Layered Learning in Genetic Programming for a Cooperative Robot Soccer Problem

Steven M. Gustafson[1] and William H. Hsu[2]

[1] ASAP Group, School of Computer Science & IT, University of Nottingham, Jubilee Campus, Nottingham, NG8 1BB, UK
smg@cs.nott.ac.uk
[2] Department of Computing and Information Sciences, Kansas State University, Manhattan, KS 66502, USA
bhsu@cis.ksu.edu

Abstract. We present an alternative to standard genetic programming (GP) that applies *layered learning* techniques to decompose a problem. GP is applied to subproblems sequentially, where the population in the last generation of a subproblem is used as the initial population of the next subproblem. This method is applied to evolve agents to play keep-away soccer, a subproblem of robotic soccer that requires cooperation among multiple agents in a dynnamic environment. The layered learning paradigm allows GP to evolve better solutions faster than standard GP. Results show that the layered learning GP outperforms standard GP by evolving a lower fitness faster and an overall better fitness. Results indicate a wide area of future research with layered learning in GP.

1 Introduction

For complex problems, such as robotic soccer [5][11], genetic programming (GP) may not be capable of finding a solution in its standard form. One reason is that the GP search space grows so large that it effectively leads to an intractable problem [17]. GP was previously used for robotic soccer to evolve teams of agents, but modifications were usually made to simplify the problem. Luke used GP to evolve high-level team strategies in [9], and Andre and Teller developed a fitness function based on human coaching principles of soccer in [1]. In a multiagent system (MAS) such as soccer, there is a definite hierarchy of behaviors that can be observed from human soccer. GP produces hierarchical programs by evolving and using automatically defined functions (ADF).

Koza gives a good description of ADFs and hierarchical programs in [7]. Hierarchical programs are evolved using ADFs to allow for code structure and code reuse. Rosca and Ballard, in [12], discuss ADFs and their importance in hierarchical programs. Although ADFs allow for the GP individuals to reuse evolved subprograms and develop solutions that have hierarchical code structure, they do not explicitly allow for *hierarchical learning* to take place. The distinction is that hierarchical learning describes a way in which behaviors are learned, not

necessarily in how the code that represents them is structured. While code reuse and program structure may help to overcome the inherent complexity of MAS problems, we suggest an approach for learning cooperative behaviors in a team-based MAS that is based upon primitive team objectives.

In many cases, teamwork can be made more tractable to learning, both in efficiency and in robustness of the performance element through a logical decomposition of the main problem [18]. For example, in the robotic soccer domain Stone and Veloso in [13] produce a very effective team of agents playing soccer by learning the overall task in a hierarchical manner. Tasks such as passing and kicking were learned before overall team strategies were learned. This technique of learning in layers is called *layered learning* and is formally described in [14]. It was applied with reinforcement learning for robotic soccer and the results indicated that layered learning may be a good adaptation for other machine learning methods such as GP.

To investigate layered learning in GP, the problem of learning keep-away soccer is chosen for its similarities to soccer and its properties of being a MAS problem. Also, keep-away soccer presents GP with a more reasonable search space than the full soccer problem and should allow for standard GP to find a solution so that comparisons can be made with the hybrid method of layered learning in GP.

Figure 1 is a screen capture of the visualization program used for the simulator built for keep-away soccer. The figure depicts the three offensive agents passing the ball in a counterclockwise motion (agent 3 passes the ball twice), with the trail of the ball denoted by the '-' character. The "*", "+", and ";" show the paths of other agents. The visualization was run for about 30 timesteps to collect the screen capture.

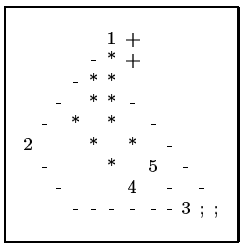

Fig. 1. Screen capture of simulator. 1,2, and 3 are offensive agents. 4 is the defender, and 5 is the ball. Ball moves from 3 to 1, 1 to 2, 2 to 3, and back towards 1.

Several variations exist for MAS and keep-away soccer can be categorized as multiagent learning with homogenous, noncommunicating agents [16]. This type of MAS problem requires robust solutions and is an interesting problem for research. A natural way to reduce complex, MAS problems, such as keep-away soccer, could prove to be useful for other MAS problems.

