

Genetic Programming Method of Evolving the Robotic Soccer Player Strategies with Ant Intelligence

R. Geetha Ramani¹, R. Subramanian², P. Viswanath³

¹Assistant Professor, Dept. Of CSE & IT, Pondicherry Engineering College

²Professor, Dept. Of Computer Science, Pondicherry University

³Department of Computer Science, Pondicherry University

Corresponding author E-mail: rgeetha@yahoo.com, visucse@yahoo.com

Abstract: This paper presents the evolved soccer player strategies with ant-intelligence through genetic programming. To evolve the code for players we used the Evolutionary Computation tool (ECJ simulator-Evolutionary Computation in Java). We tested the evolved player strategies with already existing teams in soccerbots of teambots. This paper presents brief information regarding learning methods and ant behaviors. Experimental results depicts the performance of the evolved player strategies.

Keywords: Robotic Soccer, Social Insect Behaviors, Ant intelligence, Learning methods, Genetic Programming, E CJ simulator, Teambots.

1. Introduction

Many people discovered the variety of the interesting insect or animal behaviors in the nature. A flock of birds sweeps across the sky. A group of ants forages for food. A school of fish swims, turns, flees together [1]. Hive's of bee communicates using dance language. In fact the honeybee dance language has been called one of the seven wonders of animal behaviors and is considered among the greatest discoveries of behavioral science [2]. Termites are small in size, completely blind and wingless - yet they have been known to build mounds 30 meters in diameter and several meters high [3]. We call this kind of aggregate motion "swarm behavior". Recently biologists and computer scientists in the field of "artificial life" have studied how to model biological swarms to understand how such "social animals" interact, achieve goals, and evolve. Moreover, engineers are increasingly interested in this kind of swarm behaviors since the resulting "swarm intelligence" can be applied in optimization, robotics, traffic patterns in transportation systems, and military applications etc.

In our work we used Genetic programming to evolve the code for player strategies. Genetic Programming (GP) is considered to be a domain independent problem solving approach in which computer programs are automatically evolved to solve problems.

The idea of mapping ant behaviors especially foraging and nest protecting behaviors to Robotic Soccer is presented in this paper.

This paper will highlight the important points of robotic soccer, genetic programming, and simulation tool used, in the subsequent sections. Finally it presents experimental results and conclusion.

2. Robotic Soccer

Robot Soccer Competition give an opportunity to foster intelligent techniques and intelligent robotic research by providing a standard problem where a wide spectrum of technologies such as collaborative multiple agent robotics, autonomous computing, real-time reasoning and sensor fusion can be developed, tested and integrated.

2.1. Research Issues And Approaches

Several research issues are involved in the development of real robots and software agents for RoboCup. One of the major reasons why RoboCup attracts so many researchers is that it requires the integration of a broad range of technologies into a team of complete agents, as opposed to a task-specific functional module. The following is a partial list of research areas, which RoboCup covers [7]:

- Agent architecture in general;
- Combining reactive approaches and modeling/ planning approaches;
- Real-time recognition, planning, and reasoning;
- Reasoning and action in a dynamic environment;
- Sensor fusion;
- Multi-agent systems in general;
- Behavior learning for complex tasks;
- Strategy acquisition;
- Cognitive modeling in general.

More strategic issues are dealt with in the simulation league and in the small-size real robot league while acquiring more primitive behaviors of each player is the main concern of the middle-size real robot league. Since

the simulation league is well suited for testing the various multi agent strategies without bothering about the hardware and the electrical and mechanical aspects, a number of multi agent learning methods are applied to it. Next section gives a survey of the multi agent learning methods that are applied for developing the player strategies for robotic soccer simulation.

3. Learning Methods

Learning methods play an important role in robotic soccer. A team's success in robotic soccer will depend on how efficient it can react to the uncertain environment, which in turn depends on the learning ability of the agent. In the pre RoboCup which was held in 1996, the participated teams were having fixed hand coded strategies. But in the following years the researchers found more and more efficient strategies for their teams by incorporating learning abilities to the soccer-playing agents. Some of the important methods among them are discussed below.

3.1. Reinforcement Learning

Reinforcement learning is learning what to do or how to map situations to actions, so as to maximize a numerical reward signal.[9] The learner is not told which actions to take, but instead must discover which actions yield the most reward by trying them. In the most interesting and challenging cases, actions may affect not only the immediate reward but also the next situation and, through that, all subsequent rewards. These two characteristics--trial-and-error search and delayed reward--are the two most important distinguishing features of reinforcement learning.

One of the major challenges in managing a robot-soccer team is the design of methods to assign roles to the agents dynamically. Roles have to be assigned to the agents according to some metrics like the distance between the agent and the ball. A static assignment is not usually a good practice, but it can be used effectively for the goal-keeper agent. For other agents, a dynamic scheme is needed. One such approach is to choose the agent closest to the ball as the attacker, and the agent that is closest to the own goal among the unassigned agents as the defender. Other agents can help the attacker by holding strategic positions behind the attacker.

As another efficient strategy for the role assignment problem, the Reinforcement Learning (RL) is used to learn the role assignment process. [10] RL is a learning method, which can be used when the agent is only informed about the consequences of a sequence of its actions. The reinforcement measures chosen for the soccer problem were the goals scored by the team or the opponent team. The results showed that reinforcement learning is a good solution for role assignment problem in robot soccer. The solution can also be used in other highly dynamic environments where it is possible to introduce some reinforcement measures for the team.

3.2. Inductive Learning

Inductive learning is a machine-learning framework, which is based on generalization of examples. The concept of Inductive Logic Programming (ILP) has been used for soccer agents' inductive learning.

