International Journal of Advanced Robotic Systems

OPEN ACCESS
ARTICLE

Optimization of Rules Selection for Robot Soccer Strategies

Regular Paper

Václav Snášel[1], Václav Svatoň[1], Jan Martinovič[1,*] and Ajith Abraham[1]

1 Department of Computer Science, FEECS, VŠB - Technical University of Ostrava, Ostrava-Poruba, Czech Republic
* Corresponding author E-mail: jan.martinovic@vsb.cz

Received 31 Aug 2012; Accepted 16 Jul 2013

DOI: 10.5772/56827

Abstract Mobile embedded systems belong among the typical applications of distributed systems control in real-time. An example of a mobile control system is a robotic system. The proposal and realization of such a distributed control system represents a demanding and complex task for real-time control. In the process of robot soccer game applications, extensive data is accumulated. The reduction of such data is a possible win in a game strategy. The main topic of this article is a description of an efficient method for rule selection from a strategy. The proposed algorithm is based on the geometric representation of rules. A described problem and a proposed solution can be applied to other areas dealing with effective searching of rules in structures that also represent coordinates of the real world. Because this construed strategy describes a real space and the stores physical coordinates of real objects, our method can be used in strategic planning in the real world where we know the geographical positions of objects.

Keywords Robot Soccer, Strategy, Rule

1. Introduction

This introduction contains an explanation of our view of strategies and rules and how we use them for mapping coordinates of the real world. In the next part of the article, our method is practically applied to a robot soccer game in which we can see the benefits of the presented solution to the existing algorithm. The algorithms and approaches from this article may serve beyond their mere use in games of robot soccer.

A complete set of options that are available to players in any game situation in order to achieve the objective is considered as a strategy in game theory [1, 13]. The result of this strategy depends not only on the actions of the individual player but also on the actions of other players or elements of the game. A so-called pure strategy contains a list of all possible situations that may arise in the game. For strategy [2] can be considered any mapping or description of the space in which we know the geographical positions of the objects located in it, and we have defined a finite set of rules that tell us how these objects can behave in a given situation. This principle can be applied to a number of areas of the real world and is generally called 'strategy planning' [3]. From our perspective, the best approach is to use such strategies to describe a space and the objects in it, and to use the subsequent search for the optimal path or relocation of these objects in order to achieve our desired goals. In other words, we are interested in strategies dealing with

planning. A practical example might be a strategy for traffic control at the airports [4]. The strategy represents the area of the airport and also provides rules upon which are selected the optimal paths for aircrafts towards assigned runways. The following section will explain the importance of strategy in the game of robot soccer, in which our method was also practically tested.

2. Strategies in robot soccer games

In this section, the problems of a robot soccer game are described. Robot soccer has been selected as a sample topic because it contains a number of challenges in the areas of robot control, image analysis and the above-mentioned strategic planning. Strategy is inherently connected with robot soccer games. The basic idea of robot soccer is the same as in real football, namely to win over the opponent by the number of scored goals. Just like in a real football, the team that usually wins is the team with the better strategy.

Robot soccer games can be divided into two main types: games with autonomous control and those with centralized control. In games based on autonomous control, the players are considered to be mutually independent agents. Each player only has information about its surroundings, which in most cases contains information simulating a field of view of the player. Each player also has its own strategy or a set of rules that defines how it should behave in the current situation. By a 'situation' is meant information about those objects that are currently in the player's field of view. The main representative of this category is undoubtedly the international robotics competition RoboCup [16].

The second type of game involves games with central management. We also use this type of game in this article [5]. As the name suggests, there is one central element that has information about all the objects located on the game field at every step of the game. The behaviour of the robots in specific situations is controlled by one central strategy that controls all the robots of the controlled team using globally-available information (the coordinates of the ball, the coordinates and rotations of the robots, etc.) describing the situation on the game field. This game type is represented by The Federation of International Robot-soccer Association (FIRA).

In practical terms, strategies are used according to various methods. For example, strategies based on standard decision trees [6] where actions are selected based on the robots' position in relation to the ball, the owner of the ball, the distance to the goal, etc. Another frequently-used method is a strategy based on fuzzy logic [7-9], where game situations are divided into several levels and where the corresponding types of behaviour of robots are assigned to them. Many teams participating in some of the categories of RoboCup address autonomous robot control methods based on case-based reasoning [17, 18], which is an artificial intelligence problem-solving technique that catalogues experience into cases and matches the current problem to the experience. On the other hand, teams participating in FIRA competitions use a number of methods dealing with the centralized control of robots.

We can encounter combinations of the above-mentioned approaches or even vastly different ones. The main idea is still the same: to create a system that will respond as quickly as possible to changes of situations in the game field by the best possible relocation and control of the robots in order to score in an opponent's goal. In the following sections, we present our method for the quick and effective selection of rules from a strategy in a robot soccer game.

3. Strategy for our robot soccer game

Our work is based on the FIRA category called 'SimuroSot', which uses the centralized control of robots. This category contains a FIRA simulator as the game server simulating a game environment and the two client programs with an implemented strategy for the left or the right team. Both playing teams have access to the global information describing the current situation on the field at each game moment. Next, prepared strategies decide what actions the controlled team must perform, based on this information.

