Empirical Comparison of Incremental Reuse Strategies in Genetic Programming for Keep-Away Soccer

William **H. Hsu, Scott J. Harmon, Edwin Rodríguez, and Christopher Zhong**

Department of Computing and Information Sciences
Kansas State University

Abstract. *Easy missions* approaches to machine learning seek to synthesize solutions for complex tasks from those for simpler ones. In genetic programming, this has been achieved by identifying goals and fitness functions for subproblems of the overall problem. Solutions evolved for these subproblems are then reused to speed up learning, either as automatically defined functions (ADFs) or by seeding a new GP population. Previous positive results using both approaches for learning in multi-agent systems (MAS) showed that incremental reuse using easy missions achieves comparable or better overall fitness than monolithic simple GP. A key unresolved issue dealt with hybrid reuse using ADF plus easy missions. Results in the keep-away soccer domain (a test bed for MAS learning) were also inconclusive on whether compactness-inducing reuse helped or hurt overall agent performance. In this paper, we compare monolithic (simple GP and GP with ADFs) and easy missions reuse to two types of GP learning systems with incremental reuse: GP/ADF hybrids with easy missions and single-mission incremental ADFs. As hypothesized, pure easy missions reuse achieves results competitive with the best hybrid approaches in this domain. We interpret this finding and suggest a theoretical approach to characterizing incremental reuse and code growth.

Introduction

One of the primary challenges to scalability of genetic programming is the evolution of primitive functions that are general enough to be reusable. In GP-based reinforcement learning [SV00a], this problem is related to the general one of discovering *intermediate representations* [FB85]. Koza found [Ko94] that in speedup learning, automatically-defined functions (ADFs) can be used to achieve reuse in GP – particularly more compact representations and fast evolution of complex solutions. Early research on characterizing reuse in ADFs was conducted primarily by manual or very basic automatic static analysis. For example, functions that could be seen to solve simpler subtasks were frequently evolved and reused by GP systems for symbolic regression problems $f(x) = 4x$ or $f(x) = x^4$ and for large multiplexer problems [Ko94]. In some domains, such as the multi-agent systems (MAS) intelligent control [SV00b] task we discuss here, the nature of reuse through ADFs is not clear.

One way to control reuse is through explicit specification of easier subtasks that the GP system is designed to acquire in stages. This method of *incremental* reuse works by migrating the individuals evolved to solve intermediate subtasks into a GP

for the harder overall problem. Similar approaches were independently explored by deGaris (as *behavioral memory* [De90]), Asada *et al*. (as *learning from easy missions* [ANTH98]), and Gustafson (as an adaptation of a hierarchical learning approach [St00, Gu00]). These have shown that policies can be evolved incrementally.

Previous work on reinforcement learning in homogeneous, multi-agent team problems [LS96] such as *keep-away soccer* [SS01, HG02] has demonstrated reuse in incremental evolution. For this test bed, the approach of learning from easier subtasks finds reusable solutions faster and more reliably than GP using ADFs with a *monolithic* task definition.

A key unresolved issue in previous work is why easy missions reuse achieve higher average fitness than ADFs in this domain. Gustafson [Gu00] found that slightly larger trees in the ADF-based solutions were significantly higher in fitness, but that the entire population tended to collapse into either a group of smaller trees ("Bad") or larger ones ("Good"). "Good" ADF cases achieved better fitness than the average for easy missions, but occurred in a minority of cases and could not be perfectly identified. This result also seems to contradict the hypothesis that limited tree size as induced by ADF reuse is beneficial in this domain and similar MAS domains.

In this paper, we therefore explore the "spectrum" from ADF-based reuse, which tends to induce compactness, to unrestricted easy missions reuse, which results in somewhat larger trees. We compare (1) monolithic GP with a single fitness criterion, with and without ADFs, against (2) the easy missions reuse approach previously shown to achieve better fitness [Gu00], (3) GP/ADF hybrids with easy missions, and (4) single-mission incremental ADFs. We refer to our adaptation of 2-3 as *GP-ISLES* (*Genetic Programming – Incrementally Staged Learning from Easier Subtasks*) and our implementation of 4 as monolithic, incremental ADFs. Just as ADFs provide reusable code and subroutine structure [Ko94], *GP-ISLES* provides a way to build solutions using a layered approach [St00, SV00a]. The difference between ADF learning and incremental approaches such as easy missions, using GPs or other methods, is that ADFs describe a way to implement structure in the agent representation while easy missions describes a way to train a learning intelligent agent. Both approaches achieve reuse, but ADF reuse is more automated in that it trades the ability for the GP designer to specify easy missions (intermediate fitness criteria) against the potential of discovering more generic, reusable ADFs.

