Computing and Informatics, Vol. 28, 2009, 57–80

CAOS COACH 2006 SIMULATION TEAM: AN OPPONENT MODELLING APPROACH

José Antonio Iglesias, Agapito Ledezma, Araceli Sanchis

Universidad Carlos III de Madrid
Avda. de la Universidad, 30
28911, Leganés, Madrid, Spain
e-mail: {jiglesia, ledezma, masm}@inf.uc3m.es

Manuscript received 8 January 2007; revised 22 January 2008
Communicated by Baltazár Frankovič

Abstract. Agent technology represents a very interesting new means for analyzing, designing and building complex software systems. Nowadays, agent modelling in multi-agent systems is increasingly becoming more complex and significant. *RoboCup Coach Competition* is an exciting competition in the RoboCup Soccer League and its main goal is to encourage research in multiagent modelling. This paper describes a novel method used by the team CAOS (*CAOS Coach 2006 Simulation Team*) in this competition. The objective of the team is to model successfully the behaviour of a multi-agent system.

Keywords: Team behavior, opponent modelling, robocup, coach simulation, multiagent system

Mathematics Subject Classification 2000: 68T05

1 INTRODUCTION

Agent modelling techniques have been used from the beginning of Artificial Intelligence (AI). Perhaps one of the most interesting environments where agent modelling has been satisfactorily studied is the robotic soccer domain [1]. Originated by Alan Mackworth [2], soccer domain was introduced as a great challenge for AI [3]. From then on, many researches have been done in this area.

The Robot World Cup Initiative (RoboCup) [4] is an international joint project to encourage AI, robotics and related fields. RoboCup provides a standard problem where many intelligent techniques must be integrated and examined. RoboCup chooses to use soccer game as a central topic of research, aiming at innovations to be applied for socially significant problems and industries [5].

One of the competition leagues in RoboCup is the *Simulation League* [6] which was introduced in 1995 [7]. From then, a lot of advances have been made. In this league there are three competitions with different goals: *2D*, *3D* and *Coach*. In the *2D Competition*, a soccer game is performed and technologies like multi-agent collaboration or strategy acquisition are incorporated. The *3D Competition* makes use of a simulator for physical multi-agent simulations. And, the last *Coach Competition* mainly dealt with an agent (coach) able to model a team of heterogeneous agents.

The first *Coach Competition* was held in 2001. In this competition, an online coach was able to act as an advice-giving agent [8]. In order to improve the behaviour of the coached team, the coach could receive a global view of the world environment and could communicate with the team [9]. However, *RoboCup Coach Competition* changed recently in order to emphasize opponent-modelling approaches. The main goal of the current competition is to model the behavior of a team. A play pattern (way of playing soccer) is activated in a test team and the coach should detect this pattern and then recognize the patterns followed by a team by observation. Considering this competition, our work on opponent modelling is driven by the goal to model the behavior of a soccer team by observing its players.

This paper introduces a comparing process of agents behaviours as a general framework and it could be used in many different multi-agent systems. Our proposal is implemented and evaluated in the coach soccer domain, specifically by the participation in the *RoboCup 2006 Coach Competition*.

This paper is organized as follows: Section 2 introduces the concept of agent modelling and coaching. Section 3 describes a short introduction to RoboCup, especially to Coach Competition. A summary of our approach is presented in Section 4. Sections 5 and 6 describe in detail how our team has treated the two parts of the competition: the offline analysis and the online recognition. The experimental results are shown in Section 7. Section 8 presents the related work and finally, conclusions and future works are drawn in Section 9.

2 AGENT MODELLING AND COACHING

The idea of extracting knowledge from other agents' behaviour was used, originally, in the game theory field [10]. After not considering information about opponents in game mechanics, it was realized that knowledge about the opponent's strategy can enhance the game results. Following this first advance, many others researchers have been working to develop different ways of modelling other agents: Carmel and Markovitch [11] proposed a method to infer a model of the opponent's strategy

which is represented as a Deterministic Finite Automaton (DFA). Tambe et al. [12] presented an approach for tracking agent models based on a plan recognition task: Agent tracking involves monitoring the observable actions of other agents as well as inferring their unobserved actions, plans, goals and behaviours.

There are many papers and studies related to agent modelling in this robotic soccer domain: Stone et al. [13] introduce "ideal-model-based behaviour outcome prediction" (IMBBOP) which models the result of the other agents' future actions in relation to their optimal actions based on an ideal world model. But, this first work into automated opponent-modelling assumes that the opponent plays optimally. Ledezma et al. [14] present an approach to modelling low-level behaviour of individual opponent agents: OMBO (Opponent Modelling Based on Observation). Druecker et al. [15] develop a neural network which is fed with the observed player positions and tries to classify them into a predefined set of formations. Riley et al. [16] propose a classification of the current adversary into predefined adversary classes. Based on Riley's work, Steffens [17] presents a feature-based declarative opponent-modelling (FBDOM) which identifies tactical moves of the opponent.

About the Coaching problem raised in Coach Competition, Riley et al. [18] present a general description of the coaching problem. This description is the first step in understanding advice-based relationships between automated agents. In Riley's research, one of the reasons to consider a coach role in a team of autonomous agents is that a coach role provides a method of oversight for the agents and can aid in the creation of agents with adjustable autonomy [19]. Also, Riley et al. [20] justify that coaching can help teams improve in a simulated robotic soccer domain.

3 ROBOCUP: COACH COMPETITION

RoboCup is a continuing AI research initiative that uses the game of soccer as a unifying and motivating domain [21]. The *RoboCup Competition* includes several robot leagues, a humanoid league and a simulation league [22]. One of the challenges of the simulation league for the near future is improving the opponent modelling, a research which can be used both in RoboCup domain and in general multi-agent systems.

3.1 The Soccer Server Simulator

The *RoboCup Simulation League* uses the *Soccer Server System* [23] to simulate the conditions of a real soccer match. Matches are played in a client/server environment. Each player consists of a unique process that connects via a standard network protocol to the server. This server keeps track of the current state of the world, executes the actions requested by the clients, and periodically sends each agent noisy information about the world. There are 2 teams and 11 players per team (10 fielders and 1 goalie) that can only perceive objects that are in their field of vision and both the visual information and the execution of the actions are noisy. Also, the server

is a real-time system working with discrete time intervals (or cycles). The default length for each match is 6000 simulation cycles (around 10 minutes).

3.2 The Coach Competition

From 2001 to 2004, the main goal of the *RoboCup Coach Competition* was to create an automated coach able to advise a team of autonomous agents to perform better its behaviour in front of an opponent [24]. In this case, in addition to the players, each team may connect a privileged client to the server: the coach agent. This special agent gets global and noiseless information from the Soccer Server about the position and speed of all players and the ball. Also, this coach can only support its team by giving messages to its player in a standard coach language called CLang, which was developed by members of the simulated soccer community [23].