A framework for inductive learning soccer agents (ILSAs) had been proposed [10] in which the agents acquire knowledge from their own past behavior and behave based on the acquired knowledge. The inductive learning soccer agents decides each action taken in the game to be good or bad according to the examples which are classified as positive or negative. Also the agents themselves classify their past states during the learning process. In the framework the agent is given an action strategy, ie, a state checker and an action-command translator. An inductive learning soccer agent acquires a rule from examples whose positive examples consist of states in which the agent failed an action, and uses the acquired rule as the state checker to avoid taking actions in states similar to the positive examples.

Based on this work, another agent architecture that adapts its own behavior by avoiding actions, which are predicted to be failure, is proposed in [11]. The inductive learning agent used first-order formalism and inductive logic programming (ILP) to acquire rules to predict failures. First, the ILA collects examples of actions and classifies them. Then the prediction rules are formed using ILP and uses them for their behavior. This was implemented in soccer using parts of the RoboCup-1999 competition champion CMUnited-99 [12] and an ILP system Progol [13]. It was shown that agents could acquire prediction rules and could adapt their behavior using the rules. It was found that the agents used actions of CMUnited-99 more effectively after they acquired prediction rules.

Another research has been reported, which uses ILP systems for verifying and validating multi-agents for RoboCup [14]. This concentrates on verification and validation of knowledge based system, not but prediction or discovery of new knowledge. Consequently, agents cannot adapt their own behavior using rules or knowledge acquired by ILP.

3.3 Memory Based Supervised Learning

A memory-based supervised learning strategy was introduced, which enables an agent to choose to pass or shoot in the presence of a defender. [15] Learning how to adjust to an opponent's position can be critical to the success of having intelligent agents collaborating towards the achievement of specific tasks in unfriendly environments. Based on the position of an opponent indicated by a continuous-valued state attribute the agent learns to choose an action. A memory-based supervised learning strategy, which enables an agent to choose to pass or shoot in the presence of a defender, was attempted.

In the memory model, training examples affect neighboring generalized learned instances with different

weights. Each soccer agent stores its experiences in an adaptive memory and is able to retrieve them in order to decide upon an action. It has been seen that using an appropriate memory size, the adaptive memory made it possible for the agent to learn both time-varying and non-deterministic concepts. Also short-term performance was shown to be better when acting with a memory.

3.4. Neural Networks

The Artificial Neural Network (ANN) is an information-processing paradigm that is inspired by the way biological nervous systems, such as the brain, process information. The network is composed of a large number of highly interconnected processing elements or neurons working in parallel to solve a specific problem. Neural networks learn by example. They cannot be programmed to perform a specific task. An ANN is configured for a specific application through a learning process.

Neural networks had been successfully used for learning low-level behaviors of soccer agents. [16] This learned behavior, namely shooting a moving ball, equips the clients with the skill necessary to learn higher-level collaborative and adversarial behaviors. The learned behavior enabled the agent to redirect a moving ball with varying speeds and trajectories into specific parts of the goal. By carefully choosing the input representation to the neural networks so that they would generalize as much as possible, the agent was able to use the learned behavior in all quadrants of the field even though it was trained in a single quadrant. In another work, neural networks were used to learn turn angles based on balls distance and angle as a part of a hierarchical layered learning approach.

Neuro-Evolution

A neuro-evolutionary algorithm, which was successfully used in simulated ice hockey, was employed to evolve a player who can execute a dribble the ball to the goal and score behavior in the environment of robot soccer. Both goal-only and composite fitness functions were tried. The evolved players developed rudimentary skills; however it appeared that considerably more computation time is required to get competent players. The fitness of the individuals increases with generations. Also, the goal-only fitness was found more likely to lead to success than the composite fitness function

Using this approach, some of the evolved players exhibited rudimentary dribble and score skills, but the overall results were not very good considering the lengths of the runs. More complex networks with more input variables may lead to the evolution of better players more quickly.

3.5. Layered Learning

Layered Learning [17] allows the programmer to provide a hierarchical, bottom up decomposition of the problem to be learnt. Each layer is a separate skill which is learnt and then used as a component in the learning of a future layer. Layered learning is a machine independent and

problem independent learning scheme and as such there is some variation in what is considered to constitute a layer.

Applied to soccer, three distinct layers at differing abstraction levels within a multi-agent soccer domain is defined. [18] The first is the individual skill of intercepting the ball, learnt by training a neural network. The second is ball passing skills and evaluation, learnt by a decision tree, which requires the ability of ball interception. The third layer was pass selection, learnt by reinforcement learning, which utilizes both the previous skills. Each of these layers are encapsulated and do not change during further learning. The technique of layered learning has been applied to the evolution of goal scoring behavior in soccer players. [17] It was found that the solutions evolved with layers have a higher accuracy but do not make as many goal attempts as other methods like standard genetic programming technique.

3.6. Hybrid Approaches

There have been approaches, which combines various multi agent methods for developing agent strategies for soccer agents. One of the significant works, which combines several multi agent strategies in order to arrive at efficient soccer team, was the CMUnited teams which participated in the RoboCup simulation league from the first RoboCup. The CMUnited 97 team used layered learning approach with locker room agreement as their strategy. [19] Improving upon that, the CMUnited-98 simulator team used the following multi-agent techniques to achieve adaptive coordination among team members.

