A Hybrid Genetic Programming Decision Making System for RoboCup Soccer Simulation

Amir Tavafi
Memorial University of Newfoundland
amir.tavafi@mun.ca

Wolfgang Banzhaf
Michigan State University
banzhafw@msu.edu

ABSTRACT

In this contribution we propose a hybrid genetic programming approach for evolving a decision making system in the domain of RoboCup Soccer (Simulation League). Genetic programming has been rarely used in this domain in the past, due to the difficulties and restrictions of the soccer simulation. The real-time requirements of robot soccer and the lengthy evaluation time even for simulated games provide a formidable obstacle to the application of evolutionary approaches. Our new method uses two evolutionary phases, each of which compensating for restrictions and limitations of the other. The first phase produces some evolved GP individuals applying an off-game evaluation system which can be trained on snapshots of game situations as they actually happened in earlier games, and corresponding decisions tagged as correct or wrong. The second phase uses the best individuals of the first phase as input to run another GP system to evolve players in a real game environment where the quality of decisions is evaluated through winning or losing during real-time runs of the simulator. We benchmark the new system against a baseline system used by most simulation league teams, as well as against winning systems of the 2016 tournament.

CCS CONCEPTS

•Computer systems organization → Evolutionary robotics;
•Software and its engineering → Genetic programming;

KEYWORDS

Decision Making, Genetic Programming, Multi-agent Systems, RoboCup, Soccer Simulation

ACM Reference format:
Amir Tavafi and Wolfgang Banzhaf. 2017. A Hybrid Genetic Programming Decision Making System for RoboCup Soccer Simulation. In *Proceedings of the Genetic and Evolutionary Computation Conference 2017, Berlin, Germany, July 15–19, 2017 (GECCO '17),* 8 pages.
DOI: 10.1145/3071178.3071194

1 INTRODUCTION

A multi-agent system is a system in which there are multiple interacting intelligent agents within an environment [6]. These systems can be used in different domains for solving problems where a single agent is unable to achieve the defined goal or reaching it might be time consuming and not efficient. The problem of decision making in multi-agent systems is a complex one. Each agent in a multi-agent system needs to be equipped with its own independent decision making system in order to be able to interact with other agents and also with the environment. The communication between agents is limited in these environments and the amount of information that can be sent or received from is normally limited per cycle, depending on the environment type.

Each agent has different functions and skills that it can execute in a cycle or a period of time. In order to execute the best action for reaching the goal of the system, there are two main problems. First, the functions and skills of an agent need to be optimized to reach the best possible outcome in principle. Second, when all functions and skills are available, a good decision making system is required so that the skills and functions are used at the right time so the goal can be reached in an efficient and fast way. The problem of generating a good decision making system when we have the functions and skills available is the problem we focus on in this article. All of the implementations reported in this contribution are based on the source code of the award winning *MarliK* team [20] which has developed a set of good skills. The decision making system is then subjected to a genetic programming approach for improving the outcome of an agent's actions.

Genetic Programming (GP) is a research method within the field of Evolutionary Computation (EC) [5] that generates programs and algorithms through simulated evolution automatically. In Genetic Programming, computer programs are evolved with a computer program represented as a tree (as used here) or another data structure like a sequence of instructions. The trees are composed of functions and terminals, depending on the problem definition. Crossover and mutation operators are used for evolving individuals over generations. Evaluation of trees in GP traverses the corresponding tree recursively in the evaluation method that is defined depending on the problem statement.

The particular method used here employs two phases of GP application, with different training sets and different dynamics. The first phase sets agents into game situations we call snapshots which stem from previous games with the simulator, where the potential behaviors an agent can choose are labeled by hand and trained with an offline system. The second phase uses these situations to interpolate in real-time game situations. Agents start out with what they have learned to work well in off-line situations, and are put into real games. Thus, the learning is human-like in the sense that behavior is trained for specific patterns of circumstances which are in some sense idealized typical situations with a slim chance to ever be encountered in a real game. Yet, similar situations could well be encountered and the system interpolates between those while running a real game. The evaluation in the second phase is

GECCO '17, July 15-19, 2017, Berlin, Germany

DOI: http://dx.doi.org/10.1145/3071178.3071194

based on the consequences (gaining an advantage in the game, or loosing ground). It is hoped that with an increase in the density of situations, the system becomes more and more accurate and able to win real games. This approach is similar in philosophy, yet different technically from the memory modelling approach we proposed two decades ago [16].

The rest of the paper is structured as follows: In Section 2 we discuss the soccer simulation domain and previous usage of EC in this domain, along with the challenges that were faced in that research. Section 3 details the hybrid GP method. Section 4 presents the performance analysis. Finally, conclusions and future work are found in Section 5.

2 THE SOCCER SIMULATION DOMAIN AND EVOLUTIONARY COMPUTATION

The Soccer 2D Simulation Server (RCSSServer) [15] is the software in the soccer 2D simulation league that provides a complex multi-agent system. It allows groups to develop their own soccer teams of 11 individual agents and play against another team. The soccer simulation domain is a very difficult, real-time, noisy, and highly dynamic environment.

