GP+Echo+Subsumption = Improved Problem Solving

W.F. Punch

Computer Science and Engineering
Michigan State University
punch@cse.msu.edu
http://www.cse.msu.edu/~punch

W.M. Rand

Computer Science and Engineering
Michigan State University
randwill@msu.edu
http://www.msu.edu/~randwill

Abstract

Real-time, adaptive control is a difficult problem that can be addressed by EC architectures. We are interested in incorporating into an EC architecture some of the features that Holland's Echo architecture presents. Echo has been used to model everything from cultures to financial markets. However, the typical application of Echo is a simulation to observe the dynamics of the modeled elements such as found in control problems. We show in this paper that some aspects of Echo can be incorporated into Genetic Programming to solve control problems. The paper discusses EAGP (Echo Augmented Genetic Programming), a modified GP architecture that uses aspects of Echo, and subsumption. We demonstrate the usefulness of EAGP on a robot navigation problem.

1 BACKGROUND

1.1 ECHO

Echo, as described by Holland in the book Hidden Order [Holland95], is an umbrella architecture consisting of various models for experiments with complex adaptive systems (CAS). An Echo architecture has a number of independent "agents" that exist within a particular environment. The hallmark of an Echo simulation is the development of interactions between agents that allow them to thrive in the environment, creating complex communities of different types of agents. Interactions and agent types develop over time by allowing each agent to adapt to the environment and to the other agents in the community. To create an Echo model, Holland lists six criteria:

1. Simplicity: Echo "is meant for thought experiments rather than for emulation of real systems."
2. Geography: Agents in Echo should move within a "geography", where location "matters, in terms of the input an agent receives and how interactions with other agents occur.
3. Fitness: Fitness is not fixed externally but should depend on the context of the environment.
4. Mechanisms: The mechanisms in an Echo architecture should have counterparts in real CAS.
5. Frameworks: The developed model must allow easy insertion of other established CAS frameworks.
6. Analysis: The model should be amenable to mathematical analysis.

Holland also lists a set of properties and mechanisms that should be universally available in a CAS system that is listed in Table 1.

Table 1: The basic properties and mechanisms required for a CAS system

Properties	**Mechanisms**
Aggregation	Tags
Nonlinearity	Internal Models
Flows	Building Blocks
Diversity	

Most of these are self-explanatory. Flows involve the sharing of resources in the developed community of agents. Tags provide a means of identifying agents, and are the means for most interactions. Internal models are the "code" that drives the agent's interactions with the environment. Building blocks are the basic unit of heredity, the passing of useful information from parent to child, as found in GAs.

A number of Echo implementations exist. Gecko[Booth97] is an implementation of the Echo architecture specialized for modeling ecosystems. It focuses on spatial distribution and interactions as could be found in an ecology[Schmitz96]. Swarm[swarm] is a system for simulating multiple, interacting agents and has

been used in implement a wide variety of CAS systems, including Echo-like systems.

2 FOUNDATIONS OF EAGP

Most CAS and Echo-like systems focus on the simulation of multiple agents in a complex environment. Holland purposefully states in criteria 3 above that no fixed measure of fitness should be provided. The goal is not to solve problems so much as to provide an environment in which complex interactions can develop given certain conditions. However, it seems clear that a modified Echo model, or at least models that embody Echo properties, could indeed be used to solve problems, especially complex problems that have multiple, interacting aspects such as would be found in design or planning.

Our goal is to create a Echo Augmented Genetic Programming (EAGP) by starting with the basic Echo description and modifying only those aspects necessary to introduce problem-solving capabilities. Thus we wish to keep as many aspects of the original Echo approach as possible.

