Learning Decision Trees for Action Selection in Soccer Agents

Savas Konur [1] and **Alexander Ferrein** and **Gerhard Lakemeyer** [2]

Abstract.

In highly-dynamic domains such as robotic soccer it is important for agents to take action rapidly, often in the order of a fraction of a second. This requires, a possible longer-term planning component notwithstanding, some form of reactive action selection mechanism. In this paper we report on results employing decision-tree learning to provide a ball-possessing soccer agent in the SIMULATION LEAGUE with such a mechanism. The approach has payed off in at least two ways. For one, the resulting decision tree applies to a much larger set of game situations than those previously reported and performs well in practice. For another, the learning method yielded a set of qualitative features to classify game situations, which are useful beyond reactive decision making.

1 Introduction

In highly-dynamic domains like robotic soccer it is important for agents to take action rapidly, often in the order of a fraction of a second. This is especially true in the application domain considered in this paper, the ROBOCUP SIMULATION LEAGUE with 11 players per team on a 2D playing field. Such tight time constraints require, a possible longer-term planning component notwithstanding, some form of reactive action selection mechanism. By *reactive* we mean, roughly, that decisions are made solely on the basis of a description of the current situation or world model. In particular, this precludes any evaluation of different possible courses of actions as in planning.

When presented with a game situation in the SIMULATION LEAGUE, humans are usually quite capable of choosing a reasonable action for, say, the ball-possessing agent. However, it is not at all easy to encode this "expert" knowledge in a way suitable for an artificial soccer agent for at least two reasons:

1. It is not clear what the salient features of a game situation are, which determine the action to be chosen. Presumably, these features would include qualitative descriptions such as the team member or opponent closest to the ball. But what the relevant ones?
2. Even if we were given those features, it is not clear how to translate them into rules for decision making. We could try to hand-code them, but this approach is likely error-prone, not to mention the difficulty of eliciting the rules from the human expert.

Perhaps the best way to overcome these problems is to use machine-learning techniques. Deciding what action to take next in robotic soccer can be thought of as a classification problem, where a game situation is classified according to the best next action. Machine-learning techniques suitable for classification are decision-tree learning such as ID3 [9] or C4.5 [10]), neural networks [14, 21] and reinforcement learning [17, 21, 19].

For our work we have chosen decision-tree learning, in particular, C4.5, as it is capable to deal well with both issues raised above. For one, given a sufficiently large set of training examples, the system automatically builds a decision tree, which encodes the rules for action selection. Compared to other techniques like neural networks, decision trees also have the well-known benefit that they can be inspected and understood by humans. For another, it is not necessary to decide beforehand what the relevant features are for classification. All that is needed is that the system is given a sufficiently large set. The relevant features are produced as a side-effect of building the decision tree in the sense that only those features or attributes that eventually appear as nodes in the decision tree are thought of as relevant.

We remark that we applied learning to all types of players (except the goalie) anywhere on the field, but we restricted ourselves to players in ball possession.

We believe that our results are noteworthy for at least the following reasons. For one, the resulting decision tree covers a much wider range of game situations and actions than in previous work such as [7, 20]. For another, as we will see in the discussion of experimental results, a team using this decision tree, but which is otherwise not optimized at all, performs surprisingly well. Finally, as already noted above, while decision-tree learning by itself does not come up with qualitative world descriptions, it is nevertheless useful in pruning irrelevant attributes from a given set.

This rest of the paper is organized as follows. In Section 2 we briefly discuss existing learning methods applied to robotic soccer. In Section 3, we describe our approach to decision-theoretic learning of action selection for a soccer agent in the SIMULATION LEAGUE, followed by a discussion of experimental results in Section 4. The paper ends with a brief summary and concluding remarks.

2 Related Work

In this section we present some of the work on applying machine learning techniques to robotic soccer and action selection. One focus is learning of basic agent skills such as *dribbling, passing,* and *intercepting.* [13], for example, use reinforcement learning for this purpose. In [15, 20] a form of so-called *Layered Learning* is proposed. It provides a bottom-up hierarchical approach to learning agent behaviors. In this framework, the learning at each level is directly used in the learning at the next higher level. The bottom layer considers low-level individual agent skills such as *ball interception* or *drib-*

[1] Free University of Amsterdam Artificial Intelligence Department, Amsterdam, The Netherlands, skonur@cs.vu.nl,