Hierarchical machine learning (Layered learning): Three learned layers were linked together for layered learning. [20] Neural networks were used by individual players to learn how to intercept a moving ball. With the receivers and opponents using this first learned behavior to try to receive or intercept passes, a decision tree was used to learn the likelihood that a given pass would succeed. This learned decision tree was used to abstract a very high-dimensional state-space into one manageable for the multi-agent reinforcement learning technique TPOT-RL [21]

Flexible, adaptive formations (Locker-room agreement): Locker-room agreement [22] includes a flexible team structure that allows homogeneous agents to switch roles (positions such as defender or attacker) within a single formation

Single-channel, low-bandwidth communication: Use of single-channel, low-bandwidth communication, [23] ensures that all agents must broadcast their messages on a single channel so that nearby agents on both teams can hear; there is a limited range of communication; and there is a limited hearing capacity so that message transmission is unreliable.

Predictive, locally optimal skills (PLOS): Predictive, Locally Optimal Skills [24] was another significant improvement of the CMUnited-98 over the CMUnited-97 simulator teams. Locally optimal both in time and in space, PLOS

was used to create sophisticated low-level behaviors, including dribbling the ball while keeping it away from opponents, fast ball interception, flexible kicking that trades off between power and speed of release based on opponent positions and desired eventual ball speed, a goaltender that decides when to hold its position and when to advance towards the ball based on opponent positions.

Strategic positioning using attraction and repulsion (SPAR): SPAR [25] determines the optimal positioning as the solution to a linear-programming based optimization problem with a multiple-objective function subject to several constraints. The agent's positioning is based upon teammate and adversary locations, as well as the ball's and the attacking goal's location.

Team-Partitioned, Opaque Transition Reinforcement Learning (TPOT-RL): TPOT-RL [10] allows a team of agents to learn to cooperate towards the achievement of a specific goal.

The team, which used this strategy, was the RoboCup 98 simulation league champion. Improvements upon these strategies were presented by CMUnited-99, [13] which was the world champion of RoboCup 99. The low-level skills were improved and updated to deal with server changes, the use of opponent and teammate models was introduced, and some coordination procedures were improved, and a development paradigm called layered disclosure [26] was introduced, by which autonomous agents include in their architecture the foundations necessary to allow a person to probe into the specific reasons for an agent's action.

It can be seen that hybrid approaches were effective in the soccer domain. Hybrid-methods may be able to capture the complexity of the soccer domain better because the problem itself demands solution to a combination of different multi-agent issues. Also the evolutionary approach genetic programming is also found to evolve good and effective strategies through automatically. This approach has the advantage of giving multiple strategies, which are equally good. A detailed overview of the genetic programming technique is presented in the next section.

4. Genetic Programming

Genetic Programming (GP) is a subset of Evolutionary Computing that follows the Darwinian theory of natural selection, "survival of fittest"[4]. It is considered to be a domain independent problem solving approach in which computer programs are automatically evolved to solve problems. This is achieved by genetically breeding a population of computer programs using the principles of natural selection and biologically inspired operations like crossover and mutation [4][5].

The first step in GP is initial population generation [4][5]. The success of GP depends on the trees generated in the initial population [27]. The components of GP are given below.

The terminal set: The terminal set consists of the variables and constants of the programs.

The function set: A set of domain specific functions used in conjunction with the terminal set to construct potential solutions to a given problem. For symbolic regression this could consist of a set of basic mathematical functions, while Boolean and conditional operators could be included for classification problems.

The fitness function: Fitness is a numeric value assigned to each member of a population to provide a measure of the appropriateness of a solution to the problem in question.

The termination criterion: This is generally a predefined number of generations or an error tolerance on the fitness.

Selection Methods: There are three selection methods namely Fitness Proportionate selection, Tournament selection and Roulett wheel selection methods.

The algorithm control parameters: The genetic programming paradigm is controlled by 19 control parameters, including two major numerical parameters, 11 minor numerical parameters, and six qualitative variables that select among various alternative ways of executing a run. The Table 1 lists the 19 control parameters and their default values [28].

Parameter Name	Representation	Default Value
Two major numerical parameters		
Population size = 500.	M	500
Maximum number of generations to be run	G	51
Eleven minor numerical parameters		
Probability of crossover	P_c	90%
Probability of reproduction	P_r	10%
Probability of choosing internal points for crossover	P_{ip}	90%
Maximum size for S-expressions created during the run	D_c	17
Maximum size for initial random S-expressions	D_i	6
Probability of mutation	P_m	0.0%
Probability of permutation	P_p	0.0%
Frequency of editing	F_{ed}	0
Probability of encapsulation	P_{en}	0.0%
Condition for decimation = NIL.		
Decimation target percentage	P_d	0.0%
Six qualitative variables		
Generative method for initial random population is ramped half-and-half.		
Basic selection method is fitness proportionate.		
Spousal selection method is fitness proportionate.		
Adjusted fitness is used.		
Over-selection is not used for populations of 500 and below and is used for Populations of 1,000 and above		
Elitist strategy is not used.		

Table 1. Default values of the 19 control parameters for genetic programming. [34]

Tree Generation Algorithms for GP

At the beginning of the evolution process, initial individuals must be generated at random. Genetic programming creates these individuals' trees by applying a tree generation algorithm to each tree's function set. Tree generation algorithms work by selecting and copying nodes from the templates in the function set, then hanging the copied nodes together to form the tree. The three traditional tree-generation algorithms are GROW, FULL, and RAMPED HALF-AND-HALF.

GROW begins with a set of functions F to place as nodes in the GROW randomly selects a root from the full set of functions (both terminals and non terminals), then fills the root's arguments with random functions, then *their* arguments with random functions, and so on. GROW's companion algorithm (FULL) is used to generate full trees. RAMPED HALF-AND-HALF randomly does either FULL or GROW.