The *Coach Competition* structure changed in 2005. Before going to the following description, the definition of some phrases is presented:

Play Pattern: The term "play pattern" is used to describe a simple behaviour that a team performs which is predictable and exploitable for the coach. In this paper we use the term "pattern" as a contraction of play pattern.

Base Strategy: The general strategy of the test team regardless of the pattern in it.

According to the *RoboCup 2006 Coach Competition* official rules [25], previously to the competition, a set of strategies to be used as the basic strategies of the patterns are created by the organizing committee and some matches are played (*no-pattern log files*). Then, the patterns are added to these basic strategies, and some sample matches are played again (*pattern log files*). Many pairs of log-files (*pattern log file* and its corresponding *no-pattern log file*) are created.

At the beginning of the competition, each coach team participant is provided with some[1] pattern game log-files (only one pattern is activated in a log-file) and its corresponding no-pattern game log-files (games with the same base strategy but the pattern not activated). The main goal of the current competition is to look for the qualitative differences among the pattern log file and its corresponding no-pattern log file. The coach should detect the patterns followed by the test team in the pattern log files and report them. Also, once every pattern has been detected and stored, the coach should recognize them by observing a live game. Therefore, this competition consists of two phases [25]:

Offline Analysis: The inputs of this phase are several *pattern log files* and their corresponding *no-pattern log files*. The coach has to look for the qualitative differences between the two log files in order to detect the pattern. The output of this analysis are some files where the specifications of every pattern are recorded.

[1] The number of pairs of log files received by the coach is around 20.

Online Recognition: The coach observes a live game where some patterns have been activated in the test team. The coach should recognize online the patterns activated in the test team in a 6 000 cycle and report them to the simulator. The sooner the coach sends the report, the more score it gets.

The figure 1 shows the overview structure of the RoboCup Coach Competition.

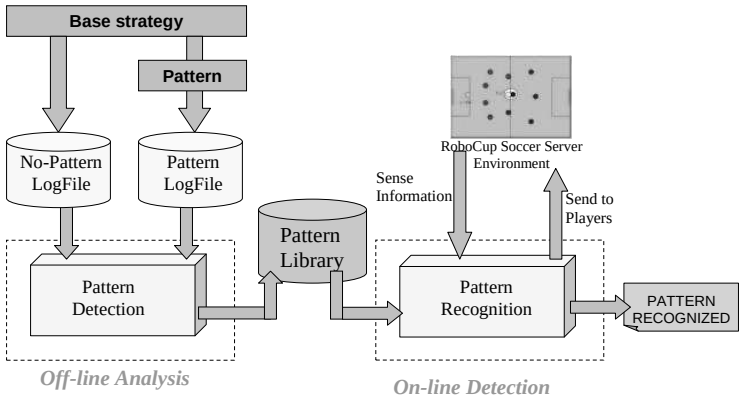

Fig. 1. The RoboCup Coach Competition. Overview structure.

Hence, the performance of a given coach is based only on its ability to detect and report patterns. The research focus is on team/opponent modelling and online recognition. The coaches work both by analyzing logs of previous games and by observing and recognizing while a game is proceeding.

4 OUR APPROACH

In this section, our approach in the two phases of the competition (Offline Analysis and Online Recognition) is described.

4.1 Offline Analysis

According to the *RoboCup 2006 Coach Competition* official rules [25], the coach receives two different game log files for each pattern to detect. Then, it looks for the qualitative differences between these two log files. This difference is considered as the pattern followed by the test team and it is stored in a file.

In our approach, a *special* event structure (that will be explained in detail in Section 5.1.3) is created for the two game log-files and then, these structures are compared in order to extract the pattern. This pattern detection task is divided into two different parts: *Behavior Extraction* and *Behavior Comparison*. These parts are shown in Figure 2.

The input of the first part (*Behavior Extraction*) is a pair of log files: the *pattern log file* and its corresponding *no-pattern log file*. This part consists of two threads. The input of the first thread is the *pattern log file*, and the second thread receives the *non-pattern log file*. These two threads are executed in parallel and each one consists of three processes: *Features Extraction*, *Events Recognition* and *Events Structure*. The output of each thread is the above-mentioned *special* events structure.

These two thread outputs are the inputs of the second part: Behavior Comparison. In order to detect the pattern, the two events structures are compared in this part. If the pattern is recognized, it is stored in a file (in a special way).

Also, this process is carried out once per each pattern to detect (each pair of log files). Figure 2 describes the Offline Analysis proposed by our CAOS Coach Team.

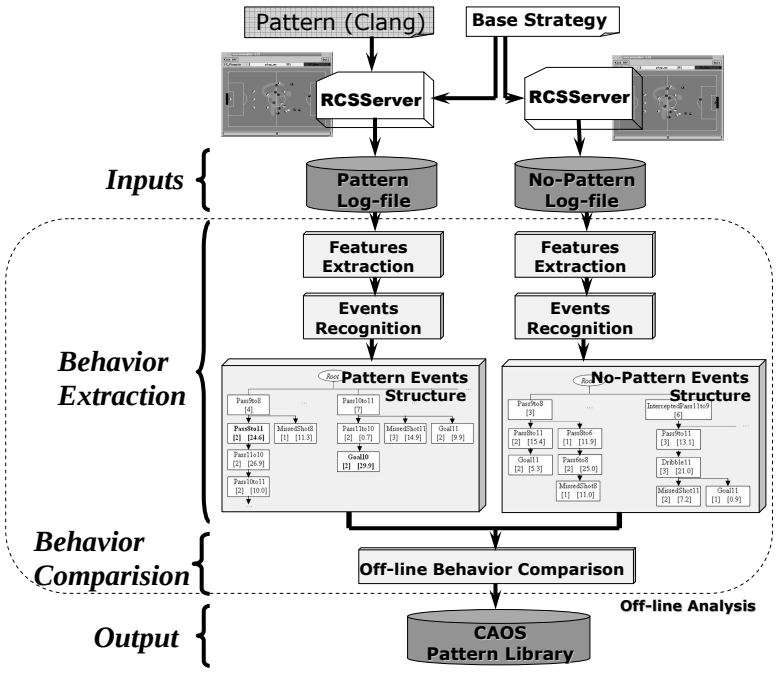

Fig. 2. Offline Analysis. Overview structure.

4.2 Online Recognition

Once every pattern has been stored in the Pattern Library, a live game is observed and the patterns followed by the opponent team must be recognized. During the live game, the coach receives data from the server. In our approach, this data is treated like in the previous phase, considering the three parts: Features Extraction, Events Recognition and Events Structure. As in the previous phase, the result of this

process is a special event structure that is matched in an online comparing method with the patterns stored in the Pattern Library. If the similarity is considerable, the pattern is recognized and reported.

Figure 3 shows the overview structure of the online phase.