[2] RWTH Aachen, Knowledge-Based Systems Group, Aachen, Germany, {ferrein,gerhard}@cs.rwth-aachen.de

bling. In contrast to [13], the behaviors are learned using a neural network. At higher levels, action selection of the ball-possessing agent is learned using multi-agent reinforcement learning. We remark that the authors consider only eight kicking actions, which is much more limited than in our case. (A comparison of multi-agent reinforcement learning methods in the soccer domain can be found in [18].) In earlier work, Matsubara et al. [7] considered action selection using neural networks. There the scope was even more limited, as they restrict themselves to the decision of whether to shoot directly to the goal or to pass to a better positioned player. Decision-tree learning has been applied in robotic soccer as well. For example, Visser and Weland [22] recently applied C4.5 to learning aspects of the strategy of the opposing team in the SIMULATION LEAGUE.

Outside of the soccer domain, action selection for robots is often addressed using reinforcement learning. For example, [11] proposes hierarchical Q-learning for action selection, where the control task of a robot is divided into a set of simpler problems each learned separately. Another reinforcement learning approach to the action selection problem was proposed by Humphrys [8]. Each behavior module proposes an action with a certain weight of which the action with the highest weight is executed. The weights of the actions are modified based on the difference between the weight of the action being executed and the action a behavior module proposed using a form of reinforcement learning. The application domain presented in [8] is a simulated environment of a house keeping robot.

3 Learning the Decision Tree

In this section we present how we applied C4.5 to our SIMULATION LEAGUE agent. We start with an overview of the categories which should be learned, i.e. the action which the agent should perform. In Section 3.2, we present the attributes which turned out to be appropriate for the SIMULATION LEAGUE before we show how we instructed the agent in Section 3.3. The consulting procedure in on-line games is represented in Section 3.4.

3.1 Skill Hierarchy and Meta-Level Actions

As C4.5 cannot deal with parameterized categories to be learned [10], we implemented special behaviors which are to be selected by the decision tree. Figure 1 gives an overview of the skill hierarchy we use in our reactive soccer agent. The low-level action layer comprises basic actions like *dashing to a position*, *accelerating the ball to a certain velocity*, or *freezing the ball*. Those commands are translated into the SOCCERSERVER commands, such as *dash*, *kick*, *turn*, etc. The intermediate action layer defines actions like *moving to a position*, or *kicking the ball to a certain point*, which are based on the low-level action layer. High-level actions use the intermediate actions for the desired behavior. *dribbling* and *passing the ball to a teammate* are two examples of high-level actions taken from [1].

C4.5 requires that output values (categories) of decision trees must be discrete and specified in advance. This means action categories which we use in the learning process should not take any argument in order to satisfy the C4.5 requirements. Therefore, we cannot directly use the high-level actions in the learning process since they require some arguments in order to be executed. For that reason, we need new actions which should have the form of a argument-free discrete category. In order to accomplish this purpose we have introduced the meta-level actions which use the high-level actions to generate the desired behavior. These skills are parameterless encapsulations of skills from the other layers suitable to deal as a category for C4.5.

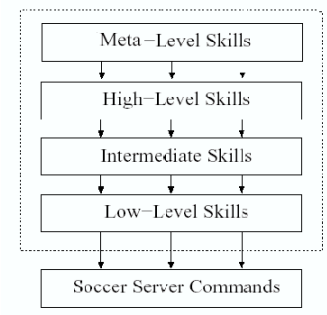

Figure 1. Skill hierarchy

The meta-level actions calculate necessary arguments before calling the corresponding high-level actions. In our current implementation, we have defined 15 meta-level actions (see below). An example for such an action is the dribble action depicted in Fig. 2. Some decisions like with which angle and speed the agent should dribble are made. For the supervision process it is very important that the supervisor has the semantics of the respective meta-level action in mind in order to give the right advice. The high-level dribble action in turn is responsible for correctly determining when to kick and intercept the ball in order to move player and the ball to the demanded position on the field.

```
dribble()
if ball is not in kickable margin then
  return intercept()
else
  if path toward opponent goal is free then
    ang ← direction to opponent goal
    type ← DRIBBLE_FAST
  else if path toward goal is fairly free then
    ang ← direction to opponent goal
    type ← DRIBBLE_SLOW
  else
    ang ← getDirectionOfWidestAngle()
    if ang = wide then
      type ← DRIBBLE_FAST
    else
      type ← DRIBBLE_SLOW
    end if
  end if
end if
return dribble(ang, type)
```

Figure 2. The dribble action which is executed by the decision tree

In the following, we give an overview of the categories (meta-level actions) which were used.

