Final Technical Report **December, 2004**

Multi-type, Self-Adaptive Genetic Programming for Complex Applications

PI: Lee Spector, Hampshire College

DARPA Agent Based Computing (ABC) Program
Taskable Agent Software Kit (TASK)

Project URL: http://hampshire.edu/lspector/darpa-selfadapt.html

Abstract

This focus of this project was the development of new forms of genetic and evolutionary computation and their application to problems in the automatic development and programming of multi-agent systems. This report summarizes the project's methods, assumptions, procedures, and results, and also provides links to related publications and software packages that were produced during the project.

Table of Contents

Summary 1
Introduction 2
Methods, Assumptions, and Procedures
Genetic Programming 5
Reducing Configuration Parameters for GP through Self-Adaptation ..6
Multi-type Genetic Programming 8
Computational Infrastructure: Asynchronous Parallelism and Simulations 9
Results and Discussion
Highlights 11
Explanations and Illustrations 13
Conclusions 27
References
Primary publications of work conducted in this project 28
Additional references 30

List of Figures

Figure 1. Genetic Programming flowchart.....6
Figure 2. The execution architecture for the Push programming language for genetic and evolutionary computation.....13
Figure 3. The flowchart of the Pushpop autoconstructive evolution system.....15
Figure 4. Diversification results from Pushpop.....15
Figure 5. A snapshot of SwarmEvolve 2.0. 16
Figure 6. A snapshot of SwarmEvolve 2.0 with lines drawn between agents that have shared energy with one another. 17
Figure 7. Results demonstrating the emergence of collective behavior in certain conditions in the SwarmEvolve 2.0 autoconstructive evolution system.....18
Figure 8. Team/agent architecture for competitive co-evolution of Quidditch-playing agents. 19
Figure 9. A snapshot of the system for competitive co-evolution of Quidditch-playing agents. 20
Figure 10. A GUI for the QGAME Quantum Gate and Measurement Emulator. 21
Figure 11. The cover of the PI's book on the evolution of quantum algorithms using QGAME and PushGP.....22
Figure 12. A better-than-classical quantum algorithm for determining whether a provided oracle gate (f) satisfies the "AND/OR" property. 23
Figure 13. The structure of the Hampshire/UMass collaboration on the UAVolve project. 24
Figure 14. Performance of the evolved UAV surveillance strategy relative to the UMass strategy and the random initial strategies. 25
Figure 15. A snapshot of the UAVolve system for the evolution of UAV surveillance strategies. 26

Summary

The core technologies developed in this project extend the genetic programming paradigm through stack-based type-handling facilities that provide several benefits simultaneously; these include support for data-rich application areas, automatic modularization (through the use of CODE and EXEC types), and self-adaptation of the evolutionary process (by leveraging the CODE type in the reproduction process). These technologies were applied in a range of complex application areas including several involving multi-agent systems (such as transport network controllers, goal-directed swarms, and surveillance UAVs) and other applications that helped to "stress test" the core technologies (for example on quantum computing problems). Several infrastructural technologies were also developed, including systems for conducting and analyzing genetic programming runs on high-performance computer clusters and simulation systems within which evolving programs can be assessed for fitness. The results of this project include both the specific technologies (most of which are available from the web; links are provided below) and documentation of the methodologies by which these technologies can be applied to the automatic development and programming of multi-agent systems.

Introduction

Automatic programming technologies hold great promise for the production of software agents for complex, dynamic environments. Genetic programming techniques, in particular, can harness the power of natural selection to produce autonomous systems well suited to their niches, even in environments that are too complex or dynamic for detailed human analysis. In a genetic programming system solutions emerge from a continuous process of adaptive engagement with the environment; in some cases this can produce solutions where other methods fail. Furthermore, automated methods can be repeatedly run and varied, and the results of these runs can be analyzed to provide detailed information about the difficulty of the agents' tasks and the suitability of the methods for the task environments. The genetic programming literature describes many experiments in which systems with agent-like properties have been automatically generated. However, most of the previously generated systems have been too simple to perform helpful tasks for actual users. A gap remains between the capabilities of current genetic programming technology and those that would be required to automatically generate robust software agents. The proposed research seeks to narrow the gap between current genetic programming technology and genetically-based "well-founded agent creation tools."

As stated in the BAA to which this project was a response, "Agent development is currently a 'black art' -- an arbitrarily complex software development" process. One way to make agent development more replicable and amenable to formal analysis is to automate the process so that controlled, repeatable experiments can be performed. Unfortunately, however, the successful application of genetic programming technology is also a "black art," often requiring repeated experiments and sophisticated understanding of interactions among a genetic programming system's many parameters. One goal of the proposed project was therefore to reduce the configuration work that a practitioner must do to apply the technique to a new problem area. The primary approach was to fold parameters into the representation over which the evolutionary search is conducted — that is, to allow the same adaptive process of natural selection to control both the search for agent programs and the search through the space of system parameters. This approach, of using *self-adaptation* to control system parameters, was studied previously in the genetic algorithms and genetic programming communities, but it had been applied much more narrowly than was accomplished here. The work in this project used self-adaptive

mechanisms to control not only mutation rates and other numeric parameters, but also to develop data representations, program representations, population structure, and reproductive strategies. When it is adaptive to do so the developed systems may evolve toward any of a broad range of genetic algorithm or genetic programming approaches, but the user is not required to specify the details of the strategy or combination of strategies in advance.

We began with standard genetic programming systems as described in, for example, (Koza, 1992, 1994; Banzhaf et al., 1998; Koza et al., 1999; Spector et al., 1999). Application to the evolution of robust agents in complex multi-agent systems required several innovations; these innovations were the focus of this project and they will be outlined below.

Agents must generally interact with a heterogeneous set of entities, each of which may use a specialized interface or language. Early genetic programming systems forced users to restrict all operations to a single data type to ensure the semantic validity of programs undergoing recombination and mutation. More recently, "strongly typed" genetic programming systems have been developed that relax this restriction, allowing the generation of programs that manipulate a diverse set of data types. This capability is critical for the evolution of agents that must interact with diverse entities and refer to diverse data. The research conducted in this project extended the idea of strongly typed genetic programming to a more general notion of "multi-type" genetic programming that has several advantages. First, it allows the arbitrary intermixing, without syntactic restrictions, of code that works with any data type or set of data types. Second, it incorporates first-class function types to simplify the dynamic evolution of subroutines, control structures, and other program representations. Third, it leverages the extended type system to allow agent programs to contain their own reproduction procedures, thereby transferring the responsibility for reproductive strategies from the user (via global, human-set parameters) to the system itself (via natural selection operating on the agents themselves). The cumulative effect of these innovations is that, in many cases, the user can specify a diverse set of primitives and related data types while simultaneously specifying little in the way of system parameters.

These ideas have, over the course of this project, led to the development of several new technologies for automatic programming of multi-agent systems. These technologies include the Push programming language for

evolutionary computation (now at version 3 and available in several programming languages for several platforms), the PushGP genetic programming system (also available in several versions), and a variety of specific “autoconstructive evolution” systems. Significant results have been obtained with these technologies in application areas ranging from competitive strategy games in complex 3D environments to the discovery of new quantum computing algorithms.

Methods, Assumptions, and Procedures

Genetic Programming

Genetic programming (GP) is a technique for the automatic generation of computer programs by means of natural selection (Koza 1992). GP is a special case of the genetic algorithm developed by Holland (Holland 1992). Whereas the conventional genetic algorithm uses evolution-inspired techniques to manipulate and produce fixed-length chromosome strings that encode solutions to problems, GP manipulates and produces computer programs.

The GP process starts by creating a large initial population of programs that are random combinations of elements from problem-specific function and terminal sets. Each of the programs in the initial population is assessed for fitness. This is usually accomplished by running each program on a collection of inputs called fitness cases, and by assigning numerical fitness values to the output of each of these runs; the resulting values are then combined to produce a single fitness value for the program. The fitness values are used in producing the next generation of programs via a variety of genetic operations including reproduction, crossover, and mutation. (For a study comparing these operations see (Luke & Spector 1997, 1998)). Individuals are randomly selected for participation in these operations, but the selection function is biased toward highly fit programs. Over many generations of fitness assessment, reproduction, crossover, and mutation, the average fitness of the population may tend to improve, as may the fitness of the best-of-generation individual from each generation. After some number of generations or improvement in fitness, the best-of-run individual is produced as the output from the GP system. Figure 1 shows a flowchart for the typical genetic programming system.

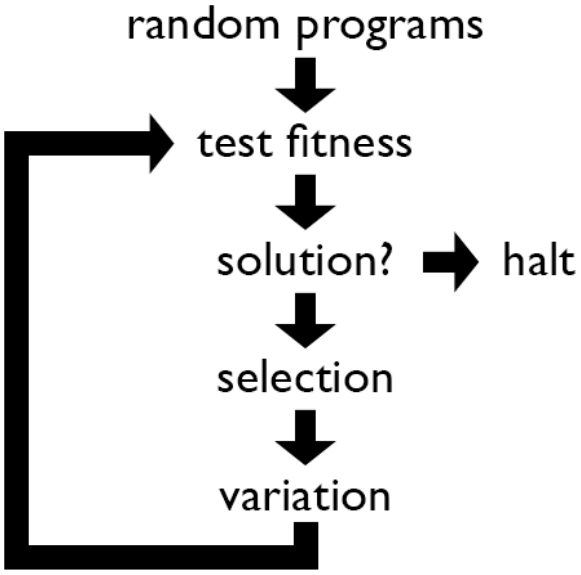


Figure 1. Genetic Programming flowchart.

