3 shows the contingency table and the average of the accuracy was $89.6 \pm 5.0\%$, where we allowed 10% of noise as a parameter for Progol.

One of the acquired prediction rules consisted of the following clauses:

```
can_kick_to_goal(A) :- my_speed(A,B), goal_distance(A,C),
                       gteq(B,0.080), lteq(C,18.260).
can_kick_to_goal(A) :- my_speed(A,B), goal_distance(A,C),
                       lteq(B,0.030), lteq(C,8.180).
```

In this time, we concentrated only on one action `kick_to_goal`, and accordingly, we let the agent find `can_kick_to_goal/1` but not `succeeded/1`. The agent can predict whether an action will succeed or not using the clauses combined with the clause:

```
succeeded(X) :- action_type(X,kick_to_goal), acted_time(X,Y),
                can_kick_to_goal(Y).
```

The clauses mean that "if the speed of the agent is greater than or equal to 0.080, the agent can get goal within the distance 18.260 from the goal, or, if it is

[4] Parameters for Soccer Server were as follows: $simulator\_step = 100$, $send\_step = 150$, and $recv\_step = 10$.

	succeeded	failed	Total
predicted to be success	6.6 ± 0.9	0.9 ± 0.7	7.5 ± 1.2
predicted to be failure	1.4 ± 0.9	13.2 ± 0.7	14.6 ± 1.3
Total	8.0 ± 0.0	14.1 ± 0.3	22.1 ± 0.3

average accuracy of 10 rules: $89.6 \pm 5.0\%$

Table 3. Contingency table of the acquired `can_kick_to_goal/1`.

	succeeded	failed	Total
predicted to be success	62.5 ± 10.4	6.7 ± 2.0	69.2 ± 11.3
predicted to be failure	18.5 ± 10.3	143.4 ± 5.7	161.9 ± 12.8
Total	81.0 ± 1.3	150.1 ± 5.8	231.1 ± 6.7

average accuracy of 10 rules: $89.1 \pm 4.2\%$
average rate of successful goals in games: $90.3 \pm 2.4\%$

Table 4. The performance of the learned shooter agent.

less than or equal to 0.030, the agent can get goal within 8.180 from the goal." Note that the background knowledge B' that is used when the agent constructed prediction rules did not include relations `albf/3`, because it requires information in the future in order to decide whether the value is true or false.

For more practical examination we measured the performance of the learned agent. We gave an agent the acquired prediction rules and made the agent behave applying the rules in the similar way. Table 4 shows the result. If the learned agent takes actions only when the prediction rules says they will success, the rate of successful actions rises to 90.3% form 36.2%.

4.2 A Passer Agent

Then we prepared two passer agents and gave them another teacher theory as follows:

$$P = \left\{ \begin{array}{l} \texttt{succeeded(X) :- action\_type(X, pass\_ball), acted\_time(X, Y),} \\ \qquad\qquad\qquad \texttt{can\_pass\_ball(Y).} \\ \texttt{can\_pass\_ball(X) :- albf(X, a, b), albf(X, a, c), albf(X, a, d).} \end{array} \right\},$$

where `a`, `b`, `c`, and `d` are states:

State a: The receiving player is kickable.
State b: There is more than one kickable player, except for the receiving player.
State c: The ball goes out of the field.
State d: Referee judges the receiving player is off-side.

We made them pass the ball each other. For 30 minutes, one of them collected 232 examples: 74 successes and 158 fails, and consequently, the rate of success

	succeeded	failed	Total
predicted to be success	1.5 ± 1.1	0.7 ± 1.0	2.2 ± 1.5
predicted to be failure	5.9 ± 1.1	15.0 ± 1.3	20.9 ± 1.8
Total	7.4 ± 0.5	15.7 ± 0.5	23.1 ± 0.5

average accuracy of 10 rules: $71.4 \pm 6.7\%$

Table 5. Contingency table of the acquired `can_pass_ball/1`.

	succeeded	failed	Total
predicted to be success	4.2 ± 1.8	7.0 ± 3.3	11.2 ± 4.7
predicted to be failure	64.7 ± 3.4	151.9 ± 16.4	221.4 ± 14.5
Total	68.9 ± 3.6	166.1 ± 16.2	235.0 ± 16.8

average accuracy of 10 rules: $69.4 \pm 2.4\%$
average rate of successful passes in games: $38.8 \pm 9.9\%$

Table 6. The performance of the learned passer agent.

was 31.9%. We also divided them and evaluated with 10 fold cross validation. The contingency table is shown in Table 5 and the average of the accuracy was $71.5 \pm 6.8\%$.