- *Outplay Opponent* The ball is played into the opponent's back followed by an intercept action.
- *Dribble* calculates the angle relative to the agent where it should dribble to. A second argument is the speed with which the agent should dribble. Two different speeds are distinguished: slow and fast.

- *Direct Pass* comprises several actions. It is distinguished between direct passes of an attacker and between a defensive player and a midfielder. Moreover, there exists a pass action for back passes and passes in front of the player. We have to split the direct pass action because the different aspects (playing a pass to a player in front or to the back) does not fit in one pass action model. C4.5 is not able to determine the differences in the semantics only by looking at the attributes. Each different instance of a direct pass share the calculation of the "least congested team-mate". Heuristically, this team-mate is chosen. The heuristics is based on the number of opponents in a certain distance around the player, there exists a free pass-way to that respective player, and some more.
- A *Through Pass* is a pass which is played behind the opponent defense. A free space behind the defensive line is found where a team-mate is able to receive.
- A *Leading Pass* is a pass in the run-way of the respective team-mate. It is calculated if a team-mate can intercept the ball after the pass in a certain amount of time.
- The *Shoot at Goal* action calculates a point on the opponent goal line with maximum distance to the opponent goal keeper.
- With *Clear Ball* the player simply kicks the ball as far away as possible. For instance, if a defender is not able to dribble or pass the ball to a team-mate it seem reasonable to bring the ball as far away from the own goal as possible.
- *Turn to Opponent Goal* When the agent is in ball possession and cannot see the opponent goal in order to perceive the opponent's goal keeper position, this action enables the agent to turn towards the goal without leaving the ball.

3.2 Constructing the Attribute Set

In order to generate a good classification by the C4.5 algorithm choosing an appropriate attribute set is a crucial task. Having irrelevant attributes in the attribute set is the main reason for *overfitting* [9, 10]. Another difficulty for finding an appropriate set lies in the nature of the soccer domain. As there are different player types and situations during a game where each player has to react in different ways according to its type and location on the field, we have to account for this by dividing the field into different regions. One such possible division is depicted in Figure 3 which was proposed in [1]. One approach to the problem could have been to learn different trees for different player types, such as *attacker*, *midfielder*, *defender*, by constructing the test sets with only the relevant information. However, this approach raises several problems: (1) it is not trivial to recognize all relevant regions; statically dividing the field into defense, midfield and attack is not sufficient because also a defender might sometimes be located in a midfield region, (2) a separate construction of the training set for each region and player type is a tedious task and available data from the LOGPLAYER[3] consists of whole game information, (3) different decision trees for each player type according to the game situations demand a selection mechanism that tells which tree should be consulted; this would take the same problem to a higher level.

Therefore, we decided to use only one decision tree containing the distinguishing features like player types and playing region as attributes. We also restrict the consulting of the decision tree to game situations where players are in ball possession.

From the considerations above and from many experiments we arrived at 35 attributes, which can be summarized as follows:

[3] The LOGPLAYER is a tool coming with the SIMULATION LEAGUE simulation server to replay recorded games.

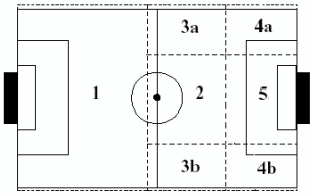

Figure 3. Possible regions a player can be in.

- *Type of player* is a discrete attribute and distinguishes between defender, midfielder, and attacker.
- *Playing region* is a discrete attribute representing in which region the player is located. The different regions are depicted in Figure 3.
- *Closest teammate to ball* is a boolean attribute denoting if the player is the closest player to the ball.
- *Distance and angle to ball, goals and opponent goal keeper* are continuous attributes determining the agent's distance and angle to the ball, the own as well as the opponent goal, and to the opponent goal keeper.
- *Distances and angles to the visible teammates and opponents* are a number of attributes denoting the visible teammates and opponents of an agent.
- *Closest team to the ball* is a boolean attribute which is true if one player of our team is the closest player to the ball and false otherwise.
- *Ball possessing team* takes three values corresponding to whether the ball is in kickable range for our team, for the opponent team, or for none.
- *Ball distances and angles to both goals* are continuous attributes representing the distances and angles of the ball to both of the goals.
- *Opponent goalie's distances and angles to its goal posts* is a number of attributes representing the the distance and angle of the opponent goalie to the opponent goal posts.