Incremental evolution can be used to break down MAS learning tasks by first evolving solutions for smaller fitness cases [ANTH98], for smaller groups of agents, and for agents with a more primitive fitness criterion [SS01]. In our test bed, we focus on developing a fitness criterion for training agents to play on a team of 3 players based upon their cooperative performance in an easier subtask (passing between 2 players [MNH97]) with no adversary. The evolved individuals are used to seed a population of agents to be further improved. This new population and the associated GP form the second layer of an incremental system. The product of *GP-ISLES* is an agent that is evolved using highly fit primitive agents for the easier subtask as raw material. The overall solutions, however, typically contain material from these agents that has been crossed over and mutated within subroutines. By contrast, GPs using ADFs produce primitive anonymous functions (i.e., macros). They may

replace them within individuals or discard them from the population, but ADF trees are held apart. Their initialization and crossover are strictly controlled and their arities are pre-determined.

This paper describes experiments with new hybrid variants of *GP-ISLES* in the *keep-away soccer* domain. The problem is to better understand reuse in ADFs and in incremental approaches, both monolithic (single mission-based) and easy mission-based. Our approach was to compare speedup learning among the variants, using simple GP and monolithic ADFs experimental controls and the 3-on-1 keep-away task as the benchmark. Our hypothesis was that hybrid approaches would outperform the controls and approach the performance of non-ADF *GP-ISLES* (called *layered learning GP* in [HG02]). The purpose of these comparisons was to characterize the impact of different types of reuse on solution quality and on code size.

Reuse Strategies for GP

This section describes the MAS test bed, keep-away soccer. It then reviews the design and implementation of GP reuse strategies using ADFs, basic *GP-ISLES* with no ADFs, hybrid variants with ADFs, and incremental ADFs. It also reviews findings from the literature using basic *GP-ISLES*, which are extended in this work.

Multi-Agent Learning Test Bed

k-versus-*t* (*k*-on-*t* or *kvt*) *keep-away soccer* is the subproblem of robotic soccer of keeping the ball away from *t* players called *takers* who are attempting to capture it from *k* players called *keepers*. We chose keep-away soccer as an MAS learning test bed because it can be easily abstracted [SSS00, HG02], is scalable [SS01], captures a compositional element of teamwork [LS96], admits a symmetric (zero-sum) reinforcement for takers and keepers, and has adjustable opponent difficulty.

Incremental learning 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 compositionality of behaviors involving larger subteams. For example, a low-level primitive in soccer is *passing* the ball, an activity among any number of keepers for which a single episode can be isolated to two keepers at a time. Passing is incorporated into several multi-agent activities: guarding the ball; moving the ball downfield; setting up for a goal attempt; etc. In this paper, we shall explore GP solutions with reuse (*ADF* and *GP-ISLES*) for multi-agent systems (MAS) problems using the 3-on-1 keep-away problem using homogeneous (identical) keeper policies.

Despite the compositionality of 3-on-1 keep-away, learning to pass the ball effectively is only part of the GP application. In real soccer, human keepers learn to minimize the number of turnovers to the taker 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 keepers 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 at least one open passing lane.

Evolutionary Computation in Java (ECJ) and Simulator

All GP experiments were all conducted using Luke's *Evolutionary Computation in Java (ECJ)* package [Lu02]. A set of operators was developed in Java for the 3-on-1 keep-away task and is described in the Experiment Design section.

Where specified, *ECJ* defaults [Lu02] were overridden. These overrides, in turn, follow Gustafson's original implementation using an earlier version of *ECJ* [Gu00]. All variations (simple GP, ADF-GP, GP-ISLES, and incremental ADF-GP) use ramped half-and-half initialization, tournament selection with tournament size 7. The genetic crossover operator generates 90 percent of the next generation; tournament selection generates the other 10 percent. [Ko92] The GP variations use no mutation, permutation, over-selection, or elitism.