Genetic programming has been applied to multi-agent coordination before. Andre in [1] evolved communication between agents with different skills. Qureshi in [18] evolved agent-based communication in a cooperative avoidance domain. Raik and Durnota in [19] used GP to evolve cooperative sporting strategies. Luke and Spector in [13] and Haynes et al. in [8] used GP to develop cooperation in predator-prey environments. Iba in [9] applied a similar approach to cooperative behavior in the TileWorld domain. These were some of the early related methods applied before Sean Luke applied Genetic Programming to a very difficult problem domain, RoboCup soccer 2D simulation.

The RoboCup soccer server is said to be not a good match for GP [11] because the soccer server domain is very complex and there are multiple options and controls with many special cases important for each decision or action. This makes it hard for GP to be integrated successfully. Another difficulty is the time factor. The soccer server runs in real-time and all players are connected separately via UDP sockets to the server. Each game takes ten minutes to play and there is an enforced 10ms delay between world model updates which results in the entire game to last about 10 minutes (equal to 6000 of 10ms cycles) [12].

In 1997, Sean Luke proposed using Genetic Programming for producing a team of competitive agents for the RoboCup97 official competition [11, 14]. The objective that Luke and his teammates set for their team was fairly modest. Their goal at first was to produce a team of agents able to play an entire game [11, 12]. They managed to produce agents that were able to decide how to disperse throughout the field, pass, kick to the goal, defend the goal, and coordinate with and defer to other teammates. At the time their team participated in RoboCup97, all other teams were hand-crafted, human coded algorithms and their team was the only one with intelligent players using an AI approach. They managed to win their first two games in the competition but lost others. In the end they won the RoboCup97 scientific challenge award for their research [11].

One of the reasons which makes it a serious problem to evolve a computer program to work successfully in this domain is that it requires a very large number of evaluations. In this case, each evaluation is equal to a run of a game in the simulator. In one of their previous results in a simpler domain, Luke and Spector found that GP would require about 100,000 evaluations in order to find a reasonable solution [13]. In a more complex domain like soccer it is likely that more evaluations would be necessary to find a reasonable solution. If we consider each evaluation in a soccer simulation server run to take 5 minutes, it would take up to a full year to perform 100,000 evaluations. The challenge of cutting down this time from years to a few months or weeks, while still being able to produce a relatively good-playing soccer team from only a small number of evolutionary runs, was one of the main problems they faced. Luke and Spector managed to take on this problem in several ways [11]:

- They used brute force in order to speed up the process, running 32 parallel games and cutting down time for each evaluation from a full game of 10 minutes to limited periods of games of between 20 seconds and one minute duration.
- They cut down population size and number of generations.
- They developed an additional layer of software to simplify the domain in order to eliminate many boundary conditions the GP programs would have to account for.
- They spent much time designing a function set and evaluation criteria to promote better evolution in the soccer simulation domain.
- They used parallel runs with different genome structures to have more options when they were close to the competition.

Their fitness function was only based on the number of goals that a team scored in a game. In order to prevent premature convergence (a problem because of their use of small population sizes) they employed a high mutation rate of 30% [3, 11].

After Luke's team pioneered GP in the soccer simulation domain in RoboCup97, a team named *Darwin United* applied GP to evolve their agents in RoboCup98 [2]. Darwin United used a different method than Luke's team for evolving their players and they employed an optional coach agent - who receives noiseless data from the server but has very limited communication with agents- for storing data and coordinating the decision for rewarding players after each command execution [17].

After Luke and Darwin United, Aronsson in [3] also employed GP to teach software robots to play soccer. He focused on designing a better fitness function. Aronsson also made several compromises to limit the duration of the evolutionary process because of excessive running times needed for evaluating each population due to the complex nature of the soccer simulation framework.

Aronsson used a similar approach as Luke for implementing trees in his GP system. Both groups evolved a move tree and a kick tree for agents and set a basic state-rule to determine which tree to call in each cycle of the game. If a player was close enough to kick the ball, the kick tree would be called and if a player could see the ball but it was not reachable, the move tree would be called. Figure 1 depicts the structure of a decision tree in Aronsson's research. Each

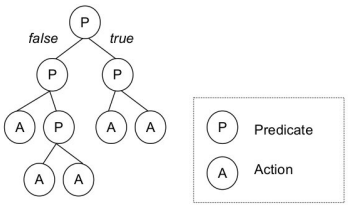

Figure 1: The structure of Aronsson's decision trees

leaf node is an action and all other nodes are predicates in his experiments. Predicates are Boolean functions that give information about the environment around the agents such as: *OpponentIsClose*, *IAmNearGoal*, and *IAmClosestToTheBall*.

One main difference in the decision making of Aronsson's agents compared to previous approaches was that players that calculated to have an obvious chance of scoring a goal would attempt to score without considering the decision of the evolved kick tree. This "action overriding" was expected to focus the agents' attention on team coordination and positioning. Improving the quality of their decision making indeed worked as expected.