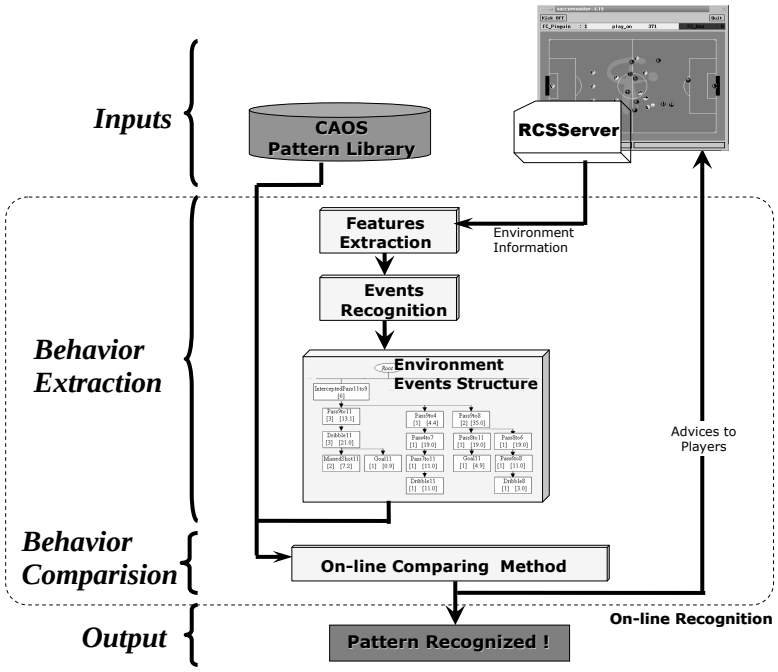

Fig. 3. Online Recognition. Overview structure.

5 OFFLINE ANALYSIS

In this section is explained in detail how the first part of the competitionL: Offline Analysis has been treated and developed by our CAOS Coach Team. In this research, the patterns recognized are related to players' actions. There are other kinds of patterns (the different field regions in which the action occurs or the cycle when some actions occur) that are not detected by our current approach.

The process explained in this section is carried out once per each pattern to detect, that is once per each pair of log files (*pattern* and *no-pattern log files*) received. The two main parts of this process are Behavior Extraction and Behavior Comparison.

5.1 Behavior Extraction

The goal of this first process is to extract the important information from log files, store it in a useful way and classify the behavior of the team. As we can see in Figure 2, this process consists of three parts: *Features Extraction*, *Events Recognition* and *Events structure*.

5.1.1 Features Extraction

Kuhlmann et al. [24] describe a procedure to identify high-level events in a soccer game. An event represents a recognized atomic behaviour. The goal of their work is to get a coach that learns to predict opponent agents' behaviour from past observations and generates advices to improve its team's performance. Based on that work, we carried out two processes: feature extraction and event recognition. Unlike Kuhlmann's work, we use the result of these processes to describe the behaviour of a team.

When running the *Soccer Server*, certain options can be used to store the data of every player in every cycle for a given match. This information is recorded in a log file, so this file is a special stream of consecutive information data. The coach receives these log files and the important features over all the information of the match are extracted.

At every cycle of the server, each agent updates its world model with data such as positions and velocities, as well as cumulative data such total travel distances and average positions. In this first phase, the most relevant information of these log files must be stored, so in every cycle the following data is extracted:

Cycle: A number that enables arrange the events.

Ball Position: Represented by a point in the Cartesian coordinate system.

Teammate Positions and Opponent Position: Each teammate and opponent position is represented by a point in the Cartesian coordinate system.

Ball Possessor: A number that indicates who is the owner of the ball.

5.1.2 Events Recognition

After extracting the data from the logfiles, it must be inferred what events have occurred. In the work mentioned above, Kuhlmann et al. [24] propose nine different events to create advices in the Simulation Soccer. In our work, seven events are identified but the way we use them is very different.There is some uncertainty inherent in the results because there are events very hard to identify, even by a soccer expert.

One of the most important data used to identify high-level events is to know which player is the ball owner in each cycle. When a ball possession change occurs, it means that an event is taking place. In our research, the following events are classified:

PassXtoY (T): If a player X of the team T kicks the ball and a teammate Y gains possession, then the ball owner made a pass. Perhaps the ball owner did not want to do this pass, but we can not consider this assumption. This event stores both the player who makes the pass and the player who gains possession.

DribbleX (T): If the ball moves a significant distance since the player X gains possession until he kicks it.

InterceptedPassXtoY (T): If the player X kicks the ball and the opponent Y gains possession within a reasonable distance of the ball owner, the event is assumed to be an intercepted pass.

StealXfromY (T): If a player X kicks the ball and an opponent Y gains possession, then the opponent stole the ball from the ball owner.

GoalX (T): If a player X kicks the ball and at the end of the interval, the ball is in the goal, the event is classified as a goal by the player X.

MissedShotX (T): If a player X kicks the ball and at the end of the interval, the ball is out of bounds, the kick is considered as a missed shot.

Clear (T): If the event cannot be classified as any of the above categories.

The result of this phase is a set of events ordered by time. Each event is labelled with a letter (L-Left or R-Right) that indicates the team that executed the event. Every stream of events of a same team is called behaviour sequence and it may look as follows:

$$\{Pass1to2(R) \rightarrow Dribble2(R) \rightarrow Pass2to10(R) \rightarrow Goal10(R)\},$$

$$\{Pass7to10(L) \rightarrow MissedShot10(L)\}$$

5.1.3 Events Structure

Once the necessary features have been extracted from the log files and every event has been recognized, the next step is to analyze the behaviour sequences and choose the sequences more related to the pattern. In this and next sections, the behaviour sequences that describe the pattern followed by the team must be identified.

As explained in Section 3.2, a pattern describes a simple behaviour that a team performs. As this pattern must be predictable for the coaches, we consider that the repeating events of the behaviour sequences and its dependence are related to the activated pattern. Because of this supposition, in this paper we propose the use of a trie structure data [26, 27] to store the results of the event recognition.

A trie (which is abbreviated from "retrieval") is a kind of search tree similar to the data structure commonly used for page tables in virtual memory systems. This special search tree is used for storing strings in which there is one node for every common prefix and the strings are stored in extra leaf nodes.

Although tries have been used for retrieving a string efficiently from a set of strings, in this research we propose to use this data structure for a different goal:

to store the behaviour sequences in an effective way. The advantage of this kind of data structure is that every sequence of events is stored in the trie just once, in a way that each event has a number that indicates how many times it has been inserted on.

The work by Kaminka [28] uses the trie data structure to learn the coordinated sequential behaviour of teams. Also, Huang et al. [29] try to create frequent patterns in dynamic scenes using the same data structure.

In this work, every event is represented as a node in the trie structure. A behaviour sequence represents the events related to a team since the team gains possession of the ball until the team loses it. Therefore, a path from the root to a node represents a sequence of events of a team in the order they were played.