2.1 PRINCIPLES USED

Our goal will be to use some of the ideas presented in Echo. In particular we will use the concept of Tags, Aggregation, Internal Models and Building Blocks. In summary:

- By using Genetic Programming [Koza92] as the underlying architecture, we incorporate the concept of the building block as represented by the subtrees of the individual trees (Section 3.1).
- For aggregation, we allow an EAGP to consist of multiple individuals, each of which can be acquired over time from other EAGPs (Section 3.3).
- For internal models, we allow the EAGP to acquire those models that best optimize the problem-solver's behavior (Section 2.3).
- For tags, we allow each EAGP to be identified based on the aspects of the fitness function that it may address (Section 3.2).

These will be explained in more detail in the indicated sections.

2.2 FITNESS

An Echo model and a genetic algorithm occupy the poles of a continuum when it comes to fitness evaluation. Holland's Echo provides no external measure of fitness. Each agent is provided with resources and the potential to gather and exchange those resources with the environment and other agents. Those agents that develop effective internal models for dealing with their environment will be successful in maintaining those resources and therefore thrive and propagate their success via their building blocks to their children. A traditional GA has a specific, external fitness function that directly measures the success of any GA solution. Those solutions that perform "better" as measured by the fitness function have a better chance of propagating their building blocks to the next generation.

EAGP must occupy a midpoint between these two poles. To be an effective problem-solving system we must provide some measure of success in order to foster strategies that better solve the problem. However, a monolithic measure of success such as a GA fitness function does not as easily foster the development of interaction between agents, the hallmark of an Echo system. EAGP therefore requires multiple fitness criteria that can be independently measured. Furthermore, these independent criteria can be combined to give higher levels of fitness to agents that can solve more than one aspect.

Consider a simple example shown in Figure 1, based roughly on the "ant" problem of Koza[Koza92]. A robot must be trained to navigate the above environment. The robot has three general goals:

- avoid hitting any walls
- avoid the light
- pick up any available food

The scoring of an agent's performance in this environment is based strictly on how well they solve each goal with an equal weight given to each task. The scoring of the problem-solver is based on how well it solves all of the goals found in the fitness function based on the performance of the agents it runs.

2.3 PROBLEM-SOLVERS AS GROUPS OF INTERACTING AGENTS, A SUBSUMPTION VIEWPOINT

Creating a multi-part fitness function does not in itself create an environment that fosters interaction of multiple agents to solve problems. We must also have multiple, independent agents that have the possibility of interacting with each other, where their fitness depends fundamentally on these interactions. We accomplish this by defining an EAGP **problem-solver** as a group of **agents**, each of which can address any or all parts of the fitness function. Those agents that can also work together, i.e. interact, can solve more of the overall problem and the overall problem-solver can accomplish more of the aspects of the multi-valued fitness function.

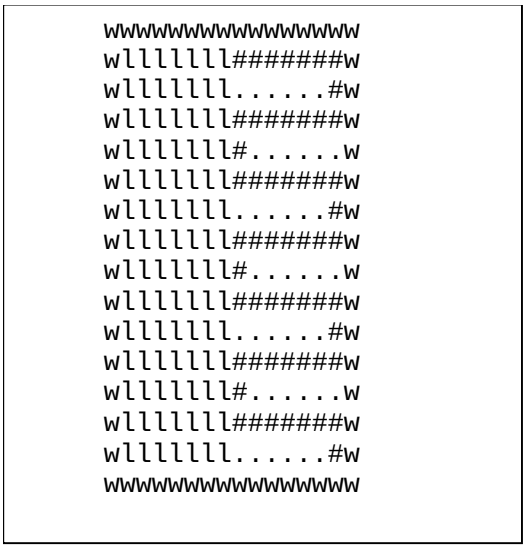

Figure 1. A robot path: w= wall, l=light, #=food, .=dark. In this case, all food is in the dark.