The reader should note that it was the decision-tree learning algorithm that ultimately decided that these are the relevant attributes of a game situation for a player in ball possession, as only these attributes were used in the decision tree. For example, it turns out that only the *five* nearest players to the ball are ever considered relevant. One possible explanation for this are the restrictions due to two dimensions of the current SIMULATION LEAGUE, where passes across the opponent defense are impossible. Other attributes which were used during tests were not contained in the resulting tree and therefore deemed irrelevant.

We also remark that the choice of attributes may likely be different for players other than the one in ball possession. For example, one would not expect the goalie to care much about the distance to the opponent's goal.

The reader should note that many of the above attributes have a continuous domain. We make use of C4.5's ability to discretize continuous attributes given the training set. This discretization sometimes results in wrong classifications during the consulting phase, as hard bounds on the attributes are drawn. Nevertheless these errors seem acceptable in practice.

```
...
MyPlayerType = 2:
|   BallDistanceToOpponentGoal <= 18.2609 :
|   |    MyCurrentPlayingRegion = 4: 9 (6.0/1.2)
|   |    MyCurrentPlayingRegion = 5:
|   |    |   MyDistanceToOpponentGoal <= 12.4953 :
|   |    |   |    MyDistanceToOurGoal <= 92.6233 : 1 (3.0/2.1)
|   |    |   |    MyDistanceToOurGoal > 92.6233 :
|   |    |   |    |    MyDistanceToOpponentGoalie > 18.5529 : 9 (3.0/2.1)
|   |    |   |    |    MyDistanceToOpponentGoalie <= 18.5529 :
|   |    |   |    |   |   OpponentGoalieGoalRightCornerAngle <= 47.7002 : 11 (40.0/2.6)
|   |    |   |    |   |   OpponentGoalieGoalRightCornerAngle > 47.7002 :
|   |    |   |    |   |  |   MyAngleToBall <= 26.746 : 11 (4.0/1.2)
|   |    |   |    |   |  |   MyAngleToBall > 26.746 : 10 (2.0/1.0)
|   |    |    MyDistanceToOpponentGoal > 12.4953 :
|   |    |   |    MyDistanceToOpponentGoalie <= 6.47349 :
|   |    |   |    |    OpponentGoalieGoalRightCornerDistance > 16.9349 : 9 (5.0/2.3)
|   |    |   |    |    OpponentGoalieGoalRightCornerDistance <= 16.9349 :
|   |    |   |    |   |    MyDistanceToSecondVisibleOpponent <= 5.27978 : 4 (3.0/1.1)
|   |    |   |    |   |    MyDistanceToSecondVisibleOpponent > 5.27978 : 11 (11.0/2.5)
|   |    |   |    MyDistanceToOpponentGoalie > 6.47349 :
|   |    |   |    |    MyAngleToOpponentGoal <= -103.167 : 10 (3.0/1.1)
|   |    |   |    |    MyAngleToOpponentGoal > -103.167 :
|   |    |   |    |   |    MyDistanceToThirdVisibleOpponent <= 8.71111 :
|   |    |   |    |   |   |   MyDistanceToOpponentGoal <= 14.6303 : 9 (2.0/1.0)
|   |    |   |    |   |   |   MyDistanceToOpponentGoal > 14.6303 :
|   |    |   |    |   |   |   |   MyAngleToFirstVisibleTeammate > 53.8028 : 8 (5.0/2.3)
|   |    |   |    |   |   |   |   MyAngleToFirstVisibleTeammate <= 53.8028 :[S7]
|   |    |   |    |   |    MyDistanceToThirdVisibleOpponent > 8.71111 :
|   |    |   |    |   |   |   ClosestTeamToBall = 0: 1 (0.0)
|   |    |   |    |   |   |   ClosestTeamToBall = 2: 11 (2.0/1.0)
|   |    |   |    |   |   |   ClosestTeamToBall = 1:
|   |    |   |    |   |   |  |   MyDistanceToOurGoal > 95.9754 : 9 (5.0/2.3)
|   |    |   |    |   |   |  |   MyDistanceToOurGoal <= 95.9754 :
|   |    |   |    |   |   |  |  |    MyDistanceToThirdVisibleTeammate <= 14.797 :[S8]
|   |    |   |    |   |   |  |  |    MyDistanceToThirdVisibleTeammate > 14.797 :[S9]
|   BallDistanceToOpponentGoal > 18.2609 :
|   |   MyAngleToOpponentGoal <= 98.1456 :
|   |   |    MyAngleToOpponentGoal <= -89.512 :
|   |   |    |    MyDistanceToFirstVisibleOpponent <= 1.87341 : 8 (21.0/2.5)
|   |   |    |    MyDistanceToFirstVisibleOpponent > 1.87341 :
|   |   |    |   MyAngleToOpponentGoal <= -118.357 :
|   |   |    |  |   OpponentGoalieGoalRightCornerAngle <= 12.9107 : 10 (53.0/3.8)
|   |   |    |  |   OpponentGoalieGoalRightCornerAngle > 12.9107 : 8 (3.0/1.1)
|   |   |    |   MyAngleToOpponentGoal > -118.357 :
|   |   |    |  |   MyAngleToFirstVisibleTeammate <= -113.017 : 4 (7.0/2.4)
|   |   |    |  |   MyAngleToFirstVisibleTeammate > -113.017 :
|   |   |    |  |  |   MyDistanceToThirdVisibleTeammate <= 13.8264 :
|   |   |    |  |  |  |    MyDistanceToThirdVisibleOpponent <= 14.2218 : 8 (9.0/2.4)
|   |   |    |  |  |  |    MyDistanceToThirdVisibleOpponent > 14.2218 : 1 (4.0/2.2)
|   |   |    |  |  |   MyDistanceToThirdVisibleTeammate > 13.8264 :
|   |   |    |  |  |  |    MyDistanceToOurGoal <= 74.9802 : 10 (15.0/4.7)
|   |   |    |  |  |  |    MyDistanceToOurGoal > 74.9802 : 4 (2.0/1.8)
|   |   |    MyAngleToOpponentGoal > -89.512 :
|   |   |    |    BallAngleToOpponentGoal <= 32.6327 :
|   |   |    |   ClosestTeamToBall = 0: 1 (0.0)
...
```