Fitness evaluations are made using Gustafson's 20-by-20 grid-based abstract simulator for keep-away soccer [Gu00]. Previous work by Stone and Sutton [SS01] and by Hsu and Gustafson [HG02] on the 3-on-1 task defined minimization of *turnovers* (change in possession) as the objective function for the full keeper policy. Let us define this problem specification as *3-on-1-turnovers* and easier subtasks, based upon the number of passes completed, as *k-on-t-passing* (for $k \leq 3$, $t \leq 1$).

For standardization, all runs use a generational GP with population size 4000 and 101 generations. Experiments with population size 1000, 2000, 4000, and 8000 and with fewer (51) and more (201) generations showed the above parameters to be effective for this test bed, as Gustafson also reports [Gu00].

Monolithic Simple GP and ADF-GP

As a baseline for comparison, we used SGP and ADF-GP with the single *monolithic*, or non-incremental, objective of minimizing the number of turnovers that occur in a simulation.

The ADF-GP is initialized with maximum size 6 for initial random programs. Hybrid variations also use this constraint, but have no restrictions on ADF seeding (as documented in the next sections). ADF-GP 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.

The next three sections describe incremental reuse: first using easy missions (*GP-ISLES*), then using single-mission incremental ADFs.

Basic GP-ISLES and Related Work

A basic version of *GP-ISLES* is described by Gustafson and called *layered learning GP* (LLGP) [Gu00]. Hsu and Gustafson initially experimented with the *3-on-0-passing* subtask [HG02], training a GP system for $g_1 \in [0, 90]$ generations with the *passing* fitness measure, then transferring the entire resultant population to a GP for evolving 3-on-1-turnover solutions and training for $g_2 = 101 - g_1$ generations. Validation experiments testing different values of g_1 found 10 to be the best value for this implementation of 3-on-1 keep-away. By running these experiments at various scals, Hsu and Gustafson found that g_1 can be automatically controlled [HG02].

To modify standard GP for incrementally staged learning, we must develop a learning objective for each layer, i.e., the fitness at each layer that selects ideal individuals for the easier subtask. The *GP-ISLES* system focuses on automatically discovering how to compose *passing agents* into *keep-away soccer agents*. *GP-ISLES* has 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.

In comparing this incremental approach to the monolithic systems (using a simple GP and a GP with ADFs), Gustafson found that it outperformed both GP and ADF-GP on average, achieving a best-of-run fitness of 5.8 turnovers in a 200-time step simulation [Gu00]. In analyzing specific ADF-GP runs, however, he found that populations tended to contain trees in one of two clusters: some runs averaging less than 110 nodes, the other averaging more than 120. Populations of larger trees had a markedly better (**lower**) best-of-run fitness (6.8) compared to those of smaller ones (16.6), which underperformed even the simple GP (9.0).

This result left open the question of whether *post-filtering* solutions evolved using ADF-GP, on the basis of code size, could result in solutions that were competitive with the staged incremental approach. That is, the "larger" solutions found by the monolithic 3-on-1-turnover ADF were not distinguishable to a high degree of statistically significance from the solutions found by the best 3-on-0-passing-3-on-1-turnover incremental GP ($g_1 = 10$).

Hybrid GP-ISLES

The results from *GP-ISLES* posed the question of whether *pre-filtering* by code size could be used as a termination criterion for the ADF-GP, or in conjunction with a parsimony-based fitness criterion. In our reimplementation of the incremental system, *GP-ISLES*, we found that while tree sizes were larger (averaging 250 nodes versus 230 for simple GP and 110 for ADF), incremental reuse appeared to be more beneficial to overall fitness in the keep-away task.

Our hypothesis was that there was some limitation of compact reuse in ADFs that did not occur among the incremental solutions. Therefore, we observed not only mean fitness (on easy missions and on the overall task) but also variance in fitness. We designed ADF/simple GP hybrids and relate them to stages of solution development in *GP-ISLES*.