One of the acquired prediction rules included the following clauses:

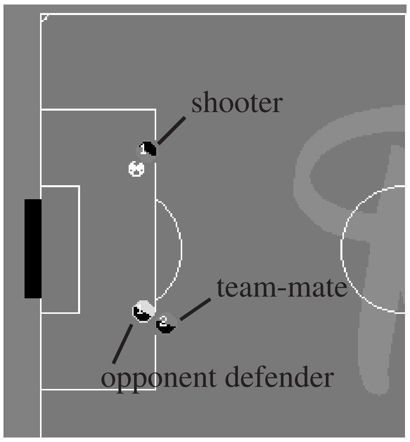
```
can_pass_ball(A) :- my_stamina(A, B), ball_angle_from_neck(A, C),
                    ball_speed(A, D), gteq(B, 1154.400), lteq(C, 0.470),
                    lteq(D, 0.030).
can_pass_ball(A) :- ball_speed(A, B), known_player(A, C),
                    player_distance(C, 22.200), lteq(B, 0.030).
can_pass_ball(A) :- ball_distance(A, 0.710).
```

The first clause means "the agent will be able to pass the ball when the stamina of the agent is greater than or equal to 1154.400 and the angle to the ball from the agent's neck is less than or equal to 0.470 and the speed of the ball is less than or equal to 0.030."

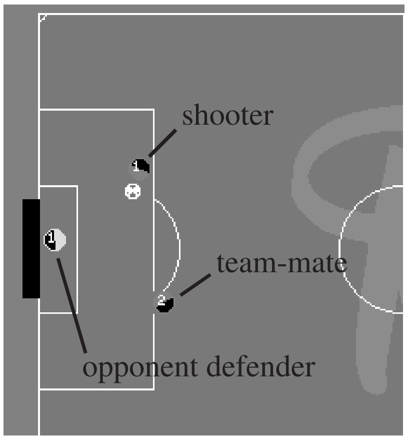
We also measured the performance of the learned agent and the result is shown in Table 6. If the learned agent behaves under the prediction rules, the rate of success rises to 38.8% from 31.9%

4.3 A Shooter Agent in More Complex Situations

We examined shooter agent again in more complex situations. We also prepared a team-mate agent and an opponent defender agent. The team-mate agent waits pass from another agent in front of the goal. The opponent defender agent behaves in two typical ways which are (a) trying to intercept passes to a player



(a)



(b)

Fig. 5. A shooter agent in more complex situations: in case of defender trying to intercept passes (a) and in case of defender trying to intercept the ball moving into the goal (B).

which is waiting the ball, and (b) trying to intercept the ball which is moving into the goal. Fig. 5 shows the typical situations. We dropped the ball at places near a certain place, and let the shooter agent try to take the action **`kick_into_goal`**. When the opponent defender tried to intercept passes to the team-mate player (a), the shooter agent succeeded in getting a goal, but otherwise (b) it failed.

The shooter agent acquired prediction rules from 20 successes and 20 fails. The acquired rules included a clause:

```
can_kick_to_goal(A) :- known_player(A,B), known_player(A,C),
                       teammate(B), opponent(C), sq_distance(B,C,D),
                       lteq(D,5.905).,
```

where `sq_distance(B,C,D)` represents "`D` is the square of the distance between player `B` and player `C`." This clause means "the agent will be able to pass the ball when the agent observes a teammate player and an opponent player, and the square of the distance between them is less than or equal to 5.905."

5 Discussion

The behavior of ILA depends on the rules. Although many good ILP systems have been developed, they cannot learn good rules without good examples. "Good examples" means examples properly classified. Accordingly, we must give ILA a teacher theory carefully in order to classify them autonomously. ILA architecture allows us to give any hand-coded teacher theory for this process.

Now consider the performance of learned agents. Passer agent could not improve the performance as much as shooter agent. The main reason is that there are too many variations of the situations. That is, passer agent almost always faced with new situations during the experiment. If agent deals with similar situations as one situation, the number of situations can be reduced. However, it will also cause a problem that agent cannot find little differences between success and failure. On the other hand, We should concentrate on adapting behavior rather than raising the accuracy, because it is impossible to predict all the result of actions perfectly. Even if an agent does not have prediction rules with high accuracy, the agent can adapt its own behavior by avoiding typical failures using the rules. In order to acquire such rules using ILP, we need to choose typical examples from all of situations.

Our agent uses first-order representation, and consequently, it is natural to use ILP for the agent's learning process. Notice that ILA may acquire rules which include relations such as a clause shown in Subsection 4.3. Although attribute-value learning such as decision tree learning cannot deal with relations, ILA can acquire prediction rules which include relations by using ILP for the learning process.

6 Related Works

Learning actions in soccer agents have been researched. Reinforcement learning has been succeeded in learning low-level, individual skills [2] and learning functions for action selection in multi-agent environment [18]. Genetic programming has also been used to learn action selection function [9]. However, these method cannot use first-order logic and we cannot give agents knowledge easily.

An autonomous agent TRAIL using ILP has also been reported [3]. However, this agent differs from ours in a planning way. TRAIL makes a plan based on programs acquired using ILP. On the other hand, our agent makes some plans using given programs. We can give hand-coded programs or programs learned using the other effective methods.