The keep-away soccer problem is described in Section 2, followed by a more detailed analysis of the application of layered learning to GP in Section 3. Section 4 describes experiments using an abstracted version of the TeamBots and SoccerServer robotic soccer simulator. Results are in Section 5 and research findings, conlusions, and future work follow.

2 Keep-Away Soccer

The RoboCup competition is an excellent testbed for MAS and are of interest to a wide variety of MAS research areas [4][3][19]. RoboCup competition occurs with real robots and in a simulation league, which presents several interesting challenges for researchers. Reinforcement learning, hierarchical sensing, neural networks, genetic programming, and a variety of hybrid combinations have been previously applied to the RoboCup simulation league [15][18] and real robot leagues. However, hand-coded and hybrid learning still outperform purely learned agent strategies. This poses a continuing challenge to researchers.

In keep-away soccer three offensive agents are located on a rectangular field with a ball and a defensive agent. The defensive agent is twice as fast moving as the offensive agents, and the ball can move, when passed, twice the speed of the defensive agent. This is similar to the predator-prey problem in [10] where more than one agent is required to solve the problem. The problem in keep-away soccer is to minimize the number of times the ball is turned over to the defender. A turnover occurs every time step that the defender is within one grid unit of the ball. Thus, the objective for offensive agents is to continously move and pass the ball to other offensive agents to keep the ball away from the defender and minimize turnovers.

Soccer, whether analyzing it as a human game or robotic game, can be broken down into subproblems of optimizing skills like ball control, passing, and moving. Keep-away soccer can be decomposed in the same manner. For the experiments here, we think of keep-away soccer as two layers of behaviors: passing accurately to other offensive agents with no defender agent present, and moving and passing with a defender to minimize the number of turnovers that occur in a game. The two layers of behaviors come from a human-like view of learning soccer, but are not heavily dependent on domain knowledge. These two types of behaviors are important to play good keep-away soccer, but these behaviors are not necessarily ways 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.

The layered learning application to GP is presented next, as we explain why keep-away soccer is a good test bed that illustrates its benefits.

3 Layered Learning

Applying the layered learning paradigm to a problem consists of breaking that problem into a bottom-up hierarchy of subproblems. When the subproblems are solved in order, where each previous subproblem's solution leads to the next

subproblem's solution, the original problem is eventually solved. This type of hierarchical solution is different than the hierarchical solution ADFs propose to find, which focus on code reuse and structure, not on how subtasks are learned.

Problems that attempt to simulate human behaviors, such as robotic soccer and keep-away soccer, lend themselves well to a bottom-up decomposition. The reason for this is because human learning usually occurs in a bottom-up fashion of first learning the smaller tasks needed to solve a larger task. In fact, when the problem is of this type and we are already using a biologically motivated method like GP, it seems very natural to use a bottom-up decomposition of the problem that simulates human learning and allows GP to learn each one of the smaller problems.

Table 1 is a modified version of the table found in [14]. Each key principle of layered learning is correlated with a property of genetic programming for keep-away soccer showing why the application of layered learning is natural and possibly beneficial to this type of problem and solution.

Table 1. Key principles of layered learning and the GP keep-away soccer correlation.

Layered Learning		**Genetic Programming**
1. Learning from raw input is not tractable	⇒	MAS problems for GP are complex problems
2. A bottom-up decomposition is given	⇒	Human-like learning problems have a natural bottom-up decomposition
3. Learning occurs independantly at each level	⇒	GP applied to each layer is independent
4. The output of one layer feeds the next layer's input	⇒	The population in the last generation of one layer is the next layer's initial population

When we modify standard GP for layered learning, we need to decide what the learning objective at each layer is, i.e., the fitness at each layer that drives the search for ideal individuals. As seen in [9], 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 multiobjective fitness functions should allow GP to evolve more complex behaviors, it becomes difficult to decide what the multiobjective fitness should be and how important each fitness is to the solution. If one fitness is clearly more important than another, it is necessary to decide to what proportion that fitness is more important. While using a multi-objective fitness is a possibility, using a singule-objective fitness seems logical for layers of a layered learning system that represents a decomposition of a larger problem.