The fitness measure used by Aronsson was based on each player's performance during a game, or on the average of its performance if that player played multiple games during the evaluation process. The fitness value was calculated as weighted sum of some parameters such as: team's score, opponent team's score, number of goals the agent scored, total number of passes made by the agent, number of shoots on goal made by the agent, and etc.

Aronsson performed two experiments, different only in the team set-up. In the first experiment, players learned to chase and kick the ball towards the goal or pass it to a teammate, but in the second experiment players converged and did not develop further than the first experiment. Also it was found that players from the first experiment were developing towards team coordination only slowly [3].

Aronsson's conclusion was that software robots are able to learn to play simulated soccer but that the strategies that the robots developed were most likely inferior to human-coded algorithms, though better than initially random strategies [3].

GP was rarely used in the domain of soccer simulation after these contributions were published. Lichocki et al. in [10] evolved team compositions by agent swapping and Aşık and Akın in [4] used genetic algorithms for solving multi-agent decision problems.

2.1 The Challenges of Evolving Agents in the Soccer Server Domain

Luke, Aronsson, and the Darwin United team used GP for evolving soccer agents in the soccer 2D simulation framework. Some important aspects and challenges of research done by Luke's team are as follows;

- In his research, all agents were evolved only during the actual run of the soccer simulator and there is no learning for agents except inside the matches.
- For evaluating individuals during a game, each agent can execute only one decision tree so that it will be evaluated by its decisions. Due to the limited number of players (11 per team) in each game and the long run times for each game, large population sizes are almost impossible.
- Since it was the first year of the competition, skills developed for agents (such as pass, dribble, shoot, etc.) were very simple and not as mature as today.

Here are some problems and challenges Aronsson faced during his research:

- All agents were evaluated during actual runs of the simulator. Evaluating individuals during running a game was very time-consuming so he had to limit evaluation times which rendered weaker agents.
- The evolved agents had difficulties with basic skills, such as ball interception. Even when GP found a very good individual, this basic skills might have resulted in bad performance, receiving a low fitness value ultimately leading to its elimination.
- Due to the nature of the soccer simulator, only small population sizes were possible which limited the exploration of the search space and had a substantial impact on the final result.
- As a result, the team was not competitive enough to play against teams that were participating in the RoboCup competition of that year.

3 HYBRID GP METHOD FOR DECISION MAKING OF AGENTS

Our new method consists of using GP with a newly proposed evaluation method in order to generate decision trees for soccer agents, followed by another GP algorithm, with a different setup and evaluation method, which uses the best individuals of the first GP as input rather than a random population.

In the first phase of the approach, a GP algorithm with a random initial population and a large population size creates decision trees for agents. The fitness function evaluates the behavior of an agent in some pre-defined situations from real games with a set of desired outputs, each of which with a pre-defined score for different actions performed. The data used for evaluating individuals in order to train our agents are generated using actions of agents from top teams of the world from the latest RoboCup competition. Individuals will be scored on the decisions they made for these specific situations using a fitness function that evaluates those decisions. The selection method in our method is tournament selection for this phase of the algorithm, with crossover and mutation for generating variant trees using the sub-tree replacement method. Function and terminal sets are player skills (such as pass, dribble,etc.) and predicates from the environment needed for making decisions (such as inOppField, oppIsClose, etc.).

In the second phase of the method, the best individuals resulting from the first phase are used as inputs to another GP system. This phase is again using a GP to evolve decision tree of agents, however, this happens in real-time games. Differences between the GP method used in second phase and first phase of the approach can be summarized as:

- In the first phase, individuals are initialized randomly using the ramped half-and-half method. In the second phase, the

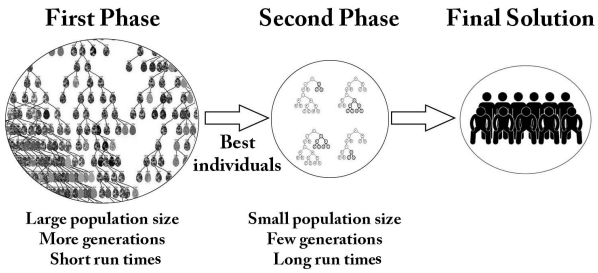

Figure 2: An overview of the proposed hybrid method

best individuals of the first phase are used as first generation individuals. Thus, in the second phase fit individuals from pre-defined scenarios of games are further evolved by taking part in real-time simulated soccer games.

- Individuals of the first phase are evaluated using some pre-defined situations and also a pre-defined scoring system as the fitness function (supervised learning of situations). In the second phase, each individual is evaluated in a real run of a simulated soccer game, mostly by feedback for each action of agents. For instance, if an agent decides to pass a ball to a teammate and that action results in losing the ball in a short period of time, or if it decides to dribble with the ball when it is not safe to do so, it will be considered a wrong decision and will negatively affect its fitness value compared to other individuals. The total result of the game also affects all team members' fitness values.
- In the second phase, the population size is much smaller due to the long run time of the games needed for evaluating an individual (about 10 minutes per run) as well as the limited number of agents that are able to be tested in the field (one individual per agent, so no more than 10 evaluations in each game excluding the goalkeeper).