An example of how to store an events sequence in the trie data structure is shown as follows. Each events sequence represents the team actions made by a team while it has possession of the ball. In this example, the trie is initially empty and the event sequences obtained from the previous phase (event recognition) are:

- First sequence:

$$\{Pass1to2(R) \rightarrow Dribble2(R) \rightarrow Pass2to10(R) \rightarrow Pass10to11(R)\}.$$

- Second sequence:

$$\{Dribble2(R) \rightarrow Pass2to10(R) \rightarrow Goal10(R)\}.$$

The first event sequence is added as the first branch of the tree. Each event is labelled with number 1 that indicates that the event has been inserted in that sequence once (in Figure 4, this number is enclosed in brackets). At this step, the trie created is as shown in Figure 4:

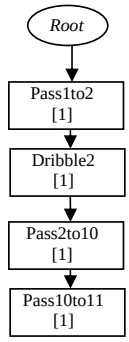

Fig. 4. Example of a trie representation (first step)

Because of repeating events sequences are important, the suffixes of the sequence (until a sequence of two events is obtained) are also added to the trie. In this

example, the two remaining suffixes of the sequence to insert are:

$$\{Dribble2(R) \rightarrow Pass2to10(R) \rightarrow Pass10to11(R)\}$$

and

$$\{Pass2to10(R) \rightarrow Pass10to11(R)\}.$$

After inserting these sequences, the trie is modified as shown in Figure 5.

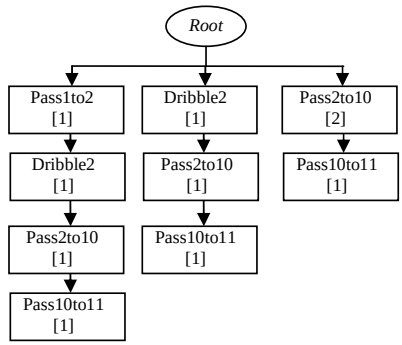

Fig. 5. Example of a trie representation (second step)

Then, the second sequence $\{Dribble2\ (R) \rightarrow Pass2to10(R) \rightarrow Goal10\ (R)\}$ and its remaining suffix: $\{Pass2to10(R) \rightarrow Goal10\ (R)\}$ are inserted. In this case, there exist sequences like these already inserted in the trie, so the counter number of the events is increased by one.

Ater inserting every sequence, the complete trie for the right team is shown in Figure 6.

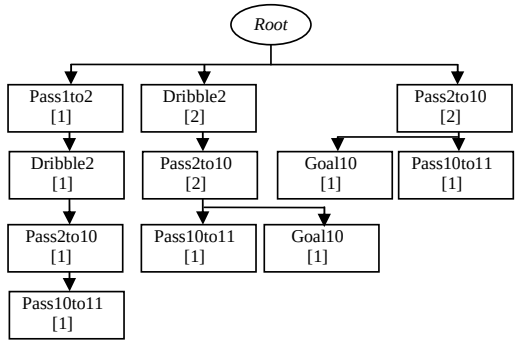

Fig. 6. Example of a trie representation (complete)

In a simulated game, the trie obtained from a log file is quite big, so in order to get the most important information and because the behaviour sequences must be

repeated to be able to extract the pattern, the tries are pruned. For pruning a trie, we eliminate the branches which have been introduced in the trie only once.

5.1.4 Evaluating Dependence

Because of the behaviour of a team has been divided into different events, a pattern could be defined by the dependence of an event and its previous ones. Also, repetitive sequences may indicate a pattern. The main idea of this process is to get the repeated relations between events in order to get a pattern.

Although there are many methods for discovering dependences between sequences and sub-sequences, in this work a statistical dependency test [29] is used. Therefore, to evaluate the relation between an event and its prefix (sequence of events previous to an event), one of the most popular statistics – the Chi-square test [30] is used. This statistical test enables to compare observed and expected sequences objectively and to evaluate whether a deviation appears. Hence, every event of the trie stores a value that determines whether an event is relevant with the previous ones in its behaviour sequence or not.

To compute this test, a 2×2 contingency table is necessary (also known as a cross-tabulation table). This table is filled with four frequency counts, as we can see in the Table 1. The counts are calculated as follows: The first number O_{11} indicates how many times the current node/event is following its prefix. The number O_{12} indicates how many times the same prefix is followed by a different event. The number O_{21} indicates how many times a different prefix of the same length is followed by the same event. The number O_{22} indicates how many times a different prefix of the same size is followed by a different event.

	Event	Different event	Total
Prefix	O_{11}	O_{12}	$O_{11} + O_{12}$
Different prefix	O_{21}	O_{22}	$O_{21} + O_{22}$
Total	$O_{11} + O_{21}$	$O_{12} + O_{22}$	$O_{11} + O_{12} + O_{21} + O_{22}$

Table 1. The contingency table

The expected values are calculated by Equation (1).

$$Expected(E_{ij}) = \frac{(Row_i Total x Column_i Total)}{GrandTotal}. \tag{1}$$

The formula to calculate chi-squared value is given in Equation (2)

$$X^2 = \sum_{i=1}^{r} \sum_{j=1}^{k} \frac{(O_{ij} - E_{ij})^2}{E_{ij}} \tag{2}$$

where O_{ij} is the observed frequency and E_{ij} is the expected frequency.

This value is calculated for each event of the trie. Hence, the structure obtained in Section 3.1.3 is modified (except the nodes of level 1 and the root) in order to include this value in every node.

5.2 Comparing Process

A very interesting part in our research is to provide a method to compare two agent teams' behaviours. This section explains the process carried out for our approach.

The inputs of this process are the two tries obtained from the previous phase. In this process, the term *pattern-trie* is used to refer to the trie obtained from the test team which followed a pattern, and *no-pattern-trie* terms the trie for the test team that did not follow the pattern. As the two tries represent the same team base strategy but one of them (*pattern-trie*) represents also the pattern followed by the team, the result of the comparison, that is, the pattern, must be the difference between the trees.

The output of this process and the input of the offline analysis is the description of the pattern detected. The pattern description is stored in a file called *Pattern Library*.

Before describing the comparing method, it is presented how to store the result of the comparison: our pattern description.

5.2.1 Our Pattern Description

In this work, a pattern defines recurring events to a recurring prefixes. Also, a pattern could consist of a set of simple sub-patterns. Because of this reason, our pattern description consists of a set of sub-patterns. A sub-pattern is defined as the possibility, measured by chi-square test value (*chi-sq*), that an event (E) occurs after a prefix (P). Let OurPDescription = $\{[\text{sub} - p_1], [\text{sub} - p_2], \ldots\}$ be the set of all sub-patterns $\text{sub} - p_i$. A sub-pattern is defined as follows:

$$\text{sub} - p_i = (E, P, \textit{chi-sq}).$$

The chosen description of the pattern is very significant because it describes the result of the offline analysis and it is used in the online recognition.