This approach is inspired by the subsumption architecture of Brooks[Brooks91,Brooks97]. The standard subsumption architecture defines multiple layers of simple problem solving agents, often represented by Finite State Machines (FSMs). The organization of these simple problem-solvers (FSMs) is based on a kind of priority hierarchy created for the subsumption architecture by the programmer. Lower level behaviors are closer to the hardware (actuators of robots for example) and can over-ride higher behaviors since the lower behaviors typically drive "survival" kinds of operations (predator avoidance, cliff avoidance, etc.). Higher level functions can come into play by driving the lower level behaviors based on an **implicit** representation of higher goals (food gathering, community activities). Thus multiple agents can examine the present environment and each try to act according to its own local model. The interaction of the agents, based on the assigned priorities, determines the overall behavior of the entity.

We incorporate, with some modifications, subsumption principles into the EAGP approach. First, we define a problem-solver as a group of interacting agents that, while working together, can solve a problem that no agent itself can fully solve. Second, we wish to limit the computational complexity of any agent. This both encourages the interactions mentioned above and makes any part of the resulting solution more understandable. Our philosophy is that the complexity of the solution should arise from the interactions of the involved agents, not the complexity of the agents themselves. Third, the interaction of the agents is provided by their organization in the problem-solver. Unlike subsumption however, we have the freedom to evolve not only the agents, but also their means of interaction. Thus the interaction need not be fixed at the time of the design of the problem-solver, but can evolve along with the agents in that problem-solver. Fourth, we do not limit the ability of any agent to evolve some measure of global state. Global state was avoided in the original subsumption work because Brooks wished to avoid model building. He wishes his systems to be reactive to the conditions of the environment directly. While his philosophy is understandable, a reaction to building non "real-world" systems in the traditional AI community, it does not obviate the usefulness of models themselves.

3 STRUCTURE OF EAGP

3.1 STRUCTURE

The foundation of an EAGP problem-solver is a modified genetic programming [Koza89, Koza92] One of Koza's extensions to basic GP is the Automatically Defined Function (ADF)[Koza94]. An ADF is a kind of subroutine to be used in many places within an overall GP. Its structure is shown in Figure 2.

The typical ADF-GP has a single Result Producing Branch (RPB) which constitutes a main program, and some number of ADF's. Execution begins in the RPB, which can subsequently call any of the ADFs. Each ADF-GP has the same number of ADFs, though what functions they consist of and how they are used via the RPB and other ADFs is individual. Crossover between ADF-GP solutions occurs on a one-to-one basis between its elements. Thus, RPB's crossover with RPBs, ADF1s with ADF1s etc.

We modify this structure to create an EAGP problem-solver. Instead of ADFs, each part of the EAGP problem-solver is an **agent**. These are individual programs that can address one or more of the multiple goals in the fitness function. They are **not** subroutines to be called by another part of the EAGP solution. They stand only as individual programs. **All** the agents in an EAGP problem-solver are executed individually when the problem-solver is evaluated. Each agent posts to the Arbitrator program the action it thinks it would take if it were given control. This posting includes a result of "do nothing" if the agent feels it is not appropriate in the present circumstances. The input data available to the agent is based on the functions it has evolved. Thus it may have access to all the input data, or some stored/abstracted data, or to no data at all depending on its structure.

The Arbitrator, once it receives all the agent responses, then decides which agent will gain control in the present circumstances. Alternately, the Arbitrator may decide based upon the present conditions which agent will gain control and then allow it to operate without examining other agents. The complexity of the Arbitrator is one of the variables in generating an EAGP system. It can range anywhere from a fixed priority schedule to a completely evolved program. However it chooses, the Arbitrator selects the agent action to perform, and performs it. Based on the action taken, the EAGP problem-solver is evaluated.