Figure 4. Excerpt from the resulting decision tree.

3.3 Gathering the Training Data

For the supervision process we extended the LOGPLAYER to be able to associate the actions described in Section 3.1 to players. This *supervisor monitor* generates the training examples by storing the output category (actions) together with the input categories (world model information).

It is important to note that while calculating the attribute values we cannot use the global information from the LOGPLAYER directly. Instead, we must calculate the supervised agent's relative world model from the global information and derive the attribute values from it. This is important because while the agent consults the decision tree in on-line games, the world model information comes from the SOCCERSERVER, which supplies the agent with relative information about the world model. Therefore, in the supervision process by calculating the relative information from global view, it is guaranteed that our training and test data are almost from the same distribution.

Another important point to be noted is that the supervisor should have a good idea of how soccer is played in order to give advice to the agent. For humans it is easier to classify a given situation including qualitative measures and give advices of what to do than to formalize a good action selection scheme. In the supervision process, the idea to specify the action classification of a play situation was that a player should select the most suitable action which provides him with the best advantage among alternatives actions. In this case, we can say that each action has a priority which depends on the player type and the region the agent plays in. Below is some part of the scheme that we used in the supervision process while advising the agents:

if scoring prob. is very high **then**
  goalshot
else if agent in defensive region close to our goal **then**
  if no opponent close and agent faces our goal **then**
    turn to opponent goal
  else if there is a very free teammate ahead **then**
    pass ball directly to this teammate
  else if trajectory to opponent goal is fairly free **then**
    dribble forward
  else
    clear ball
  endif
. . .
  else if agent in wings close to opponent goalie **then**
. . .

One might ask why we simply did not implement the above scheme instead of using a learned decision tree. As motivated in the introduction human beings are good in classifying the world into qualitative categories but encoding this as agent control software is much harder. As one can see from the scheme it uses qualitative statements like "very" or "fairly". When supervising "fairly" is evaluated by the supervisor in the complex situation the agent is in. On the other hand, by having a qualitative world model it would be interesting to compare an agent using the supervision scheme as action selection with the decision tree learned by C4.5.

Naturally, the supervisor makes mistakes in the classification or decides on the border line, giving contradictory advices. But as C4.5 is very robust against such mistakes they do not matter that much as they would in a hand-coded variant of the scheme.