We implemented two hybrid versions of *GP-ISLES*:

1. **GP-to-ADF**: A simple GP in the first layer, with results encapsulated into ADFs and random trees generated in the second layer
2. **ADF-to-GP**: A GP with ADFs in the first layer, with ADF crossover and creation suppressed in the second layer
3. **ADF-to-ADF**: A GP with ADFs in both layers

Justification for these two variants is as follows:

1. **GP-to-ADF**: This type of reuse corresponds to delayed definition of macros (ADFs) in speedup learning. That is, individuals – candidate solutions to

the easy mission – are allowed to evolve freely the until they have reached some minimum level of reusable complexity. We expected this approach to yield results similar to **GP-to-GP** (the basic *GP-ISLES* technique), but with potentially more compact individuals due to late reuse.

2. **ADF-to-GP**: This type of reuse corresponds to seeding a *GP-ISLES* population with solutions to the easy mission that may contain ADFs. We hypothesized that this approach makes a worse tradeoff than **GP-to-ADF** because it commits to the ADFs early (possibly *too* early) and then does not have the means to perform separate crossover on the ADF trees.
3. **ADF-to-ADF**: This type of reuse corresponds to starting with easy missions using a GP with ADFs and seeding into a GP with the overall criterion. We hypothesized that the fitness for this approach would be comparable to or worse than that for the monolithic, non-incremental ADF.

All variations were implemented by using command-line overrides of the *ECJ* parameters (configuration file). In each csae, we used a 10-generation GP for the first layer with *3-on-0-passing* as the easy mission (fitness criterion), and a 91-generation GP for the second layer with *3-on-1-turnover* as the fitness criterion. *3-on-0-passing* is implemented by switching the taker off and measuring the number of successful passes subject to zone constraints (keepers must maintain distance or the ball position will be reset). This is similar to the *simultaneous attraction and repulsion (STAR)* fitness measure developed by Stone and Veloso [SV98].

In all experiments, we migrated the full population (4000 individuals). Gustafson reports [Gu00] that this approach achieved better results than seeding the best-of-run individual throughout the second stage, by preserving diversity. Our preliminary experiments confirmed that full migration consistently outperforms best-of-run individual migration.

Monolithic Incremental ADF-GP

Two final variants we implemented are the **monolithic, ADF-to-ADF and GP-to-ADF incremental GP**. In both of these versions, only the *3-on-1-turnover* target is used, but population seeding occurs. We expected that results for this approach would also be comparable to monolithic, non-incremental ADF, though the early packaging of subroutines into first-layer ADFs may pose a tradeoff.

Experiment Design

The main objective of this work was to identify the best hybrid incremental reuse strategies for GP in the MAS domain of keep-away soccer. Furthermore, we sought to derive a plausible explanation for the performance of the basic *GP-ISLES* versus ADF-GP, both non-incremental and incremental. The long-term purpose of this research is to move towards a more general theory of incremental reuse by ADFs versus seeding, that accounts for code size and code growth.

Our approach was to compare the three hybrid versions above with our reimplementation of Gustafson and Hsu's basic *GP-ISLES* [Gu00, HG02]. The new approaches were predicted to perform better than or equal to ADF-GP, with the better

hybrid strategies achieving equal or better overall fitness than basic *GP-ISLES*. We sought to explain this phenomenon by comparing the fitness curves of all stategies and examining them in the context of descriptive statistics, such as average tree size, that are related to reuse.

We adapted the grid-based simulation developed by Gustafson [Gu00] for keep-away soccer, which abstracts some of the low-level details of agents playing soccer from the *TeamBots* [Ba01] and *SoccerServer* [An98] environments. Abstractions of this type allow the keep-away soccer simulator to be incorporated later to learn strategies for more fine-grained environments. 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.

Table 1: Terminals (egocentric vectors)

Terminal	Description
Defender	Opponent (taker)
Mate1	First teammate
Mate2	Second teammate
Ball	Ball

Table 2: Keep-away soccer function set

Func (arity)	Description
Rotate90(1)	Current vector 90° counter-clockwise
Random(1)	New vector $\in [0, arg]$
Negate(1)	Reverse direction
Div2(1)	Divide magnitude by 2
Mult2(1)	Multiply 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 $\mathbf{v_4}$

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 1 and functions listed in Table 2.