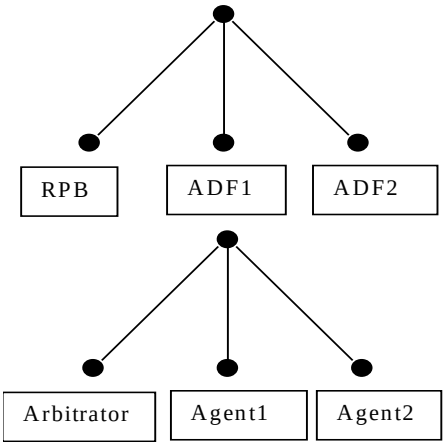

Figure 2. The upper tree represents a typical ADF-GP. The lower tree indicates the modifications for an EAGP problem-solver

There are a few other, important differences, between a ADF-GP system and an EAGP system. An EAGP problem-solver can have 1 or more agents, limited only by the initial setup of the system. Thus in the population of an EAGP system each problem-solver can have different numbers of active agents, unlike an ADF-GP where each GP solution has the same number of ADFs. We also purposefully limit the computational capabilities of each agent by severely limiting the agent's tree. We limit the depth and/or the number of nodes that the agent can contain.

3.2 TAGS

The normal operators of GP crossover and mutation are maintained in an EAGP system. However, because each EAGP problem-solver can have different numbers of agents, we cannot simply perform crossover between the problem-solvers on a one-to-one basis as in ADF-GP. Instead, we must identify "similar" agents in each problem-solver and initiate crossover between those agents. We do this by **tagging** each agent. Tagging occurs by evaluating each individual agent and identifying which aspect of the multi-part fitness function it best addresses. Continuing with our robot path example, if an agent scores well on wall avoidance, but not well on light-avoidance or food gathering, then we tag that agent as a "wall-avoider". Those agents can then crossover with other "wall-avoider" agents in other EAGP problem-solvers. This is an important concept because one of the fundamental mechanisms of an Echo-like system (or any GA derived EC approach) is the passing of building blocks between like entities in the population. Were the agents in an EAGP problem-solver not tagged, then we could not **appropriately** pass building blocks between agents and therefore could not take advantage of the principle so well proven in GAs.

3.3 OPERATORS

Besides crossover and mutation, we add new operators that allows **aggregation** of agents in an EAGP problem-solver. The **extend** operator adds a new agent *of a different tag* to the present EAGP problem-solver by copying that agent from a different problem-solver. It works as follows. Under the normal rules of crossover, two EAGP problem-solvers are selected. If one such problem-solver has an "unfilled tag", that is it does not yet contain an agent that addresses one of the aspects of the multi-valued fitness function, then that agent can copy an agent of the unfilled tag from the other problem-solver, if such an agent exists. The donating problem-solver is not affected by the donation, but it too is given an opportunity to extend itself if appropriate.

The **swap** operator plays a similar role. Two problem-solvers are selected under the current rules of crossover. If both problem-solvers have agents with the same tag, they can swap those agents. Unlike the extend operation, both problem-solvers are modified. As in crossover and mutation, the frequency of occurrence of these operations plays a role in the development of the problem-solving capabilities of an EAGP system.

4 IMPLEMENTATION AND TESTS

This version of EAGP was implemented as a highly modified version of the GP programming system, lilgp[lilgp]. Lilgp is a public domain, C based implementation of a GP that includes support for ADF, as well as multiple populations, threading and others features (see http://garage.cse.msu.edu/software/lilgp for more details). Lilgp is very modular, has proven relatively easy to modify, is well documented and has been used by a number of other researchers to extend GP research (constrained GP[lilgp-ct], strongly-typed GP[lilgp-st]). Thus we have not yet implemented a full EAGP system from scratch, something we hope to do in the near future. As of now, the EAGP system is relatively difficult to work with given the confounding of GP and EAGP in the implementation.

Each problem-solver has a maximum number of agents it can contain. For these experiments the number of aspects in the multi-valued fitness function determined the maximum. Each problem-solver begins with only one agent, though in fact the problem-solvers are initialized with the maximum number and those extra agents are essentially ignored until the EAGP system recognizes it (it is extended). The maximum tree depth for each agent was kept relatively low, only 7, so as to limit the computational abilities of each agent. The swap operator was not used in the experiments described below.