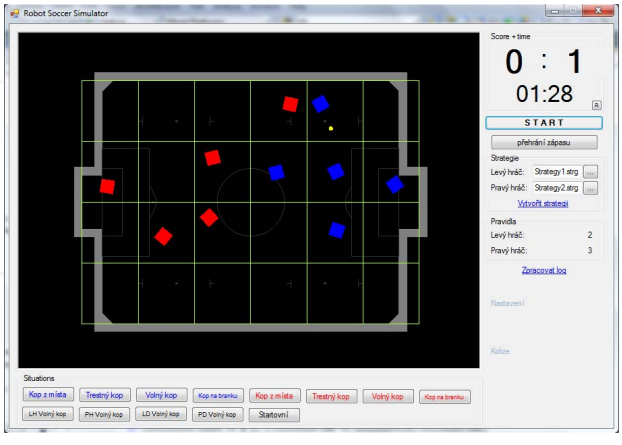

Figure 1. 2D robot soccer simulator

Our goal was to create a unified and robust library for a robot soccer game that could be used in a real robot game and a game running within the simulator. Such a proposed system was built from interconnected modules containing the necessary functionality for prediction, image analysis and robot control, etc. This architecture allowed us to perform experiments with different

methods used for the efficient processing of the strategy or else to create a partially-simulated game containing both real and simulated robots. For the extensive testing of these methods, we created a 2D robot soccer simulator (see Figure 1) that was also used for the experiments presented in this paper.

In our work, we have two types of representation of the game field. The first type is an abstract coordinate system. In it, the physical coordinates of the real objects on the field are mapped to a standard coordinate system (see Figure 2). Because this logical representation is very detailed, it is used mainly for actual robot control and image analysis, which requires the greatest possible accuracy. However this very detailed representation is not suitable for the strategy's description. As mentioned above, the strategy contains a set of rules that describes how the robots should behave in a given situation. A strategy based on a very detailed coordinate system would have to include a large number of rules, and most of these rules would have been very similar so as to almost describe the actual situation on the game field.

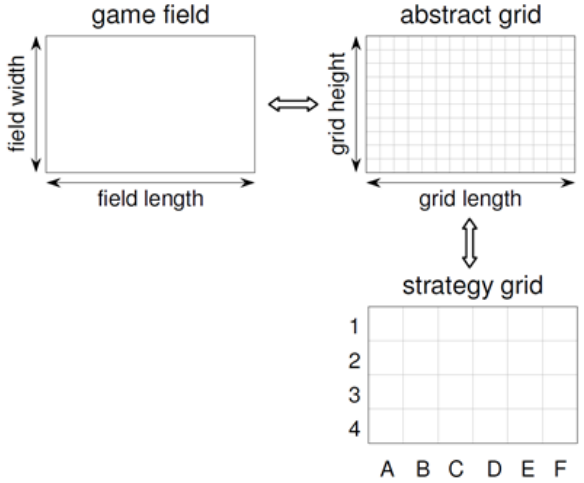

Figure 2. Inner game field representation

Therefore we introduce the term so-called 'grid coordinates'. Grid coordinates have much lower resolution and are only used for the definition of strategies and underlying rules (see Figure 2). This simplification is sufficient for us because for the purposes of the strategic planning we do not need to know the exact position of each robot. All we need is the approximate location of the robot represented by the grid coordinates.

By using grid coordinates, we reduce the accuracy of the mapping of the physical coordinates of the robots into the logical ones but, on the other hand, we dramatically reduce the number of the rules needed to create a strategy. If necessary, it is possible to convert the grid coordinates back to the physical and to use them for the aforementioned robot control. The robot's behaviour in the grid coordinates is then controlled by so-called 'tactics' [10]. The tactics contain functions for robot control, such as turning the robot, shooting at goal or passing the ball. Therefore, in terms of hierarchy, the strategy takes care of the placement of the robots in the grid coordinates while at the lower level, under the strategy, the tactics are executed. The tactics work with the physical coordinates and controls the robot inside the grid. The strategy, as we understand it, is the quaternion $< X, Y, p, m >$ where:

- $X \subset X^U$ where X^U is the universe of all situations that may occur during the game and X is a set of game situations defined by the strategy,
- $Y \subset Y^U$ where Y is a set of instructions that the strategy uses to control the robots and Y^U is the universe of all possible instructions,
- p is a projection $X \to Y$,
- m is a projection $X^U \to X$,
- $p(m(\overline{x}_u)) \in X^U$ where $\overline{x}_u \in X^U$: every real situation on the game field is assigned by a game situation from the strategy, which itself also contains instructions as to where to move the robots.

The selection of the appropriate rule from the strategy depends on the current game situation. This situation is represented by the current state of the game in robot soccer. This state contains information about the position of the robots and the ball on the game field as well as their rotations and speeds. However, the most important data for the strategy includes the grid coordinates of all the objects on the game field.