GP Breeding

In standard genetic programming, the breeding procedure picks from among the three breeding strategies reproduction, mutation, and crossover then applies one strategy to selected individuals and returns the results. Reproduction and mutation will return one individual, while crossover returns two. Traditionally, genetic programming chooses reproduction 10% of the time and crossover 90% of the time. The mutation operator, which is a secondary operator, is not used usually. The other secondary operators can also be used along with these three operators.

To perform crossover and mutation, genetic programming must implement 'recombine' and 'modify'. The standard recombination method in genetic programming, *sub-tree crossover*, starts with two individuals selected and copied from the old population. Classically, a random point is selected within one tree of each copied individual with non terminals selected 90% of the time and terminals selected 10% of the time.

Then crossover swaps the sub-trees rooted at these two nodes and returns the modified copies. If the crossover process results in a tree greater than a maximum depth bound (except where noted, the bound is 17), then the modified child is discarded and its parent (the tree into which a sub-tree was inserted to form the child) is simply copied through reproduction. Crossover may only occur if two trees share the same function set. This process is illustrated in Fig. 1. and Fig. 2.

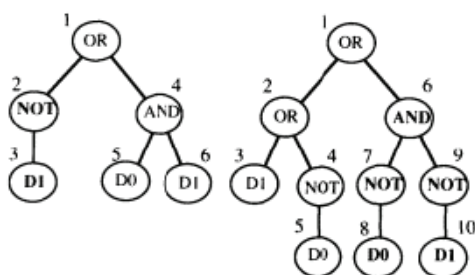


Fig. 1. Two parental computer programs.

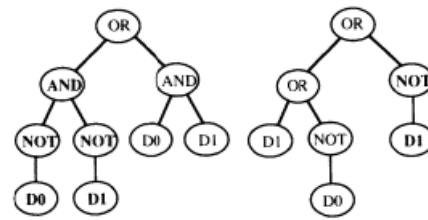


Fig. 2. Two offspring produced by crossover, resulting from selection of point 2 of the first parent and point 6 of the second parent as crossover points.

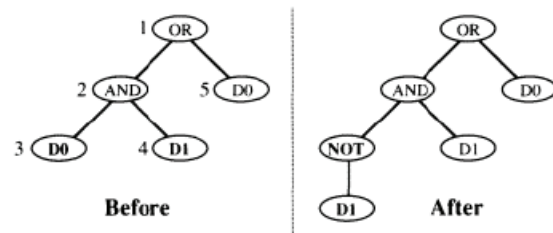


Fig. 3. A computer program before and after the mutation operation is performed at point 3.

The most common 'modify' step uses *mutation operator*, shown in Figure 3. Point mutation starts with a single individual selected and copied from the old population. One node is selected from among of the copied individual's trees, using the same node-selection technique as described for sub-tree crossover. The sub-tree rooted at this node is removed and replaced with a randomly-generated sub-tree, using the GROW algorithm and the appropriate function set for the tree. If mutation results in a tree greater than the maximum depth, then the copy is discarded and its parent is reproduced instead.

5. GP Specification for Robotic Soccer Player Strategies (without Ant-Intelligence)

Individual Representation: Tree representation

Selection Method: FitnessProportionate Selection

Operators: Crossover, Reproduction, Mutation

Termination Criteria: Number of generations.

Terminal Set: The following table 2 depicts the terminal set for evolving the robotic soccer player strategies.

Terminal	Description
Ball	Vector to the ball
Mate1	Vector to Teammate 1
Mate2	Vector to Teammate 2
Mate3	Vector to Teammate 3
Mate4	Vector to Teammate 4
Opp1	Vector to the Opponent1
Opp2	Vector to the Opponent1
Opp3	Vector to the Opponent1
Opp4	Vector to the Opponent1
Goal	Vector to the Goal
OppGoal	Vector to the Opponent Goal
Rand	A random Vector

Table 2. The terminals used for evolving the robotic soccer player strategies

Function Set: The following table 3 depicts the function set for evolving the robotic soccer player strategies.

Function	Arity	Description
Rotate (v)	1	Rotate the vector v 90 degrees, Return the result
Reverse (v)	1	Reverse the direction of Vector v, Return the result
Mul2 (v)	1	Multiply the vector v by 2, Return the result
Div2 (v)	1	Divide the vector v by 2, Return the result
Add (v1,v2)	2	Add vector v1 and vector v2, Return the result
Sub (v1,v2)	2	Subtract vector v2 from vector v1, Return the result.
Vecomp(v1,v2,v3,v4)	4	Compare the magnitude of vectors v1 and v2, if $ v1 < v2 $ return v3 else return v4
NearestToBall(v1,v2)	2	Checks whether the player of the team who is in the position nearest to the ball is the current robot. If so return v1 else return v2.
MateNearBall(v1,v2)	2	Checks whether the player who is nearest to the ball is a teammate. If so return v1 else return v2.
OppNearBall (v1,v2)	2	Checks whether the player who is nearest to the ball is an opponent. If so return v1 else return v2.
BallNearOppGoal (v1,v2)	2	Checks whether the ball is near the opponent goal post. If so return v1 else return v2.
IfMateNear(v1,v2)	2	Checks whether a teammate is near the current robot. If so return v1 else return v2.
IfOppNear(v1,v2)	2	Checks whether an opponent is near the current robot. If so return v1 else return v2.
BallNearGoal(v1,v2)	2	Checks whether the ball is near the goal post. If so return v1 else return v2.
OppNearGoal (v1,v2)	2	Checks whether an opponent is near the goal post. If so return v1 else return v2.