Reducing Configuration Parameters for GP through Self-Adaptation

Rothlauf et al. describe the problem of configuring a genetic algorithm (GA) as follows:

> Users are often confronted with the situation that a GA does a good job for a small or easy problem, but when scaled up to bigger or more complicated problems, the traditional GA degrades or even breaks down. The user is frustrated and starts knob twiddling until he gets good solutions (or not!). Research is under way to help the user to overcome this problem and to adjust the parameters of a GA autonomously (Harik & Lobo, 1999). Nevertheless, finding good encodings, operators, and parameters that all fit together is a difficult task and needs work, time, and sometimes luck. The process of matching the encoding and the GA often draws the line between failure or success of optimisation. (Rothlauf, Goldberg and Heinzl, 2000)

One goal of this project was therefore to reduce the configuration work that a practitioner must do to apply the technique to a new problem area. The

primary approach was to fold parameters into the representation over which the evolutionary search is conducted — that is, to allow the same adaptive process of natural selection to control both the search for agent programs and the search through the space of system parameters. This approach, of using self-adaptation to control system parameters, had been studied previously in the genetic algorithms and genetic programming communities (see especially (Angeline, 1995)) but it had been applied much more narrowly than was accomplished here. In this project self-adaptive mechanisms were used to control not only mutation rates and other numeric parameters, but also to develop data representations, program representations, population structure, and reproductive strategies. When it is adaptive to do so the resulting system may evolve toward any of a broad range of genetic algorithm or genetic programming approaches, but the user will not be required to specify the details of the strategy or combination of strategies in advance.

Self-adaptation of program representations had been explored in previous genetic programming work through the use of automatically defined functions (Koza, 1994), architecture-altering operations (Koza et al. 1999), automatically defined macros (Spector, 1996), and to some extent self-modifying code (Nordin and Banzhaf, 1996; Spector and Stoffel, 1996a). The new mechanisms developed here, building on multi-type genetic programming extensions, generalize these mechanisms.

Self-adaptation of population structure and reproductive strategies has been studied in various forms, often involving the adjustment of parameters that control the frequency with which various genetic operators will be applied or details about how those operators will operate (e.g., with adaptive mutation) (Angeline, 1995). In contrast, the work in this project was more thorough-going in its approach to self-modification: the entire reproductive strategy, including population structure (that is, which other individuals may be accessed for crossover operations), the implementation of particular genetic operators (such as mutation and crossover), and any numeric control parameters were represented explicitly in the code of the evolving programs. They were therefore be manipulable by genetic operators and subject to variation and natural selection, providing for self-adaptation over evolutionary time. This notion of reproduction being the “responsibility” of individuals is widespread in the artificial life literature but has not been adequately explored in engineering-oriented genetic algorithms or genetic programming systems. The potential benefit is that a wide range of

parameters and design choices that are normally under the control of human experimenters are transferred to the control of the algorithm. We coined the expression "Autoconstructive evolution" to describe evolutionary computation systems that are self-adaptive in this way.

Multi-type Genetic Programming

Agents must generally interact with a heterogeneous set of entities, each of which may use a specialized interface or language. Early genetic programming systems forced users to restrict all operations to a single data type to ensure the semantic validity of programs undergoing recombination and mutation. More recently, "strongly typed" genetic programming systems have been developed that relax this restriction, allowing the generation of programs that manipulate a diverse set of data types (Montana, 1995). This capability is critical for the evolution of agents that must interact with diverse entities and refer to diverse data. In this project we extended the idea of strongly typed genetic programming to a more general notion of "multi-type" genetic programming. Multi-type genetic programming builds on the stack-based genetic programming paradigm, in which all intermediate values are stored on a global stack (Perkis, 1994; Stoffel and Spector, 1996). In multi-type genetic programming this concept is extended to use multiple stacks, one for each data type that may be important in the application environment. For example, one would use independent stacks for integers, symbols, strings, URLs, HTML tags, DAML tags, and so on. For each stack there is a set of primitive stack-manipulation operators (push, pop, duplicate, etc.) and higher-level functions that take any number of arguments of any of the types can be included in the function set. To implement the self-adaptive features described above several additional data types and corresponding high-level functions and stacks were developed; the additional data types include CODE and EXEC, and related high-level functions include DEFINE, DO, a general set of list manipulation primitives, and so forth. As with ordinary stack-based genetic programming, a function that is called in a context in which the required arguments are not available (because one or more stacks are empty) is simply ignored.

Multi-type genetic programming has several advantages over the standard strongly-typed genetic programming methods. First, it allows the arbitrary intermixing, without syntactic restrictions, of code that works with any data type or set of data types. Second, it incorporates first-class function types

and code entry-point types to simplify the dynamic evolution of subroutines, control structures, and other program representations. Third, it leverages the extended type system to allow agent programs to contain their own reproduction procedures, thereby transferring the responsibility for reproductive strategies from the user (via global, human-set parameters) to the system itself (via natural selection operating on the agents themselves). The cumulative effect of these innovations is that the user is able to specify a diverse set of primitives and related data types while simultaneously specifying little in the way of system parameters. As a result, these techniques are able to produce human-competitive results in a variety of difficult problem areas related to multi-agent systems.

Computational Infrastructure: Asynchronous Parallelism and Simulations

The work in this project was computationally intensive and required the development and deployment of several significant pieces of computational infrastructure.

First, it was necessary to deploy the developed evolutionary computation systems across a "Beowulf-style" computing cluster. We used an asynchronous "deme" or "island" model for our genetic programming runs, with one sub-population per computational node, and we developed software to support the execution and analysis of runs conducted in this way.

Second, it was necessary for several of the applications to develop simulation environments within which the evolving programs would be assessed for fitness. In particular, we developed and/or enhanced four significant simulation environments

- Breve: a 3d simulation environment for decentralized simulations and artificial life. This is an open source software project; see http://www.spiderland.org/breve.

- SwarmEvolve: an system for the evolution of goal-directed collective activity in a 3D environment. See http://hampshire.edu/lspector/gecco2003-collective.html.

- QGAME: Quantum Gate And Measurement Emulator, a quantum computer simulator. See http://hampshire.edu/lspector/qgame.html.

- UAVolve: a system for the evolution of strategic behavior of UAV swarms (developed jointly with the University of Massachusetts at Amherst TASK team, David Jensen P.I.).

Results and Discussion

We first provide a quick listing of the major results (highlights) of the project to provide an overview. We then augment the highlights with explanatory text and illustrations relating to some of the major results. Full details of each specific result, each publication, and each collaborative effort can be found in the archived technical reports and publications listed at http://hampshire.edu/lspector/darpa-selfadapt.html.

Highlights

- Development of the Push programming language for evolutionary computation (Push 1, Push 2, and Push 3, now available in C++ and Lisp; see http://hampshire.edu/push.html).

- Development of the PushGP multi-type, self-adaptive genetic programming system (now available in C++ and Lisp; see http://hampshire.edu/push.html).

- Development of the Pushpop autoconstructive evolution system (see see http://hampshire.edu/push.html).

- Development of the Breve simulation environment for complex agent-based systems (see http://www.spiderland.org/breve).

- Integration of Push/PushGP with Breve (first as a plugin, now through native support in Breve 2.0 and higher).

- Integration of PushGP/Pushpop with MIT/BBN-derived transport network agent simulator.

- Integration of Elementary Adaptive Modules into a framework for the evolution of multi-agent systems.

- Demonstration of the use of servos in evolved agent architectures.

- Demonstration of the utility of a genetic algorithm in conjunction with the Dartmouth 3D Opera problem simulator.

- Evolution of transport network control agents.

- Demonstration of the evolution of modularization in PushGP.

- Demonstration of the efficacy of size-fair genetic operators in PushGP.

- Development of diversity metrics in PushGP and Pushpop.

- Demonstration and extension of the "Van Belle/Ackley effect" (UNM).

- Demonstration of reliable auto-diversification in the Pushpop autoconstructive evolution system.

- Development of the QGAME Quantum Gate and Measurement Emulator to support the evolution of new quantum algorithms (now available in C++ and Lisp; see http://hampshire.edu/lspector/qgame.html).

- Discovery of several significant new quantum algorithms using the PushGP genetic programming system in conjunction with Qgame.

- Evolution of goal-directed 3D swarms driven by parameterized flocking equations (in SwarmEvolve 1.0; see http://hampshire.edu/lspector/gecco2003-collective.html).

- Evolution of goal-directed 3D swarms driven by open-ended programs in a Turing complete representation (in SwarmEvolve 2.0; see http://hampshire.edu/lspector/gecco2003-collective.html).

- Demonstration of the emergence of collective behavior and multicellular organization in 3D swarms (in SwarmEvolve 1.0 and 2.0; see http://hampshire.edu/lspector/gecco2003-collective.html).

- Analysis of the relations be environmental stability, genetic stability, and adaptation (in SwarmEvolve 2.0).

- Co-evolution of teams players for a complex, dynamic, 3D game (quidditch; see http://alum.hampshire.edu/~rpc01/vww.html, http://hampshire.edu/lspector/quidditch-movies/, and http://hampshire.edu/lspector/pubs/virtual_witches_and_warlocks.pdf).

- Production of course materials using multi-agent simulations (WUB World, Capture the Flag; see http://hampshire.edu/lspector/cs263/cs263s04.html).