5.2.2 Off-Line Comparing Algorithm

The algorithm to compare two tries (*pattern-trie* and *no-pattern-trie*) used to get the pattern description that the *pattern-trie* follows is described in this section. Before describing the algorithm, the following concepts about the trie data structure must be considered:

- As described in Section 3.1.4, every node is represented by:

  Event: A word that indicates a specific event and the agent(s) that take part in it (*passXtoY*, *InterceptedPassXtoY*, etc.).

  Prefix: A set of the previous events in the trie.

  Chi-Sq: A number that indicates the chi-square test value for the node.

- The *depth* of a trie is the maximum depth of any of its leaves.
- The *root* in the trie is the node of level 0.

The main features of the algorithm are:

1. A threshold value (*ThresholdChiSqValue*) has to be established for accept or reject the chi-square test hypothesis in the events.
2. If the event of a node is represented in both tries at the same level with the same prefix: Chi-square value of both nodes is compared. Only if the difference is higher than a threshold value, the information of the node of the *pattern-trie* is stored as a sub-pattern in the pattern description. It means that a different behaviour in the trie has been found (and it is classified as a sub-pattern).
3. If the event and prefix of a node are represented only in the *pattern_trie*: If the chi-square value of the event is higher than a threshold value (*ThresholdChiSq-Value*), the information of the node is stored as a sub-pattern in the pattern description.

Also, in order to make the proposed algorithm efficient and more comprehensible, the following functions are used:

depthTrie(Trie T): This function returns the maximum depth of any of its leaves.

getSetOfNodes (Level L, Trie T): This function retrieves a result set that contains every node of the trie T in the level L.

getNode(Event E, Prefix P, SetOfNodes S): This function returns a node consisting of the event E and whose prefix is P, and is obtained from the set of nodes S. (If a node with these parameters does not exist in the trie T, the function returns *null*).

chi-Sq(node N): This function returns the chi-square value of the node N.

Prefix(node N): This function returns the prefix (set of events) of the node N.

AddToOurPatternDesc(*Event E, Prefix P, Chi-Sq Chi-SqValue*): This function adds description the new sub-pattern consisting of the event E, the prefix P and the chi-square value *Chi-SqValue* to our pattern.

The basic structure of the algorithm is shown in Figure 7.

6 ONLINE RECOGNITION

Once every game has been analyzed and the pattern library has been filled in, the patterns must be recognized by observing a live game. Therefore, the main goal of this section is to recognize (from the detected patterns in the previous phase) the behaviour (set of patterns) followed by the opponent team by observing a live game.

As described in the *2006 Coach Competition* official rules [25], in this phase the coach is connected, in online mode, to the RoboCup Soccer Server. As we could see

```
CompareTries (pattern_trie,  no_pattern_trie)
  begin
     patternDesc ← null
     maxDepth ← depthTrie (pattern_trie)

    for level_i ← 2 to maxDepth do
       {Root is considered in level 0 and the events in level 1 have no prefix}

      set_P ← getSetOfNodes (level_i, pattern_trie)
      set_NP ← getSetOfNodes (level_i, no_pattern_trie)

         set_P ← getSetOfNodes (level_i, pattern_trie)
         set_NP ← getSetOfNodes (level_i, no_pattern_trie)

         for all node_p in Set_P  do
            node_np ← getNode (event(node_p); prefix(node_p); set_NP)
            if (node_np = null)
            {The node is only in the pattern_trie}

               if (chi-Sq(node_p) > ThresholdChiSqValue)
                  AddToPatternDesc (event(node_p), prefix(node_p), chi_sq(node_p))
               fi

            else (if node_np ≠ null)

               if (chi_sq(node_p)>ThresholdChiSqValue + chi_sq(node_np))
               {The node is in both tries, so the difference between them must be considered}
                 DifChiSqValue = chi_sq(node_p)- chi_sq(node_np)
                 patternDesc←AddToPatternDesc(event(node_p),prefix(node_p), DifChiSqValue)
               fi

            fi
         od
    od
    return (patternDesc)
  end
```

Fig. 7. Basic structure of the algorithm *CompareTries*

in Figure 3, the inputs of this part are the output of the offline analysis (the Pattern Library) and the environment information of the live game. Then, the coach receives data of every player and ball from the server. This data is analyzed for matching it with the patterns stored in the Pattern Library. The matching is done by comparing every pattern description (event, prefix and chi-square value) with the events and prefix of the created trie. If the similarity is *considerable* (higher than a threshold), the pattern is recognized and reported.

6.1 Creating the Online Trie

Once the coach is connected to the RoboCup Soccer Server, it receives global *see messages* (environment data). The data of these messages is very similar to the data obtained from the log files, although the format is very different. Hence, in order to analyze the behaviour of the opponent, the same method used in the offline analysis can be applied in this phase: the environment information is obtained from the server, then the features are extracted and the events are recognized and finally, the trie (*behavior detection trie*) is created in the same way than the previous phase.

6.2 Online Comparing Algorithm

As explained in Section 5.2.1, a pattern consists of sub-patterns. Also, every sub-pattern is stored with its prefix and its dependence value (chi-square). In this phase, a trie (*on-line trie*) is created for matching the patterns store in the Pattern Library with the patterns created during the live game.

Because the on-line trie does not change in every cycle, it is only created a few times per match, exactly 6 times. This value has been chosen because the length of a match is always 6 000 cycles and at least 1 000 cycles are needed to get a significantly different trie.

Every 1 000 cycles, an *on-line trie* is created with the events recognized from the server data. The way to create patterns from this tree is the same as in the previous phase. The patterns created are matched with every pattern stored in the Pattern Library. A pattern P is recognized if the similarity of its sub-patterns and the sub-patterns of a pattern stored in the Pattern Library is higher than the threshold value.

7 EXPERIMENTAL RESULTS

This section is divided into two parts: In the first one, the experiments carried out to test our method are explained. In the second one, the result of our team in the RoboCup 2006 Coach Competition is analyzed.

7.1 Previous Experiments

We carried out a set of experiments in order to test whether our method is able to compare a game played by a test soccer team which follows a pattern to a game in which the test team does not follow that pattern. Also, the result must be a description of the detected pattern.

Robocup 2005 Coach Competition created a set of simple strategies to be used as the basic strategies of the patterns [31]. The patterns are added to these basic strategies. As it was explained, a *no-pattern log file* describes the basic strategy and the *pattern log file* describes a simple behaviour that the opponent team performs.

The first step in our experiment is to obtain some pattern log files and their corresponding no-pattern log files. The files which we use to test and explain our method have been obtained from the *Patterns* section of *RoboCup 2005 Coach Competition* web page [32]. The two logfiles last around 3 500 time steps (a complete game lasts 6 000 time steps). The pattern and no-pattern descriptions (defined by CLang rules) are described as follows:

Pattern Description: The attackers (i.e. players 10 and 11) pass to each other right in front of the goal, before shooting.

No-pattern Description: The attackers (i.e. players 10 and 11) dribble to the goal and shoot (i.e. they do not pass each other)

The *CAOS Coach* is able to extract the main features from the game log file to analyze the behaviour of the test team. Also, this coach is able to recognize and report the events of each game. The result is a data structure for each game (pattern game and no-pattern game) where every event of the game is defined. This events representation consists of the event realized and the player/s (and team) who executed the event.