The last issue to address for layered learning in GP is that of transferring the population of the last generation of the previous layer to the initial population of the next generation. Because the ideal team will consist of individuals with

high fitness on the coordinated MAS task, and in every population there are certain individuals that have a better fitness than others, we might want to copy that best individual many times to fill the initial population of the next layer. However, this duplication removes the diversity that was evolved from the previous layer, which seems counterintuitive, because the best individual may only be a suboptimal solution. Therefore, we propose two experiments with layered learning GP, one that duplicates the best individual and one that simply copies the entire population.

4 Experiments

Four initial experiments were chosen to investigate the performance of layered learning GP, standard GP (**SGP**), GP with ADFs (**ADFGP**), layered learning GP with the best individual duplicated to fill initial populations (**LL1GP**), and layered learning GP with the entire last population copied for the next initial population (**LL2GP**). SGP and ADFGP use the single fitness function of minimizing the number of turnovers that occur in a simulation. ADFGP allows each tree for kicking and moving to each have two additional trees that represent ADFs, where the first ADF can call the second ADF, and both have access to the full function set, as in SGP. LL1GP and LL2GP both have two layers, the first layer's fitness function is to maximize the number of accurate passes, and the second layer's fitness function is to minimize the number of turnovers.

Six variations of each experiment were developed that use standard GP parameter settings as described in [7] and vary the maximum generations allowed per run and the population size. Maximum generation values are 51 and 101, and population sizes of 1000, 2000, and 4000 are used. Six different runs are done for each type of experiment, SPG, ADFGP, LL1GP and LL2GP. The stopping criterion of each run is when an ideal fitness is found, a fitness equal to 0 or the maximum generation is reached. The genetic operators crossover and reproduction create 90 and 10 percent of the next generation, respectively. Tournament selection is used of size 7 with maximum depth 17. Table 2 summarizes the function set used and is similar to function sets used in [9] and [1]. Terminals are vectors, egocentric, or relative, to the agent whose tree is being evaluated, and all functions operate on and return vectors.

The GP system used was developed by Luke and is called Evolutionary Computation in Java [8] (ECJ). The simulator designed for keep-away soccer abstracts some of the low-level details of agents playing soccer from the TeamBots [20] environment, which abstracts low-level details from the SoccerServer [2]. Abstractions of this type would allow the keep-away soccer simulator to be incorporated later to learn strategies for the TeamBots environment and the SoccerServer.

In the 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 eleminate the need for low-level ball possession skills and allow offensive agents to have possession of the ball. Once an agent has possession,

possession is only lost when the ball is kicked, according to the evaluation of the agent's kick tree. Because we used 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 allow agents to act, if the agent has possession of the ball (i.e. the agent is on top of the ball in the grid) the agent's kick tree is evaluated. The kick tree evaluates to a vector that gives direction and distance to kick the ball. Otherwise, the agent's move tree is evaluated. Table 2 gives the terminals, functions, and their description.

For layered learning experiments, 40 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 which are passed to a location that is within 3 grid units of another agent. The fitness is then $200 - passes$, 200 timesteps in a simulation and a fitness of 0 is best and 200 the worst. The remaining 60 percent of generations are spent in layer 2 with a fitness value based on the number of turnovers that occur with a defender present. The defender uses a hand coded stratey and always moves towards to the ball to cause a turnover.

Table 2. Keep-away soccer terminal (egocentric vectors) and function set

terminals	functions(args)	Description
defender	rotate90(1)	rotate current vector 90 degrees counter-clockwise
mate1	random(1)	new random magnitude between 0 and current value
mate2	negate(1)	negate vector magnitude
ball	div2(1)	divide vector magnitude by 2
	mult2(2)	multiply vector magnitude by 2
	vadd(2)	add two vectors
	vsub(2)	substract two vectors
	iflte(4)	if v1 < v2 then v3, else v4 (comparing magnitudes)

Each evaluation of an individual in the simulator takes 200 timesteps, where the ball can move on each step, the defender moves on every other step, and all offensive agents move together on every fourth timestep. The initial setup of the simulation places the defender agent in the middle of a 20 by 20 unit grid. The field is then partitioned into three sections, the top half and the bottom left and right halves. Offensive agents are placed randomly within those sections, one in each, 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 was all the offensive agents crowding the ball and preventing the defender from causing a turnover. To overcome this, the defender, if blocked from the ball, can move through an offensive agent without the ball by simply trading

places with the agent if the two are within one unit on the grid. Each experiment was run 10 times and averages were taken across those runs. Running on a 16-processor 400 MHz Sun Ultra-Enterprise 10000 machine, evaluation of one generation took approximately 2 seconds for population size of 1000 and approximately 4-8 seconds for population size of 4000.