Example rule	**Red team rule**
Mine A1 A2 B1 B2	**M**ine 11 12 21 22
Opponent B1 B2 C1 C2	**O**pponent 21 22 31 32
Ball B2	**B**all 22
Destination A1 B2 C1 C2	**D**estination 11 22 31 32

Table 1. Strategy rule

As we have explained, the strategy is a finite set of rules that describe the current situation on the game field and, thus, says where to move the robots in that situation. Each rule can be easily expressed as a quaternion (M, O, B, D), where M denotes the grid coordinates of the robots, O denotes the coordinates of the opponent's robots, B denotes the coordinates of the ball and D denotes the coordinates of the desired destination of the robots. An example of such a rule is given in Table 1.

Because this system is based on the centralized control of every step of the game, we have access to the grid coordinates of each object on the game field. Thus, at every step of the game we can compute how similar the

real situation on the game field is to that described by the selected rule from our strategy. Table 1 gives an example of a rule created for a 5-member team of robots. The rule consists of just 4 robots because the strategy does not include the goalkeeper. The only role of the goalkeeper is to protect the goal, and so it is not necessary to include it in the strategy. For graphical representation of this rule see Figure 3.

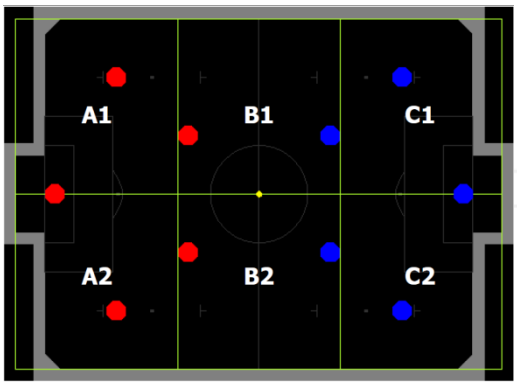

Figure 3. Situation on the game field

4. Existing problems

The current approach to our strategies is based on the description presented above. Therefore, at every step of the game we compare the current situation on the field with those situations described in the rules of the strategy. By 'comparing' is meant the computation of the Euclidean distance between the real situation on the field and the situation described by the selected rule. The Euclidean distance is computed from the grid coordinates of the ball, the friendly robots and the opponent robots. The resulting number is computed as the sum of these three individual distances and represents the distance between the real situation and the situation in a given rule. Afterwards, the rule with a minimum Euclidean distance is selected from the strategy. From this selected rule, the grid coordinates marked as 'destination' are used to move the friendly robots to designated locations on the game field.

From this description, one of the main problems of this approach becomes obvious. At each step of the game, we must constantly calculate the distance between the real situation on the game field and all the rules contained in the used strategy. With any increase of the number of rules in the strategy, we also increase the time needed to find the closest rule. This at first glance may not seem like a serious problem, but when it is realized that a given moment of one cycle of the game takes several milliseconds, we must perform this computation almost constantly. Thus, any time saved while finding the closest rule allows us to respond to changing situations on the game field as quickly as possible.

Another problem with this architecture relates to the identification of the robots. For identification, we consider their coloured markings. Our designed architecture does not include robots that are uniquely identifiable by IDs or assigned roles. This is because of the desire for greater discretion during the game. For example, in a specific game situation, a defender and an attacker can switch places during an attack on goal, if possible. The players do not need to be identified by unique IDs or roles. However, this poses problems during the selection of the closest rule from the strategy. Each rule only tells us that some robot can be found in the grid coordinates. However, it does not tell us specifically that this particular robot must be located in the grid coordinates. This was also one of the requirements of the design of our architecture (i.e., the mutual independence of strategies and the robots' identification).

To calculate the distances of robots, it is necessary to work with the real robots. This issue was solved using a permutation without repetition. For each rule the, distance between robots is not computed only once but k times, where k means a number of robots. Then, it is always selected from this permutation the mapping with the smallest value of the calculated distance and it is considered to be the final distance between the real situation and the situation described by the selected rule. This approach causes that the increasing number of robots in the game also increases exponentially the number of operations required to find the closest rule from the strategy. Our presented method attempts to preserve the concept of a general and independent strategy but with the eliminating the above mentioned shortcomings.

5. Our approach

The main area of our interest lies in the optimization of rule searching from the strategy using a graph and so-called 'space-filling curves' [11]. A space-filling curve (SFC) is a way of mapping a multidimensional space onto a one-dimensional space [19, 11]. An SFC is a path through the points of a discrete Cartesian space that passes through each point exactly once. There are many types of SFCs. Typically, an SFC is more likely to connect points that are close. The family of Hilbert curves [22] is known to have good locality-preserving properties. However, they are complex to construct. A much simpler curve to construct is the *Z-curve* [12], which is the main reason for using this type of SFC in this article. Like other space-filling curves, a *Z-order* curve maps data points in a multidimensional space onto a one-dimensional space, where each point is represented by a unique number, called *Z-address* or a *Morton code*. [23, 24]. A *Z-address* is a bit string calculated by interleaving the bits of all the coordinate values of a data point. For a d-dimensional