- Evolution of high-performance surveillance strategies for UAVs.

Explanations and Illustrations

Many of the project results built on the development of the Push programming language for evolutionary computation. Push is described in over a dozen documents accessible from http://hampshire.edu/lspector/push.html, but the most current incarnation of the language as of this report, Push 3, is described in:

> Spector, L., C. Perry, J. Klein, and M. Keijzer. 2004. Push 3.0 Programming Language Description. http://hampshire.edu/lspector/push3-description.html.

Figure 2 provides a schematic diagram of the Push execution architecture.

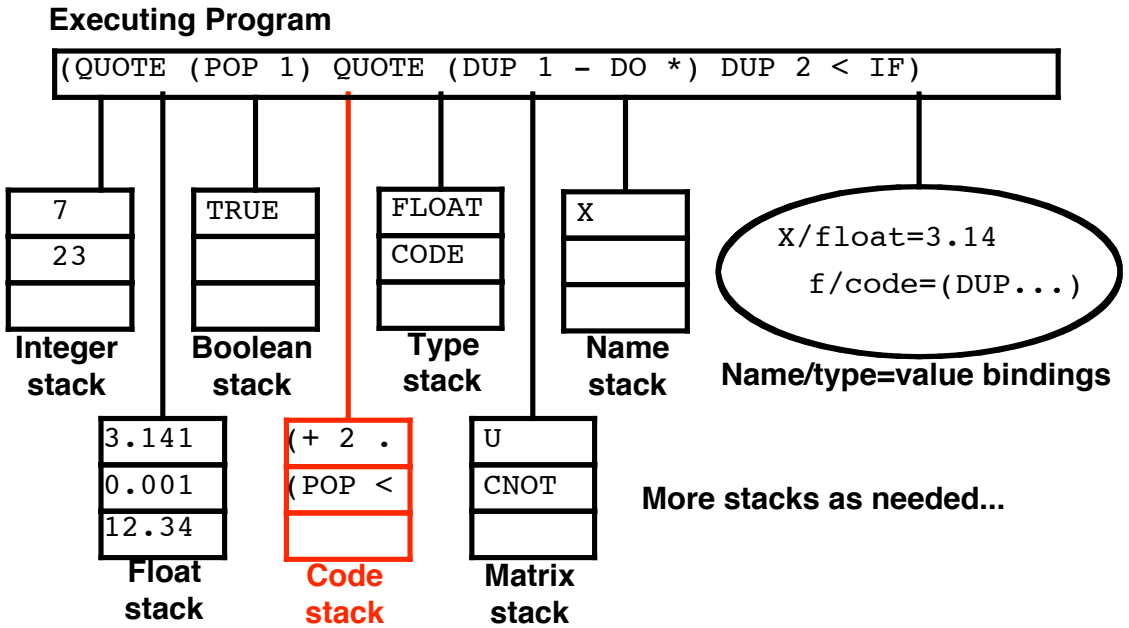


Figure 2. The execution architecture for the Push programming language for genetic and evolutionary computation.

Push was used in several genetic and evolutionary computation systems, the most widely-used of which was PushGP. PushGP:

- Evolves Push programs using (mostly) standard GP.

- Multiple types handled without syntactic constraints.

- Evolves modules and control structures automatically, in some cases more effectively than ADF mechanisms.

- See, e.g., results in GPEM 3:1 (2002), ECOMAS-2002, GECCO-2001, GECCO-2002, GECCO-2004.

A greater level of self-adaptation was achieved in our "autoconstructive evolution" systems such as Pushpop and SwarmEvolve 2.0. In an autoconstructive evolution system:

- Individuals make their own children.

- Agents thereby control their own mutation rates, sexuality, and reproductive timing.

- The machinery of reproduction and diversification (i.e., the machinery of evolution) evolves.

- Radical self-adaptation is achieved.

- See, e.g., results in ALife8, GECCO-2003, AAAI 2004 Symposium on Artificial Multiagent Learning, etc.

Figure 3 shows a flowchart for the Pushpop autoconstructive evolution system. Figure 4 shows results, published in the proceedings of the ALife8 conference, demonstrating that in adaptive Pushpop populations specied are more numerous and diversification processes are more reliable. This also demonstrates how selection can promote diversity and provides a possible explanation for the evolution of diversifying reproductive systems.

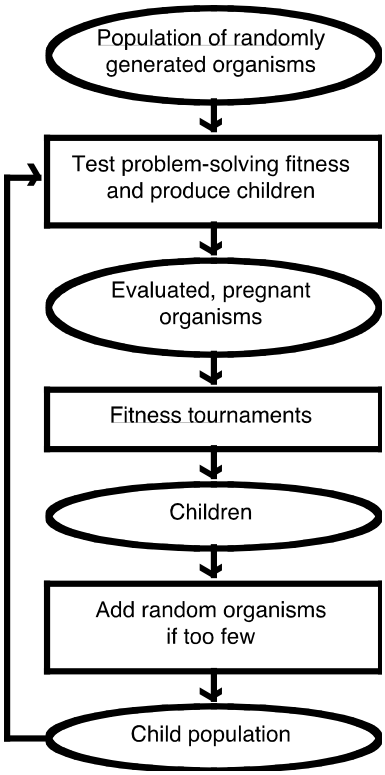
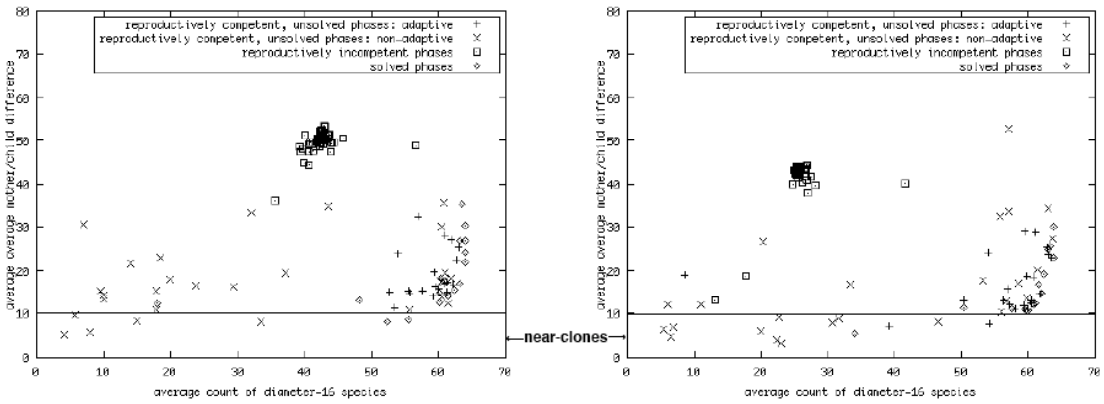


Figure 3. The flowchart of the Pushpop autoconstructive evolution system.

Note distribution of "+" points: adaptive populations have many species and mother/daughter differences in a relatively high, narrow range (above near-clone levels).



Runs including sexual instructions **Runs without sexual instructions**

Figure 4. Diversification results from Pushpop (see text).

Figures 5 and 6 show snapshots from the SwarmEvolve 2.0 system, while Figure 7 presents results demonstrating the emergence of collective behavior (energy sharing) in this dynamic, goal-oriented environment.

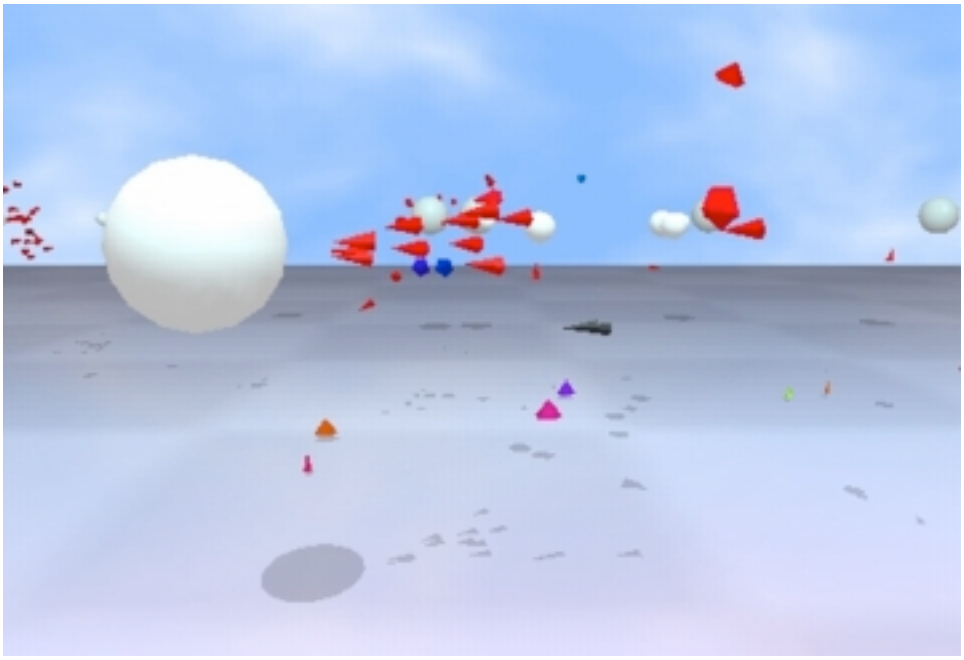


Figure 5. A snapshot of SwarmEvolve 2.0.