5 Results

The simplifications made to the problem allow SGP to find keep-away soccer agents with good fitness. For all experiments, the parameter of 101 generations always showed the best convergence and lowest fitness. For the remainder of the paper, we consider only that parameter setup.

ADFGP experiments converged to two clusters of fitnesses, one being better than SGP, and the other much worse. When observations of the individual size are accounted for, it appears that the bad cluster contains individuals with about half the number of nodes as individuals in the good cluster. Prefiltering ADFGP runs based on individual size may be appropriate to remedy this, but since we are not explicity studying ADFGP, we still use the averages here as this is only a hypothesized explanation of ADFGP. LL1GP, with duplicating the best individual from the previous layer, did much worse than SGP and ADFGP. LL2GP was competitive with SGP and ADFGP, remembering that the initial 40 generations are spent in layer 1 learning accurate passing.

These results do not highlight a strength or weakness of layered learning for GP, except that we can get nearly the same solutions with LL2GP as with SGP and ADFGP. However, when we look at the learning curve for best fitness per generation of layer 1 in LL2GP, we notice that convergence takes place in about 15 generations and settles to the same value for the rest of the run. This hints that perhaps we do not gain anything from running for a total of 40 generations. Two new experiments are then developed to test this hypothesis.

New layered learning GP, **nLL2GP** and **n2LL2GP**, are exactly the same as LL2GP, except that for nLL2GP the first layer is only run for 20 generations, and the second is run for 81. For n2LL2GP, the first layer is run for 10 generations and the second for 91. Figure 3 shows the learning curves for the new experiment, n2LL2GP, with the fitness of the last generation labeled. Figure 2 shows the same learning curves for SGP and ADFGP. We see a steeper drop in fitness in n2LL2GP, layer 2, and a better resulting fitness. The nLL2GP experiment showed some improvement, but not as much as n2LL2GP.

The same learning curves for mean fitness show that n2LL2GP, nLL2GP, SGP and ADFGP all produce the same values, approximately, with n2LL2GP being slightly better. These results suggest that a natural breakdown of the problem into subproblems, where GP solves each of the subproblems, could allow for a better overall fitness and a speed up in the learning over SGP. The standard deviation of the several runs of SGP, ADFGP, nLL2GP, n2LL2GP were 4.98, 17.45, 2.73 and 2.28 respectively, showing good stability for n2LL2GP. Table 3 gives some interesting values across all experiments. Note that individual size is

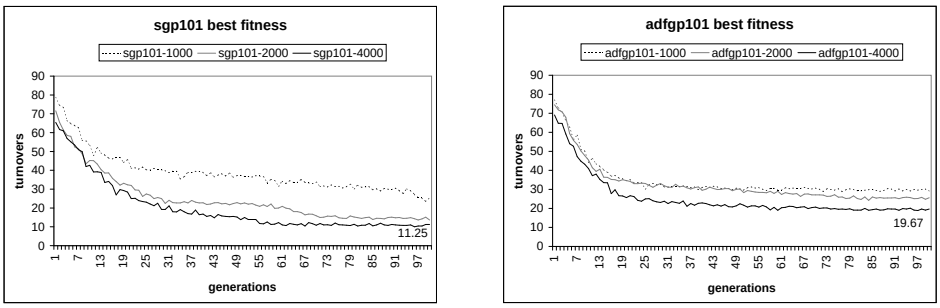

Fig. 2. Best fitness learning curve for SGP and ADFGP experiments

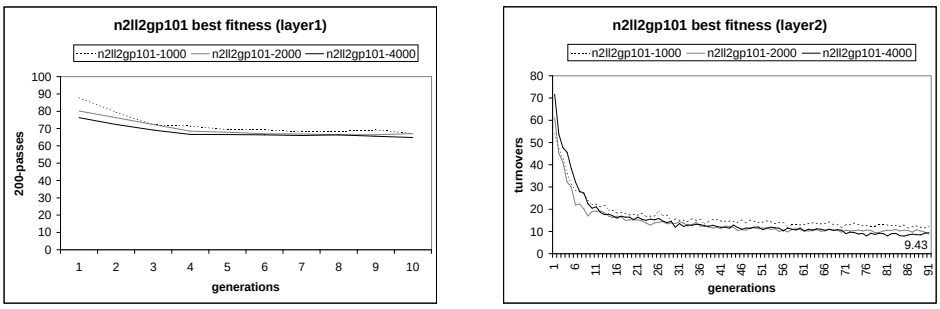

Fig. 3. Best fitness learning curve for n2LL2GP experiment. The left graph shows the learning of accurate passing in layer one, in 10 generations. The last generation is then transferred to layer 2, which is represented in the right graph where the minimizing of turnovers with a defender present is the objective.