space with ([0; 2^v-1]) as the coordinate value domain ranges, the *Z-address* of a data point contains *dv* bits. The *i*-th bit of a *Z-address* is contributed by the *(i/d)*-th bit of the *(i%d)*-th coordinate. For example, the *Z-address* of p = (1; 6; 3; 7) (i.e., (001; 110; 011; 111) in binary) is 010101111011. This *Z-address* calculation is reversible, so the original coordinate can be recovered from its corresponding *Z-address*. *Z-order* is a good approximation of geometric similarity [21]. The *Z-order* can be computed very efficiently using bit interleaving of the point coordinates in the Cartesian space [23].

```
RecursiveZ(int level, int x, int y)
{
  if(level=0)
    return;
  if(level>1){
    RecursiveZ(level-1, 2*x, 2*y) ;
    RecursiveZ(level-1, 2*(x+1)+1, 2*y) ;
    RecursiveZ(level-1, 2*x, 2*(y+1)) ;
    RecursiveZ(level-1,2*(x+1)+1, 2*(y+1)) ;
  }
  addToIndexList(x, y);
  addToIndexList(x+1, y);
  addToIndexList(x+2, y);
  addToIndexList(x+1, y+1);
  addToIndexList(x+2, y+1);
}
```

Table 2. Z-order pseudocode

The algorithm for computing the *Z-order* curve is based on this method by using its recursive nature. Due to its properties, it is suited for converting a two-dimensional matrix representing the playing field into a one-dimensional array of the coordinates of the individual robots.

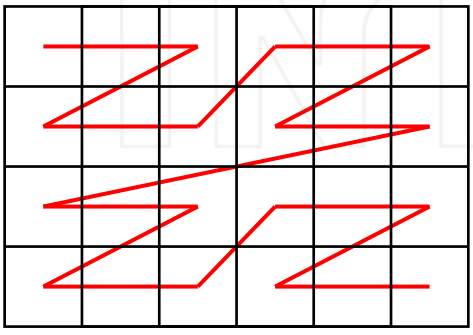

Figure 4. Grid 6x4 with Z-order curve

Thus, if we use the *Z-order* to arrange the robots' coordinates in the real situation on the game field and then to arrange the robots coordinates within each set of rules, all we need to do next is compute the distance between these two sorted sequences. Accordingly, we do not have to solve the issue of the ambiguity of the robots using permutations. The removal of these permutations will mean that even with a large number of robots on the game field there is no longer an exponential increase in the time needed for computation. The algorithm in its original form is created for a square space, so it was slightly modified for our game field with dimensions 6x4. A pseudocode algorithm for the computation of the *Z-order* curve can be seen in Table 2 and Figure 4 shows the final relocation of the robots located on the game field.

However using a *Z-order* curve does not eliminate the need to compute the distances between the real situation and all the rules defined in the strategy in every step of the game. Therefore, it was necessary to completely change the principle of how the individual rules from the strategy would be chosen. To address this problem, we decided to use an undirected connected graph with edge valuations. It is suitable for our purposes both in terms of the effective searching of similar rules as well as from the perspective of a relatively easy visualization of the rules contained in the strategy.

Let the graph be defined as a pair G = <V, E> where V is a non-empty set of vertices and E is a set of two-element sets of vertices, also called 'undirected edges'. In our case, the set of vertices consists of the individual rules from the strategy. The edges contain an evaluation that corresponds to a distance between the two neighbouring vertices (rules). A distance is considered to be a normalized value of a Euclidean distance computed from two sorted sequences using the above-mentioned *Z-order* applied to the neighbouring vertices that contain the robots' grid coordinates.

However, this graph will contain all the edges between all the defined vertices. The main idea of this graph lies in the fact that very similar rules to those of the strategy will be connected by a strong bond (a small distance) while the rules that differ fundamentally will be connected with a weak bond (a large distance). Thus, we do not have to compare the current situation on the game field with every rule from the strategy at each game step. It is sufficient to compare the current situation only with the neighbouring rules connected by the strong bond. Basically, each current step of the game is represented by some real situation on the game field and is mapped to a certain rule from the strategy. We assume that in the next step (which, in our simulator, takes only several ms) the situation on the game field does not change sufficiently to have required the choice of some completely different rule from the strategy. In other words, the robots will not have enough time to move a sufficiently large distance over the game field and, therefore, the currently selected rule will not change or else some very similar rule will be selected instead. We can pre-compute this strategy graph before the start of the game because the rules of the strategy do not change during the game. During the game itself, we use it only to quickly find those rules of the

strategy that most closely match the real situation on the game field.

For greater clarity, we compare the time complexity of the two presented algorithms. The first algorithm at each game moment compares the current situation on the field with all those situations that are represented by the rules in the strategy. In addition, the algorithm needs to compute the permutations of all the robots on the field (except for the goalkeeper) for both the left and the right team. This time complexity can be expressed as:

$$O (N_{rules} * (P (N_{left} - 1) + P (N_{right} - 1)))$$

where N_{rules} is the number of rules in the strategy, P is the permutation without repetition, N_{left} is the number on robots of the left team and N_{right} is the number of robots on the right team.