Then, a trie is built for each game with the event sequences of the test team (this is the only team analyzed for our purpose). The *pattern trie* represents the game in which the opponent follows a pattern and the *no-pattern trie* represents the game in which the pattern is not followed. As we have seen, a trie node consists of (1) the description of an event (2) a number that indicates how many times the event has been inserted in that sequence (3) a chi-square value for the node (except for the nodes of level 1 and the root).

In this experiment, after analyzing the log files and creating the tries of the corresponding games, the number of nodes in each trie is as follows: 176 (*pattern trie*) and 121 (*non-pattern trie*). As explained in Section 5.1.3, in order to reduce the amount of not significant nodes, the trie is pruned and the branches which have been introduced only once are eliminated.

The main characteristics of the pruned tries are:

Pruned pattern trie: *Amount of nodes*: 49 nodes. *Depth*: 6. *Chi-square Value*: Maximum (35.0); minimum (1.3) and average (11.85).

Pruned no-pattern trie: *Amount of nodes*: 56 nodes. *Depth*: 5. *Chi-square Value*: Maximum (25.0); minimum (0.2) and average (11.91).

Some important branches of each trie are shown in Figures 8 and 9.

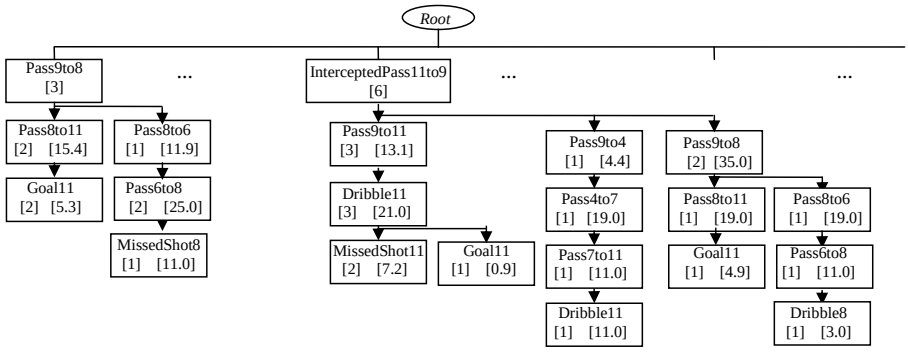

Fig. 8. Most important branches of the *pattern trie*

After obtaining the two tries that represent the two games, we can apply the corresponding algorithm described in Section 5.2.2: At level 2 of the pattern trie there is an event *pass10to11* (and prefix *pass11to10*) which does not exist in the *no-pattern trie*. Also, this event has the highest chi-square value (35.0); therefore,

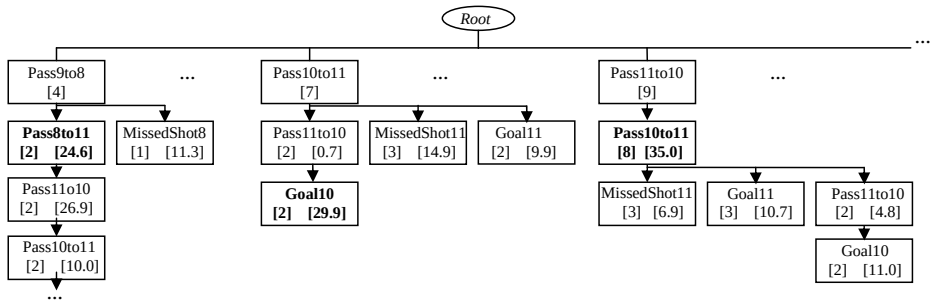

Fig. 9. Most important branches of the *no-pattern trie*

this event and its prefix are inserted in the pattern description. Other node with a high chi-square level (compared with the average value) in this level is the event *Pass8to11* (and prefix *Pass9to8*), but, in this case, this event appears also in the no-pattern trie with a similar chi-square value, so this event is not considered as a possible sub-pattern.

At level 3 of the pattern trie, the node *Goal10* is detected as a sub-pattern because its chi-square value is much higher than the average value and the node is not repeated in the no-pattern trie. Therefore, this sub-pattern is inserted into the pattern description. After analyzing every node in the pattern trie, *Our Pattern Description* is as follows: $OurPDescription = [\{\text{Event:(pass10to11)}, \text{Prefix} : \text{(pass11to10)}, \textit{Chi-Sq:}(35.0)\}, \{\text{Event} : \text{(Goal10)}, \text{Prefix:(pass10to11} \rightarrow \text{pass11to10)}, \textit{Chi-Sq:}(29.9)\}]$.

The above pattern description represents a behaviour description very similar to the behaviour of the pattern represented by the original description. In this case, our method is able to extract the qualitative difference between the pattern trie and its corresponding no-pattern trie. The only difference between our pattern description and the original pattern description is that the event *pass11to10* has not been identified as a sub-pattern. It is because this event is in level 1 and at this level the nodes have no chi-square value.

Once the pattern has been obtained, it is stored in the *Pattern Library* and it will be used in the online recognition.

7.2 The Competition

The *RoboCup Coach Competition* was held in Bremen (June, 2006). There were nine qualified teams, but finally only six teams participated. The competition consisted of three rounds. In each round, 17 patterns were used for the offline phase. Therefore, every coach received 17 patterns log files and its corresponding no-pattern log files. Then, the coaches read, modeled and stored these patterns in the chosen way. An average of 5 minutes for processing time of a pattern was considered.

In the on-line phase, a round consisted of 3 iterations. In each iteration, a test team was created with a behaviour consisted of 4 or 5 patterns (previously considered in the offline phase). The coaches observed that team during a 6 000 cycles game (around 10 minutes), and they should recognize and report, as soon as possible, the 4 or 5 patterns activated.

In order to recognize an activated pattern, it should be predictable and exploitable. But sometimes, when it was activated with other patterns, it was not much repeated or predictable, and it made the task even more difficult. Coaches were evaluated based on correctly and incorrectly reported patterns. A coach was rewarded if he reported an activated pattern correctly. The sooner s/he detected the pattern, the higher score s/he got. Also, penalty was imposed to any incorrect report.

To calculate the performance of a given coach the following formulas were used:

$$\begin{aligned} \textit{score} &= \alpha \times \textit{Correct report reward} - \textit{Incorrect report penalty} \\ \textit{Correct report reward} &= 3\,000 \times N_c + \sum_{i=1}^{N_c} (6\,000 - tc_i) \\ \textit{Incorrect report penalty} &= 3\,000 \times N_i + \sum_{j=1}^{N_i} (6\,000 - ti_j) \end{aligned}$$

where:

$$\alpha = \frac{\textit{Total number of flaws} - \textit{Number of activated flaws}}{\textit{Number of activated flaws}}$$

$$\begin{aligned} N_c &= \textit{Number of correct reports} \\ tc_i &= \textit{Time of the } i^{\text{th}} \textit{ correct report} \\ N_i &= \textit{Number of incorrect reports} \\ ti_j &= \textit{Time of the } j^{\text{th}} \textit{ incorrect report.} \end{aligned}$$

According to the previous formulas, the score and rankings for the teams in the first round are shown in Table 2[2].