Another research has been reported, which uses ILP systems for verifying and validating multi-agents for RoboCup [6]. This concentrates on verification and validation of knowledge based system, not but prediction or discovery of new knowledge. Consequently, agents cannot adapt their own behavior using rules or knowledge acquired by ILP.

This work builds upon earlier ideas on using ILP to adapt behavior of software agents for RoboCup [11]. It does not include formalization, but only another simple implementation of ILSA.

7 Conclusion

We proposed new architecture of intelligent agent which can adapt its own behavior by predicting the results, and we name it inductive learning agent (ILA). ILA treats actions in human tractable levels. It causes that we can give ILA a teacher easily and we need not think low-level behavior. ILA uses first-order logic for its observation language, and consequently, ILA can use ILP as the learning method.

The good points of using ILA architecture are as follows.

- Agent can learn symbolic and relational knowledge at a strategy level.
- Agent can use first-order background knowledge and we can describe our knowledge easily.
- Being accorded with basic agent architecture, we have possibility to use other technologies developed in it.

Actually, we left motor control and knowledge representation to parts of the latest champion CMUnited-99. We also used an ILP system Progol to acquire prediction rules. We have implemented soccer agents using this architecture, and confirmed the utility. Our agents have become to use actions of CMUnited-99 more effectively after the agents acquired prediction rules. Since the agents learn with respect to the other parts in our architecture, the agents can improve the performance of them.

Our future works include increasing refinement of prediction by iterative acquisition of rules, real-time acquisition of prediction rules, and adapting cooperative behavior by cooperative learning with other player's agents.

References

1. D. Andre, E. Corten, K. Dorer, P. Gugenberger, M. Joldos, J. Kummeneje, P. A. Navratil, I. Noda, P. Riley, P. Stone, T. Takahashi, and T. Yeap. *Soccerserver Manual*, 1999. Ver. 4 Rev. 02.
2. M. Asada, H. Uchibe, and K. Hosoda. Cooperative behavior acquisition for mobile robots in dynamically changing real worlds via vision-based reinforcement learning and development. *Artificial Intelligence*, 110:275–292, 1999.
3. S. S. Benson. *Learning action models for reactive autonomous angets.* Ph.D. thesis, Stanford University, 1996.
4. The RoboCup Federation. RoboCup Official Site, Since 1998. http://www.robocup.org/.
5. E. Guéré and R. Alami. A possibilistic planner that deals with non-determinism and contingency. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*, volume 2, pages 996–1001, 1999.
6. N. Jacobs, K. Driessens, and L. D. Raedt. Using ilp-systems for verification and validation of multi-agent systems. In D. Page, editor, *Proceedings of the 8th International Conference on Inductive Logic Programming (ILP-98).* Springer Verlag, 1998.
7. H. Kitano, M. Tambe, P. Stone, M. Veloso, S. Coradeschi, E. Osawa, H. Matsubara, and M. Asada. The RoboCup Synthetic Agent Challenge 97. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intellegence*, volume 1, pages 24–29, 1997.
8. W. Łukaszewicz and E. Madalińska-Bugaj. Reasoning about plans. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence*, volume 2, pages 1215–1220, 1997.

9. S. Luke. Genetic programming produced competitive soccer softbot teams for RoboCup97. In *Proceedings of the Third Annual Genetic Programming Conference (GP98)*, pages 204–222. San Fransisco: Morgan Kaufmann, 1998.
10. S. M. Majercik and M. L. Littman. Contingent planning under uncertainty via stochastic satisfiability. In *AAAI-99/IAAI-99 Proceedings*, pages 549–556, 1999.
11. T. Matsui, K. Kashiwabara, N. Inuzuka, H. Seki, and H. Itoh. Soccer agents learning from past behavior with inductive logic programming. In *Proceedings of the IMC Workshop on Knowledge Mining in the Real-world*, pages 9:1–12, 1999.
12. S. Muggleton. Inverse entailment and Progol. *New Generation Computing*, 13:245–286, 1995.
13. I. Noda, H. Matsubara, K. Hiraki, and I. Frank. Soccer server: A tool for research on multiagent systems. *Applied Artificial Intelligence*, 12:233–250, 1998.
14. S. Russell and P. Norvig. *Artificial Intelligence, A Modern Approach*, chapter 2. Prentice Hall, 1995.
15. S. Russell and P. Norvig. *Artificial Intelligence, A Modern Approach*, chapter 7. Prentice Hall, 1995.
16. P. Stone, P. Riley, and M. Veloso. CMUnited-99 source code, 1999. http://www.cs.cmu.edu/~pstone/RoboCup/CMUnited99-sim.html.
17. P. Stone, P. Riley, and M. Veloso. The CMUnited-99 champion simulator team. In M. Veloso, E. Pagello, and H. Kitano, editors, *RoboCup-99: Robot Soccer World Cup III*, Springer Verlag, 2000.
18. P. Stone and M. Veloso. Team-partitioned, opaque-transition reinforcement learning. In M. Asada and H. Kitano, editors, *Proceedings of the Third International Conference on Autonomous Agents*. Springer Verlag, 1999.