Genetic Programming and Multi-Agent Layered Learning by Reinforcements

William H. Hsu
bhsu@cis.ksu.edu
Department of Computing and Information Sciences
Kansas State University
Manhattan, KS USA 66506-2302

Steven M. Gustafson
smg@cs.nott.ac.uk
School of Computer Science and Information Technology
University of Nottingham
Jubilee Campus, Nottingham UK NG8 1BB

Abstract

We present an adaptation of the standard genetic program (GP) to hierarchically decomposable, multi-agent learning problems. To break down a problem that requires cooperation of multiple agents, we use the *team objective function* to derive a simpler, intermediate objective function for pairs of cooperating agents. We apply GP to optimize first for the intermediate, then for the team objective function, using the final population from the earlier GP as the initial seed population for the next. This *layered learning* approach facilitates the discovery of primitive behaviors that can be reused and adapted towards complex objectives based on a shared team goal. We use this method to evolve agents to play a subproblem of robotic soccer (keep-away soccer). Finally, we show how layered learning GP evolves better agents than standard GP, including GP with automatically defined functions, and how the problem decomposition results in a significant learning-speed increase.

1 INTRODUCTION

For complex problems with low-level primitive operations, such as robotic soccer [Ki97, MNH97], it is intractable to search for a direct solution using genetic programming (GP). This is due in part to the combinatorial explosion of the GP search space as a function of the problem state space – e.g., the size of the playing field. [SVR99] Other factors, such as operator granularity, also contribute to this growth. Many of GP researchers who have worked on robotic soccer have simplified the GP search space through problem redefinition: raising the level of terminals in order to evolve higher-level behaviors [Lu98] or using a more sophisticated fitness function [AT99]. Because robotic soccer is a multi-agent system (**MAS**) problem that is based upon a real game played by humans, it is helpful to compare learning strategies with those of human teams, even if we use a different approach to automatically develop a solution. One important observation is that the structure of team training in real soccer involves individual, pair, and small group drills, resulting in a well-defined hierarchy of behaviors. Traditional GP produces hierarchical programs by evolving and reusing automatically defined functions (ADFs). [Ko94, RB94]

In this paper, we show how *layered learning* can also achieve reuse – faster and more reliably than GP with ADFs – in developing a solution to an MAS subproblem of robotic soccer. Just as ADFs provide reusable code and subroutine structure [Ko94], layered learning provides a way to build solutions using a divide-and-conquer approach [St00, SV00a]. The difference between ADF learning and layered learning, using GPs or other methods, is that layered learning describes a way to *train* a learning intelligent agent, while ADFs describe a way to *implement structure* in the agent representation – i.e., code.

Layered learning GP (**LLGP**) [GH01] can be used to break down MAS learning tasks by first evolving solutions for smaller fitness cases or for smaller groups of agents with a more primitive fitness criterion. While our adaptation of layered learning to GP is based in part upon Stone and Veloso's work in reinforcement learning [SV00a], similar approaches have been developed that perform sequential evolution of populations using different fitness functions [De90, HHC94].

This paper extends our previous study of LLGP for an MAS task in the robotic soccer domain [GH01] with further experiments and analysis of LL behavior. We focus on automatic tuning and validation of *intermediate representations* in incremental LL. The purpose of our test bed is to facilitate development of fitness criteria for "coaching" or training agents based upon their strictly cooperative performance in a two-agent task. We then use the evolved individuals to seed a population of agents to be further improved in three-way competitive interaction against a fourth agent, the opponent. This new population and the associated GP form the second layer of the LLGP system. The product of LLGP is an agent that is *evolved* using highly fit primitive agents, but does not necessarily contain exact copies of these primitive agents as subroutines.

Another advantage of layered learning is that it provides a logical methodology for implementing a hierarchical approach to teamwork. In order to evolve more complex teamwork, we may be able to take advantage of the dependency of behaviors involving three or more teammates upon primitive behaviors involving just two. For example, a low-level primitive in soccer is passing the ball, a two-agent activity that is incorporated into several multi-agent activities: guarding the ball; moving the ball downfield; setting up for a goal attempt; etc. In the rest of this paper, we shall explore LLGP for MAS problems using *keep-away soccer*, a subproblem of robotic soccer [SV00a, GH01] that shows how complex teamwork can be hierarchical in nature and therefore can be learned efficiently in a hierarchical fashion.