A sketch of the proposed hybrid method and the main characteristics of both GP systems used in the two phases is shown in Figure 2.

3.1 Addressing Challenges

This hybrid GP method will address some of the main challenges that were faced in previous works using GP in the soccer simulation environment such as:

3.1.1 Small Population Size. Because each evaluation was done previously during a partial run of a game in the simulator, the limit of evaluating 10 individuals per game prevented researchers from having a large population size for each generation of the GP run. Using the first phase of GP with a fitness function that can evaluate individuals without the need of running a full game enables us to execute millions of evaluations in a much shorter period of time.

3.1.2 Time-consuming Evaluations. Each run of a full game in the soccer simulator takes about 10 minutes. If an individual is to be evaluated during one game, it therefore takes 10 minutes to evaluate. The largest number of evaluations that can be done in a game is 10 as we have 10 players excluding the goalkeeper agent. In order to achieve a desired number of evaluations we would need months and years for evaluations. The first phase of GP helps finding some intelligent agents that can properly behave in certain situations (games snapshots). They use this knowledge in the second phase to "interpolate" between situations generating better auspices for improvement of behavior in the time-consuming evaluations of full real-time games.

3.1.3 High Level Skills of Agents. This research uses source code and skills of the *MarliK* team that is an internationally recognized team in the RoboCup soccer 2D simulation league and was placed third of the world in 2011 and 2012 and also won first place in some international competitions such as DutchOpen and IranOpen from 2009 to 2013. Instead of agents with very basic skill functions such as intercepting, passing, and shooting skills used for evolution in earlier work, the high level skills of the *MarliK* team are used here, which lowers the chance of eliminating good individuals during the evolution process due to errors caused by basic skill functions, as happened in previous research.

3.2 Implementation

We use the GPC++ library [7] for implementing GP in both phases of the hybrid method. The structure of a tree in our tree-based GP system is very similar to what Aronsson applied in his research, shown in Figure 1. Here, we implement and test the hybrid method for the kick decision tree of agents, the decision tree for when agents have the ball.

The function set for both phases of the method includes agent skills such as *pass*, *dribble*, and *clear*. Terminals are also chosen from the following predicates: *nearGoal*, *nearOwnGoal*, *inOppField*, *oppIsFar*, *oppIsClose*, *oppIsVeryClose*, *weAreWinning*, *weAreDefending*, *weAreAttacking*, *tmmAvailable*, *pathClear*, and *ballInDangerArea*.

3.3 Fitness of First Phase

Each individual in our tree-based GP is evaluated by the actions it takes in some predefined simulated situations that are derived from real games. These situations are extracted using the decisions that were made by agents of the strongest teams from the RoboCup 2016 competition. Each of these situations is called a "snapshot" and it contains information about the field, such as position of teammates and opponents in that specific cycle of the game. Each snapshot is associated with a reward for each particular action that could be made in that situation. A decision in a snapshot is classified into one of the following categories associating a number with each of them that increases as the action become worse;

- **perfect:** The best possible action for that situation of the game which was chosen in the real game by the agents of top teams.
- **good:** The action is good to be executed and there is not much risk included in the action in that situation of the game.
- **bad:** The action is not a good choice and might result in losing possession of the ball or an opportunity, and there is at least one better option to choose from.
- **veryBad:** The action is a very bad choice and it is very risky for that cycle of the game.

Figure 3: A snapshot from final game of RoboCup 2016

- **worst:** The action is an obvious bad choice and will immediately result in losing a great opportunity or losing possession of the ball.

Figure 3 shows a snapshot of the RoboCup 2016 final game between the *Helios* and *Gliders* teams that was used in our fitness function. In this situation, player number 9 of the attacking team on the left side of the field has a great opportunity to score a goal and decided to dribble with the ball towards the goal to improve its chance for scoring. For this snapshot, *dribble* action is set as *perfect* decision, and *pass* and *clear* actions are classified as *bad* and *worst* decisions, respectively.

We used 100 snapshots from games of the final round of RoboCup 2016 in order to evaluate the first phase of genetic programming.

3.4 Fitness of Second Phase

Due to the complexity and nature of the soccer simulator, evaluation of individuals in this phase of the algorithm is very time consuming compared to the first phase. The fitness function for this phase of the algorithm is reward-based and each individual will get a reward based on the action that it took in each cycle of the game. Evaluation of these actions is achieved using a combination of some parameters from consequences of these actions and some basic rules of common sense, such as when the agent does not clear the ball during an attack situation in front of the opponent's goal.

Each population is evaluated during one full game run in the soccer simulator. An individual is randomly assigned to a player before beginning of a match and the fitness value of that individual is the sum of its rewards received from all the actions taken during the game.

Here is a list of events that affect fitness values of individuals of a population:

- If ball possession is lost to the opponent team in the following 20 cycles after the agent executed an action.
- If a goal is scored in the next 100 cycles.