Table 3. The functions used for evolving the robotic soccer player strategies.

6. Ant Behaviors in Robotic Soccer

As in many other multi agent systems, the soccer robots might benefit from some of the insights from social ants to model their strategies of play. For instance the tendency of nest protecting exhibited by ants can be a strategy for the goalkeeper robot in robotic soccer. Also if an opponent player tries to go near the goal post, the players should try to prevent the attempt.

Likewise the division of labor among ants can be imitated by the robots for dividing the various sub tasks among themselves in a coordinated way. The ants follow the pheromone paths to find shortest path to food sources. This pheromone laying behavior may be applied to the robotic soccer game in another sense so as to help the players to disperse throughout the field or to communicate some other information.

A swarm of small ants will be able to defeat a larger ant and make it their food. But a large ant can catch a small ant as food. In the game of soccer, to deal with a quick and strong opponent player, two or more players will be needed. Analogous to ants rearing insects without eating, in-order to get sweet sap from them, a weak player among the opponents may be given access to the ball in the hope that he will play badly always.

These are some of the examples, which show that the ant behaviors may be helpful in soccer domain also. We attempted to use one of these concepts while evolving the player strategies for robotic soccer.

6.1. Evolving Soccer Players strategies with Real Ant Behaviors

The technique behind the foraging behavior of real ants is the use of a chemical called pheromone, which was laid by the ants while coming from a food source. Ants from the same nest recognize this and follow the trail to arrive at the food source. This also enables them to find the shortest route to the nest from the food source.

This concept of applying the pheromone to convey some information to the nest mates is adapted to the soccer-playing agents. The agent when perceives the ball which is near it as well as near the opponent goal may put some pheromone in its position to enable other teammates to know that there is a good chance of putting goal. If the other teammates also come near the pheromone, one of them may get a chance to put the ball to the opponent goal. This may help the forwarder agent to score more goals against the opponent.

The persistence of the pheromone is for only one time-step, within which all the robots execute one command each. By using a function to check the presence of pheromone, the strategy to be followed when the pheromone is present in the field may be evolved through genetic programming. The revised function set and terminal set are given in the table 4 and table 5.

Function	Arity	Description
Rotate (v)	1	Rotate the vector v 90 degrees, Return the result
Reverse (v)	1	Reverse the direction of Vector v, Return the result
Mul2 (v)	1	Multiply the vector v by 2, Return the result
Div2 (v)	1	Divide the vector v by 2, Return the result
Add (v1,v2)	2	Add vector v1 and vector v2, Return the result
Sub (v1,v2)	2	Subtract vector v2 from vector v1, Return the result.
Vcomp (v1,v2,v3,v4)	4	Compare the magnitude of vectors v1 and v2, if $ v1 < v2 $ return v3 else return v4
NearestToBall (v1,v2)	2	Checks whether the player of the team who is in the position nearest to the ball is the current robot. If so return v1 else return v2.
MateNearBall (v1,v2)	2	Checks whether the player who is nearest to the ball is a teammate. If so return v1 else return v2.
OppNearBall (v1,v2)	2	Checks whether the player who is nearest to the ball is an opponent. If so return v1 else return v2.
BallNearOppGoal (v1,v2)	2	Checks whether the ball is near the opponent goal post. If so return v1 else return v2.
IfMateNear(v1,v2)	2	Checks whether a teammate is near the current robot. If so return v1 else return v2.
IfOppNear(v1,v2)	2	Checks whether an opponent is near the current robot. If so return v1 else return v2.
BallNearGoal(v1,v2)	2	Checks whether the ball is near the goal post. If so return v1 else return v2.
OppNearGoal (v1,v2)	2	Checks whether an opponent is near the goal post. If so return v1 else return v2.
IfPhNew()	2	The function checks the presence of pheromone and executes the child trees according to whether pheromone is present in the field or not.

Table 4. Revised Function set of Genetic Programming to adopt the real ant behavior in player strategies.

Terminal	Description
Ball	Vector to the ball
Mate1	Vector to Teammate 1
Mate2	Vector to Teammate 2
Mate3	Vector to Teammate 3
Mate4	Vector to Teammate 4
Opp1	Vector to the Opponent1
Opp2	Vector to the Opponent1
Opp3	Vector to the Opponent1
Opp4	Vector to the Opponent1
Goal	Vector to the Goal
OppGoal	Vector to the Opponent Goal
Rand	A random Vector
GoToPh	The terminal directs the agent to move to the location of pheromone

Table 5. Revised Terminal set of Genetic Programming to adopt the real ant behavior in player strategies.

Other than this method, this concept of the real ants behavior may be used in various other senses also. An ant usually identifies the pheromone laid by its nest-mates. Pheromone lay by other types of ants or ants from some other nest will be ignored. Like wise, while adopting this concept to soccer, different kinds of pheromone can be used to flag different situations or the forwarder and defender agents may use different types of pheromone, the meaning of which may be different to them.

In order to test our work we used a simulation tool called Evolutionary Computation in Java tool. Next section gives details about the simulation tool and the modifications made to the tool.