3.4 Consulting the Decision Tree

In the previous sections, we considered how the attribute and data sets are gathered through the supervision process (the *training phase*). After the training phase the *model generation phase* starts in which these input files are passed through the C4.5 system, and the decision tree model is produced by executing the C4.5 program. That is, at the end of these phases we have acquired a model which can be used by an agent to classify unseen cases. In the ROBOCUP context, classification means offering a convenient action to the agent as it is playing an on-line game in the SOCCERSERVER. This task is done in a different process which we call the *consult phase* in which an agent consults the resulting decision tree to select an action in a play situation. An agent consults the decision tree model when the ball is kickable for him. In this case, a new process is started in which the attribute values are calculated according to the world situation. Based on these values the decision tree offers an action category which will be performed by the agent in this particular world situation, and the consult process halts for this time instance. Whenever the agent is in the ball-kickable margin, this process is started again. This process repeats until the game finishes. The hierarchical relationship between these phases is shown in Figure 5.

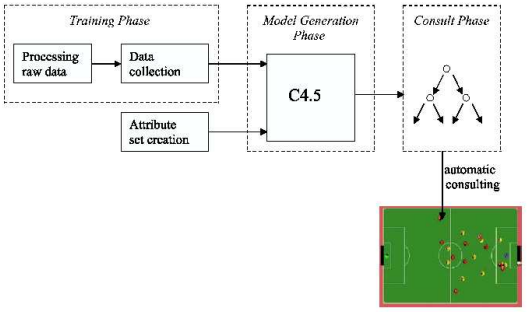

Figure 5. Overview of processing the *Reactive Component* for our soccer agents

Figure 4 shows parts of the resulting decision tree for a midfield player. Attributes are the nodes in this tree, e.g. *MyPlayerType* or *BallDistanceToOpponent Goal*. The leaves of the tree can be identified by numbers which correspond to a respective meta-level action, e.g. action 1 stands for *dribble*, 4 represents a *through pass*, and 11 means *shoot to goal*. The pair which follows an action shows the number of training instances and the number of misclassifications. The numbers in square brackets represent another subtree which is not shown here for readability.

4 Empirical Results

For assessing the quality of the learned decision tree we conducted several experiments.

The first question of interest was the accuracy of our training data. In total, we collected 3000 training examples and grouped them in training sets in steps of 500 examples up to the largest set containing the whole number of training examples (see Table 1). For each set size we built several instances choosing randomly from the whole set of training data. Table 2 shows the classification error rates. The column *Tree size* reflects the number of nodes the tree contains. Based on this table we can make the following observations:

First, the results show a (slightly) decreasing error rate with an increasing number of examples. The fact that we are left with an error rate of almost 9 % even before pruning has at least two reasons. One reason is that the supervisor makes mistakes giving contradictory examples. The other is that we use a large number of continuous attributes. For a continuous attribute, C4.5 finds a split value which maximizes information gain for the respective attribute. This discretization leads to misclassifications.

Second, in each category, we see the error rate of the pruned tree is higher than that of the original tree. Actually, this result is expected since in the pruning process some branches of the tree are replaced by a leaf node, yielding misclassification of some of the examples which were previously classified correctly.

Finally, it should be noted that the size of the trees gradually increases as the size of training data gets bigger, since C4.5 adds new branches to the decision trees in order to correctly classify the data instances.

Category	1	2	3	4	5	6
Set Size	500	1000	1500	2000	2500	3000

Table 1. Sizes of the test sets.

Cat.	Before Pruning		After Pruning	
	Tree size	Error(%)	Tree size	Error
1	174	8.40	144	9.74
2	353	7.82	296	9.40
3	493	8.76	422	9.94
4	672	8.60	588	9.75
5	846	8.20	722	9.50
6	979	8.02	847	9.30

Table 2. Error rates for the training set.

The next interesting question was how good the decision tree classifies unseen examples. We therefore played a large number of games against several teams with a different tree for each category from our training set (For each category we collected 1500 test examples). For assessing "ground truth" we classified for each logged game the situations according to the supervision scheme we presented in Section 3.3. The results over the training games are presented in Table 3 and Figure 6

Category	1	2	3	4	5	6
Classification Ratio (%)	35.1	46.3	59.8	64.1	66.8	66.5

Table 3. Results of training games.

By looking at the table and figure we can see that there is a sharp increase in the correctness between the Category 1 and Category 3. However, the performance increases only slightly between Categories 3 and 5. In the last category we even see a small reduction in the correctness of the classification. This suggests that the optimal size of the training set is reached at around 2500 examples.