We tested the EAGP system on three subsumption-like problems involving robot navigation. The three problems are shown in Figure 3. The problems present a range of difficulty. The Dark trail requires only navigating to the dark area to find food. The Sidewinder requires moving to the dark area, but then must navigate a trail to find food.

The modified Santa Fe trail is based on the Santa Fe trail presented in the ant problem in Koza[Koza92]. We have added more walls and some light sources to make the problem more difficult and to have more aspects for the fitness function (light and wall).

Modified Santa Fe Trail

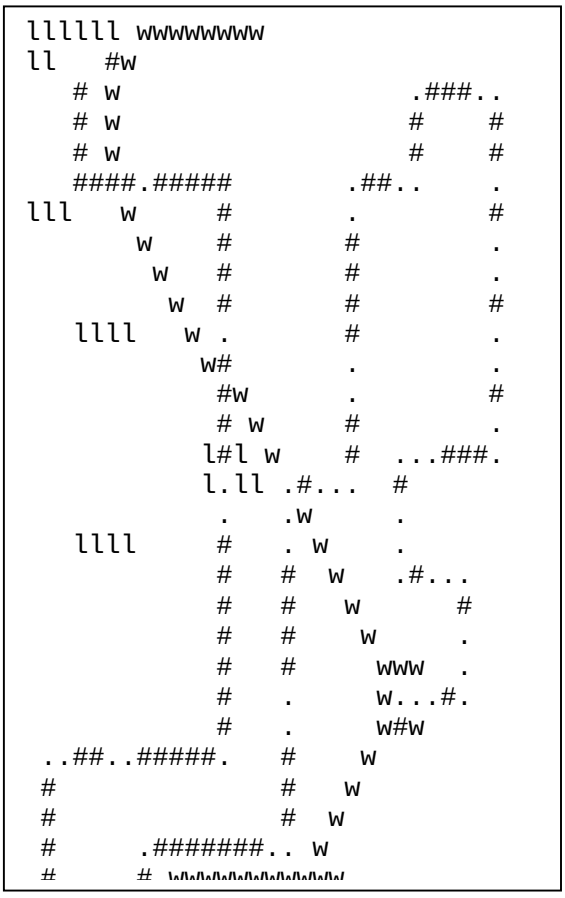

Figure 3. w=wall, l=light source, #=food. Darkness is assumed unless a light source is directly indicated. Thus the Dark trail and the Sidewinder trail all have their food in darkness. All problem-solvers start in the top-left corner.

The fitness measures were the ones mentioned before, namely wall-avoidance, light-avoidance and food-gathering. The fitness for wall-avoidance and light-avoidance was a percentage based on the number of times run and the number of time successfully run. Thus an 85% wall-avoidance means that 85% of the time, the wall-avoiding agent when run avoided a wall. The food-gathering evaluation was the percentage of food picked up versus the food available. The overall score was presented as a six-digit number. The first two digits are the wall-avoidance percentage, the middle two is the light-avoidance percentage and the last two are the food-gathering percentage. All evaluations had a time limit on the number of steps that could be taken (400).

5 RESULTS

We conducted experiments on all three trails listed in Figure 3, and compared the results to running an unmodified version of lilgp using ADFs. The conditions were kept exactly the same for the corresponding parameters (crossover %, tree depth max, etc.).

Our first experiments used a simple, fixed arbitrator. The three fitness function aspects were ordered, that is the problem-solver would choose an agent for execution whose tag matched the highest priority fitness aspect available. The order established was (in order): wall-avoidance, light-avoidance and food-gathering. If the agent returned the "do nothing" result, then the next agent in the priority list was run.

The reasoning for this was straightforward. We wanted the problem-solver to have a kind of "survival" mode so that it always dealt with problems in importance order. Thus it would be most important to not bang into walls and damage yourself, while avoiding light sources was also important so as to not exposure yourself to danger, while food gathering was something one could do when not threatened.

Table 2: The functions and terminals used for the subsumption problem.