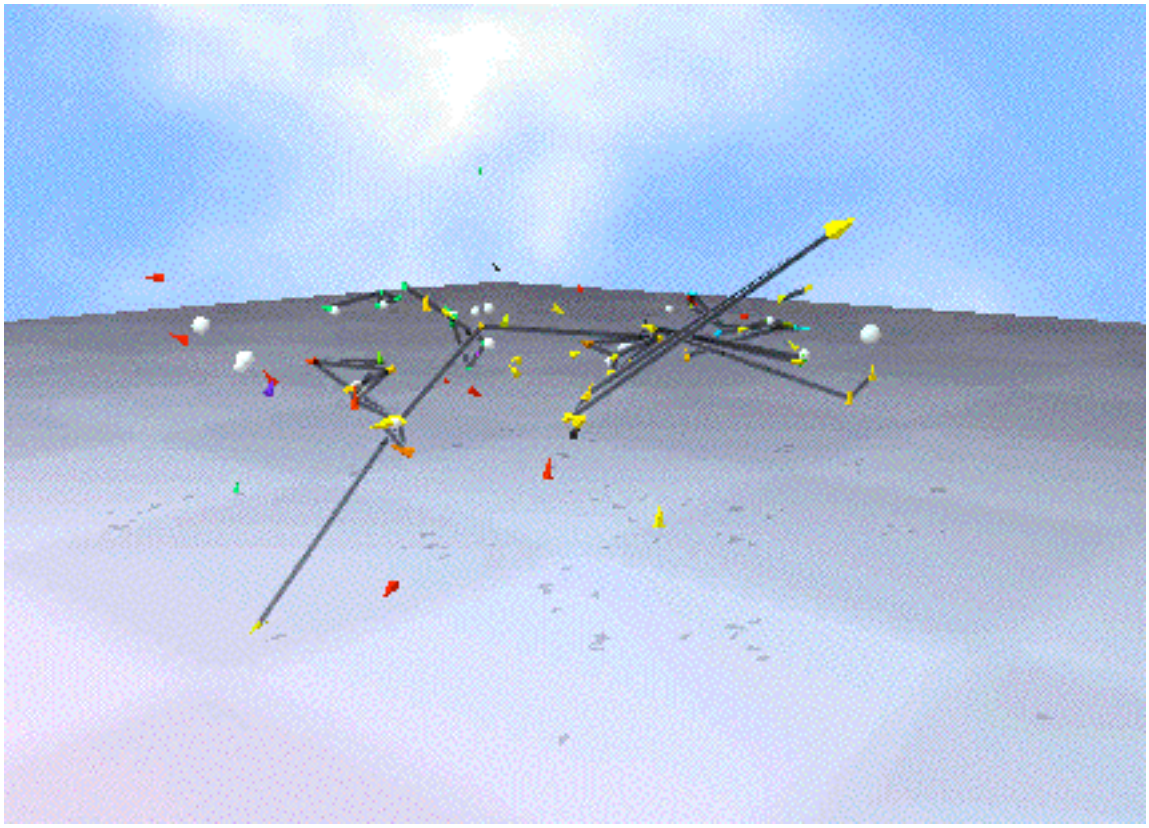


Figure 6. A snapshot of SwarmEvolve 2.0 with lines drawn between agents that have shared energy with one another.

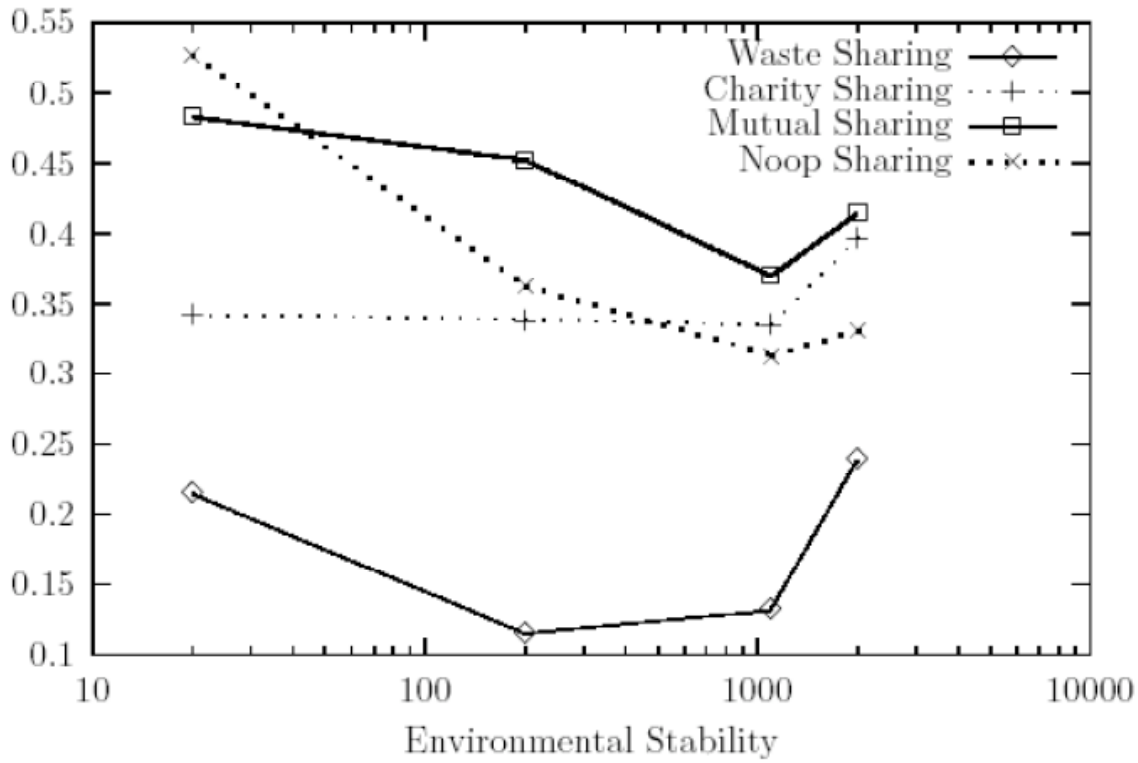


Figure 7. Results demonstrating the emergence of collective behavior in certain conditions in the SwarmEvolve 2.0 autoconstructive evolution system (see GECCO-2003 paper for full explanation).

In another demonstration of our combination of 3D simulation and self-adaptive evolutionary computation we evolved agents to play the "Quidditch" game from the popular Harry Potter books. This domain was chosen both because it attracts general interest and because of several useful technical features:

- Richly heterogeneous—player roles, balls themselves are active/intelligent.

- Richly 3-dimensional—flying game, full use of the third dimension.

- Extensible—rules not uniquely determined by the Rowling books; physics based on magic spells so the sky is the limit!

- Beyond human experience—unlike soccer, few intuitions about strategy to bias methods.

We used competitive co-evolution with teams represented as shown in Figure 8. Figure 9 shows a snapshot of the system. The source code for the system is available from http://hamp.hampshire.edu/~rpc01/vww.html.