For *GP-ISLES* experiments, the first 10 percent[1] 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

[1] Hsu and Gustafson report a 10% / 90% division of generations between stages (with equal population size) to achieve the best results [Hs02].

fitness of 0 is best and one of 200 is worst. The remaining 90 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.

Table 1 summarizes the terminal set used, consisting of vectors that are egocentric, or relative to the agent whose tree is being evaluated. Table 2 summarizes the function set used, where all functions operate on and return vectors. Both sets are similar to those used in [Lu98] and [AT99].

Results

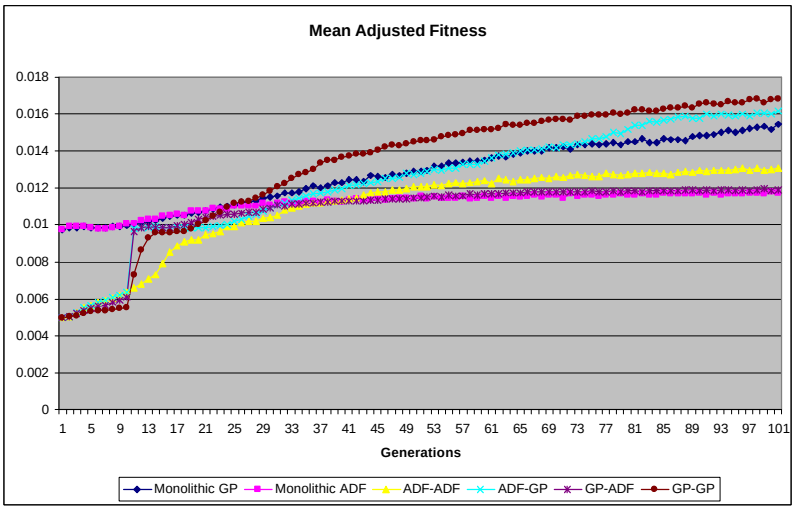

Figure 1. Mean adjusted fitness (10 replications) – *higher is better*

Each experiment was replicated 10 times, with the mean and variance taken across these runs. Table 3 reports experimental results on 20-by-20 keep-away task for monolithic GP and ADF-GP, the four *GP-ISLES* variants (three hybrid and one basic[2] with no ADFs), and GP-to-ADF. For 6 of the 8 methods, the mean adjusted

[2] For comparison and context, we also experimented with multi-deme parallel GP-ISLES (implemented by concatenating `.individuals` files from *ECJ* [Lu02]), where

fitness curve is plotted in Figure 1; the tree size, in Figure 2; and the max tree depth, in Figure 3. (All of these are mean values over 10 replications.) The monolithic incremental statistics are similar to that for GP-to-ADF.

Table 3. Mean adjusted fitness, tree size, and max tree depth

Method	**Mean Adjusted Fitness**	**Mean Tree Size**	**Mean Tree Depth**
Basic *GP-ISLES*	0.0178 ± 0.004	201.1	14.1
ADF-to-GP	0.0162 ± 0.004	165.1	13.4
Monolithic GP	0.0155 ± 0.003	174.3	13.7
ADF-to-ADF	0.0131 ± 0.003	113.1	7.5
GP-to-ADF	0.0119 ± 0.001	73.2	7.0
Mon ADF	0.0117 ± 0.001	52.5	5.7

Having found that the 10-91 *GP-ISLES* exhibited a better learning speed curve, Hsu and Gustafson [HG02] repeated the incrementally staged experiment with population size 4000 and found that it was able to match the best 10 of 20 runs of ADF performance, converged at least as quickly as any other GP, and resulted in lower mean-of-run and best-of-run fitness values found (fewer than 6 turnovers per simulation). We found that the mean adjusted fitness was ranked as follows, in best-to-worst order: basic ***GP-ISLES***, **ADF-to-GP**, monolithic GP, **ADF-to-ADF**, **GP-to-ADF**, monolithic ADF-to-ADF, monolithic GP-to-ADF, and monolithic ADF. **Bold** methods are *GP-ISLES* variants and underlined ones are incremental. The best-of-run fitness was achieved by **ADF-to-GP** (best adjusted fitness 0.073 ± 0.038) vs. monolithic GP (0.071 ± 0.045), but the confidence intervals for the **best** of run individual overlap greatly, so they are not significantly distinguishable on that basis in the above experiments. However, the last four variants in the above list have standard deviation less than 0.003 and are thus significantly outperformed by the first four. Standard deviation for **mean** adjusted fitness is much lower (0.004 or lower), as shown in Table 3, and so the ordinal ranking of ***GP-ISLES***, **ADF-to-GP**, monolithic GP (the three best performers) is better supported by the data, though the confidence intervals still overlap slightly.[3]