Functions:

- If_food_ahead(true result)(false result)
- If_wall_ahead(true result)(false result)
- If_light_ahead(true result)(false result)
- If_light(true result)(false result)

Terminals:

- Do_nothing
- Move_forward
- Turn_left
- Turn_right

While well meaning, this proved to be a rather poor choice of arbitration. The reasons are obvious. The problem-solvers often evolved a wall-avoiding agent, and since this was the priority agent it was always run first. The hope was that this agent would "do nothing" when it didn't apply (there were no walls around), but in fact there was no evolutionary pressure in this model to induce this behavior. It was more "profitable" for the wall-avoidance agent to simply "turn left" in a circle, meaning that it never hit a wall (or rarely) and it always applied to

any situation, so it always got a very high score. Thus no other agents were ever run and no other behaviors ever induced

5.1 DYNAMIC PRIORITY ARBITRATION

We modified the priority arbitration in the following way. The fitness aspects were still prioritized as before, but each was also given a "reservoir" or cache of reward. Every action that an agent of a particular tag took would remove some value from the reservoir. Every time that an agent with a particular tag was in a situation where it **could** have acted then some value added was to the reservoir. The evaluation of this potential action was done externally in the fitness functions using the same functions that were available to the agent and determining if there was a wall, light or food nearby. The latter concept is the more difficult to understand. In the case of the wall-avoiding function, if the wall-avoiding agent acted, then value is removed from the reservoir. If that same agent was near a wall, value was put back in the reservoir.

The size of the reservoir would then modify the actions of the priority list. The agents were queried in priority order, but their chance of being run was based on the reservoir size. If the reservoir was very low, then the probability of running that agent was low, and vice versa. The result is a modification of the priority order based on present conditions. If the problem-solver is not hitting any walls but the wall-avoider is running, then running the wall-avoiding agent becomes less and less likely to be called in the future, giving other agents better opportunities to run. If hitting walls or approaching walls is a problem, then the wall-avoiding agents become more necessary and its action more likely in the future. Most importantly, each agent is now given some evolutionary pressure to act only when necessary, as doing so means it does not lose value in the reservoir.

5.2 PROBLEM SET RESULTS

We ran both EAGP and ADF-GP on all three robot navigation problems. All results represent the average based on a series of ten runs. Both EAGP and ADF-GP were able to solve the simple problems of the Sidewinder trail and Dark trail so the more interesting comparison comes form examining the results of the Santa Fe Trail. The results for the three fitness functions are shown in Table 3 below. These results are expressed as percentages as described above.

Table 3: Fitness for each fitness function for the modified Santa Fe Trail.

	Light	**Food**	**Wall**
ADF-GP	77.28	6.96	92.56
EAGP	84.87	16.18	97.07

It is clear that the EAGP performs well on simple as well as complex problems, though it is better used on more complex problems (such as modified Santa Fe) where it shows better performance. For instance, if you look at the percentage of wall and light that both versions avoided, they are fairly close but EAGP was able to score five percent higher than the ADF-GP. However, where EAGP truly does well is when gathering food. Here, the EAGP more than doubles the score of the ADF-GP. In order to avoid the wall and light the agent need only detect that it will head into the wall or light on the next turn and turn away from it. However, in order to gather food the agent has to explore randomly looking for food, then when it finds food concentrate on following the trail. EAGP has the ability to switch between agents depending on which situation it needed to address, which allows it to move between behaviors, such as avoiding walls to following a trail. This is probably one of most important features of EAGP.