7. ECJ Simulator

ECJ is a Java-based evolutionary computation research system. [31] It comes with a comprehensive suite of libraries for doing Java-based genetic programming. ECJ 4 was the first most full-featured genetic programming distribution, which was made publicly available. [32] It provides

- Multithreaded and multi process evolution,
- Both steady-state and generational evolutionary computation,
- Multiple populations, inter-population exchanges, and co-evolutionary hooks,
- Set-based strongly-typed genetic programming,
- Automatically defined functions and macros,
- Ephemeral random constants,
- Check-pointing, logging, and reading populations from files,
- Hooks for evolution with multi-objective fitness,
- A highly flexible breeding architecture with six selection methods, twelve breeding operators, and four tree-generation algorithms,
- Customizable individuals with multiple trees,
- Pre-done problem domains (Artificial Ant, Multiplexer, Symbolic Regression, Parity, Lawnmower, Edge Encoding),

- Abstractions for many evolutionary computation approaches, apart from genetic programming.

ECJ was designed to have the features like extensibility, portability, and also to execute reasonably fast. It has also got rich set of features for doing evolutionary computation.

In order to evolve the code for robotic soccer players, we coded an application package (Robotic Soccer) in ECJ Simulator. The formulated application package comprises of java and class files of the above said function sets and terminal sets referred in Table 4 and Table 5.

The fitness of the individuals (Robotic Soccer Player Strategies) is evaluated against the teams in soccerbots of Teambots. Next section gives information regarding Teambots package.

8. TeamBots

TeamBots is a Java-based collection of application programs and Java packages for multi agent mobile robotics research. [33] The TeamBots distribution is a full source-code release, in which the simulation environment is written in Java. One of the most important features of the TeamBots environment is that it supports prototyping in simulation of the same control systems that can be run on mobile robots.

TeamBots supports prototyping, simulation and execution of multi robot control systems. Robot control systems developed in TeamBots can run in simulation using the TBSim simulation application, and on mobile robots using the TBHard robot execution environment.

The TeamBots simulation environment is extremely flexible. It supports multiple heterogeneous robot hardware running heterogeneous control systems. Complex or simple experimental environments can be designed with walls, roads, opponent robots and circular obstacles. All of these objects may be included in a simulation by editing an easily understandable human-readable description file.

Four application programs are included in the TeamBots distribution to aid the deployment of a multi agent system:

- TBSim: the TeamBots simulator.
- TBHard: the real robot execution environment for control systems developed using TeamBots.
- RoboComm: RoboComm simplifies asynchronous robot to robot communication. Any Java object can be sent from one robot to another. The RoboComm server forwards messages between agents.
- JCye: These programs demonstrate how to use our Java package to control a Cye robot.
- The TeamBots distribution includes several example systems that can run in simulation. The following are the domains which are included in the teambots package.
- Cye: this is a simulation of Probotic's Cye robot.
- SoccerBots: this is a simulation of RoboCup F-180 league soccer.

- Book: includes example exercises being developed for the book *Designing Robot Behavior* by Tucker Balch. These primarily illustrates principles of behavior-based control.
 - CTF: an implementation of Capture-The-Flag, a multi agent adversarial game.
 - Forage: a basic implementation of multi agent foraging.
 - ForageAAAI97: the actual source code used to win the AAAI-97 mobile robot competition.
 - Roads: two example uses of simulation environments including roads. One example includes two different types of roads and a bridge. The other is an entire city map. Any arbitrarily complex environment including different traversability constraints can be built.
 - Walls: an example simulation environment with walls.
- Out of these domains, the one which supports soccer domain is the soccer bots. Brief overview of SoccerBots is given below.

8.1. SoccerBots

SoccerBots is a simulation of RoboCup soccer. SoccerBots simulates the dynamics and dimensions of a regulation RoboCup small size robot league game. Two teams of five robots compete on a ping-pong table by pushing and kicking an orange golf ball into the opponent's goal.

SoccerBots runs in the TBSim application using a configuration file that tailors it for soccer games. The same control systems that run in simulation can also operate real robot hardware using the TBHard application.

SoccerBots has 20 soccer teams with different strategies in the teams package. The following table depicts the list of teams in teambots.

The soccer games conducted among the set of teams has identified AIKHomoG, DoogHeteroG, DoogHomoG, MattiHeteroG as the top 4 leading teams. Among these 4 teams AIKHomoG is the never outbeaten by other teams and hence its the best team in soccerbots.

Next section presents experimental results obtained through genetic programming.

9. Experimental Results

The genetic programming system for evolving the player strategies with ant intelligence was designed and implemented. The evolutionary computation system ECJ simulator was used for this purpose. The genetic programming parameters for this robotic soccer application are fine tuned by conducting a series of test runs with various parameters.

The following experiments were conducted for finding the suitable genetic programming parameter setting for evolving the better strategies for the agent. Here we tested the evolved team with four top leading teams in soccerbots of teambots by varying the genetic programming parameters. The corresponding results obtained are shown:

Experiment 1:

Genetic Programming Parameters	Values
Maximum Generations G	10
Population Size M	10
Crossover Probability P_c	90%
Reproduction Probability P_r	10%
Selection Method	Fit Proportionate Selection
Tree Generation	Ramped Half and Half

Teams	Hits
AIKHomoG	3
DoogHeteroG	2
DoogHomoG	3
MattiHeteroG	2

Experiment 2:

Genetic Programming Parameters	Values
Maximum Generations G	15
Population Size M	10
Crossover Probability P_c	90%
Reproduction Probability P_r	10%
Selection Method	Fit Proportionate Selection
Tree Generation	Ramped Half and Half

Teams	Hits
AIKHomoG	3
DoogHeteroG	3
DoogHomoG	4
MattiHeteroG	1

Experiment 3:

Genetic Programming Parameters	Values
Maximum Generations G	20
Population Size M	10
Crossover Probability P_c	90%
Reproduction Probability P_r	10%
Selection Method	Fit Proportionate Selection
Tree Generation	Ramped Half and Half