More important than the minor improvement over basic *GP-ISLES* is the empirical property we observe in Figure 2: that ADF-to-GP nearly tracks monolithic GP in code size and has smaller trees than basic *GP-ISLES.* In this sense, it is a "best of

---

the tasks in layer were either monolithic (3-on-1-turnover with two demes in the first layer) or *GP-ISLES* (multi-deme 3-on-0-passing). The monolithic multi-deme parallel GP resulted in significantly worse performance than either SGP or *GP-ISLES*. Slightly better results are obtained using single-deme, three-layer *GP-ISLES* (3-on-0-passing, 3-on-1-passing, 3-on-1-turnover), but the improvement is not statistically significant. This minor improvement seems to be due to the relative simplicity of the passing-under-interference subtask compared to the full keep-away task.

[3] Continuing replications (in batches of 100) of these experiments, now in progress, are expected to reduce this sample variance significantly and corroborate the above finding.

both worlds" tradeoff, as it achieves slightly better fitness **and** slightly lower code size. We note that our hypothesis that it would be better to start with GP rather than ADFs in the easy mission is disconfirmed: in fact, ADFs in the second phase consistently perform worse. A likely interpretation is that this reflects more thorough and diverse exploration of the "hard mission" (turnover) search space for the by *GP-ISLES* than by ADF-GP. This explanation is consistent with the poorer performance by small-tree ADFs that both we and Gustafson [Gu00, HG02] found. Langdon and Poli note that *diffusion* is often a fitness-based driver of code growth [LP97], and the trends in tree size for all *GP-ISLES* variants indicate that this is a main cause of growth.

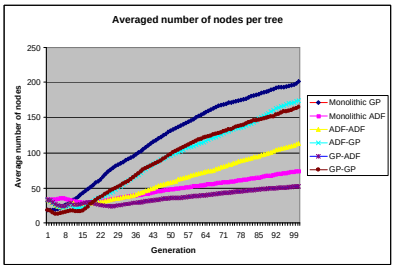

Figure 2. Average tree size (10 reps)

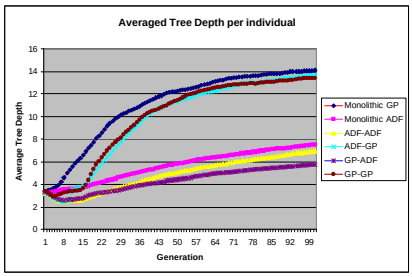

Figure 3. Max tree depth (10 reps)

Summary and Continuing Work

Using incrementally staged learning, genetic programming can evolve intelligent agents for a cooperative MAS task such as keep-away soccer more quickly, with better fitness. The keep-away soccer problem is a good test bed for abstracting away the complexities of simulated soccer and allows for different incremental 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 Gustafson's simulator [Gu00] and *ECJ* [Lu02] are both written in Java.

GP-ISLES allows for a natural decomposition of the MAS learning problem into easier subtasks. Experiments using four variants of *GP-ISLES* corroborated the result for basic *GP-ISLES* obtained by Hsu and Gustafson [HG02] and improved on them slightly using a hybrid variant, ADF-to-GP, that preserved compactness somewhat (as a tradeoff). This is an interesting characteristic of incremental reuse and it serves as another demonstration of how *compact reuse does not necessarily increase fitness* in incremental learning problems. We believe that constrained compactness can hurt fitness and that a formal theory for this phenomenon is needed.