the average number of nodes, where each node represents a function, terminal, or ADF call, of an individual in a generation. Figure 4 shows the learning curves for best fitness per generation and best generation so far in the run, where the n2LL2GP line only includes data for layer 2, omitting layer 1 that was attempting to maximize accurate passing.

Table 3. Data for experiments with population size=4000, max generations=101, and averaged over 10 runs. Good-ADFGP represents the average of the 10 best runs selected from 20 runs of ADFGP.

	SGP	ADFGP	Good ADFGP	LL1GP	LL2GP	nLL2GP	n2LL2GP
best fit.gen.101	11.25	19.67	8.75	23.71	12.67	9.67	9.43
mean fit.gen.101	66.89	60.21	64.27	82.03	64.64	74.78	70.39
ave.ind.size gen.101	228.74	113.25	123.07	161.71	171.40	217.36	249.21
1^{st} gen.fit.$\leq$ 20	33	62	22	101	55	31	26
best fit.run	9.0	16.56	6.83	19.29	9.0	7.32	5.78

Examining best-of-run individuals show the emergence of several behaviors, moving without the ball to avoid defenders, passing to open agents, and spreading out across the field, i.e. not crowding other agents or the ball. All the results highlight several other areas of interesting and worthwhile research with layered learning and keep-away soccer, and for other domains as well.

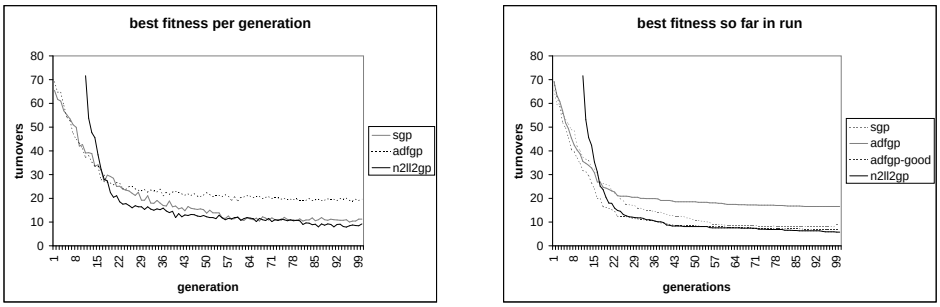

Fig. 4. The 'best fitness per generation graph' (left) compares the SGP, ADFGP and the n2LL2GP experiments for 101 generations and a population size of 4000. Because n2LL2GP spends the initial 10 generations learning accurate passing, only layer 2 is shown here, the layer for reducing the number of turnovers. The 'best fitness so far in run' (right) compares the same experiments with the addition of the Good-ADFGP experiment, the average of the best 10 out of 20 runs of ADFGP.

6 Conclusions

We showed that using layered learning for GP can evolve more fit individuals than standard GP. Additionally, layered learning GP allows for a natural decomposition of a large problem into subproblems. Each subproblem is then more easily solved with GP. The keep-away soccer problem is a good testbed for abstracting away the complexities of simulated soccer and allow for different GP methods to evolve good solutions for comparing methods. It is also an easily extended problem to the full game of soccer and transferred across platforms to other domains such as TeamBots and the SoccerServer from the simulator that was written here in ECJ.

Intuitively, we can liken our success with layered learning in GP with 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. The n2LL2GP experiment simulates this kind of behavior, where we attempt to minimize the number of generations needed per layer. The results indicate that it is beneficial to learn complex behaviors in a layered learning approach with GP, instead of standard GP, as it is easier to decide on fitness functions and natural to decompose the overall problem.