Teams	Hits
AIKHomoG	3
DoogHeteroG	4
DoogHomoG	4
MattiHeteroG	2

Experiment 4:

Genetic Programming Parameters	Values
Maximum Generations G	15
Population Size M	10
Crossover Probability P_c	90%
Reproduction Probability P_r	10%
Selection Method	Fit Proportionate Selection
Tree Generation	PTC 1
Depth (min-max)	2-10
Size	10

New Teams Vs	Hits
AIKHomoG	2
DoogHeteroG	2
DoogHomoG	3
MattiHeteroG	0

Experiment 5:

Genetic Programming Parameters	Values
Maximum Generations G	15
Population Size M	10
Crossover Probability P_c	90%
Reproduction Probability P_r	10%
Selection Method	Fit Proportionate Selection
Tree Generation	Grow Method
Depth (min-max)	2-10

New Teams Vs	Hits
AIKHomoG	1
DoogHeteroG	2
DoogHomoG	2
MattiHeteroG	1

Experiment 6:

Genetic Programming Parameters	Values
Maximum Generations G	15
Population Size M	10
Crossover Probability P_c	90%
Reproduction Probability P_r	10%
Selection Method	Fit Proportionate Selection
Tree Generation	Full Method
Depth (min-max)	2-10
Size	10

New Teams Vs	Hits
AIKHomoG	2
DoogHeteroG	2
DoogHomoG	2
MattiHeteroG	1

Experiment 7:

Genetic Programming Parameters	Values
Maximum Generations G	15
Population Size M	10
Crossover Probability P_c	90%
Reproduction Probability P_r	10%
Selection Method	Tournament Selection
Tournament size	7
Tree Generation	Ramped Half and Half
Depth(min-max)	2-10

New Teams Vs	Hits
AIKHomoG	2
DoogHeteroG	3
DoogHomoG	3
MattiHeteroG	1

Experiment 8 :

Genetic Programming Parameters	Values
Maximum Generations G	15
Population Size M	10
Crossover Probability P_c	90%
Reproduction Probability P_r	10%
Selection Method	Fitness Proportionate Selection
Tree Generation	Ramped Half and Half
Depth(min-max)	2-10

New Teams Vs	Hits
AIKHomoG	3
DoogHeteroG	3
DoogHomoG	3
MattiHeteroG	1

Experiment 9 :

Genetic Programming Parameters	Values
Maximum Generations G	15
Population Size M	10
Crossover Probability P_c	80%
Reproduction Probability P_r	20%
Selection Method	Fitness Proportionate Selection
Tree Generation	Ramped Half and Half
Depth(min-max)	2-10

New Teams Vs	Hits
AIKHomoG	3
DoogHeteroG	3
DoogHomoG	3
MattiHeteroG	1

Experiment 10 :

Genetic Programming Parameters	Values
Maximum Generations G	15
Population Size M	10
Crossover Probability P_c	80%
Reproduction Probability P_r	10%
Mutation Probability P_m	10%
Selection Method	Fitness Proportionate Selection
Tree Generation	Ramped Half and Half
Depth(min-max)	2-10

New Teams Vs	Hits
AIKHomoG	2
DoogHeteroG	3
DoogHomoG	2
MattiHeteroG	1

Experiment 11 :

Genetic Programming Parameters	Values
Maximum Generations G	15
Population Size M	10
Crossover Probability P_c	70%
Reproduction Probability P_r	10%
Mutation Probability P_m	20%
Selection Method	Fitness Proportionate Selection
Tree Generation	Ramped Half and Half
Depth(min-max)	2-10

New Teams Vs	Hits
AIKHomoG	1
DoogHeteroG	2
DoogHomoG	2
MattiHeteroG	1

Through the experimental results it can be seen that when the concept of real ant food foraging behavior were applied in evolving the player strategies, the results obtained are quite promising. At the outset the evolved player strategies are capable of winning against the best team in soccerbots (AIKHomoG).

10. Conclusion and Future Work

This paper made an attempt to evolve the robotic soccer player strategies with ant intelligence through genetic

programming. It was seen that Genetic Programming is giving promising results. Hence application of natural ants' behavior helps to improve the performance of the players. This work is done for the AICTE- Research promotion scheme project titled «BIOINSPIRED BEHAVIORS IN ROBOTIC SOCCER». As part of our future work we are trying to adopt few more social insect behaviors in evolving better player strategies.