On the other hand, the presented optimization utilizing a *Z-order* to arrange the robots' coordinates and an undirected connected graph with edge valuations for the quick searching of the rules can be expressed by the following two steps:

1. Pre-computation of the graph of rules: $O (P (N_{rules})$, where P is a permutation without repetition and N_{rules} is the number of rules in the strategy,
2. Rule selection: $O (S_{rules} * (Z_{left} + Z_{right}))$, where S_{rules} is the selected number of neighbouring vertices for searching, Z_{left} is one comparison of two *Z-order* arrays representing robots on the left team and Z_{right} is one comparison of two *Z-order* arrays representing robots on the right team.

This idea is also based on the experience that a majority of strategies consist of several so-called 'actions'. These actions are based on the selected subset of rules from the strategy. These actions are not strictly defined in the strategy, but may be created by a user during the construction of the strategy. For example, we can create a strategy that consists of 15 rules. The first five rules can represent the action for an attack on the right flank, the next five rules can represent an attack on the left flank and the last five rules can be for defence. Thus, we have created a strategy consisting of 15 rules which represents only three actions as intended by the user. We are mainly interested in the fast searching of the closest rules from the strategy and in the smoothest execution of the follow up actions, which are defined by these rules.

6. Experiments

For our experiments, we created a sample strategy consisting of 18 rules. This strategy was created for a game with 11 robots on each team. Therefore, every rule in the strategy contains 10 coordinates for each of the robots (the goalkeeper is not controlled by the strategy). All the rules of this strategy can be seen in table 3. This strategy was created in our tool for the simple and intuitive creation of strategies, called the 'strategy creator'.

Example strategy	
Rule 1	
Mine	3,2 3,3 3,2 3,3 2,2 2,4 2,1 2,4 1,1 1,4
Opponent	4,2 4,3 4,2 4,3 5,2 5,4 5,1 5,4 6,1 6,4
Ball	4,3
Move	4,2 4,2 3,2 4,3 2,2 2,3 3,1 3,3 2,1 2,4
Rule 2	
Mine	4,2 4,2 3,2 4,3 2,2 2,3 3,1 3,3 2,1 2,4
Opponent	4,2 4,3 4,2 4,3 5,2 5,3 4,1 4,4 6,1 6,4
Ball	4,2
Move	5,2 5,2 4,2 5,3 3,2 3,3 3,1 3,4 2,1 2,4
Rule 3	
Mine	5,2 5,2 4,2 5,3 3,2 3,3 3,1 3,4 2,1 2,4
Opponent	4,2 4,3 5,2 4,3 5,2 5,3 4,1 4,4 6,1 6,4
Ball	5,2
Move	6,2 5,2 4,2 6,3 3,2 3,3 3,1 3,4 2,1 2,4
Rule 4	
Mine	6,2 5,2 4,2 6,3 3,2 3,3 3,1 3,4 2,1 2,4
Opponent	4,2 4,3 6,2 5,3 5,2 6,3 5,1 4,4 6,1 5,4
Ball	6,2
Move	6,2 5,2 4,2 6,3 3,2 3,3 3,1 3,4 2,1 2,4
Rule 5	
Mine	3,2 3,3 3,2 3,3 2,2 2,4 2,1 2,4 1,1 1,4
Opponent	4,2 4,3 4,2 4,3 5,2 5,4 5,1 5,4 6,1 6,4
Ball	4,3
Move	4,3 4,3 3,2 3,3 3,2 2,3 2,2 3,4 2,1 2,4
Rule 6	
Mine	4,3 4,3 3,2 3,3 3,2 2,3 2,2 3,4 2,1 2,4
Opponent	4,2 4,3 4,2 5,3 4,1 5,4 5,2 4,4 6,1 6,4
Ball	4,3
Move	4,3 4,4 3,2 3,4 3,1 3,3 2,2 4,4 2,1 2,4
Rule 7	
Mine	4,3 4,4 3,2 3,4 3,1 3,3 2,2 4,4 2,1 2,4
Opponent	4,3 4,3 4,2 5,3 4,1 5,4 5,1 4,4 6,1 6,4
Ball	4,4
Move	4,3 5,3 3,2 4,4 3,1 3,3 2,2 5,4 2,1 3,4
Rule 8	
Mine	4,3 5,3 3,2 4,4 3,1 3,3 2,2 5,4 2,1 3,4
Opponent	4,3 4,3 4,2 5,3 4,1 5,4 5,1 4,4 6,1 6,4
Ball	5,3
Move	5,3 6,3 4,2 5,4 4,1 3,3 2,2 6,3 2,1 3,4
Rule 9	
Mine	5,3 6,3 4,2 5,4 4,1 3,3 2,2 6,3 2,1 3,4
Opponent	5,3 4,3 4,2 5,2 5,2 6,3 5,1 5,4 6,1 6,4
Ball	6,3
Move	5,3 6,3 4,2 5,4 4,1 3,3 2,2 6,3 2,1 3,4
Rule 10	
Mine	3,2 3,3 3,2 3,3 2,2 2,4 2,1 2,4 1,1 1,4
Opponent	4,2 4,3 4,2 4,3 5,2 5,4 5,1 5,4 6,1 6,4
Ball	4,3
Move	3,2 3,3 3,2 3,3 2,2 2,3 3,1 3,4 2,1 2,4