2 THE KEEP-AWAY SOCCER DOMAIN

2.1 DEFINITION AND JUSTIFICATION

We call *keep-away soccer* the task of keeping the ball away from a defensive player who is attempting to capture it from multiple offensive opponents. We chose keep-away soccer as a learning test bed for MAS because it:

1. captures a compositional element of teamwork, composing and refining passing behaviors to achieve full keep-away soccer behavior, that occurs in real and robotic soccer
2. elides some objectives of soccer (such as moving the ball downfield and attempting to score) that, while crucial, would overcomplicate our study of basic low-level MAS
3. allows us to easily adjust opponent difficulty

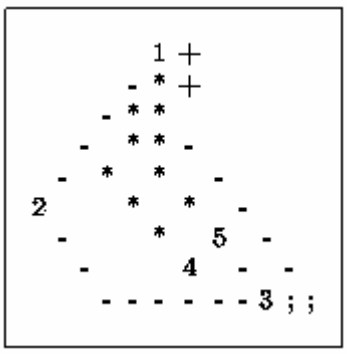

Figure 1. Screen capture of simulator. 1, 2, and 3 are offensive agents, 4 is the defender, and 5 is the ball, which moves in trajectory 3-1-2-3-1.

Although there is a strong compositional element, learning to pass the ball effectively is only part of the keep-away soccer learning task. In real soccer, human players learn to minimize the number of turnovers to the offensive opponent by passing accurately, move to receive a pass, and make themselves open to receive a pass, and control the ball effectively. For 3 or more agents to coordinate effectively, each must be able, when in possession of the ball, to: select a teammate to pass to, time the pass appropriately, and maintain open at least one passing lane.

Figure 1 shows a text-mode screen capture from the simple program that we used to visualize and animate games of keep-away soccer. The figure depicts three offensive agents passing the ball in a counterclockwise motion (agent 3 passes the ball twice) about a defender. The trail of the ball is denoted by '-'. The symbols '+', ';', and '*' show the paths of agents 1, 3, and 4, respectively. The simulation and visualization were run for about 30 time steps to collect the screen capture.

Several MAS variants of robotic soccer exist; keep-away soccer belongs to the category of multi-agent learning with *homogeneous, noncommunicating* agents [SV00b] – those that share identical code but have no direct channels of communication other than by observing the behavior of teammates. This type of problem requires more robust, autonomous solutions and is therefore an interesting framework for teamwork learning.

Soccer, whether analyzed as a human or robotic game, can be broken down into skill-optimization subproblems such as ball control, passing, and moving. Keep-away soccer can be decomposed in the same manner. A natural way to reduce complex, MAS problems, such as that investigated in keep-away soccer, could generalize to other cooperative MAS problems [Ta97].

2.2 PROBLEM SPECIFICATION

Test beds for robotic soccer-playing agents have been framed through the *RoboCup* competition [KAK+95, As99]. These have been found to be rich experimental environments for many MAS research areas, including flexible teamwork learning [TAA+99], and methodologies, including hierarchical sensing and reinforcement learning by *Q*-learning, temporal differences, team-partitioned algorithms [SV98], artificial neural networks, and genetic programming [As99]. At present, however, hand-coded and hybrid learning techniques that employ a large amount of hand-coded domain-specific knowledge still outperform strategies that are learned automatically.

In keep-away soccer, three offensive agents are located on a rectangular field with a ball and a defensive agent. The defensive agent moves twice as quickly as the offensive agents, and the ball, when passed, moves twice as quickly as the defensive agent. This is similar to the predator-prey problem in [LS96], where more than one agent is required to solve the problem. The objective in keep-away soccer is to minimize the number of times the ball is turned over to the defender. A turnover occurs at every discrete time step in which the defender is within one grid unit of the ball. Thus, subsidiary objectives for offensive agents are to continuously move and pass the ball to one another in order to minimize turnovers.

We think of keep-away soccer as consisting of two layers of behavior: passing accurately with no defensive agent present, and moving and passing with a defender to minimize the number of turnovers that occur during a game. The two layers of behaviors come from a human-like view of soccer, but are not heavily dependent upon domain knowledge. Both types of behavior are important to playing good keep-away soccer, but the operational definition does not necessarily give us a way to measure the effectiveness of a team of agents who have just played keep-away soccer, which would be useful for finding a fitness function.