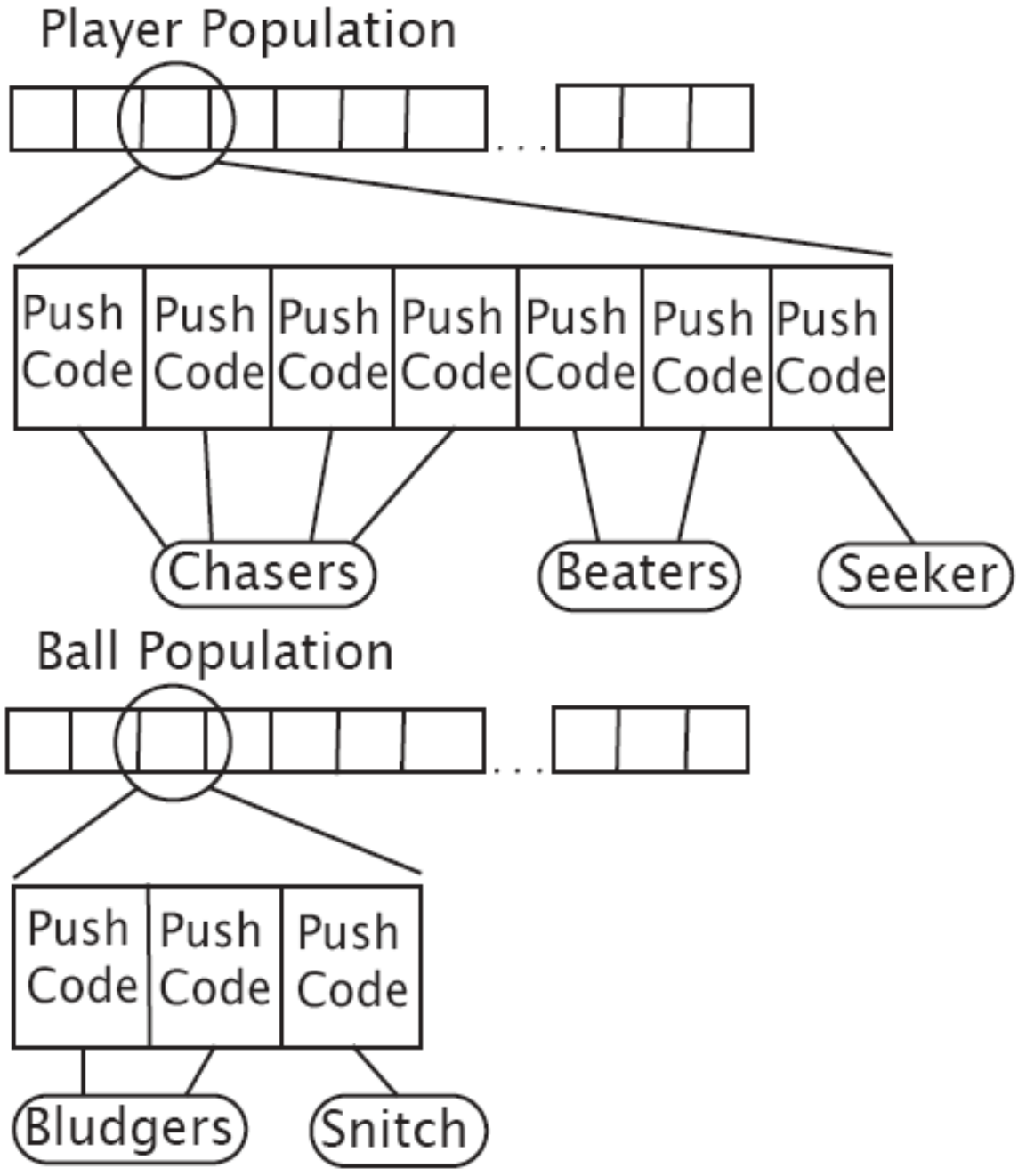


Figure 8. Team/agent architecture for competitive co-evolution of Quidditch-playing agents.

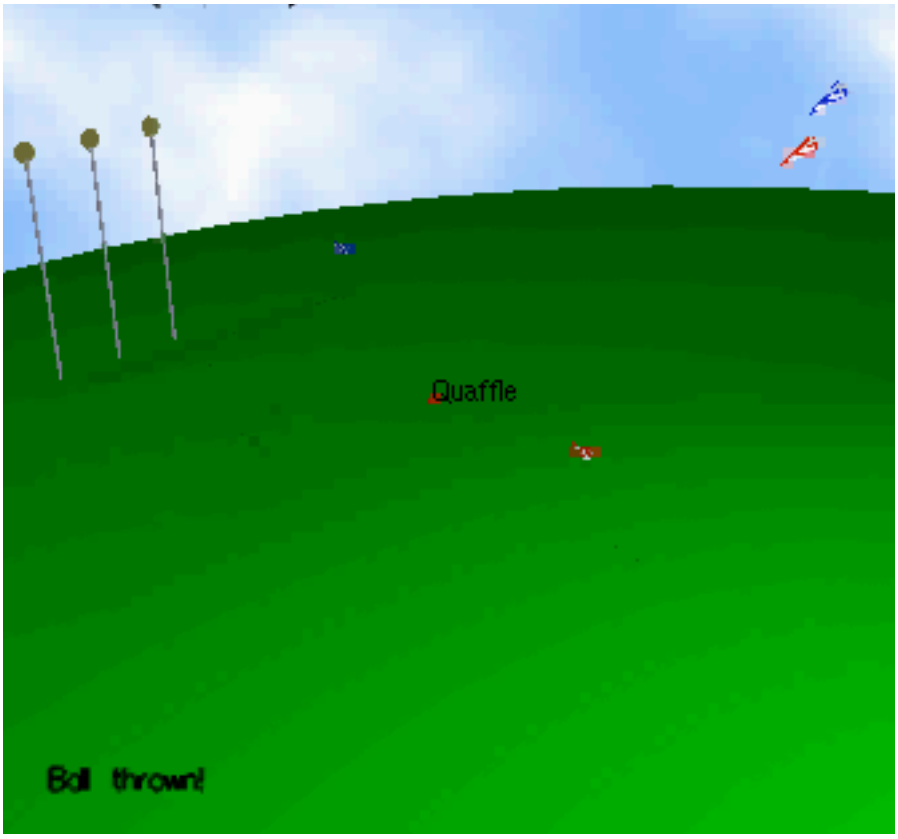


Figure 9. A snapshot of the system for competitive co-evolution of Quidditch-playing agents.

Several core technologies in the project were tested on applications to quantum computing which, while not directly involved in multi-agent systems, provide difficult test problems for machine learning technologies. These applications were supported by the development of the QGAME Quantum Gate and Measurement Emulator, a GUI for which is shown in Figure 10. Versions of QGAME are now available both in Lisp and in C++ from http://hampshire.edu/lspector/qgame.html. Work on QGAME in conjunction with PushGP formed the foundation for the book shown in Figure 11.

Several significant results were obtained in quantum computing, including results on:

- The 1-bit Deutsch-Jozsa (XOR) problem

- The 2-bit Grover database search problem

- The 1-bit OR problem

- The 2-bit AND/OR problem (see the algorithm shown in Figure 12)

- Communication and entanglement capacities of “Smolin” and “Bernstein-Spector” gates.

- Re-discovery of quantum dense coding.

Several of these results were significant both with respect to the genetic programming techniques that were employed in their production and in virtue of their importance in the study of quantum computing and quantum information theory, irrespective of their means of production. This significance is demonstrated by the fact that the results were published both in the computer science literature and in the physics literature. Several of these results were the basis for a gold medal award granted to the PI in the Human Competitive Results competition at the 2004 Genetic and Evolutionary Computation Conference (GECCO-2004).

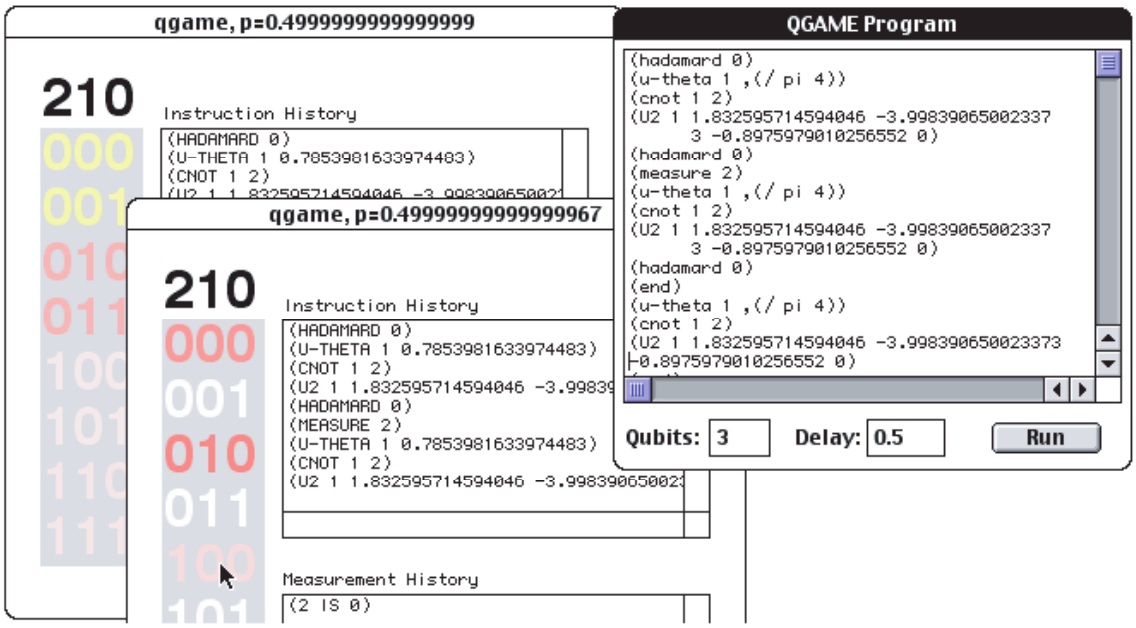


Figure 10. A GUI for the QGAME Quantum Gate and Measurement Emulator.

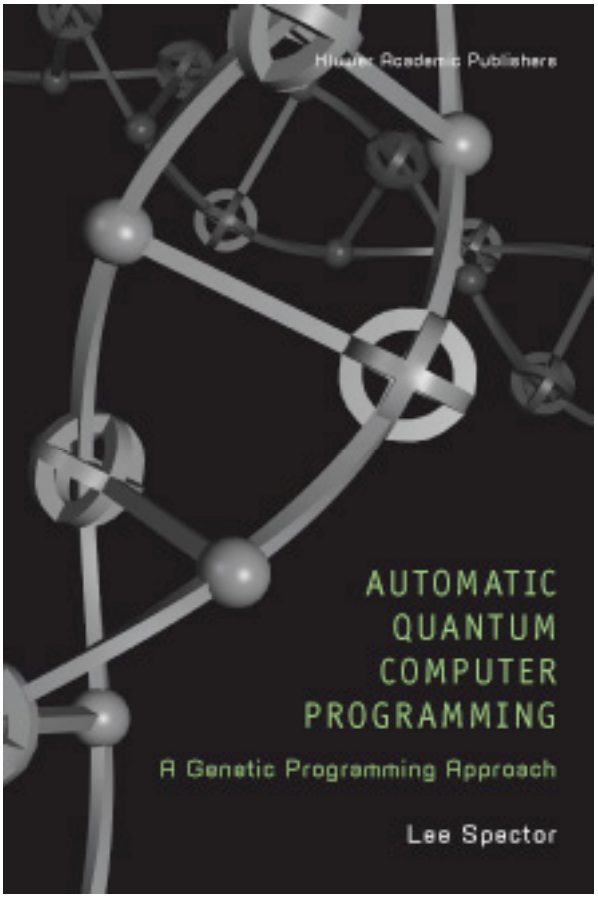


Figure 11. The cover of the PI's book on the evolution of quantum algorithms using QGAME and PushGP.