The highest ratio of correct classifications we obtained is 66.8 % (Category 5). If we take the RoboCup's domain characteristics and

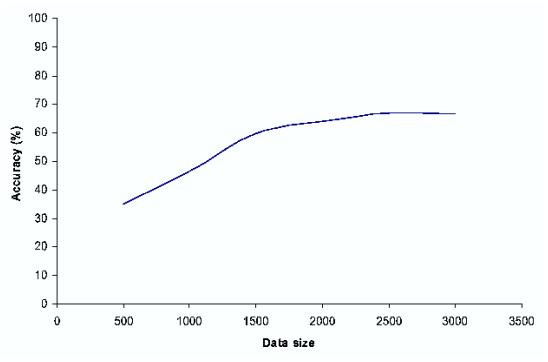

Figure 6. Learning curve of the agent

restrictions into account, we can say that this value is quite reasonable. Especially our results seem to compare favorably with other case studies. For example, Matsubara *et. al.* [7] performed an experiment, in which only the simple situation of two attackers attempting to score a goal against a single opponent is examined. In this experiment, the attackers learned when to select either 'pass' or 'shoot' actions. The ratio of the correct classification that the results showed was 68 %. Note also that the choices in this experiment are far simpler than in our case where we consider all skills for all types of players.

AllemaniACs: Robolog	2 : 0
AllemaniACs: VirtualWerder	1 : 0
AllemaniACs : UvaTrilearn	0 : 9
AllemaniACs : WrigthEagle	0 : 0

Table 4. Some test game results

We played several games against SIMULATION LEAGUE teams from 2003 showing the performance of the learned decision tree. Table 4 shows the results of some of these games. Against mediocre teams like Robolog or Virtual Werder we are able to win. Against the world champion Uva Trilearn our approach leaves room for improvement. One has to note that for these games the agent used the decision tree when a player was in ball possession and executed some standard behavior like "move to strategic position" or "search ball" otherwise. The agent was not highly tuned as we wanted to see the performance of the decision tree.

5 Conclusions

In this paper we described an application of the decision-tree learning method C4.5 to RoboCup's SIMULATION LEAGUE. The method was used to learn the action selection strategy of the whole team, that is, defenders, midfielders, and attackers, when a player is in ball possession. We were able to obtain decision trees which performed surprisingly well in real game situations. Moreover, the method is suitable for selecting the relevant attributes from a given set of qualitative world descriptors.

While this paper focusses on reactive action selection, we believe that cooperative team-play cannot be achieved by reactive control alone, taking only the current game situation into account. Consider, for example, the situation where a wing-change would be necessary

because one side of the field is blocked by opponents. A good choice would be to shift the game to the other wing of the field. It is hard to imagine how such behavior could come about without some form of deliberation where different possible courses of action are considered and evaluated.

For this purpose we have developed an architecture which provides for reactive control as well as a *deliberative component* using the logic-based language Golog [6]. Golog is a language for reasoning about actions and change and is based on the situation calculus [16]. Recent extensions like dealing with a continuously changing world [5] and the integration of a form of decision-theoretic planning [2] to account for the uncertainty arising in the soccer domain makes it a suitable language to reason about scenarios like a wing-change and to coordinate the agents accordingly (cf. [3] for an example in the soccer domain).

When using deliberation one needs a symbolic representation of the environment. Therefore, we are interested in building up a qualitative world model which can be used for the deliberative component. One of the central problems is finding the appropriate attributes to describe the environment in a qualitative way. Recently, Dylla et al. [4] approached this problem by looking at the issue of specifying soccer moves based on the knowledge from a domain expert's (from [12] in their case) for different ROBOCUP leagues. As soccer theory is described in a very abstract fashion, qualitative descriptions clearly seem important, but the theory itself does not answer the question of which qualitative descriptions are most suitable.

The present paper can perhaps be thought of as one step in this direction. As we saw, one interesting outcome is that for the player in ball possession only the five nearest team-mates and opponents seem to matter. Applying the presented approach also for other players like the goal keeper one probably can learn more about the relevant information in robotic soccer.

We believe that the proposed method for reactive action selection is not restricted to the ROBOCUP domain. Highly dynamic domains have in common that actions must be performed rapidly, even if those actions seem to be sub-optimal. Applying decision-tree learning yields one method for implementing a reactive action selection mechanism. In future work we will apply this approach to other dynamic real-time domains, for instance to soft-bots in computer games, to get comparable results. Also the suitability of decision-tree learning for achieving good attribute sets for qualitative world modeling will be further investigated.

Acknowledgments

This work was supported by the German National Science Foundation (DFG) in the Priority Program 1125, *Cooperating Teams of Mobile Robots in Dynamic Environments*. We would like to thank the anonymous reviewers for their comments.