For future investigation of this hypothesis, we are developing a system for visualizing code growth and especially reuse descriptors for ADFs and subtrees, in order to determine the effectiveness of reuse. This will help to assess scalability and to measure the usefulness of parsimony (as a means of limiting code growth) and em-

pirical termination criteria for evolution. Another potentially useful utility for *ECJ* is a validation system, using a Hoeffding race, for termination of evolution or selection of the migration point. A related question is the degree to which *GP-ISLES reuses* code versus *refining* it in higher layers. We believe that the code growth monitoring system provides a first step towards code-size-driven termination criteria and a general technique for assessing scalability. Our future work includes refining this tool and using it to *diagnose* code growth by classifying it among the types (intron hitchhiking, defense against crossover [NB95], removal bias [SF98], and diffusion [LP97]) surveyed by Luke [Lu00]. As mentioned above, we believe the code growth in *GP-ISLES* for MAS learning to be due primarily to diffusion rather than to other causes.

We have considered several extensions to this research. One of these is to developing a full-scale team for the *RoboCup* competition [Ki97] using *GP-ISLES*. While this could be a good way to test its abilities more thoroughly, the focus in this paper was on evaluating the scalability of the incremental solution, characterizing code size in ADF-GP and *GP-ISLES*, and design of hybrid incremental reuse. Results concerning incremental transfer are likely to transfer to other MAS learning domains besides keep-away soccer. We intend to look at other teamwork and coordination problems such as network optimization and insect colony simulations. Equally important is the problem of reuse in compositional problems such as symbolic regression and modular circuit synthesis. We hypothesize that easy missions approaches can generalize to some of these domains, as well.

References

[ANTH98] M. Asada, S. Noda, S. Tawaratsumida, and K. Hosoda. Purposive Behavior Acquisition for a Real Robot by Vision-Based Reinforcement Learning. *Machine Learning 23*:279-303, 1998.

[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.

[Gu00] S. M. Gustafson. *Layered Learning in Genetic Programming for A Cooperative Robot Soccer Problem.* M.S. thesis, Department of Computing and Information Sciences, Kansas State University, 2000.

[HG02] W. H. Hsu and S. M. Gustafson. Genetic Programming and Multi-Agent Layered Learning by Reinforcements. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2002)*, New York, NY, 2002.

[Ki97] H. Kitano. The RoboCup Synthetic Agent Challenge 97. In *Proceedings of the International Joint Conference on Artificial Intelligence*, 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.
[LP97] W. B. Langdon and R. Poli. Fitness causes bloat. In P. K. Chawdry, R. Roy, and K. R. Pant, eds., Soft Computing in Engineering Design and Manufacturing, p. 13-22. Springer-Verlag, Londok, UK, 1997.
[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.
[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.
[Lu02] S. Luke. *Evolutionary Computation in Java v*9. Available from URL: http://www.cs.umd.edu/projects/plus/ec/ecj/.
[Lu98] S. Luke. Genetic Programming Produced Competitive Soccer Softbot Teams for RoboCup-97. In *Proceedings of the 3rd Genetic Programming Conference (GP98).* J. Koza *et al*, eds. p. 204-222. Morgan Kaufmann, Los Altos, CA, 1998.
[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*, AAAI Technical Report SS-96-01, p. 63-67. AAAI Press, Menlo Park, CA, 1996.
[NB95] P. Nordin and W. Banzhaf. Complexity compression and evolution. In Proceedings of the Sixth International Conference on Genetic Algorithms (ICGA-95), p. 310-317. Morgan Kaufmann, Los Altos, CA, 1995.
[SS01] Scaling Reinforcement Learning toward RoboCup Soccer. In *Proceedings of the 18th International Conference on Machine Learning*, Williamstown, MA, 2001.
[SF98] T. Soule and J. A. Foster. Removal bias: a new cause of code growth in tree based evolutionary programming. In *Proceedings of the IEEE International Conference on Evolutionary Computation (ICEC-1998)*, p. 781-786. IEEE Press, 1998.
[SSS00] P. Stone, R. S. Sutton, and S. Singh. Reinforcement learning for 3 vs. 2 keepaway. In P. Stone, T. Balch, and G. Kraetszchmar, eds., *RoboCup-2000: Robot Soccer World Cup IV*. Springer-Verlag, Berlin, 2000.
[St00] P. Stone. *Layered Learning in Multiagent Systems: A Winning Approach to Robotic Soccer*. MIT Press, Cambridge, MA, 2000.
[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, 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.