Next, we present the application of layered learning to GP and explain further how the keep-away soccer is a good, illustrative test bed for LLGP.

3 LAYERED LEARNING

Layered learning is a term used in the machine learning and intelligent agents literature [St00, SV00a] to describe a task-driven and often incremental approach to acquiring hierarchies of behaviors by reinforcement learning.

de Garis [De90] introduced a very similar concept that he called *behavioral memory* for a genetic algorithm that encoded neural networks. Weights and signs of the networks evolved for one behavior were used to construct a new population, evolved for a second behavior. Some of the initial network persisted in the solutions for the new behavior. Schoenauer and Xanthakis [SX93] then later applied this concept for constrained genetic algorithm optimization.

Harvey *et al* [HHC94] used a layered, incremental learning approach to robot control in a vision-based navigation system. The authors achieved this by sequentially evolving a population using a range of targets from simple to complex. Winkeler and Manjunath [WM98] and Eriksson [Er00] later analyzed this approach toward incremental learning.

Dorigo *et al* [DC97] developed another hierarchical learning system that is somewhat different from layered learning as we have adapted it. In this method, inputs and processing elements are organized into a hierarchy (from simple to complex), each of whose layers is incrementally trained and frozen. This is similar to previous work applied in domains such as robot soccer, but is not identical to layered learning or behavioral memory as these methods do not arrest learning in a particular portion of the hierarchical model.

Applying the layered learning paradigm to a problem consists of breaking that problem up into a hierarchy of subproblems. The original problem is then solved sequentially, by using the learning results from all the member problems of each layer in the next layer. This is conceptually similar to many other divide-and-conquer learning paradigms, but a key difference is that the structure of the *solution* does not necessarily reflect this procedural hierarchy of *training*. For example, programs evolved for a subtask in LLGP are used to seed an initial population for the next layer, but they may not be incorporated verbatim in the overall solution as ADFs are. This type of hierarchical solution is different from the type that ADF-based GP learning proposes to find, which focuses on code reuse and structure rather than on how the subtasks are learned.

Problems that attempt to achieve human-competitive behaviors [Ko98], such as robotic soccer and keep-away soccer, lend themselves well to bottom-up decomposition. This is because human task learning, especially of cooperative multi-agent behavior, often occurs in a bottom-up fashion where individuals or small groups first learn smaller tasks, then how to compose and coordinate them to solve larger tasks. When the problem is of this type and we are already using a biologically motivated method such as GP, it seems very natural to use a bottom-up decomposition of the problem that simulates this aspect of human learning and allows GP to learn each of the smaller problems.

Table 1 is a variant of the table found in [SV00a], which we have adapted to correlate each prerequisite of layered learning with a property of genetic programming for keep-away soccer.

Table 1: Requirements for using layered learning and GP keep-away soccer justifications.

Layered Learning	**Genetic Programming**
1. Learning from raw input is not tractable	Complex MAS problems for GP need to be defined at multiple levels ✓
2. A bottom-up decomposition is given	MAS learning task is compositional ✓
3. Learning occurs independently at each level	GP can be applied to each layer independently ✓
4. The output of one layer feeds the next layer's input	The population in the last generation of one layer is used as the next layer's initial population ✓

When we modify standard GP for layered learning, we need to develop a learning objective for each layer, i.e., the fitness at each layer that selects ideal individuals *for the subtask*. As seen in [Lu98], using a single-objective fitness value often leads to the best performance, and is much easier than trying to define multi-objective fitness functions. While multi-objective fitness functions should allow GP to evolve more complex behaviors, it becomes more difficult to decide what the components of fitness should be and how important each one is to the solution. In preliminary experiments, we found that it was infeasible to develop either a set of Pareto optimization criteria or a weighted function over multiple objectives for keep-away soccer. Instead, we chose to focus on

automatically discovering how to compose *passing agents* into *keep-away soccer agents*.

Another issue we addressed for layered learning in GP is the transfer of the population from the last generation of previous layer to the initial population of the next. The ideal team will consist of individuals with high fitness on the coordinated MAS task. Meanwhile, in every population, there are certain individuals that have a better fitness than others. We might therefore consider copying that best individual only and seeding the entire initial population of the subsequent layer with it. However, this duplication removes the diversity that was evolved in the previous layer, which may be detrimental because the best individual on the subtask may be a suboptimal problem solver for the overall coordinated team activity. Thus, we designed two experiments using LLGP: one that duplicates the best individual and one that simply copies the entire population.