6 DISCUSSION

```
        hits: 995099                number of agents:2
      raw fitness: 400.0000
  standardized fitness: 0.1667
    adjusted fitness: 0.8571

ADF0: (type=light)
 (if-light-ahead left
          (if-wall-ahead (if-light nothing
                                  (if-light-ahead right left))
                        (if-wall-ahead (if-light (if-wall-ahead right
                                                        (if-food-ahead left move)) nothing) move)))
ADF1: (type= wall)
 (if-light (if-wall-ahead (if-light left move) right)
       (if-food-ahead (if-light-ahead (if-light (if-wall-ahead right right) move) move)
              (if-light-ahead left right)))
ADF2: (not used)
 (if-light-ahead (if-light left
                  (if-food-ahead move
                           (if-wall-ahead right right)))
          (if-wall-ahead (if-wall-ahead left left) move))
```

Figure 4: The Best EAGP Solution to the Modified Santa Fe Trail

Our goal was to create an Echo-like problem-solving system based on the interaction of multiple agents and a more loosely defined sense of adaptive fitness via a fitness function. EAGP accomplishes these goals and appears to perform well on some standard problems of robot navigation. It is interesting to note that despite the fact that we severely limited the tree depth, the EAGP solutions often contained only one or two agents, where one agent could actually perform multiple functions. The best EAGP approach can be seen in Figure 4. The "hits" is a composite number for EAGP evaluation. In the figure, the composite indicates the percentage of each

goal accomplished; 99/100 for light, 50/100 for food, and 99/100 for the wall. We determine each number in the composite value by evaluating each agent (subtree) on the aspect of the multi-part fitness for which it was tagged. Thus the first subtree is tagged as a "light" agent and is evaluated on its light avoidance behavior. For this solution , we list agent tags for readability sake. Note the last agent exists but is not yet being used in evaluation.

In Figure 4, the solution contains only a wall-avoider and a light-avoider, but those two agents were still capable of addressing all three aspects of the fitness function since the wall-avoider was quite capable of gathering food. We need to investigate how stringently we can limit the agents, pushing them to be less complex while not overly restricting their abilities.

It is also worth noting that the behavior of the best ADF-GP eventually got stuck in a loop. The virtual agent runs back and forth across the board along a straight-line path. Since it encounters no obstacles along this path it simply repeats the behavior endlessly. The EAGP solution would never remain in such a loop due to the fact that the problem-solver's agent reservoirs would be affected by the looping and its overall behavior would eventually change. Furthermore, the best EAGP solutions often had fairly divergent agent behaviors that would prevent another agent from picking up the same loop. Thus EAGP has an innate ability to avoid looping behavior.

Creating subsumption like architectures to solve path problems is not a simple task and has not yet really been addressed by the Evolutionary Computation community. Koza[Koza92] has claimed to create subsumption architectures using a basic GP approach, but his claim is based on the fact that the generated programs have if-then-else statements which he equates with separate agents interacting on solving a problem. This is not truly the case. The if-then-else statements are not in fact **separate** agents, but a single agent with conditions. The clauses do not evolve independently nor do they get evaluated independently. Finally, their fitness is not directly based on interaction. We think EAGP is a more faithful approach to subsumption problem-solving.

6.1 FUTURE WORK

The work described here is an outline of all the possibilities we could address with EAGP. There are many aspects we have not examined yet, and many more we have not even considered. For example:

- We would like to test EAGP on more realistic and difficult problems, to show what it can do. EAGP appears to be most capable on complex, interactive problems. Beyond reactive control problems, we believe that design and planning problems would also be appropriate and we plan to investigate these problems more deeply in the future.
- Clearly more work needs to be done on models of Arbitration. Presently we have not attempted to co-evolve an Arbitrator along with the agents, an important topic that needs to be addressed. We would also like to investigate an arbitration directly between the individual agents of the problem-solver that is not centrally controlled. As mentioned, this is much more Echo-like, though more expensive to run.
- The tagging system needs to be better implemented. In the present system, the type of an agent is set at initialization and maintained for the entire run. This was a result of the expense of constantly evaluating the appropriate tag for each agent. This could be addressed with some caching of tags and the genetic operations performed on the agent (subtree). Updating types and maintaining multiple types for more complex agents are but a few of the other issues we need to address.