There are several extensions to this research that would be of interest. Developing a team for RoboCup competition using layered learning in GP would be a good way to test its ability more thoroughly. Studying other statistics about SGP, ADFGP, and n2LL2GP experiments could lead to other interesting conclusions, as would attempting to better optimize the number of generations needed in each layer. Diversity in populations is also an interesting issue, and whether layered learning promotes diversity. Other interesting modifications include developing heterogenous teams, adding additional lower and higher-level layers, and allowing ADFs in layered learning.

References

1. Andre, D., A. Teller. 1998. Evolving Team Darwin United. Lecture Notes in Artificial Intelligence: RoboCup-98: Robot Soccer World Cup II. vol 1604. Springer-Verlag.
2. Andre, D. et al. 1999. Soccerserver Manual. Ver. 4, Rev. 02. Available through the World-Wide Web at http://www.robocup.org.
3. Asada, M. et. al. 1999. Overview of RoboCup-98. Lecture Notes in Artificial Intelligence: RoboCup-98: Robot Soccer World Cup II. vol 1604. Springer-Verlag.
4. Kitano, H., Asada, M., Kuniyoshi, Y., Noda, I., Osawa, E. 1995. RoboCup: The Robot World Cup Initiative. In Proceedings of the IJCAI-95 Workshop on Entertainment and AI/ALife.
5. Kitano, H., et al. 1997. The RoboCup Synthetic Agent Challenge 97. In Proceedings of the IJCAI-97 Conference.
6. Koza, J.R. 1992. Genetic Programming: On the Programming of Computers by Means of Natural Selection. MIT Press.
7. Koza, J.R. 1994. Genetic Programming 2. MIT Press.

8. Luke, S. 2000. Issues in Scaling Genetic Programming: Breeding Strategies, Tree Generation, and Code Bloat. Ph.D. Dissertation, Department of Computer Science, University of Maryland, College Park, Maryland.
9. Luke, S. 1998. Genetic Programming Produced Competitive Soccer Softbot Teams for RoboCup97. In Proceedings of the Third Annual Genetic Programming Conference (GP98). J. Koza et al, eds. 204-222. San Fransisco: Morgan Kaufmann.
10. Luke, S. and L. Spector. 1996. Evolving Teamwork and Coordination with Genetic Programming. In Genetic Programming 1996: Proceedings of the First Annual Conference. J. Koza et al, eds. Cambridge: MIT Press. 141-149.
11. Matsubara, H., Noda, I., Hiraku, K. 1997. Learning of Coorperative actions in multi-agent systems: a case study of pass play in Soccer. In Adaption, Coevolution and Learning in Multiagent Systems: Papers from the 1996 AAAI Spring Symposium, pages 63-67, Menlo Park, CA. AAAI Press. AAAI Technical Report SS-96-01.
12. Rosca, J.P. and Ballard, D.H. 1994. Hierarchical Self-Orgainization in Genetic Programming. Proceedings of the Eleventh International Conference on Machine Learning. pp.251-258. Morgan Kaufmann Publishers, Inc.
13. Stone, P., Veloso M., Riley, P. 1999. The CMUnited-98 Champion Simulator Team. Lecture Notes in Artificial Intelligence: RoboCup-98: Robot Soccer World Cup II. vol 1604. Springer-Verlag.
14. Stone, P., Veloso, M. 2000. Layered Learning. Eleventh European Conference on Machine Learning (ECML-2000).
15. Stone, P., Veloso, M. 1998. A Layered Approch to Learning Client Behaviors in the RoboCup Soccer Server. Applied Artificial Intelligence (AAI), Volume 12.
16. Stone, P., Veloso, M. 2000. Multiagent Systems: A Survey from a Machine Learning Perspective. Autonomous Robots, volume 8, number 3.
17. Stone, P., Veloso, M. 1999. Team-Partitioned, Opaque-Transition Reinforcement Learning. Lecture Notes in Artificial Intelligence: RoboCup-98: Robot Soccer World Cup II. vol 1604. Springer-Verlag.
18. Tambe, M. 1997. Towards Flexible Teamwork. Journal of Artificial Intelligence Research, Volume 7, Pages 83-124.
19. Tambe, M., Adibi, J., Alonaizon, Y., Erdem, A., Kaminka, G., Marsella, S. and Muslea, I. 1999. Building agent teams using an explicit teamwork model and learning. Artificial Intelligence, volume 110, pages 215-240.
20. TeamBots software and documentation. Available through the World-Wide Web at http://www.teambots.org.