			Iteration 1		Iteration 2		Iteration 3	
Rank	Team	Country	Rank	Score	Rank	Score	Rank	Score
1	Mehr	Iran	1	34 750.0	1	14 300.0	2	34 700.0
2	MRL	Iran	3	11 900.0	4	11 289.6	1	48 789.6
3	CAOS	Spain	2	25 051.2	3	13 091.5	5	957.3
4	Austin Villa	USA	4	5 434.2	2	14 054.4	4	8 312.5
5	Caspin	Iran	5	−8 000.0	5	−8 000.0	3	9 600.0
6	Pasagard	Iran	6	−10 497.0	6	−10 497.0	6	−10 497.0

Table 2. Results of Coach RoboCup competition (first round)

[2] The results are available at `http://ssil.uni-koblenz.de/RC06/`.

As we can see, the score of our team was changing considerably. These changes depend mainly on the type of patterns to recognize. The type of activated patterns was very different in each iteration and our team only was able, mainly, to recognize patterns related to players' actions. Every pattern related to the ball owner actions was recognized correctly. Also, and it is an important point to consider, there were some patterns which were not related to players' actions, but when those patterns were executed, the players made specific actions that our coach was able to recognize as a pattern. This means that the pattern, as it was defined, was not recognized, but the way the players played with that pattern activated made that our coach could recognize it.

In the second round, and because the majority of the patterns to recognize were not related with players' actions, our team got the 5^{th} position. The results of every team are shown in Figure 10.

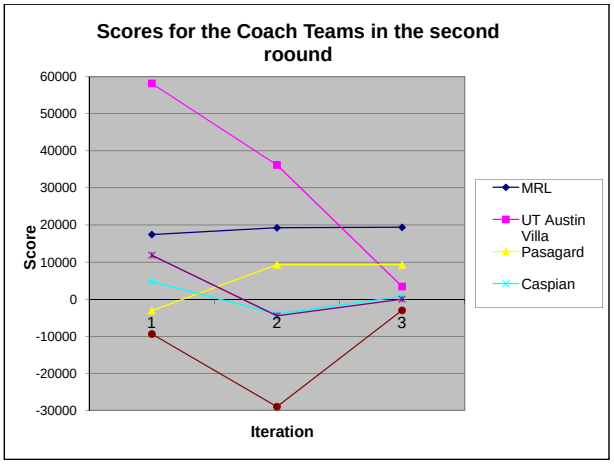

Fig. 10. Scores for the coach teams in the (second round)

As we could see, the proposed method works when the patterns are related to actions, but this coach still needs to be improved with other techniques for recognizing other different kinds of patterns.

8 RELATED WORK

MRL Coach 2006 Team, the winner team of the *2006 Coach Competition*, modelled the opponent using a learning method to predict the opponent behaviour. Also, in the online mode, the coach used recognition algorithms in the same way as in offline phase to identify the opponent behaviour.

The *UT Austin Villa 2006 Simulator Coach* Team characterized each team by a set of features calculated from statistics gathered while observing a game. Features

such as the accurate formation are considered estimating the home positions and ball attraction vectors for each player. Also, this coach identifies high-level events from cycle-by-cycle positions and velocities of both the players and the ball. With this data, a model is created to be used in the online phase.

Pasargard 2006 Coach Team is mainly based on movement detection and learning automata. *Mehr Coach 2006* Team focuses the work in two divisions: offline learning (creating a set of CLang rules from high-level actions) and online exploration (reducing the number of observations of similar events in the two log-files).

9 CONCLUSIONS AND FUTURE WORK

In this research a comparing method of two different multi-agent systems behaviours has been presented. This method has been applied with success in the soccer domain, especially in the *RoboCup Coach Competition*. In order to compare two different team behaviours (one of them is following a pattern), a trie data structure is used in a novel way. It was demonstrated that this structure can be very useful to detect predictable and exploitable behaviours. Also, there are many other domains related to multi-agent systems where this method could be applied.

The comparing methods applied in the RoboCup 2006 Coach Competition and presented in this paper work successfully when the pattern followed by a team is related to the players' actions. Any other kind of patterns (related to the different field regions in which the action occurs or the cycle when some actions occurs) could be detected or not; if it is detected, the coach can not ascertain the pattern description, but can recognize the specific way the players play following this pattern.

Although the proposed method has been applied in the soccer domain, it could be applied in another kind of multi-agent systems where there are dependences between different events.

The competition changes every year to make it more difficult and useful and the future of this sub-league is very interesting. There are many proposals where both opponent modelling and learning should be used. About this improvement, the most interesting aspect is to use the created model in this competition in order to improve the behaviour of a multi-agent system.

Acknowledgments

This work has been supported by the Spanish Ministry of Education and Science under project TRA-2007-67374-C02-02.

REFERENCES

[1] Sahota, M.—Mackworth, A. K: Can Situated Robots Play Soccer? In: Proc. Artificial Intelligence 94, Banff, AB, 1994, pp. 249–254.

[2] Sahota, M. K.—Mackworth, A. K.—Barman, R. A.—Kingdon, S. J.: Real-Time Control of Soccer-Playing Robots Using Off-Board Vision. In: IEEE International Conference on Systems, Man, and Cybernetics, 1995, pp. 3690–3693.

[3] Kitano, H.—Asada, M.—Kuniyoshi, Y.—Noda, I.—Osawa, E.—Matsubara, H.: RoboCup: A Challenge Problem for AI and Robotics. In: RoboCup, 1997, pp. 1–19.

[4] Kitano, H.—Tambe, M.—Stone, P.—Veloso, M.—Coradeschi, S.—Osawa, E.—Matsubara, H.—Noda, I.—Asada, M.: The RoboCup synthetic agent challenge 97, in: International Joint Conference on Artificial Intelligence (IJCAI 97), 1997.

[5] Kitano, H.—Asada, M.—Kuniyoshi, Y.—Noda, I.—Osawa, E.: RoboCup: The Robot World Cup Initiative. In: Agents, 1997, pp. 340–347.

[6] The soccer server web page (December 2006). Available on: `http://sserver.sourceforge.net`.

[7] Noda, I.: Soccer Server: A Simulator of RoboCup. In: J. S. for AI (Ed.), in Proceedings of AI Symposium '95, 1995, pp. 29–34.

[8] Noda, I.—Matsubara, H.—Hiraki, K.—Frank, I.: Soccer Server: A Tool for Research on Multiagent Systems. In: Applied Artificial Intelligence, Vol. 12, Taylor and Francis Ltd, 1998, pp. 233–258.