The final issue we address for LLGP is learning-speed improvement: to what degree can layered learning simplify the learning problem, allowing the target fitness to be reached faster than with standard GP? This increase in the slope of the learning-speed curve [Ka95] is to be distinguished from *speed-up learning*, wherein the efficiency of the learned problem solver is improved. We show how layered intermediate and team fitness objectives achieve greater learning-speed than a monolithic fitness objective in the keep-away soccer test bed. We also demonstrate a technique for empirically choosing a point at which to stop learning primitive MAS behaviors and switch to the high-level MAS behavior.

4 GP AND EXPERIMENT DESIGN

We designed four initial GP experiments to investigate and benchmark the performance of LLGP: standard GP (**SGP**), GP with ADFs (**ADFGP**), LLGP with the best individual duplicated to fill initial populations (**LLGP-Best**), and LLGP with the entire final population of the first layer used to seed the next (**LLGP-All**). SGP and ADFGP use the single *monolithic* (i.e., non-layered) fitness function of minimizing the number of turnovers that occur in a simulation. ADFGP allows each tree for kicking and moving to have two additional trees that represent ADFs, where the first ADF can call the second, and both have access to the full function set available for SGP. LLGP-Best and LLGP-All both have two layers; the fitness objective for the first layer is to maximize the number of accurate passes (a two-agent task evaluated over teams of three copies of the same individual, on the same size field as the keep-away soccer task), while fitness objective for the second layer is to minimize the number of turnovers.

We developed two variations on each experiment, with maximum generation values of 51 and 101. The stopping criterion for both variations is achieved when an ideal fitness measure of 0 (where fewer turnover turns are better) is found, or the maximum generation is reached. Our preliminary experiments indicated that a population size of 2000 yielded good results for the keep-away soccer domain using both SGP and ADFGP. We also found that the 101-generation SGP achieved better convergence in fitness and individual size and the 51-generation SGP, with negligible fitness improvement after 101 generations.

The genetic crossover operator generates 90 percent of the next generation; tournament selection generates the other 10 percent. [Ko92] The tournament size is 7, with maximum depth 17. Table 2 summarizes the terminal set used, consisting of vectors that are egocentric, or relative to the agent whose tree is being evaluated. Table 3 summarizes the function set used, where all functions operate on and return vectors. Both sets are similar to those used in [Lu98] and [AT99].

Table 2: Keep-away soccer terminals (egocentric vectors)

Terminal	**Description**
Defender	Vector to opponent
Mate1	Vector to first teammate
Mate2	Vector to second teammate
Ball	Vector to ball

Table 3: Keep-away soccer function set

Function (arguments)	**Description**
Rotate90(1)	Rotate current vector 90 degrees counter-clockwise
Random(1)	New random vector with magnitude between 0 and current value
Negate(1)	Reverse vector direction
Div2(1)	Divide vector magnitude by 2
Mult2(2)	Multiply vector magnitude by 2
VAdd(2)	Add two vectors
VSub(2)	Subtract two vectors
IFLTE(4)	if $\|\|\mathbf{v_1}\|\| < \|\|\mathbf{v_2}\|\|$ then $\mathbf{v_3}$ else $\mathbf{v_4}$

The GP system we use was developed by Luke and is called Evolutionary Computation in Java (ECJ) [Lu00]. The simulator we developed for keep-away soccer abstracts some of the low-level details of agents playing soccer from the *TeamBots* [Ba01] environment, which in turn abstracts low-level details from the *SoccerServer* [An98] environment. Abstractions of this type allow the keep-away soccer simulator to be incorporated later to learn strategies for the *TeamBots* environment and *SoccerServer*.

In *SoccerServer* and *TeamBots*, players push the ball to maintain possession. To kick the ball, the player needs to be within a certain distance. For keep-away soccer, we eliminate the need for low-level ball possession skills and allow offensive agents to have possession of the ball. Once an agent has possession, it can only lose possession by kicking the ball, i.e., by evaluating its kick tree. Because we use vectors that have direction and magnitude, this implementation would allow for dribbling actions to be learned, where the agent simply passes the ball a few units away. This abstraction greatly simplifies the problem and still allows for a wide range of behaviors to be learned.