Table 1: Scoring system of individuals

Parameter	Value
perfect	0
good	10
bad	20
veryBad	40
worst	80

- If ball object's x dimension is increased or decreased (an increase means ball is moved toward the opponent's goal).
- If the player is not a defender and cleared the ball while in opponent's field.
- If opponent team scored a goal in the next 50 cycles.
- If the player decided to dribble with the ball while teammate goalie agent is very close (chance of giving a back pass fault to opponent).

Each of the above events will positively or negatively affect the fitness value of individuals in the population, depending on the effectiveness or severity of the event on the whole team's gameplay.

4 PERFORMANCE ANALYSIS

4.1 Runs of the First Phase

We performed 10 runs with different random seeds for the first phase of the method. During these runs, as it was expected, initial individuals made good progress toward reaching the target fitness. A smaller fitness value for an individual means that it performed better actions in more snapshots of the game. For example if an individual makes the *perfect* decision in 75 out of a total of 100 snapshots, and makes 15 *good*, 6 *bad*, 2 *veryBad*, and 2 *worst* decisions, it will end up with a fitness value of 510. The scores associated with each of these actions are shown in Table 1.

The best individual after 100 generations had a fitness value of 180 which means it had a very good performance in the simulated snapshots of the game and it chose the *perfect* decision in at least 82 out of 100 snapshots.

Figure 4 shows the development of the fitness of the best and worst individual of each population as well as the average fitness of individuals in each generation during evolution of the first phase. The data shown in Figure 4 gives the average of 10 runs performed during our experiment. Smaller fitness values represent better performance.

Figure 5 shows error bars on the average fitness of individuals over generations in the runs of the first phase using standard deviation of individuals in each generation indicating what fitness the majority of each population achieved.

It can be observed that in all runs of the first phase, major progress of individuals is made in the first 20 generations and the latest improvement observed in the best fitness was in generation 83. Destructive mutation and crossover operators are the main reasons that cause significant changes in the fitness of worst individual over the generations. It can also be observed that the population seems to converge repeatedly in certain generations, just to later break out again. This seems to happen when the worst individual

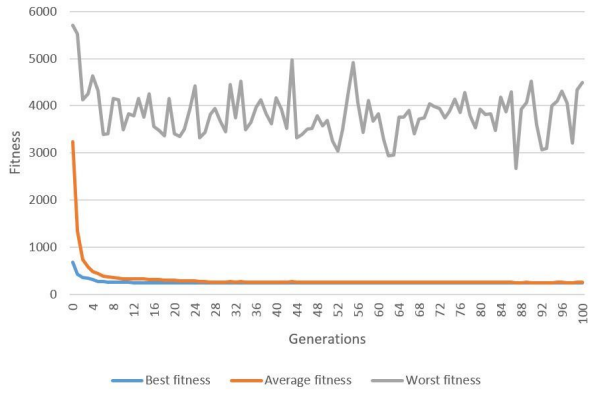

Figure 4: Evolution of best, average, and worst fitness in runs of the first phase

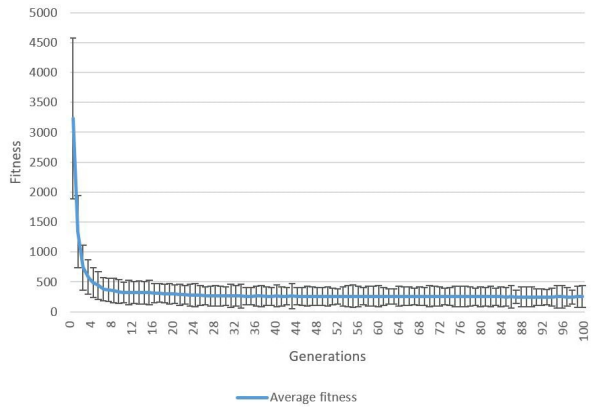

Figure 5: Error bars indicating (one) standard deviation of individuals in runs of the first phase

of population has a better fitness and it seems to break out when a significant increase in worst proportion of the population happens due to a destructive mutation or crossover as it happened in generations 52 and 98.

4.2 Run of the Second Phase

The top 10 individuals of the first phase of running GP were then used as the initial population of the GP in our second phase in order to be evolved during real runs of the soccer simulator. A significant improvement in best fitness was not expected since the initial population was not randomly generated and individuals were already evolved during the first phase. This also demonstrates that the off-game evolution using snapshots was useful as guidance in real-time games of this phase.

The individuals that were evolved during the first phase of the algorithm were only tested against 100 specific snapshots from real games. The fitness function of the second phase of the algorithm evaluates the individuals during real games where they could get actual feedback of their behavior to ensure the decisions made from simulated snapshots actually work well during real games as well.