Rule 11	
Mine	3,2 3,3 3,2 3,3 2,2 2,3 3,1 3,4 2,1 2,4
Opponent	3,2 3,3 3,1 4,3 4,2 4,4 4,1 5,3 6,1 5,4
Ball	3,2
Move	3,2 3,3 2,2 2,3 2,2 1,3 2,1 3,4 1,2 2,4
Rule 12	
Mine	3,2 3,3 2,2 2,3 2,2 1,3 2,1 3,4 1,2 2,4
Opponent	2,2 2,3 2,1 3,3 3,2 4,4 4,1 4,3 6,1 5,4
Ball	2,2
Move	3,2 3,3 1,2 2,3 2,2 1,3 1,1 2,4 1,2 1,4
Rule 13	
Mine	3,2 3,3 1,2 2,3 2,2 1,3 1,1 2,4 1,2 1,4
Opponent	2,2 2,3 2,1 3,3 3,2 4,4 4,1 4,3 6,1 5,4
Ball	1,2
Move	3,2 3,3 1,2 2,3 2,2 1,3 1,1 2,4 1,2 1,4
Rule 14	
Mine	3,2 3,3 3,2 3,3 2,2 2,4 2,1 2,4 1,1 1,4
Opponent	4,2 4,3 4,2 4,3 5,2 5,4 5,1 5,4 6,1 6,4
Ball	4,3
Move	3,2 3,3 3,2 3,3 2,2 2,4 2,1 2,4 1,1 1,4
Rule 15	
Mine	3,2 3,3 3,2 3,3 2,2 2,4 2,1 2,4 1,1 1,4
Opponent	3,2 3,3 4,2 4,3 4,1 4,4 5,2 5,4 5,1 6,4
Ball	3,3
Move	2,3 3,4 2,2 3,4 2,2 2,4 3,1 2,4 1,2 1,3
Rule 16	
Mine	2,3 3,4 2,2 3,4 2,2 2,4 3,1 2,4 1,2 1,3
Opponent	3,2 3,4 4,2 4,3 4,1 4,4 5,2 5,4 5,1 6,4
Ball	3,4
Move	2,3 3,3 2,2 3,4 2,2 2,4 3,1 2,4 1,2 1,3
Rule 17	
Mine	2,3 3,3 2,2 3,4 2,2 2,4 3,1 2,4 1,2 1,3
Opponent	2,2 2,3 3,2 3,3 4,1 4,4 4,2 4,3 5,1 5,3
Ball	2,3
Move	2,3 3,3 2,2 3,4 2,1 2,4 3,1 1,4 1,2 1,3
Rule 18	
Mine	2,3 3,3 2,2 3,4 2,1 2,4 3,1 1,4 1,2 1,3
Opponent	1,2 1,3 2,2 2,3 4,1 3,4 4,2 4,3 5,1 5,3
Ball	1,3
Move	2,3 3,3 2,2 3,4 2,1 2,4 3,1 1,4 1,2 1,3

Table 3. Example strategy used in the experiments

As mentioned in section 4, one of the main reasons why we decided to use a graph and space-filling curves instead of the currently-used permutations was time efficiency. The previous implementation was used mainly for games with five robots on each team. This number of robots was sufficient for the initial experiments. Despite the number of permutations of the all robots and the constant need to search through every rule in the strategy, we achieved an average time of 1 ms per search (one strategy moment). However, with an increasing number of robots on the game field, there was a rapid deceleration in time efficiency. With 11 robots per team, one search took an average of 142 seconds, which for the real game is a completely unusable solution.

Previous version using permutations	
Robots per team	Time of one strategy moment (one search)
5	0.9642 ms
7	198.5552 ms
9	1 589.3497 ms
11	142 917.1025 ms
Current version using a *Z-order* and graph with 11 robots per team	
Number of neighbouring vertices for searching	Time of one strategy moment (one search)
5	0.2444 ms
10	0.2672 ms
15	0.2979 ms

Table 4. Time comparison of two implementations of rules selection from the strategy

A new implementation utilizing a *Z-order* for the sorting of the robots' coordinates and a graph for the representation of the rules achieved significantly better results. The experiments with the new implementation were always created for games with 11 robots per team. The only parameter that has been changed was the number of neighbouring vertices used for the searching during each game step. The sample strategy contains a total of 18 rules, and so every vertex in the graph will have 17 neighbours. For each search, the following rule is chosen from a selected number of the nearest neighbours. Table 4 shows the final comparison of both approaches in terms of the time required for the computation of the following rule.