At each simulation step that allows agents to act, if the agent has possession of the ball – i.e., the agent and ball occupy the same grid position – the agent's kick tree is evaluated. The kick tree evaluates to a vector that gives the direction and distance to kick the ball. Otherwise, the agent's move tree is evaluated. Both trees are composed of terminals listed in Table 2 and functions listed in Table 3.

For layered learning experiments, the first 5-50 percent of the maximum number of generations are spent in Layer 1 learning accurate passing without a defender present. To evaluate accurate passes, we count the number of passes that are made to a location within 3 grid units of another agent. The fitness function for this *intermediate objective* is then (200 – *passes*), where there are 200 time steps per simulation; a fitness of 0 is best and one of 200 is worst. The remaining 50-95 percent of the generations are spent in Layer 2 with a fitness value that is inversely proportional to the number of turnovers that occur with a defender present. This is the *team objective*. The defender uses a hand-coded strategy, based upon one of the standard *TeamBots* [Ba01] defensive agents, that always moves towards the ball to cause a turnover.

Each evaluation of an individual in the simulator takes 200 time steps, where the ball can move on each step, the defender moves on every other time step, and all offensive agents move together on every fourth time step. The initial configuration of the simulation places the defensive agent in the center of a 20-by-20 unit grid. The field is then partitioned into three sections: the top half and the bottom left and right quadrants. One offensive agent is placed randomly in each section, and the ball is placed a few units from one of the offensive agents, chosen at random.

Early runs of the system resulted in local optima being achieved; the most common of these was a control policy in which all offensive agents crowded the ball to prevent a defender from stealing it, causing turnover. To eliminate this "loophole", the defender, if blocked from the ball, can move *through* an offensive agent without the ball by simply trading places with the opponent if the two are adjacent on the grid.

5 RESULTS

Each experiment was run 10 times, and averages were taken across the runs. For all experiments, we achieved the best convergence behavior with 100 generations, so this was used as the baseline for SGP, ADFGP, and all LLGP variants.

Table 4 shows our initial experimental results. For ADFGP, Good-Average represents the average of the 10 best runs selected from among 20. ADFGP experiments converged to two clusters of fitnesses – one better than SGP, the other much worse. When we considered the individual size of the good cluster, we found that the poor cluster contains individuals with about half the number of nodes as individuals in the good cluster. Prefiltering ADFGP runs based upon individual size may be an appropriate remedy, but this is beyond the scope of this paper, as we are focusing on LLGP. We report both overall and good averages here, however, to show that LLGP can achieve performance as high as the good cluster's.

As shown in Table 4, our first LLGP experiment divided 101 generations into 40 for Layer 1 (successful pass criterion) and 61 for Layer 2 (minimum turnover criterion). Copy-Best represents the LLGP-Best seeding method for Layer 2; Copy-All, the LLGP-All method. These initial results did not indicate any notable advantage or disadvantage of LLGP, indicating only that we can obtain comparable solutions using LLGP-All, SGP, and ADFGP.

Table 4: Results for experiments with population size = 4000, max generations = 101, averaged over 10 runs. *Lower f (anti-fitness) values are better.*

	SGP	**ADFGP**		**LLGP, 40-61**	
		Avg.	**Good-Avg.**	**Copy-Best**	**Copy-All**
Best *f* gen. 101	11.25	19.67	8.75	23.71	12.67
Mean *f* gen. 101	66.89	60.21	64.27	82.03	64.64
Avg. ind. sz. gen. 101	228.74	113.25	123.07	161.71	171.40
First gen. *f* ≤ 20	33	62	22	101	55
Best *f* of run	9.0 ± 4.98	16.56 ± 17.45	6.83	19.29	9.0 ± 2.73

Table 5: Results for different Layer 1 durations (population size = 2000), averaged over 10 runs. *Lower f (anti-fitness) values are better.*

Layer 2 Start Generation	First Gen. $f \leq 20$	First Gen. $f \leq 15$	Best f Gen. 101	Best f of Run
5	62	79	12	11.75
10	18	30	11.4	9.7
15	24	43	9.63	9.38
20	38	47	9.6	9.6
25	47	80	11.88	11.25
30	51	58	14.1	12.7
35	46	55	7.75	7.25
40	57	82	13.6	13.2
45	67	93	14.11	13.11
50	62	82	11.5	11.1