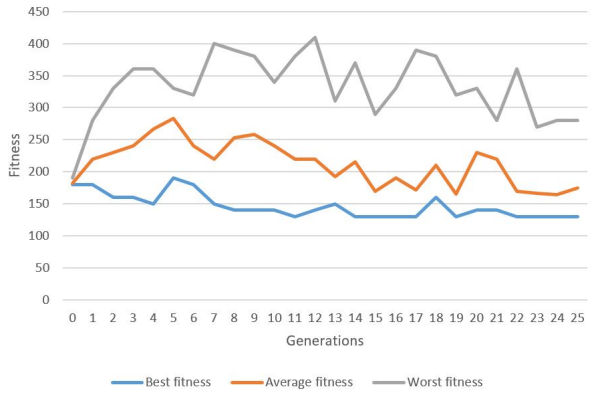

Figure 6: Fitness evolution in second phase over generations

As expected, a smaller improvement in fitness values and behavior of agents was observed during the runs of GP in this phase of the method for 25 generations.

Figure 6 illustrates evolution of the best, worst, and average fitness in each generation for this phase of the algorithm. Elitist selection is used in both phases of the method and the best individual will always survive to the next generation. The reason for a worsening best fitness value in some consecutive generations is the fact that roles of individuals are assigned to players randomly in each generation. For example, the best individual of generation 4 was the one that was assigned randomly to a defender role during the evaluation run and it received a fitness value of 150. The same individual was assigned to an attacker role during the next generation and its fitness value worsened as it did not make good decisions as an attacker during the second time that it was being tested. As it was explained, this process makes sure that during the evolutionary process, the best individuals are the ones that are most likely to perform well in all different roles.

Looking at the average fitness values in the run of this phase, we can see that in the first generation all individuals of the population had good fitness since they were the best ones from the previous phase of the method. However, in the course of 25 generations, some changes take place that obviously made some individuals perform better and some worse than at the outset. The significant changes in average fitness of different generations compared to first phase can be explained by the small population size for this phase of the algorithm.

For the same reason, the worst fitness of the first generation is significantly lower than most of the next generations as shown in Figure 6. Destructive crossover and mutation operators are the main reasons for the significant changes of the worst individual's fitness in each generation.

Figure 7 shows the distribution of individuals in each generation using one standard deviation of the individuals in each population.

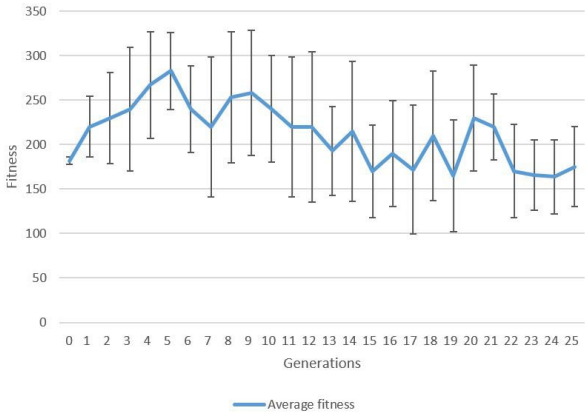

Figure 7: Error bars indicating (one) standard deviation of individuals in the second phase

Table 2: Performance of hybrid GP method vs. old *MarliK*

Tested team	Opponent	Avg. GF	Avg. GA	Win %
Hybrid method	Agent2D	6.12 (2.27)	0.40 (0.57)	100%
Old MarliK		4.36 (1.81)	0.44 (0.50)	92%
Hybrid method	Helios	1.00 (0.89)	1.56 (1.20)	20%
Old MarliK		0.72 (0.78)	1.96 (1.31)	8%
Hybrid method	Gliders	2.08 (1.81)	1.12 (0.77)	40%
Old MarliK		1.84 (1.41)	1.12 (0.99)	40%

4.3 Hybrid GP Method vs. MarliK's Old Decision Making System

The final version of *MarliK* without the new decision making system built by the hybrid GP method was tested in 25 games against the *Agent2D*, *Helios*, and *Gliders* teams. Then, the new decision making system which consists of the best individual from the last generation of the second phase of the method was added to *MarliK* for all of the agents and tested against same teams.

Agent2D is an open-source base code that most of the teams in soccer 2D simulation league, as well as *MarliK*, are currently using, and *Gliders* and *Helios* are the most powerful teams from latest (2016) RoboCup competition.

Table 2 shows the results of these experiments. Data from each row of table represents results of 25 games. Average number of goals scored and conceded per game with their standard deviation shown in parentheses as well as percentage of wins of each version of MarliK that was used is shown in the table.

We observe that the hybrid GP method of decision making outperformed *MarliK*'s previous static decision making system in most cases, leading to better overall results. Looking at the results of Table 2, it can be concluded that the attacking power of the team was more positively influenced than the defending power of the team. An explanation for this fact is that the hybrid GP method is only used for with-ball decision making of agents here. Power of defense in a soccer team is more influenced by skills such as *block*, *mark*, and *intercept* which are executed in the without-ball decision making of players.