[9] Carpenter, P.—Riley, P.—Veloso, M. M.—Kaminka, G. A.: Integration of Advice in an Action-Selection Architecture. In: G. A. Kaminka, P. U. Lima, R. Rojas (Eds.), RoboCup, Vol. 2752 of Lecture Notes in Computer Science, Springer 2002, pp. 195–205.

[10] Rasmusen, E.: Game and Information – An Introduction to Game Theory. 2nd Edition, Blackwell, Cambridge, MA, USA & Oxford, UK, 1994.

[11] Carmel, D.—Markovitch, S.: Opponent Modeling in Multi-Agent Systems. In: G. Weiß, S. Sen (Eds.), Adaptation and Learning in Multi-Agent Systems, Springer-Verlag: Heidelberg, Germany, 1996, pp. 40–52.

[12] Tambe, M.—Rosenbloom, P. S.: Architectures for Agents That Track Other Agents in Multi-Agent Worlds. In: ATAL, 1995, pp. 156–170.

[13] Stone, P.—Riley, P.—Veloso, M.: Defining and Using Ideal Teammate and Opponent Agent Models. In: Proceedings of the 7th Conference on Artificial Intelligence (AAAI-00) and of the 12th Conference on Innovative Applications of Artificial Intelligence (IAAI-00), AAAI Press, Menlo Park, CA, 2000, pp. 1040–1045.

[14] Ledezma, A.—Aler, R.—Sanchis, A.—Borrajo, D.: Predicting Opponent Actions by Observation. 2004).

[15] Drücker, C.—Hübner, S.—Schmidt, E.—Visser, U.—Wieland, H.: Virtual Werder: Using the Online-Coach to Change Team Formations. In: 4th International Workshop on RoboCup, Carnegie Mellum University Press, Melbourne, 2000.

[16] Riley, P.—Veloso, M. M.: On Behavior Classification in Adversarial Environments. In: L. E. Parker, G. A. Bekey, J. Barhen (Eds.), DARS, Springer, 2000, pp. 371–380.

[17] Steffens, T.: Feature-Based Declarative Opponent-Modelling in Multi-Agent Systems. Master's Thesis, Institute of Cognitive Science, Osnabrück, 2002.

[18] Riley, P.—Veloso, M. M.—Kaminka, G. A.: Towards Any-Team Coaching in Adversarial Domains. In: AAMAS, ACM, 2002, pp. 1145–1146.

[19] Scerri, P.—Pynadath, D.—Tambe, M.: Adjustable Autonomy in Real-World Multi-Agent Environments. In: AGENTS'01: Proceedings of the Fifth International Conference on Autonomous Agents, ACM Press, New York, NY, USA, 2001, pp. 300–307.

[20] Riley, P.—Veloso, M. M.—Kaminka, G. A.: An Empirical Study of Coaching. March 5, 2002.

[21] Veloso, M. M.—Pagello, E.—Kitano, H. Eds.: RoboCup-99: Robo Soccer World Cup III. Vol. 1856 of Lecture Notes in Computer Science and Lecture Notes in Artificial Intelligence, Springer-Verlag Inc., pub-SV:adr, 2000.

[22] The RoboCup Web Page. Available on: `http://www.robocup.org`.

[23] Chen, M.—Foroughi, E.—Heintz, F.—Huang, Z.—Kapetanakis, S.—Kostiadis, K.—Kummeneje, J.—Noda, I.—Obst, O.—Riley, P.—Steffens, T.—Wang, Y.—Yin, X.: Soccer Server Manual. Version 7. 2002.

[24] Kuhlmann, G.—Stone, P.—Lallinger, J.: The Champion UT Austin Villa 2003 Simulator Online Coach Team. In: D. Polani, B. Browning, A. Bonarini, K. Yoshida (Eds.), RoboCup-2003: Robot Soccer World Cup VII, Springer Verlag, Berlin 2004.

[25] R. Committee, RoboCup 2006 official rules for the competition. Available on: `http://www.caos.inf.uc3m.es/caoscoachteam/otros/rules0.0.pdf`.

[26] Knuth, D. E.: The Art of Computer Programming. Vol. 3. Addison-Wesley, Reading, 1973.

[27] Fredkin, E.: Trie Memory. Comm. A. C. M., Vol. 3, 1960, No. 9, pp. 490–499.

[28] Kaminka, G. A.—Fidanboylu, M.—Chang, A.—Veloso, M. M.: Learning the Sequential Coordinated Behavior of Teams from Observations. In: G. A. Kaminka, P. U. Lima, R. Rojas (Eds.): RoboCup, Vol. 2752 of Lecture Notes in Computer Science, Springer, 2002, pp. 111–125.

[29] Huang, Z.—Yang, Y.—Chen, X.: An Approach to Plan Recognition and Retrieval for Multi-Agent Systems. In: Proceedings of AORC, 2003.

[30] Chiang, C. L.: Statistical Methods of Analysis. World Scientific, Suite 202, 1050 Main Street, River Edge, NJ 07661, 2003.

[31] R. Committee, RoboCup 2005 Game Logs (2005 Coach Competition). Available on: `http://sserver.jpn.org/RoboCup/log/RoboCup05/`.

[32] The RoboCup 2005 Coach Competition Web Page. Available on: `http://staff.science.uva.nl/~jellekok/robocup/rc05`.

Araceli Sanchis has been a University Associate Professor of computer science at Carlos III University of Madrid (UC3M) since 1999. She received her Ph.D. in physical chemistry from Complutense University of Madrid in 1994 and in computer science from Politecnic University of Madrid in 1998. She has a B. Sc. in chemistry (1991) from the Complutense University of Madrid. She had been Vice-Dean of the Computer Science degree at UC3M and, currently, she is head of the CAOS group (Grupo de Control, Aprendizaje y Optimización de Sistemas) based on machine learning and optimization. She has published over 50 journal and conference papers mainly in the field of machine learning.

Agapito Ledezma is an Associate Professor of computer science and member of the CAOS research group at Carlos III University of Madrid (UC3M). He obtained his Ph. D. in computer science from UC3M in 2004. He has a B. S. in computer science (1997) from Latinoamericana de Ciencia y Tecnología University (ULACIT) of Panamá. His research interests span data mining, agent modeling, ensemble of classifiers and cognitive robotics. He has written more than 30 technical papers for computer science journals and conferences.

José Antonio Iglesias is a Teaching Assistant of computer science and member of the CAOS research group at the University Carlos III de Madrid (UC3M), Spain. He received his M. Sc. in computer science at UC3M in 2006 and he is pursing his Ph. D. since then. He has a B. Sc. in computer science (2002) from Valladolid University. He has published over 10 conference papers and he is a committee member of the International Conference on Intelligent Systems and Agents (ISA). Research interests: agent modelling, plan recognition, sequence learning, machine learning and evolving fuzzy systems. His web page: `http://www.caos.inf.uc3m.es/~jiglesia/`.