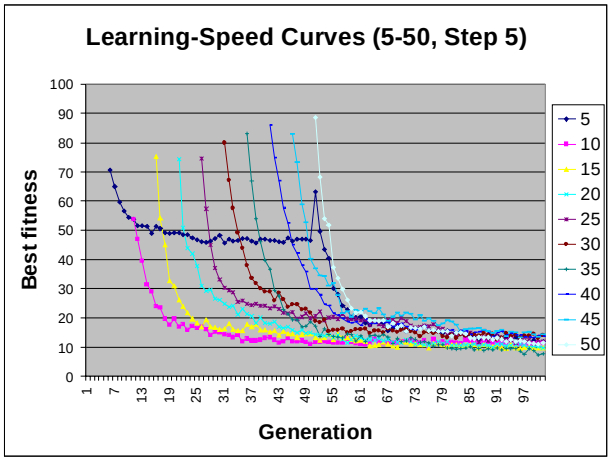

Figure 2. Layer 2 learning-speed curves (each starting at the end of Layer 1) for different Layer 1 durations. Lower is better.

We hypothesized that we were not yet realizing the full improvement in learning-speed that could be achieved using LLGP. To test this hypothesis, we plotted the Layer 2 learning-speed curves [Ka95] shown in Table 5 and Figure 1 for the following LLGP-All configurations: 5 generations in Layer 1 and 96 in Layer 2, 10 and 91, up to 50 and 51. The 10-91, 15-86, and 20-81 versions of LLGP-All achieve better convergence than those that start Layer 2 later, except for 35-66. Even accounting for the "early start", we can see that the convergence rate is faster and the final fitness is better for LLGP when Layer 1 lasts between 10 and 20 generations. We ran a second series of Layer 2 learning-speed curves (6 through 15, step 1) that indicated that the learning rates for 10 through 15 were not significantly different. We have not yet evaluated the inherent benefit to generalization quality – i.e., overfitting control and reusability – of stopping Layer 1 earlier, though this may be a good question for future experimentation.

A population size of 2000 is used for the fitness curves, as the performance for 2000 is similar to that for 4000, as reported in [Gu00]. Note that if the learning-speed curve for Layer 1 duration of 0 were plotted in Figure 2 above, it would be equivalent to that of the SGP, because the SGP runs for 101 generations with only the team objective function (Layer 2 fitness).

Table 6: Results for experiments with population size = 4000, max generations = 101, averaged over 10 runs. *Lower f (anti-fitness) values are better.*

	SGP	Good-ADFGP	LLGP-All, 10-91
Best f gen. 101	11.25	8.75	9.43
Mean f gen. 101	66.89	64.27	70.39
Avg. ind. sz. gen. 101	228.74	123.07	249.21
1st gen. $f \leq 20$	33	22	26
Best f of run	9.0 ± 4.98	6.83	*5.78 ± 2.28*

Having found that the 10-91 LLGP exhibited a better learning speed curve, we repeated the LLGP-All experiment with population size 4000 and found that it was able to match the Good-ADFGP performance, converged at least as quickly as any other GP, and resulted in the lowest best-of-run fitness values we found (fewer than 6 turnovers per simulation). This result is shown in Table 6, with the SGP and Good-ADFGP results repeated for comparison. We note that the Layer 2 individuals are much larger for LLGP-All-10-91 than for LLGP-All-40-61. That is, while stopping Layer 1 early yields a slight improvement in overall fitness and a significant improvement in learning-speed, it does not necessarily result in a more streamlined agent code. This is intuitive because more learning is deferred to Layer 2, where "passing" behavior is incorporated into the more sophisticated "keep-away" agents.

6 CONCLUSIONS

We have shown that using layered learning, genetic programming can evolve intelligent agents for a cooperative MAS task such as keep-away soccer more quickly, with better fitness. Additionally, layered learning GP allows for a natural decomposition of the MAS learning problem into subproblems, each of which is more easily solved with GP. The keep-away soccer problem is a good test bed for abstracting away the

complexities of simulated soccer and allows for different GP methods to be evaluated and their relative merits compared. It is also easily extended to the full game of robotic soccer, and is highly portable across platforms because our simulator, *TeamBots* [Ba01], *SoccerServer* [An99], and *ECJ* [Lu00] are all written in Java.