ACKNOWLEDGEMENTS

We like to thank John Holland and Rick Riolo for their helpful discussion and comments on this work. We would also like to acknowledge David Fogel's book[Fogel98] on the history of Evolutionary Computation. It has been invaluable for this research. Finally, we would like to thank our reviewers for their insightful comments.

REFERENCES

[Booth97] G. Booth (1997), "Gecko: A continuous 2-D world for ecological modeling," *Artificial Life Journal*, Vol. 3, No. 3, pp. 147-163.

[Brooks91] R.A. Brooks (1991), "Intelligence Without Representation," *Artificial Intelligence Journal*, Vol. 47, pp. 139-159.

[Brooks97] R.A. Brooks (1997), "From Earwigs to Humans," *Robotics and Autonomous Systems*, Vol. 20, No. 2--4, pp. 291-304.

[Fogel98], (ed.) D. Fogel (1998), *Evolutionary Computation:the Fossil Record*, IEEE Press, New York,NY.

[Holland62] J.H. Holland (1962), "Outline for a logical theory of adaptive systems," *Journal of the Association of Computing Machinery*, Vol. 9, pp. 297-314.

[Holland73] J.H. Holland (1973), "Genetic algorithms and the optimal allocation of trials," *SIAM Journal of Computing*, Vol. 2, pp. 88-105.

[Holland75] J.H. Holland (1975), *Adaptation in Natural and Artificial Systems*, University of Michigan Press, Ann Arbor, MI.

[Holland78], J.H. Holland and J.S. Reitman (1978), "Cognitive Systems Based on Adaptive Algorithms," *Pattern-Directed Inference Systems*, D. A. Waterman and F. Hayes-Roth(eds.), Academic Press, NY, pp. 313-329.

[Holland95], J.H. Holland (1995), *Hidden Order:How Adaptation Builds Complexity*, Addison-Wesley, Reading, MA.

[Holland98] J.H. Holland (1998), *Emergence:from Chaos to Order*, Addison-Wesley, Reading, MA.

[Koza89] J.R. Koza (1989), “Hierarchical genetic algorithms operation on populations of computer programs,” *Proceedings of the 11th Joint Conference on Artificial Intelligence*, N.S. Sridharan (ed.), Morgan Kaufmann, San Mateo, CA, pp. 768-774,

[Koza92] J.R. Koza (1992), *Genetic Programming: On the Programming of Computers by Means of Natural Selection*, MIT Press, Cambridge, MA.

[Koza94] J.R. Koza (1994), *Genetic Programming II: Automatic Discovery of Reusable Programs*, MIT Press, Cambridge MA.

[lilgp] D. Zongker and W.F. Punch, “lilgp, a genetic programming system in C,” http://garage.cps.msu.edu/software/software-index.html#lilgp

[lilgp-st] S. Luke, “Strongly-typed lilgp,” http://www.cs.umd.edu/users/seanl/gp/patched-gp/

[lilgp-ct] C. Janikow, “Constrained lilgp,” http://laplace.cs.umsl.edu/~janikow/cgp-lilgp

[Lin94] S-C Lin, W.F. Punch and E.D. Goodman (1994), “Coarse-grain Genetic Algorithms, Categorization and New Approaches,” *Sixth IEEE Parallel and Distributed Processing*, pg. 28-37.

[Punch98] W.F. Punch (1998), “How Effective are Multiple Populations in Genetic Programming*", Genetic Programming 1998: Proceedings of the Third Annual Conferenc*e, pp. 308-313, July 22-25, 1998, University of Wisconsin, Madison, Wisconsin.

[Schmitz96] O.J.Schmitz and G. Booth (1996) “Modeling food web complexity: the consequence of individual-based spatially explicit behavioral ecology on trophic interactions,” *Evolutionary Ecology*, Vol. 11, pp. 379-398.

[swarm] C. Langton, “The Swarm Simulation System,” http://www.santafe.edu/projects/swarm