11. References

- Kiyohiko Hattori, Yoshimasa Narita, Yoshiki Kashimori, and Takeshi Kambara, «Self-Organized critical behavior of fish school and emergency of group intelligence», IEEE, 1999.
- E. Crist, «Can an Insect Speak? The case of the Honeybee Dance Language». Social Studies of Science. SSS and Sage Publications. 34(1), pp. 7-43.
- John S. Montgomery, Termite Mound Simulator, lecturer : Dr L. Jankovic, May 6, 2004.
- The official website of Robotic soccer <http://www.robocup.org/>
- Mackworth, A., "On Seeing Robots", *Vision Interface* 1992
- Parpinelli, R.S., Lopes, H.S. and Freitas, A.A. An ant colony based system for data mining: applications to medical data. *Proc. 2001 Genetic and Evolutionary Computation Conf. (GECCO-2001)*, pp. 791-798. Morgan Kaufmann, 2001.
- David Andre, Emiel Corten, Klaus Dorer, Pascal Gugenberger, Marius Joldos, Johan Kummeneje, Paul Arthur Navratil, Itsuki Noda, Patrick Riley, Peter Stone, Romoichi Takahashi, and Travlex Yeap. *Soccer server manual*, version 4.0. Technical Report RoboCup-1998-001, RoboCup, 1998.
- Minoru Asada, Hiroaki Kitano, Itsuki Noda, Manuela Veloso "RoboCup: Today and tomorrow—What we have learned" *Artificial Intelligence* 110 (1999) 193–214
- Tatlydede, Kemal Kaplan, Hatice Kose, and H. Levent Akyn "Reinforcement Learning for Multi-Agent Coordination in Robot Soccer Domain", *Utku Fifth European Workshop on Adaptive Agents and Multi-Agent Systems*, Paris. March 2005. pp. 21-22
- Matsui, Kashiwabara, Inuzuka, Seki, and Itoh. "Soccer agents learning from past behavior with inductive logic programming." In *Proceedings of the IMC Workshop on Knowledge Mining in the Real-world*, pages 9:1–12, 1999
- Matsui, Inuzuka, Hirohisa Seki: "A Proposal for Inductive Learning Agent Using First-Order Logic" *ILP Work-in-progress reports* 2000
- Muggleton, S "Inverse Entailment and Progol," *New Generation Computing*, 13:245-286, (1995).
- Peter Stone, Patrick Riley, and Manuela Veloso. "The CMUnited-99 Champion Simulator Team." In M. Veloso, E. Pagello, and H. Kitano, editors, *RoboCup-99: Robot Soccer World Cup III*, Springer, Berlin, 2000, pp. 35–48
- Jacobs, N., K. Driessens, and L. D. Raedt. "Using ilp-systems for verification and validation of multi-agent systems." In D. Page, editor, *Proceedings of the 8th International Conference on Inductive Logic Programming (ILP-98)*. Springer Verlag, 1998.
- Peter Stone and Manuela Veloso. "Beating a Defender in Robotic Soccer: Memory-Based Learning of a Continuous Function" In *Advances in Neural Information Processing Systems* 8, pp. 896–902, MIT Press, Cambridge, MA, 1996.
- Peter Stone and Manuela Veloso "Towards Collaborative and Adversarial Learning: A Case Study in Robotic Soccer" In *International Journal of Human Computer Studies*, 48, 1998
- Uchibe E., Asada M., Noda S., Takahashi Y., Hosoda K., "Vision-Based Reinforcement Learning for RoboCup: Towards Real Robot Competition". *Proc. of IROS 96 Workshop on RoboCup*, 1996.
- Kitano H., "RoboCup: The Robot World Cup Initiative." In *proceedings of The First Int. Conf. on Autonomous Agent (Agents-97)*, Marina del Ray, The ACM Press, 1997.
- Peter Stone and Manuela Veloso. "The CMUnited-97 Simulator Team." In Hiroaki Kitano, editors, *RoboCup-97: Robot Soccer World Cup I*, Springer Verlag, Berlin, 1998, pp. 387–397
- Peter Stone and Manuela Veloso. "A Layered Approach to Learning Client Behaviors in the RoboCup Soccer Server." *Applied Artificial Intelligence*, 12:165–188, 1998
- Peter Stone and Manuela Veloso. "Team-Partitioned, Opaque-Transition Reinforcement Learning." In Minoru Asada and Hiroaki Kitano, editors, *RoboCup-98: Robot Soccer World Cup II*, Springer Verlag, Berlin, 1999. Also in *Proceedings of the Third International Conference on Autonomous Agents*, 1999.
- David Andre and Astro Teller, "Evolving team Darwin United", *RoboCup II: Proceedings of the second annual conference*, Springer-Verlag, 1999 RoboCup 1998: 346-351
- Peter Stone and Manuela Veloso. "Task Decomposition, Dynamic Role Assignment, and Low-Bandwidth Communication for Real-Time Strategic Teamwork." *Artificial Intelligence*, 110(2):241–273, June 1999
- Peter Stone, Manuela Veloso, and Patrick Riley. "The CMUnited-98 Champion Simulator Team." In M. Asada and H. Kitano, editors, *RoboCup-98: Robot Soccer World Cup II*, Springer, 1999, pp. 61–76
- Manuela Veloso, Peter Stone, and Michael Bowling. "Anticipation as a Key for Collaboration in a Team of Agents: A Case Study in Robotic Soccer." In *Proceedings of SPIE Sensor Fusion and Decentralized Control in Robotic Systems II*, pp. 134–143, SPIE, Boston, September 1999
- Patrick Riley, Peter Stone, and Manuela Veloso. "Layered Disclosure: Revealing Agents' Internals." In C. Castelfranchi and Y. Lespérance, editors, *Intelligent Agents VII. Agent Theories, Architectures, and Languages*

- 7th. *International Workshop, ATAL-2000*, Boston, MA, USA, July 7--9, 2000, *Proceedings, Lecture Notes in Artificial Intelligence*, Springer-Verlag, Berlin, Berlin, 2001
- Balch T. R., "Integrating RL and Behaviour-based Control for Soccer." *Proc. IJCAI Workshop on RoboCup*, 1997.
- Koza, J. R. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. The MIT Press, Cambridge MA, 1992.
- Montana, D. J. 1995. "Strongly typed genetic programming". *Evolutionary Computation* 3(2):199-230.
- Koza, J. R. *Genetic Programming II: Automatic Discovery of Reusable Programs*. Cambridge Massachusetts: MIT Press. 1994.
- ECJ, www.cs.umd.edu/projects/plus/ec/ecj/.
- Luke, S. *Issues in Scaling Genetic Programming: Breeding Strategies, Tree Generation, and Code Bloat*, DPhil Thesis, Department of Computer Science University of Maryland, 2000
- Teambots page www.teambots.org