Table 3: Performance of homogeneous approach vs. heterogeneous approach

Tested team	Opponent	Avg. GF	Avg. GA	Win %
Homogeneous	Agent2D	6.12 (2.27)	0.40 (0.57)	100%
Heterogeneous		5.72 (1.82)	0.36 (0.56)	100%
Homogeneous	Helios	1.00 (0.89)	1.56 (1.20)	20%
Heterogeneous		0.92 (0.89)	1.32 (1.05)	16%
Homogeneous	Gliders	2.08 (1.81)	1.12 (0.77)	40%
Heterogeneous		2.48 (1.70)	1.16 (1.08)	52%

4.4 Homogeneous vs. Heterogeneous Approach

In this research, we have followed the homogeneous approach meaning that we evolved/used the same decision tree for all agents of a team, without considering their specific role. A heterogeneous approach is also possible to follow which means to evolve separate decision trees for each player, or for different groups of players with similar roles (defender, midfielder, and attacker).

Whether to have the same decision tree for all players or separate decision trees for different player roles (defenders, midfielders, and attackers) is one of the big challenges for RoboCup teams. Here we try to answer this question by comparing outcomes for these two approaches to our method.

For answering the question of whether a homogeneous or a heterogeneous approach works better for our method, we ran a parallel experiment with the second phase of the algorithm where agents were divided into the 3 main categories of *defenders*, *midfielders*, and *attackers*. Each of these categories had a separate (evolving) population and the experiment was done using the same GP configuration as the second phase of our method before. The best individuals from the latest population of the first phase GP were used as initial population for both teams.

Table 3 compares performance of these two approaches of our method. We observe that the heterogeneous team slightly outperforms the homogeneous against one opponent. From these results and the experiments from previously discussed research, we expect that over more generations, the heterogeneous approach has more chance of evolving better individuals compared to the homogeneous approach. Specializing in one behavioral role showed slightly better performance, in particular against the *Gliders* team.

4.5 Hybrid GP Method vs. Base Algorithms

In order to make sure that the hybrid GP method works better than the old decision system of *MarliK* as well as either phase of the method separately in face-to-face matches, we did another experiment.

First, the top individual from the last generation of the first phase was chosen as the decision tree of all players in *Team A*. Then, the second GP of the hybrid method which had a small population size and used time-consuming runs of the soccer simulator for evaluating individuals was executed again, but instead of using the best individuals from the first GP as initial population, a randomly

Table 4: Performance of hybrid GP method against base algorithms

Tested team	Opponent	Avg. GF	Avg. GA	Win %
Hybrid method	Old MarliK	1.68 (1.43)	1.56 (1.44)	56%
Hybrid method	Team A	1.64 (1.16)	1.44 (0.94)	64%
Hybrid method	Team B	5.36 (2.04)	0.00 (0.00)	100%

generated initial population was used and individuals were evolved for 25 generations (same as it was done in second phase of the hybrid GP method). The best individual from the last generation was used as decision tree of all players in *Team B*.

We tested the hybrid GP team in face-to-face matches against previous version of *MarliK* with the old decision making system, Team A, and Team B. Results of running 25 games between each of these teams against the hybrid GP team are shown in Table 4. These results support our previous experiments and ensure better performance of our method's final solution against each of the base algorithms.

5 CONCLUSIONS AND FUTURE WORK

In this research we proposed and implemented a new hybrid GP method to improve the decision making of soccer simulation teams. The proposed approach consists of two phases each of which tries to cover the other's restrictions and limitations. The first phase will produce some evolved individuals based on a GP algorithm with an off-game evaluation system and the second phase uses the best individuals of the first phase as input to run another GP algorithm to evolve players in the simulated game environment.

To test the method, we used the *MarliK* team skill set. Our hybrid GP method improved agent performance as well as the overall team performance against three other teams. Two of these teams are top teams from the latest (2016) RoboCup competition and the third team is the base code that is used by most of the RoboCup teams as a benchmark. Comparing to previously mentioned work similar to the second phase of our method, we observed that adding the first phase of GP with its different type of fitness function, improved overall performance of final solutions substantially.

Since evaluations in the second phase of our method are very time-consuming, we only implemented the method for the kick decision tree of agents. In future work, the move tree can also be implemented using the same method.

Another important factor in this research that will affect the results is the opponent team against which the individuals are evaluated in the second phase. Depending on the behavior and power of the opponent team, the process of evolution for individuals might be different. This can also be investigated in future work to see how this factor might influence the final solutions.

Because of the nature of the pass skill function in *MarliK*, we only included one pass function in the terminal set of our GP and let *MarliK*'s pass function decide whether to perform a *direct pass*, a *leading pass*, a *through pass*, or a *cross pass*. Depending on the implementation type for pass skills, each type of these passes can also be used in the function set of both GPs. Moreover, other predicates might be considered for adding to the function set of both GPs that will directly affect the evolving decision trees.

ACKNOWLEDGEMENT

Funding to W.B. from NSERC under the Discovery Program, RGPIN 283304-12, is gratefully acknowledged.

REFERENCES

[1] David Andre. 1995. The Automatic Programming of Agents That Learn Mental Models and Create Simple Plans of Action. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence - Volume 1 (IJCAI'95)*. Morgan Kaufmann Publishers Inc., San Francisco, 741–747. http://dl.acm.org/citation.cfm?id=1625855.1625952

[2] David Andre and Astro Teller. 1998. Evolving team Darwin United. In *RoboCup-98: Robot SoccerWorld Cup II*, M. Asada and H. Kitano (Eds.). Springer Verlag, Berlin, 346–351.