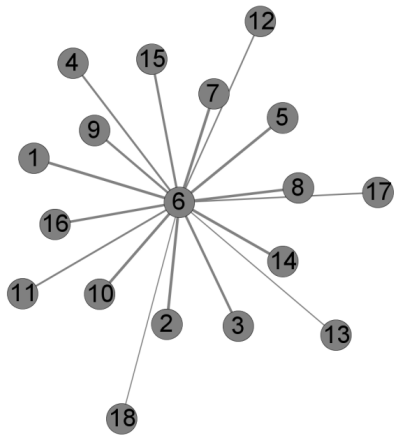

Figure 5. Rule selection using a graph

Figure 5 illustrates how the following rule is selected during each game step. For simplicity, let us say that the currently-used rule is rule 6. From our pre-computed graph, we can get all 17 neighbours of rule 6 and the evaluation of all the connected edges. Next, we can create a selected subset of rules according to the required number of nearest neighbours we want to use for the comparison.

In the next step, we only need to compare the real situation on the game field with the situations described by the selected neighbours and the currently selected rule. This is because the current rule does not necessarily have to be changed at each game step. The most similar rule is then returned as a result of searching.

Time needed to pre-compute the strategy graph	Number of rules in the strategy			
	18	50	100	200
	14 ms	18 ms	25 ms	48 ms
	Number of rules in the strategy			
	500	1000	2000	5000
	224 ms	854 ms	3411 ms	22144 ms

Table 5. Pre-computation of the strategy graph

The time of the first search [ms]				
Neigh. vertices	Number of rules in the strategy			
	18	50	100	200
5	12.4671	13.1436	12.3526	12.4121
10	13.1754	14.0021	13.4584	15.9173
15	13.1247	14.0465	14.0344	13.4878
25	-	14.4588	15.4874	14.1972
Neigh. vertices	Number of rules in the strategy			
	500	1000	2000	5000
5	15.8301	16.6385	17.9907	18.5728
10	15.9452	17.1577	18.4847	19.1544
15	15.4454	16.4877	17.9443	18.7314
25	15.9879	17.7984	18.4772	20.1478

Table 6. The first search

The time of the common search [ms]				
Neigh. vertices	Number of rules in the strategy			
	18	50	100	200
5	0.2993	0.2087	0.2179	0.2379
10	0.3539	0.4143	0.4575	0.4972
15	0.4235	0.4798	0.5646	0.7871
25	-	0.4587	0.5887	0.6448
Neigh. vertices	Number of rules in the strategy			
	500	1000	2000	5000
5	0.4949	0.6094	0.9411	1.8443
10	0.5421	0.7989	0.9781	1.7854
15	0.7554	0.8121	1.4875	1.6778
25	0.7945	0.9877	1.5751	1.7887

Table 7. The common search

It is important to emphasise that because we compare the real situation on the game field (which may not be described by rule of the strategy) and the situations described by the rules of the strategy, the winning rule may not be always the same as the nearest neighbour from the graph.

For further experiments, we prepared strategies of 50, 100, 200, 500, 1,000, 2,000 and 5,000 rules. For each strategy, we measured the time required to pre-compute the strategy graph (Table 5) and the average search time of one rule during the game (Table 7). It is important to note that at the beginning of the game (i.e., during the first game moment) the rule from the previous game moment is not known. Therefore, it is necessary to search from all of the neighbouring vertices in the graph instead of just a selected subset of the most similar rules. Table 6 contains the measured times of the first search. Table 5 shows that the time required to pre-compute the graph grows rapidly with the increasing number of rules. On the contrary, the time required to search for the rule changes slowly. The time of the first search is in the tens of ms and the times of the subsequent searches are in units of ms. The experiments were performed on a desktop computer with a processor Intel Core i5 3.4GHz with 8GB RAM DDR3 1600MHz.

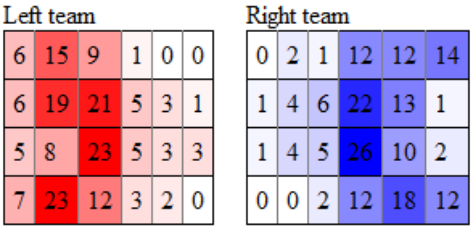

Figure 6. The places with the most frequent occurrence of the robots

The places with the most frequent occurrence of the robots on the playground could also be determined by a simple analysis of the created strategy. Figure 6 shows the players of both teams on the game field defined by the rules of this strategy (grid coordinates for friendly and opponent robots). The evaluation of the individual grid cells was calculated as the sum of all the robots of the team in the cell which is defined by rule.

7. Conclusion

In this work, strategies for robot soccer games were discussed. The description of the approach for strategies' definitions was presented. The main part of the article discussed the optimization of the best rule selection in a real-time running robot soccer game based on a *Z-order* and graphs.

A *Z-order* is representative of the category of space-filling curves. It is a function for mapping data from a multidimensional space onto a one-dimensional space while preserving the locality of data points. A *Z-order* is therefore highly suitable for the quick mapping of the two-dimensional array, representing the game field, onto a one-dimensional array containing the coordinates of each robot. With this ordered array of robots, it is thus substantially easier to work, especially during the phase of rule selection. Utilizing a graph to represent the rules of the strategy brings various benefits, not only in terms of computing speed but also in terms of the relatively easy visualization of the strategy which, for example, allows us to reveal its weaknesses.

Games can in general represent any situation in nature. In [14], game theory is applied for adversarial reasoning in security resource allocation and scheduling problems. In [14], the authors showed that randomized policies mitigate a key vulnerability of human plans: predictability. In [15] methods for strategy extraction are described.