Conceptually, we can liken our success with LLGP to the success of human soccer teams. Successful teams are usually made up of players with unique strategies, where learning took place in a bottom-up fashion and individuals first learned to play well together in pairs and small groups, then as a coordinated team. The LLGP-All experiments simulate this kind of behavior, where we attempt to minimize the number of generations needed per layer. Our results indicate that layered learning in GP yields benefits over both standard GP and over hand-coded hierarchical approaches that depend on a large volume of domain knowledge. This is because it is easier and more natural to use the team fitness function to derive an intermediate fitness function, evolve primitive MAS agents, then let the higher-level (Layer 2) GP discover how to compose and refine primitive MAS behavior into complex MAS behavior.

We have considered several extensions to this research. First, developing a full-scale team for the *RoboCup* competition using LLGP would be a good way to test its abilities more thoroughly (however, the focus in this paper was on evaluating MAS task decomposition and improvement of learning accuracy and learning speed). Diversity in populations is also an interesting issue, and our continuing research in LLGP investigates how and whether LLGP promotes diversity. A related question is the degree to which LLGP *reuses* code versus *refining* it in higher layers. Other interesting modifications include developing heterogeneous teams, adding additional lower- and higher-level layers, and hybridizing ADFs and layered learning GP.

Acknowledgments

Support for this research was provided in part by the Army Research Lab under grant ARL-PET-IMT-KSU-07 and by the Office of Naval Research under grants N00014-00-1-0769 and N00014-01-1-0917. We also thank Edmund Burke for providing support for the second author in continuing this work. Finally, thanks to Sean Luke for providing help with ECJ, the GP library used to develop our system.

References

[An99] D. A. Andre. *SoccerServer Manual Ver. 4, Rev. 02*. Available through the World-Wide Web at http://www.robocup.org, 1999.

[As99] M. Asada. Overview of RoboCup-98. In *RoboCup-98: Robot Soccer World Cup II (Lecture Notes in Artificial Intelligence Vol. 1604)*. Springer-Verlag, New York, NY, 1999.

[AT99] D. A. Andre and A. Teller. Evolving Team Darwin United. In *RoboCup-98: Robot Soccer World Cup II (Lecture Notes in Artificial Intelligence Vol. 1604)*. Springer-Verlag, New York, NY, 1999.

[Ba01] T. Balch. *TeamBots* software and documentation. Available through the World-Wide Web at http://www.teambots.org, 2001.

[De90] H. deGaris. Genetic Programming: Building Artificial Nervous Systems Using Genetically Programmed Neural Network Modules". In B. W. Porter *et al*, editors, *Proceedings of the Seventh International Conference on Machine Learning (ICML-90)*, p. 132-139, 1990.

[DC97] M. Dorigo and M. Colombetti. *Robot Shaping: An Experiment in Behavior Engineering.* MIT Press/Bradford Books, 1997.

[Er00] R.I. Eriksson. An initial analysis of the ability of learning to maintain diversity during incremental evolution. In A. A. Freitas, editor, *Data Mining with Evolutionary Algorithms*, p. 120-124. 2000.

[HHC94] I. Harvey, P. Husbands, and D. Cliff. Seeing the light: artificial evolution, real vision. In D. Cliff et al, editors, *From Animals to Animats 3: Proceedings of the Third International Conference on Simulation of Adaptive Behavior*. MIT Press/Bradford Books, Boston MA, 1994.

[GH01] S. M. Gustafson and W. H. Hsu. Layered Learning in Genetic Programming for A Cooperative Robot Soccer Problem. In J. F. Miller *et al*, editors, Proceedings of the European Conference on Genetic Programming (EuroGP-2001). Lake Como, Italy. Springer-Verlag, 2001.

[Ka95] C. M. Kadie. *Seer: Maximum Likelihood Regression for Learning-Speed Curves.* Ph.D. Dissertation, University of Illinois at Urbana-Champaign (Technical Report UIUC-DCS-R1874). August, 1995.

[KAK+95] H. Kitano, M. Asada, Y. Kuniyoshi, I. Noda, and E. Osawa. RoboCup: The Robot World Cup Initiative. In *Proceedings of the 1995 International Joint Conference on Artificial Intelligence (IJCAI-95)* Workshop on Entertainment and AI/Alife. Montréal, Canada, 1995.