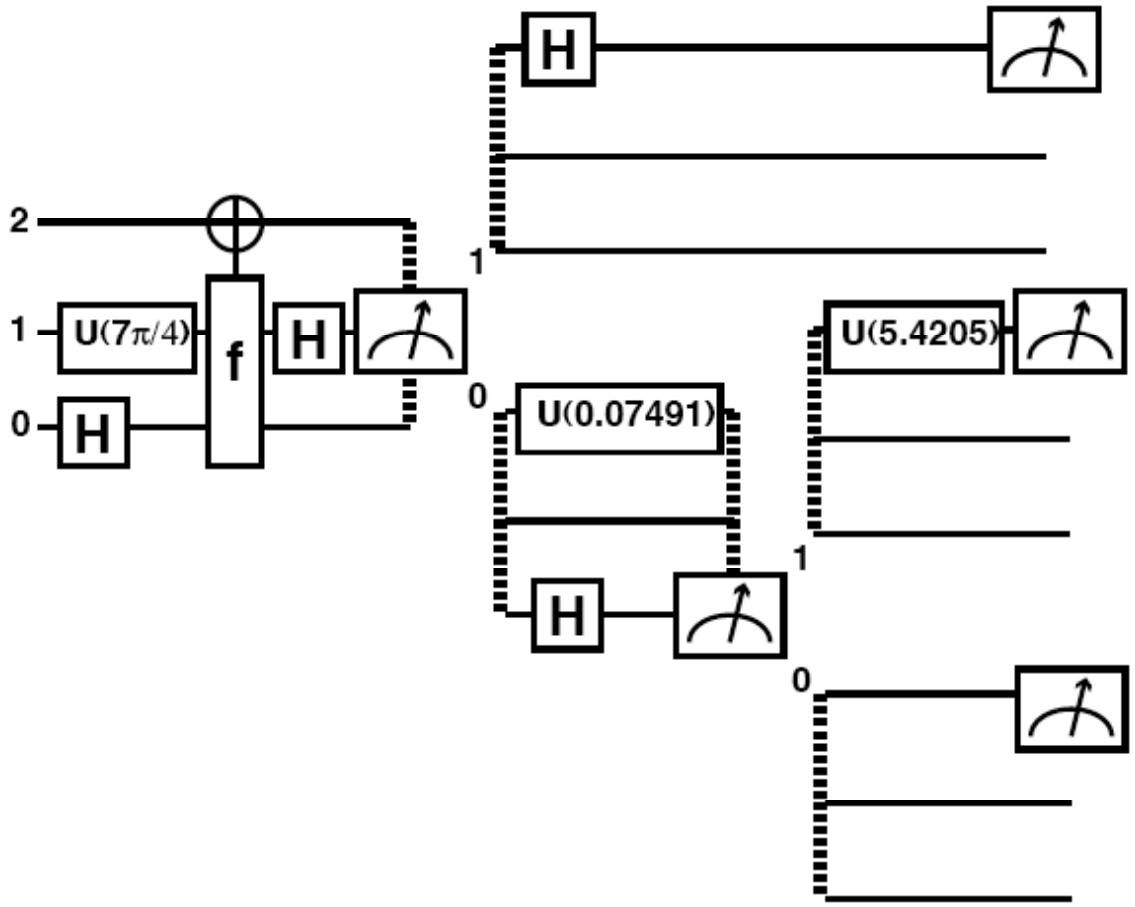


Figure 12. A better-than-classical quantum algorithm for determining whether a provided oracle gate (f) satisfies the "AND/OR" property. The algorithm was found by PushGP and is documented in the book described in the text.

The project culminated in a collaboration with the UMass TASK group on the UAV project organized by the TASK OEF group. Figure 13 shows the structure of the interaction between Hampshire College and UMass researchers.

As demonstrated at the August, 2004 TASK demonstration our techniques enabled us to evolve a UAV surveillance strategy that is robust in the face of unexpected target behaviors and threat areas. The evolved behavior appears disorganized at first, but after a short time, an “emergent” tour is established.

Figure 14 shows the performance of the evolved strategy, demonstrating that our techniques are able to improve on the strategies produced by well-trained humans (the UMass development team). Figure 15 shows a snapshot of the evolved strategy, in which an agent is skirting a threat area.
.

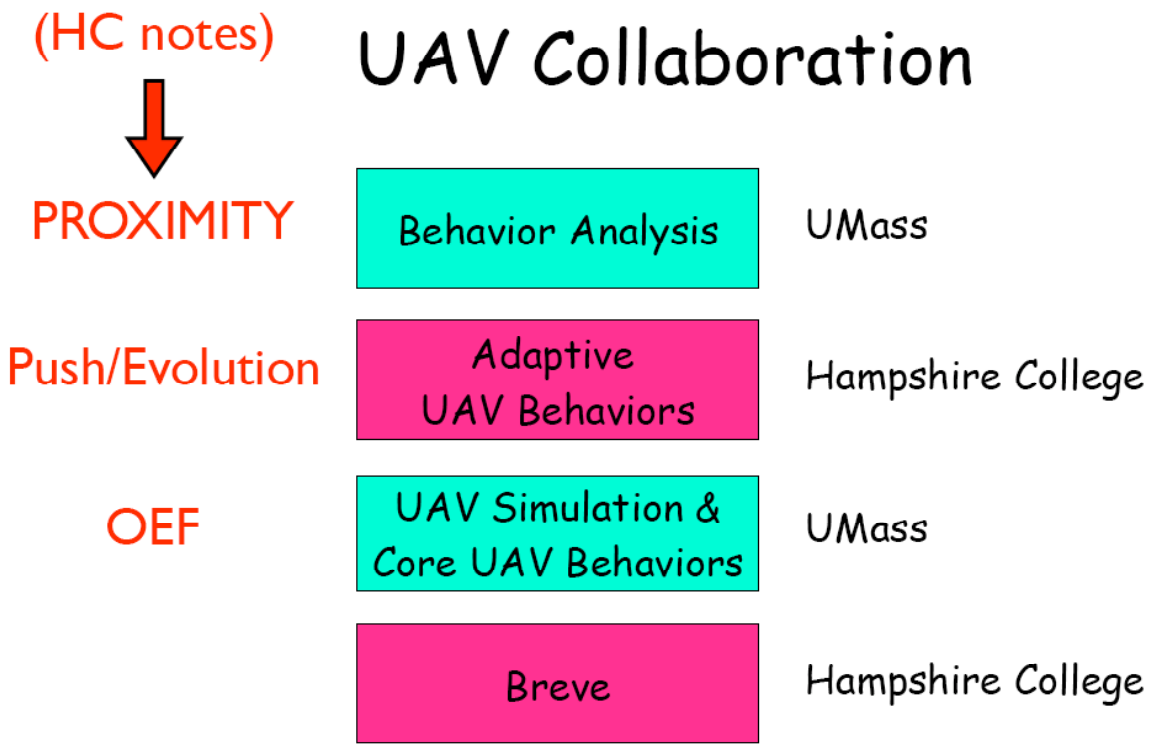


Figure 13. The structure of the Hampshire/UMass collaboration on the UAVolve project.

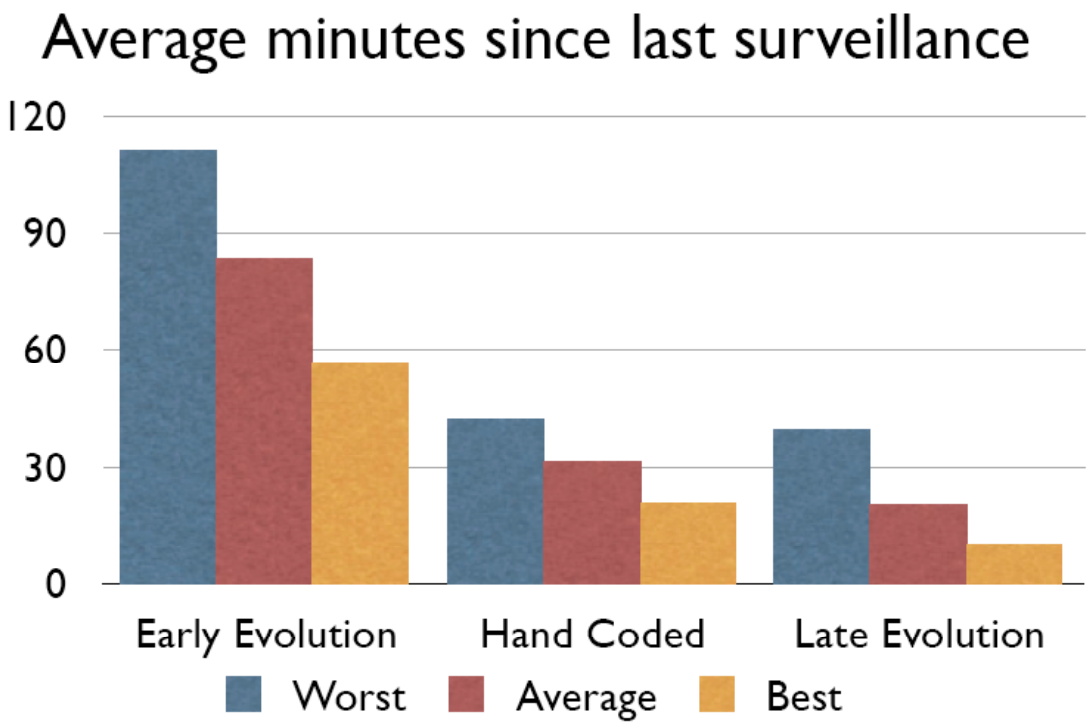


Figure 14. Performance of the evolved UAV surveillance strategy (Late Evolution) relative to the UMass strategy (Hand Coded) and the random initial strategies (Early Evolution). Lower is better.

