Automatic Generation of Intelligent Agent Programs[*]

Lee Spector
lspector@hampshire.edu
http://hampshire.edu/lspector

An “agent,” for the sake of these comments, is any autonomous system that perceives and acts to achieve a narrow set of goals within a specific virtual or real environment. This definition, while broad, nonetheless highlights several contrasts between agents and traditional AI systems. Traditional AI systems have usually been designed to be operated under the control and watchful eye of a user, while agents are sent out into the world to act autonomously, usually on their owners' behalf. Traditional AI systems automate a single step on the path from perception to action — for example, vision or reasoning about action — while agents must manage the entire path. And traditional AI technologies aspire to provide systems with general knowledge about a domain, while agents need know only about their own limited concerns within their own particular environments. We only demand that an agent do one thing for us, but it should do it *all by itself*.

One reason for excitement about agent-oriented AI is the belief that it is easier to write programs for many narrow, autonomous systems than it is to write programs for fewer, broader, user-driven systems. A related reason for excitement is the primary message of these comments: it is also easier to *automatically generate* agent programs.

The artificial life, machine learning, and automatic programming literatures already describe many experiments in which systems with agent-like properties have been automatically generated (see e.g., [Cliff et al. 1994]).

[*]Full citation: Spector, L. 1997. Automatic Generation of Intelligent Agent Programs. In *IEEE Expert.* Jan-Feb 1997, pp. 3-4.

Although most of the previously generated systems have been too simple to perform helpful tasks for actual users, recent advances suggest that useful intelligent agents will soon be within reach of these techniques. The remainder of these comments will focus on one particular technology, genetic programming (GP) [Koza 1992], but related points should apply to other automatic programming technologies.

In GP, computer programs are generated by natural selection. The process starts with a large initial population of programs that are random combinations of problem-specific primitives. Each program is assessed for fitness, and the fitness values are used to control genetic operations (usually fitness-proportionate reproduction, crossover, and mutation) that produce the next generation. After a preestablished number of generations, or after the best fitness improves to some preestablished level, the best-of-run program is produced as the output from the GP system.

The two key steps in the application of GP to a new problem are the design of the fitness test and the choice of primitives. The design of the fitness test for agents is straightforward. In traditional AI systems many features of a system's input and output — for example their representation and level of abstraction — are up for grabs. But agents must manage the entire path from percepts to actions, so it's usually obvious what the inputs and outputs must be, and what should count as good input/output behavior. The fitness of a candidate agent can therefore be assessed simply by running it in a captured fragment of the actual target environment and by collecting statistics about its behavior (see Figure 1).

Useful agent primitives are under construction by several research groups (e.g., [Coen 1994]). The goals of such groups are generally to provide very-high-level language support for human programmers, but the resulting primitives may provide equal benefit to automatic programming processes. In any case the same features of the agent paradigm that make it possible to provide high-level tools for human programmers (for example, narrow focus) should also make it possible to provide good sets of high-level primitives to GP processes.

In many cases it is even possible for a GP system to *build its own* primitives, bootstrapping from low level operations such as arithmetic or basic network functions. Existing techniques allow one to simultaneously evolve a main program and a set of *automatically defined functions* (ADFs) used by that program [Koza 1994]. For agents specifically, primitives that pro-

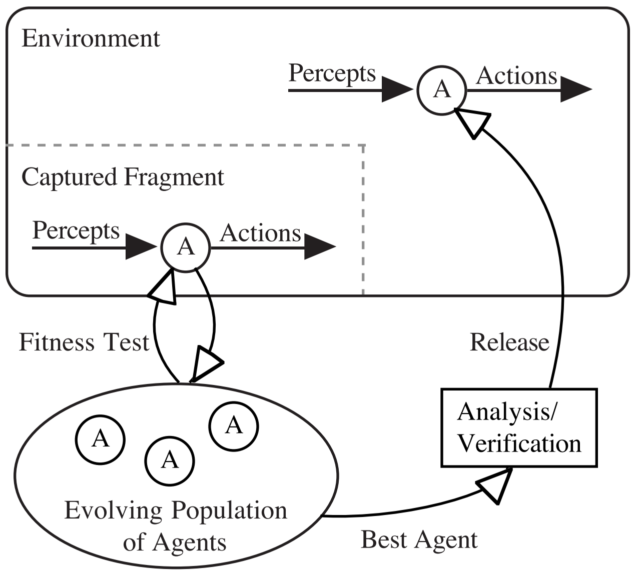

Figure 1: Framework for the evolution of artificial agents.

vide special-purpose *control structures* (for example, tasks) may be particularly useful; by use of *automatically defined macros* (ADMs) these may also be evolved simultaneously with the agent programs that use them [Spector 1996]. Both ADMs and ADFs have been shown to decrease the computational effort required to produce programs in certain environments.

Several other recent developments in GP technique dovetail nicely with the requirements for the generation of intelligent agents. For example, it is important for agents to be able to *adapt* to their environments as they run [Maes 1994]. The GP technique of *indexed memory* [Teller 1994] allows for the evolution of agents that record features of their environments and change their behavior accordingly. The technique of *ontogenetic programming* [Spector and Stoffel 1996] allows for the evolution of programs that adapt by modifying their own code throughout their lifetimes. The technique of *cultural memory* [Spector and Luke 1996] allows populations of evolving programs to collectively build repositories of information that may support more robust, adaptive behavior by individuals.

Some words of caution are also in order. In particular, the possibility of automatically generated agents forces us to confront the issue of agent *trustworthiness* in a serious way. After all, an agent acts on its owner's behalf, and with this autonomy comes responsibility. Although users may initially be more prepared to trust human-coded agents, precautions must be taken to analyze and verify agent code whatever its source (see Figure 1). The careful use of automatic programming techniques need not exacerbate this problem.

In sum, recent advances have significantly increased the power of automatic programming technology. Simultaneously, the development of the intelligent agent paradigm has provided an easier target that is particularly well-matched to the specific technological advances: autonomous systems with narrow expertise. The future looks bright for agent-oriented AI, and if the trends described here continue we won't have to write agent programs by hand — we will be able to generate them automatically.

References

Cliff, D., P. Husbands, J-A. Meyer, and S.W. Wilson, Eds. 1994. *From Animals to Animats 3.* MIT Press.

Coen, M. 1994. SodaBot: A Software Agent Environment and Construction System. MIT AI Lab Technical Report 1493.

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

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

Maes, P. 1994. Modeling Adaptive Autonomous Agents. In *Artificial Life Journal*, Vol. 1, No. 1 & 2. MIT Press.

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

Spector, L., and S. Luke. 1996. Cultural Transmission of Information in Genetic Programming. In *Proceedings of the Genetic Programming 1996 Conference.* MIT Press. To appear.

Spector, L., and K. Stoffel. 1996. Ontogenetic Programming. In *Proceedings of the Genetic Programming 1996 Conference.* MIT Press. To appear.

Teller, A. 1994. "The Evolution of Mental Models." In K.E. Kinnear Jr., Ed., *Advances in Genetic Programming*, pp. 199–219. MIT Press.