REFERENCES

[1] R.de Boer and J.Kok, *The Incremental Development of a Synthetic Multi-Agent System: The UvA Trilearn 2001 Robotic Soccer Simulation Team*, Master's thesis, University of Amsterdam, 2002.

[2] Craig Boutilier, Ray Reiter, Mikhail Soutchanski, and Sebastian Thrun, 'Decision-theoretic, high-level agent programming in the situation calculus', in *Proc. of AAAI-00*, pp. 355–362. AAAI Press, (July 30– 3 2000).

[3] F. Dylla, A. Ferrein, and G. Lakemeyer, 'Specifying multirobot coordination in ICPGolog – from simulation towards real robots', in *Proc. of the Workshop on Issues in Designing Physical Agents for Dynamic Real-Time Environments: World modeling, planning, learning, and communicating (IJCAI 03)*, (2003).

[4] Frank Dylla, Alexander Ferrein, Gerhard Lakemeyer, Jan Murray, Oliver Obst, Thomas Röfer, Frieder Stolzenburg, Ubbo Visser, and Thomas Wagner, 'Towards a League-Independent Qualitative Soccer Theory for Robocup'. accepted at 8th RoboCup International Symposium as poster.

[5] Henrik Grosskreutz and Gerhard Lakemeyer, 'On-line execution of cc-Golog plans', in *Proc. of IJCAI-01*, (2001).

[6] H.Levesque, R.Reiter, Y.Lesperance, F.Lin, and R.Scherl, 'Golog: A logic programming language for dynamic domains', *Journal of Logic Programming*, (1997).

[7] H.Matsubara, I.Noda, and K.Hiraki., 'Learning of cooperative actions in multi-agent systems: a case study of pass play in soccer', *In S. Sen, editor, AAAI Spring Symposium on Adaptation, Coevolution and Learning in multi-agent systems*, (1996).

[8] M. Humphrys, 'Action selection methods using reinforcement learning', in *From Animals to Animats 4: Proceedings of the Fourth International Conference on Simulation of Adaptive Behavior*, eds., P. Maes, M. Mataric, J.-A. Meyer, J. Pollack, and S. Wilson. MIT Press, (1996).

[9] J.Quinlan, 'Induction of decision trees', *Machine Learning, Kluwer Academic Publishers*, (1986).

[10] J.Quinlan, *C4.5 Programs for Machine Learning*, Morgan Kaufmann, 1993.

[11] L.-J. Lin, 'Scaling up reinforcement learning for robot control', in *Proc. 10th Int. Conf. on Machine Learning*, (1993).

[12] Massimo Lucchesi, *Coaching the 3-4-1-2 and 4-2-3-1*, Reedswain Publishing, 2001.

[13] M.Riedmiller, A.Merke, D.Meier, A.Hoffman, A.Sinner, O.Thate, and R.Ehrmann, 'Karlsruhe brainstormers - a reinforcement learning approach to robotic soccer', in *RoboCup 2000*, Lecture Notes in Artificial Intelligence, Springer-Verlag, (2001).

[14] P.Antognetti and V.Milutinovic, *Neural Networks: Concepts, Applications, and Implementations, Vol. II*, Printice Hall, 1991.

[15] P.Stone, *Layered Learning in Multiagent Systems: A Winning Approach to Robotic Soccer (Intelligent Robotics and Autonomous Agents)*, MIT Press, 2000.

[16] R. Reiter, *Knowledge in Action*, MIT Press, 2001.

[17] R.Sutton and A.Barto, *Reinforcement Learning: An Introduction*, MIT Press, 1998.

[18] R. Salustowicz, M. Wiering, and J. Schmidhuber, 'Learning team strategies: Soccer case studies', *Machine Learning*, **2/3**(33), 1–19, (1998).

[19] P.Norvig S.Russell, *Artificial Intelligence: A Modern Approach-Second Edition*, Printice Hall, 2002.

[20] S.Whiteson and P.Stone, 'Concurrent layered learning', in *Proceedings of the second international joint conference on Autonomous agents and multiagent systems*, pp. 193–200. ACM Press, (2003).

[21] T.Mitchell, *Machine Learning.*, McGraw-Hill, 1997.

[22] U.Visser and H.G.Weland, 'Using online learning to analyze the opponent's behavior', in *RoboCup 2002*, Lecture Notes in Artificial Intelligence, Springer-Verlag, (2003).