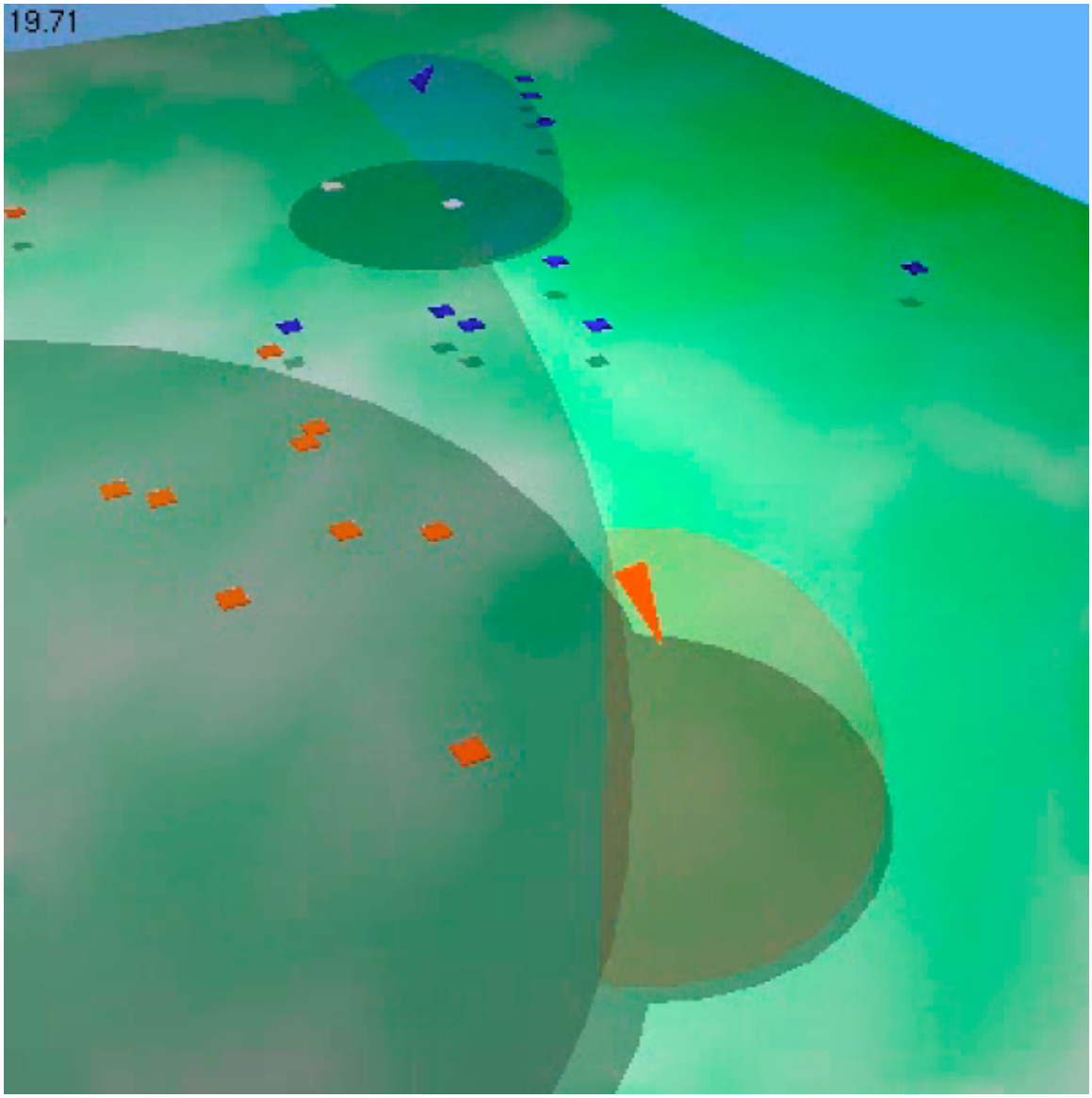


Figure 15. A snapshot of the UAVolve system for the evolution of UAV surveillance strategies.

Conclusions

This project produced several new technologies to support the automated or semi-automated development of multi-agent systems in complex, dynamic environments. These technologies, all of which are based on the concepts of self-adaptive, multi-type genetic programming, were demonstrated in a variety of application areas that range from transport network and UAV control to quantum computing. A large number of results were produced in these applications that are significant in their own right. The developed technologies are now available for deployment on other problems, both in the domain of multi-agent systems and in other problem areas.

References

Primary publications of work conducted in this project (note: many of these publications are available for download from http://hampshire.edu/lspector/darpa-selfadapt.html)

Spector, L., J. Klein, C. Perry, and M. Feinstein. To appear. Emergence of Collective Behavior in Evolving Populations of Flying Agents. In *Genetic Programming and Evolvable Machines*.

Spector, L., C. Perry, J. Klein, and M. Keijzer. 2004. Push 3.0 Programming Language Description. http://hampshire.edu/lspector/push3-description.html.

Spector, L. 2004. *Automatic Quantum Computer Programming: A Genetic Programming Approach*. Boston, MA: Kluwer Academic Publishers.

Spector, L., J. Klein, and C. Perry. 2004. Tags and the Evolution of Cooperation in Complex Environments. In *Proceedings of the AAAI 2004 Symposium on Artificial Multiagent Learning*. Melno Park, CA: AAAI Press.

Crawford-Marks, R., L. Spector, and J. Klein. 2004. Virtual Witches and Warlocks: A Quidditch Simulator and Quidditch-Playing Teams Coevolved via Genetic Programming. In *Late-Breaking Papers of GECCO-2004, the Genetic and Evolutionary Computation Conference*. Published by the International Society for Genetic and Evolutionary Computation.

Spector, L., J. Klein, C. Perry, and M. Feinstein. 2003. Emergence of Collective Behavior in Evolving Populations of Flying Agents. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2003)*. Springer-Verlag. pp. 61-73. **Winner, Best Paper Award** (AAAA Track).

Spector, L. 2003. An Essay Concerning Human Understanding of Genetic Programming. In Genetic Programming: Theory and Practice, edited by R.L Riolo and W. Worzel, pp. 11-24. Boston, MA: Kluwer Academic Publishers.

Spector, L., and H.J. Bernstein. 2002. Communication Capacities of Some Quantum Gates, Discovered in Part through Genetic Programming. In J.H.

Shapiro and O. Hirota, Eds., *Proceedings of the Sixth International Conference on Quantum Communication, Measurement, and Computing (QCMC)*. Princeton, NJ: Rinton Press.

Spector, L. 2002. Adaptive populations of endogenously diversifying Pushpop organisms are reliably diverse. In R. K. Standish, M. A. Bedau, and H. A. Abbass (eds.), *Proceedings of Artificial Life VIII, the 8th International Conference on the Simulation and Synthesis of Living Systems*, pp. 142-145. Cambridge, MA: The MIT Press.

Spector, L., and J. Klein. 2002. Evolutionary Dynamics Discovered via Visualization in the BREVE Simulation Environment. In Bilotta et al. (eds), *Workshop Proceedings of the 8th International Conference on the Simulation and Synthesis of Living Systems*, pp. 163-170. Sydney, Australia: University of New South Wales.

Spector, L., and A. Robinson. 2002. Multi-type, Self-adaptive Genetic Programming as an Agent Creation Tool. In *Proceedings of the Workshop on Evolutionary Computation for Multi-Agent Systems, ECOMAS-2002, International Society for Genetic and Evolutionary Computation.*

Robinson, A., and L. Spector. 2002. Using Genetic Programming with Multiple Data Types and Automatic Modularization to Evolve Decentralized and Coordinated Navigation in Multi-Agent Systems. In *Late-Breaking Papers of GECCO-2002, the Genetic and Evolutionary Computation Conference*. Published by the International Society for Genetic and Evolutionary Computation.

Spector, L., and A. Robinson. 2002. Genetic Programming and Autoconstructive Evolution with the Push Programming Language. In *Genetic Programming and Evolvable Machines*, Vol. 3, No. 1, pp. 7-40

Crawford-Marks, R., and L. Spector. 2002. Size Control via Size Fair Genetic Operators in the PushGP Genetic Programming System. In W. B. Langdon, E. Cantu-Paz, K. Mathias, R. Roy, D. Davis, R. Poli, K. Balakrishnan, V. Honavar, G. Rudolph, J. Wegener, L. Bull, M. A. Potter, A. C. Schultz, J. F. Miller, E. Burke, and N. Jonoska (editors), *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO-2002*, pp. 733-739. San Francisco, CA: Morgan Kaufmann Publishers.

Spector, L. 2002. Hierarchy Helps it Work That Way. In Philosophical Psychology, Vol. 15, No. 2 (June, 2002), pp. 109-117.