[Ki97] H. Kitano. The RoboCup Synthetic Agent Challenge 97. In Proceedings of the *1997 International Joint Conference on Artificial Intelligence (IJCAI-97)*, Nagoya , Japan, 1997.

[Ko92] J. R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection.* MIT Press, Cambridge, MA, 1992.

[Ko94] J. R. Koza. *Genetic Programming II: Automatic Discovery of Reusable Programs*. MIT Press, Cambridge, MA, 1994.

[Ko98] J. R. Koza. *Genetic Programming III: Darwinian Invention and Problem Solving*. Morgan Kaufmann, Los Altos, CA, 1998.

[LS96] S. Luke and L. Spector. Evolving Teamwork and Coordination with Genetic Programming. In *Genetic Programming 1996: Proceedings of the First Annual Conference*. J. Koza *et al*, eds. p. 141-149. MIT Press, Cambridge, MA, 1996.

[Lu98] S. Luke. Genetic Programming Produced Competitive Soccer Softbot Teams for RoboCup-97. In *Proceedings of the Third Annual Genetic Programming Conference (GP98)*. J. Koza *et al*, eds. p. 204-222. Morgan Kaufmann, Los Altos, CA, 1998.

[Lu00] S. Luke. *Issues in Scaling Genetic Programming: Breeding Strategies, Tree Generation, and Code Bloat*. Ph.D. Dissertation, Department of Computer Science, University of Maryland, College Park, MD, 2000.

[MNH97] H. Matsubara, I. Noda, K. Hiraku. Learning of Cooperative Actions in Multi-agent Systems: A Case Study of Pass Play in Soccer. In *Adaptation, Coevolution, and Learning in Multiagent Systems: Papers from the 1996 American Association for Artificial Intelligence (AAAI) Spring Symposium*, AAAI Technical Report SS-96-01, p. 63-67. AAAI Press, Menlo Park, CA, 1996.

[RB94] J. P. Rosca and D. H. Ballard. Hierarchical Self-Organization in Genetic Programming. In *Proceedings of the Eleventh International Conference on Machine Learning (ICML-94)*, p. 251-258. Morgan Kaufmann, Los Altos, CA, 1994.

[St00] P. Stone. *Layered Learning in Multiagent Systems: A Winning Approach to Robotic Soccer*. MIT Press, Cambridge, MA, 2000.

[SV98] P. Stone and M. Veloso. A Layered Approach to Learning Client Behaviors in the RoboCup Soccer Server. *Applied Artificial Intelligence (AAI) 12(3):165-188*. Taylor and Francis, London, UK, 1998.

[SV00a] P. Stone and M. Veloso. Layered Learning. In *Proceedings of the Eleventh European Conference on Artificial Intelligence (ECAI)*. 2000.

[SV00b] P. Stone and M. Veloso. Multiagent Systems: A Survey from a Machine Learning Perspective. *Autonomous Robots*, 8(3): 345-383. Kluwer Academic Publishers, Norwell, MA, 2000.

[SVR99] P. Stone, M. Veloso, and P. Riley. The CMUnited-98 Champion Simulator Team. In *RoboCup-98: Robot Soccer World Cup II (Lecture Notes in Artificial Intelligence Vol. 1604)*. Springer-Verlag, New York, NY, 1999.

[SX93] M. Schoenauer and S. Xanthakis. Constrained GA Optimization. In S. Forrest, editor, *Proceedings of the Fifth International Conference on Genetic Algorithms (ICGA-93)*, p. 573-580. Morgan Kaufmann, San Mateo, CA, 1993.

[Ta97] M. Tambe. Towards Flexible Teamwork. *Journal of Artificial Intelligence Research*, 7: 83-124, 1997.

[TAA+99] M. Tambe, J. Adibi, Y. Alonaizon, A. Erdem, G. Kaminka, S. Marsella, and I. Muslea. Building Agent Teams using an Explicit Teamwork Model and Learning, *Artificial Intelligence*, 110:215-240. Elsevier, 1999.

[WM98] J.F. Winkeler and B.S. Manjunath. Incremental Evolution in Genetic Programming. In J.R. Koza, editor, *Genetic Programming 1998: Proceedings of the Third Annual Conference*, p. 403-411. Morgan Kaufmann, San Mateo, CA, 1998.