[3] Jonatan Aronsson. 2003. *Genetic programming of multi-agent system in the robocup domain.* Master's thesis. Lund Institute of Technology, Sweden.

[4] Okan Aşık and H. Levent Akın. 2013. Solving Multi-agent Decision Problems Modeled as Dec-POMDP: A Robot Soccer Case Study. In *RoboCup 2012: Robot Soccer World Cup XVI*, Xiaoping Chen, Peter Stone, Luis Enrique Sucar, and Tijn van der Zant (Eds.). Springer Verlag, Berlin, 130–140. DOI: http://dx.doi.org/10.1007/978-3-642-39250-4_13

[5] Wolfgang Banzhaf. 2013. Evolutionary Computation and Genetic Programming. In *Engineered Biomimicry*, Akhlesh Lakhtakia and Raul Jose Martin-Palma (Eds.). Elsevier, Boston, 429–447.

[6] Jacques Ferber. 1999. *Multi-Agent Systems: An Introduction to Distributed Artificial Intelligence.* Vol. 1. Addison-Wesley Longman, Harlow.

[7] Adam Fraser. 1993. *GPC++ genetic programming C++ class library.* http://www0.cs.ucl.ac.uk/staff/ucacbbl/ftp/weinbenner/gp.html.

[8] Thomas Haynes and Sandip Sen. 1997. Crossover operators for evolving a team. In *Genetic Programming 1997: Proceedings of the Second Annual Conference*, J. R. Koza, K. Deb, M. Dorigo, D. B. Fogel, M. Garzon, H. Iba, and R. L. Riolo (Eds.). Morgan Kaufmann Publishers Inc., San Francisco, 162–167.

[9] Hitoshi Iba. 1996. Emergent cooperation for multiple agents using genetic programming. In *International Conference on Parallel Problem Solving from Nature*, H. M. Voigt, W. Ebeling, I. Rechenberg, and H. P. Schwefel (Eds.). Springer Verlag, Berlin, 32–41.

[10] Paweł Lichocki, Steffen Wischmann, Laurent Keller, and Dario Floreano. 2013. Evolving team compositions by agent swapping. *IEEE Transactions on Evolutionary Computation* 17, 2 (2013), 282–298.

[11] Sean Luke. 1998. Genetic programming produced competitive soccer softbot teams for RoboCup97. In *Genetic Programming 1998: Proceedings of the Third Annual Conference*, J. R. Koza and others (Eds.). Morgan Kaufmann Publishers Inc., San Francisco, 214–222.

[12] Sean Luke, Charles Hohn, Jonathan Farris, Gary Jackson, and James Hendler. 1998. Co-evolving soccer softbot team coordination with genetic programming. In *RoboCup-97: Robot Soccer World Cup I*, H. Kitano and others (Eds.). Springer Verlag, Berlin, 398–411.

[13] Sean Luke and Lee Spector. 1996. Evolving Teamwork and Coordination with Genetic Programming. In *Proceedings of the 1st Annual Conference on Genetic Programming*, J. R. Koza, D. E. Goldberg, D. B. Fogel, and R. L. Riolo (Eds.). MIT Press, Cambridge, 150–156.

[14] Sivadev Nadarajah and Kenneth Sundaraj. 2013. A survey on team strategies in robot soccer: team strategies and role description. *Artificial Intelligence Review* 40, 3 (2013), 271–304.

[15] Itsuki Noda. 1995. Soccer server: a simulator for RoboCup. In *JSAI AI-Symposium 95: Special Session on RoboCup*. Japanese Society for Artificial Intelligence, Tokyo, 29–34.

[16] Peter Nordin and Wolfgang Banzhaf. 1997. An on-line method to evolve behavior and to control a miniature robot in real time with genetic programming. *Adaptive Behavior* 5 (1997), 107–140.

[17] Liviu Panait and Sean Luke. 2005. Cooperative multi-agent learning: The state of the art. *Autonomous agents and multi-agent systems* 11, 3 (2005), 387–434.

[18] Adil Qureshi. 1996. Evolving agents. In *Proceedings of the 1st Annual Conference on Genetic Programming*, J. R. Koza, D. E. Goldberg, D. B. Fogel, and R. L. Riolo (Eds.). MIT Press, Cambridge, 369–374.

[19] Simon Raik and Bohdan Durnota. 1994. The evolution of sporting strategies. In *Complex Systems: Mechanism of Adaptation*, Russel Stonier and Xing Huo Yu (Eds.). IOS Press, Amsterdam, 85–92.

[20] Amir Tavafi, Nima Nozari, Reza Vatani, Mani Rad Yousefi, Sepideh Rahmatinia, and Pouyan Pirdir. 2012. MarliK 2012 Soccer 2D Simulation Team Description Paper. In *RoboCup 2012: Robot Soccer World Cup XVI*, X. Chen, P. Stone, L. E. Sucar, and T. van der Zant (Eds.). Springer Verlag, Berlin.