Spector, L. 2001. Autoconstructive Evolution: Push, PushGP, and Pushpop. In Spector, L., E. Goodman, A. Wu, W.B. Langdon, H.-M. Voigt, M. Gen, S. Sen, M. Dorigo, S. Pezeshk, M. Garzon, and E. Burke, editors, *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO-2001*, 137-146. San Francisco, CA: Morgan Kaufmann Publishers.

Spector, L., R. Moore, and A. Robinson. 2001. Virtual Quidditch: A Challenge Problem for Automatically Programmed Software Agents. In E.D. Goodman, editor, *Late-Breaking Papers of GECCO-2001, the Genetic and Evolutionary Computation Conference*. Published by the International Society for Genetic and Evolutionary Computation.

Barnum, H., H.J. Bernstein, and L. Spector. 2000. Quantum circuits for OR and AND of ORs. *Journal of Physics A: Mathematical and General*, Vol. 33 No. 45 (17 November 2000), pp. 8047-8057.

Additional references

Angeline, P. J. 1995. Adaptive and Self-Adaptive Evolutionary Computations, In *Computational Intelligence: A Dynamic Systems Perspective*, M. Palaniswami, Y Attikiouzel, R. Marks, D. Fogel and T. Fukuda (eds.), Piscataway, NJ: IEEE Press, pp 152-163.

Banzhaf, Wolfgang, Nordin, Peter, Keller, Robert E., and Francone, Frank D. 1998. *Genetic Programming -- An Introduction; On the Automatic Evolution of Computer Programs and its Applications*. Morgan Kaufmann, dpunkt.verlag.

Barnum, H., Bernstein, H. J., and Spector, L. 1999. Better-than-classical Circuits for OR and AND/OR Found Using Genetic Programming. Los Alamos Preprint Archive, http://xxx.lanl.gov/abs/quant-ph/9907056

Bruce, W.S. 1996. Automatic Generation of Object-Oriented Programs Using Genetic Programming. In *Genetic Programming 1996: Proceedings of the First Annual Conference*, pp. 267-272, MIT Press, 28-31.

Bruce, W.S. 1997. The Lawnmower Problem Revisited: Stack-Based Genetic Programming and Automatically Defined Functions. In *Genetic Programming 1997: Proceedings of the Second Annual Conference*, pp. 52-57, Morgan Kaufmann, 13-16.

Holland, J. H. 1992. *Adaptation in Natural and Artificial Systems*. Cambridge, MA: The MIT Press.

Koza, John R. 1992. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press.

Koza, John R. 1994. *Genetic Programming II: Automatic Discovery of Reusable Programs*. MIT Press.

Koza, J.R., D. Andre, F. H Bennett III, and M. Keane. 1999. *Genetic Programming 3: Darwinian Invention and Problem Solving*, Morgan Kaufman.

Luke, S., and L. Spector. 1996. Evolving Graphs and Networks with Edge Encoding: Preliminary Report. In Koza, John R. (editor), *Late-Breaking Papers at the Genetic Programming 1996 Conference*. Palo Alto, CA: Stanford Bookstore (ISBN 0-18-201-031-7).

Luke, S. and L. Spector. 1997. A Comparison of Crossover and Mutation in Genetic Programming. In *Genetic Programming 1997: Proceedings of the Second Annual Conference*, 240–248.Cambridge, MA: The MIT Press.

Luke, S., and L. Spector. 1998. A Revised Comparison of Crossover and Mutation in Genetic Programming. In *Genetic Programming 1998: Proceedings of the Third Annual Conference*, edited by J.R. Koza, W. Banzhaf, K. Chellapilla, K. Deb, M. Dorigo, D.B. Fogel, M.H. Garzon, D.E. Goldberg, H. Iba, and R.L. Riolo. pp. 208–214. San Francisco, CA: Morgan Kaufmann.

Montana, D.J. 1995. Strongly Typed Genetic Programming. *Evolutionary Computation*, 3(2), pp. 199-230.

Nordin, P. 1994. A Compiling Genetic Programming System that Directly Manipulates the Machine Code. In *Advances in Genetic Programming*, edited by K.E. Kinnear, Jr. pp. 311-331, MIT Press, 1994.

Nordin, P., and W. Banzhaf. 1995. Evolving Turing-Complete Programs for a Register Machine with Self-modifying Code Genetic Algorithms. In *Proceedings of the Sixth International Conference (ICGA95),* pp. 318-325, Morgan Kaufmann.

Perkis, T. 1994. Stack-Based Genetic Programming. In *Proceedings of the 1994 IEEE World Congress on Computational Intelligence*, Vol. 1, pp. 148-153, IEEE Press, 27-29.

Robinson, A. 2001. *Genetic Programming: Theory, Implementation, and the Evolution of Unconstrained Solutions*. Hampshire College Division III (senior) thesis.

Rothlauf, F., D. Goldberg and A. Heinzl. 2000. Bad Codings and the Utility of Well-designed Genetic Algorithms. IlliGAL Report #2000007. Illinois Genetic Algorithms Laboratory, University of Illinois at Urbaba-Champaigne.

Spector, L. Genetic Programming and AI Planning Systems. In *Proceedings of the Twelfth National Conference on Artificial Intelligence, AAAI-94*, 1329–1334. Menlo Park, CA and Cambridge, MA: AAAI Press/The MIT Press. 1994.

Spector, L. 1996. Simultaneous Evolution of Programs and their Control Structures. In *Advances in Genetic Programming 2*, edited by P. Angeline and K. Kinnear. Cambridge, MA: MIT Press.

Spector, L. 1997. Automatic Generation of Intelligent Agent Programs. In *IEEE Expert*. Jan–Feb 1997, pp. 3–4.

Spector, L. 2000. LGP2, a linear, steady-state genetic programming engine in Common Lisp. http://hampshire.edu/lspector/code.html

Spector, L., H. Barnum, and H. J. Bernstein. 1998. Genetic Programming for Quantum Computers. In J. R. Koza, W. Banzhaf, K. Chellapilla, K. Deb,

M. Dorigo, D. B. Fogel, M. H. Garzon, D. E. Goldberg, H. Iba, R. L. Riolo, editors, *Genetic Programming 1998: Proceedings of the Third Annual Conference*. Morgan Kaufmann. pp. 365-374.

Spector, L., H. Barnum, H. J. Bernstein, and N. Swamy. 1999a. Quantum Computing Applications of Genetic Programming. In Spector, Lee, W. B. Langdon, Una-May O'Reilly, and Peter J. Angeline, editors, *Advances in Genetic Programming 3*. MIT Press. pp. 135-160.

Spector, L., H. Barnum, H.J. Bernstein, and N. Swamy. 1999b. Finding a Better-than-Classical Quantum AND/OR Algorithm using Genetic Programming. In *Proceedings of the 1999 Congress on Evolutionary Computation*. IEEE Press. pp. 2239-2246.

Spector, L., and S. Luke. 1996a. Culture Enhances the Evolvability of Cognition. In G. Cottrell (editor), *Proceedings of the Eighteenth Annual Conference of the Cognitive Science Society*, 672–677. Mahwah, NJ: Lawrence Erlbaum Associates, Publishers.

Spector, L., and S. Luke. 1996b. Cultural Transmission of Information in Genetic Programming. In Koza, John R., Goldberg, David E., Fogel, David B., and Riolo, Rick L. (editors) *Genetic Programming 1996: Proceedings of the First Annual Conference*, 209–214. Cambridge, MA: The MIT Press.

Spector, L., and K. Stoffel. 1996a. Ontogenetic Programming. In Koza, John R., Goldberg, David E., Fogel, David B., and Riolo, Rick L. (editors), *Genetic Programming 1996: Proceedings of the First Annual Conference*, 394–399. Cambridge, MA: The MIT Press.

Spector, L., and K. Stoffel. 1996b. Automatic Generation of Adaptive Programs. In P. Maes, M. Mataric, J.-A. Meyer, J. Pollack, and S.W. Wilson (editors), *From Animals to Animats 4: Proceedings of the Fourth International Conference on Simulation of Adaptive Behavior*, 476–483. Cambridge, MA: The MIT Press.

Spector, Lee, W. B. Langdon, Una-May O'Reilly, and Peter J. Angeline, editors. 1999. *Advances in Genetic Programming 3*. MIT Press.

Spector, L., and J. Klein. 2002. Complex Adaptive Music Systems in the BREVE Simulation Environment. In Bilotta et al. (eds), *Workshop*

Proceedings of the 8th International Conference on the Simulation and Synthesis of Living Systems, pp. 17-23. Sydney, Australia: University of New South Wales

Stoffel, K., and L. Spector. 1996. High-Performance, Parallel, Stack-Based Genetic Programming. In Koza, John R., Goldberg, David E., Fogel, David B., and Riolo, Rick L. (editors), *Genetic Programming 1996: Proceedings of the First Annual Conference*, 224–229. Cambridge, MA: The MIT Press.

Walsh, P. 1999. A Functional Style and Fitness Evaluation Scheme for Inducting High Level Programs. In Proceedings of the Genetic and Evolutionary Computation Conference, Vol. 2, pp. 1211-1216, Morgan Kaufmann, 13-17.

Yu, T., and Clack, C. 1998. Recursion, Lambda Abstraction and Genetic Programming. In Genetic Programming 1998: Proceedings of the Third Annual Conference, edited by J.R. Koza, W. Banzhaf, K. Chellapilla, K. Deb, M. Dorigo, D.B. Fogel, M.H. Garzon, D.E. Goldberg, H. Iba, and R.L. Riolo. San Francisco, CA: Morgan Kaufmann.