8. Acknowledgments

This work was supported by Grant SGS No. SP2013/70, VŠB - Technical University of Ostrava, Czech Republic and by Grant SGS No. SP2013/167, VŠB - Technical University of Ostrava, Czech Republic. We would like to thank the students on the bachelor's and master's courses at the VŠB - Technical University of Ostrava, who participated in the development and further improvement of the simulator for robot soccer that was used to create the above-mentioned experiments.

9. References

[1] M. J. Osborne (2004) An introduction to game theory. New York Oxford, Oxford University Press.

[2] C. F. Camerer (2003) Behavioral Game Theory: Experiments in Strategic Interaction. Princeton University Press.

[3] S. Ontanón, K. Mishra, N. Sugandh and A. Ram (2007) Case-based planning and execution for real-time strategy games. Lecture Notes in Computer Science, Volume 4626, 164-178.

[4] M. A. Stamatopoulos, K. G. Zografos and A. R. Odoni (2004) A decision support system for airport strategic planning. Transportation Research Part C: Emerging Technologies, Volume 12, 91–117.

[5] J. Martinovic, V. Snasel, E. Ochodkova, L. Zołta, J. Wu and A. Abraham (2010) Robot soccer - strategy description and game analysis. Modelling and Simulation, 24th European Conference ECMS 2010.

[6] H. P. Huang and C. C. Liang (2002) Strategy-based decision making of a soccer robot system using a real-time self-organizing fuzzy decision tree. Fuzzy Sets and Systems 127, 49-64.

[7] P. J. Thomas and R. J. Stonier (2003) Fuzzy control in robot-soccer, evolutionary learning in the first layer of control. Journal of Systemics, Cybernetics and Informatics, Volume 1, 75-80.

[8] H. L Sng., G. S. Gupta and C. H. Messom (2002) Strategy for collaboration in robot soccer. Electronic Design, Test and Applications, 347-351.

[9] K. Jinwook Kim, K. Yoon-Gu and A. Jinung (2011) A fuzzy obstacle avoidance controller using a lookup-table sharing method and its applications for mobile robots. International Journal of Advanced Robotic Systems, Vedran Kordic, Aleksandar Lazinica, Munir Merdan (Ed.), InTech

[10] G. Klancar, M. Lepetic, R. Karba and B. Zupancic (2003) Robot soccer collision modelling and validation in multi-agent simulator. Mathematical and Computer Modelling of Dynamical Systems: Methods, Tools and Applications in Engineering and Related Sciences, Volume 9, 137-150.

[11] H. Sagan (1994) Space-filling curves. Springer-Verlag.

[12] G. M. Morton (1966) A computer oriented geodetic database and a new technique in file sequencing. Technical Report, IBM Ltd. Ottawa, Canada.

[13] J-H. Kim, D-H. Kim, Y-J. Kim, K. T. Seow (2010) Soccer robotics. Springer Tracts in Advanced Robotics

[14] M. Jain, J. Tsai, J. Pita, C. Kiekintveld, S. Rathi, M. Tambe and F. Ordonez (2010) Software assistants for randomized patrol planning for the LAX airport police and the Federal Air Marshals Service. Interfaces, Volume 40, 267–290.

[15] V. Srovnal, B. Horak, R. Bernatik and V. Snasel (2004) Strategy extraction for mobile embedded control systems apply the multi-agent technology. International Conference on Computational Science, Springer LNCS, 631-637.

[16] H. Kitano, M. Asada, Y. Kuniyoshi, I. Noda and E. Osawa (1997) RoboCup: the Robot World Cup Initiative. AGENTS '97 Proceedings of the First International Conference on Autonomous Agents, 340-347.

[17] A. Aamodt and E. Plaza (1994) Case-based reasoning: foundational issues, methodological variations and system approaches. Artificial Intelligence Communications, Volume 7, Number 1, 39-52.

[18] R. Ros, J. L. Arcos, R. L. de Mantaras and M. Veloso (2009) A cased-based approach for coordinated action selection in robot Soccer. Artificial Intelligence, Volume 173, Numbers 9-10, 1014-1039.

[19] W. G. Aref and I. Kamel (2000) On multi-dimensional sorting orders. DEXA 2000, Springer Verlag, 774–783.

[20] M. Connor and P. Kumar. (2010) Fast construction of k-nearest neighbor graphs for point clouds. Visualization and Computer Graphics, IEEE Transactions on, Volume 16, Number 4, 599 -608.

[21] M. F. Mokbel, W. G. Aref and I. Kamel (2003) Analysis of multi-dimensional space-filling curves, GeoInformatica, Volume 7, Number 3, 179–209.
[22] A. R. Butz (1971) Alternative algorithm for Hilbert space filling curve. IEEE Trans. on Computers, 20:42442.
[23] H. Samet (2006) Foundations of multidimensional and metric data structures, The Morgan Kaufmann Series in Computer Graphics.
[24] T. Skopal, M. Kratky and V. Snasel (2002) Properties of space filling curves and usage with UB-trees. ITAT 2002.