An Evolutionary Programming Approach to Self-Adaptation on Finite State Machines

Lawrence J. Fogel, Peter J. Angeline and David B. Fogel

Abstract

Evolutionary programming was first offered as an alternative method for generating artificial intelligence. Experiments were offered in which finite state machines were used to predict time series with respect to an arbitrary payoff function. Mutations were imposed on the evolving machines such that each of the possible modes of variation were given equal probability. The current study investigates the use of self-adaptive methods of evolutionary programming on finite state machines. Each machine incorporates a coding for its structure and an additional set of parameters that determine in part how it will distribute new trials. Two methods for accomplishing this self-adaptation are implemented and tested on two simple prediction problems. The results appear to favor the use of such self-adaptive methods.

1 INTRODUCTION

Evolutionary computation has a long history (Fogel 1995, Ch. 3). Some of the first efforts modeled evolution as a genetic process (Fraser 1957; Bremermann 1962; Holland 1975). In these simulations, a population of abstracted chromosomes are modified via operations of crossover, inversion and simple point mutation. An external selection criterion (objective function) is used to determine which chromosomes to maintain as parents for successive generations. These procedures have come to be termed *genetic algorithms*. Alternatively, Rechenberg (1965) and Schwefel (1965), and also Fogel (1962, 1964), offered methods for simulating evolution as a phenotypic process, that is, a process emphasizing the behavioral link between parents and offspring, rather than their genetic link. These simulations also maintain a population of abstracted organisms (either as individuals or species) but emphasis is placed on the use of mutation operations that generate a continuous range of behavioral diversity yet maintain a strong correlation between the behavior of the parent and its offspring. These methods are known as *evolution strategies* and *evolutionary programming*, respectively.

This paper focuses on experiments with evolutionary programming. In particular, self-adaptive parameters that provide information on the generation of offspring represented as finite state machines are incorporated into evolving solutions and are simultaneously subjected to mutation and selection. Such operations have been applied to real-valued function optimization problems, but can be extended to problems in discrete combinatorial optimization. The paper begins with background on the use of self-adaptation in evolutionary computation. The results of experiments comparing the efficiency of two self-adaptive methods on finite state machines for time series prediction are described. Finally, potential avenues for further investigation are discussed.

2 SELF-ADAPTIVE EVOLUTIONARY COMPUTATIONS

The ultimate effectiveness of any evolutionary optimization algorithm is determined by the relationship between the shape of the response surface (landscape) being searched and the mutation operations that are used to generate new trial solutions. The rate of optimization may be much greater if the mutative distribution can be tuned to follow grooves and valleys on the surface, rather than simply spray new trials with equal average step sizes in each dimension. The idea for allowing an evolutionary algorithm to self-adapt the manner in which it distributes new trials goes back at least to I. Rechenberg in 1967 (Rechenberg 1994), but was more explicitly detailed in Schwefel (1981).

For example, consider the problem of finding the r eal-valued n-dimensional vector **x** that minimizes $F(\mathbf{x})$. Each trial solution is taken to be a pair of vectors $(\mathbf{x}, \sigma)$, where **x** is the vector of object variables to be assessed by $F(\mathbf{x})$, and σ is a vector of standard deviations (often described as *strategy parameters*) corresponding to the step sizes of a zero mean multivariate Gaussian random variable. Offspring are created from each parent by the following rules:

$$x_i' = x_i + \mathrm{N}(0, \sigma_i) \qquad (1)$$

$$\sigma_i' = \sigma_i \cdot \exp(\tau \cdot \mathrm{N}(0, 1) + \tau' \cdot \mathrm{N}_i(0, 1)) \qquad (2)$$

where τ and τ' are operator-set parameters, $\mathrm{N}(\mu, \sigma)$ is a normally distributed random variable with mean μ and standard deviation σ, and $\mathrm{N}_i(0,1)$ describes a standard Gaussian resampled anew for the ith component of σ. Figure 1 indicates the potential for such a method to distribute trials in relation to the contours of the adaptive landscape. The technique distributes solutions in directions that have provided improved solutions in the past. Schwefel (1981) extended this method to allow for arbitrary correlations between perturbations.

Fogel et al. (1991) independently offered a similar self-adaptive procedure for evolutionary programming in which the standard deviations are altered using a Gaussian random variable. Specifically , the method is:

$$x_i' = x_i + \mathrm{N}(0, \sigma_i) \qquad (3)$$

$$\sigma_i' = \sigma_i + \alpha \cdot \sigma_i \cdot \mathrm{N}(0, 1) \qquad (4)$$

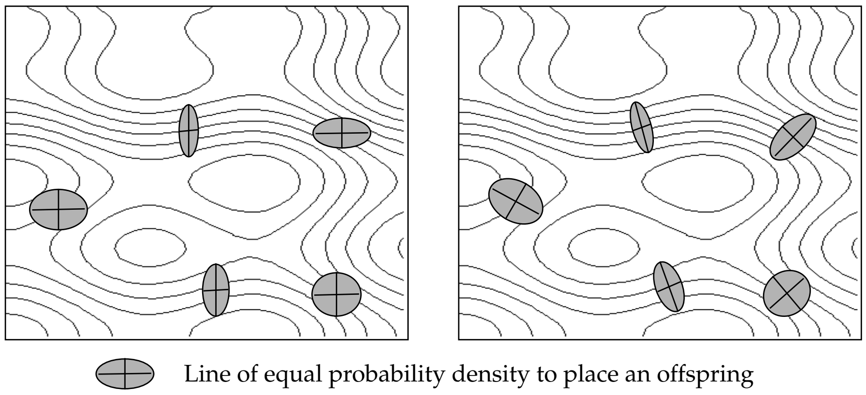

Figure 1: Contour plots of a response surface mapped onto two variable dimensions (after Bäck et al. 1991). Under independent Gaussian perturbations to each component of every parent, new trials are distributed such that the contours of equal probability are aligned with coordinate axes (left picture). This will not be optimal in general because the contours of the response surface are rarely similarly aligned. Schwefel (1981) suggested a mechanism for incorporating self-adaptive covariance terms. Under this procedure, new trials can be distributed in any orientation (right picture). The evolutionary process adapts to the contours of the response surface, distributing trials so as to maximize the probability of discovering improved solutions.

where α is a scaling parameter. If any value σ_i' becomes nonpositive, it is reset to a small arbitrary value ε. Fogel et al. (1992), at the suggestion of Sebald (1991), incorporated an additional procedure to allow for arbitrary correlations between the strategy parameters. Comparisons in Saravanan and Fogel (1994) indicate that the method of Schwefel (1981) generally outperforms the method of Fogel et al. (1991) when limited to uncorrelated perturbations of the strategy parameters. No comparisons have been made between the methods incorporating complete covariance matrices.

The idea for self-adapting the distribution of new trials also arose independently in genetic algorithms and genetic programming. Schaffer and Morishima (1987) offered a method for self-adapting crossover points. Each binary string encoded not only the *n*-bit solution vector, but an additional *n*-bit binary mask that determined the crossover points on the solution vector and was itself subject to mutation. Angeline and Pollack (1992) added mutation operators to a genetic program (Koza 1992) that protected entire subtrees from both crossover and mutation. Angeline and Pollack (1994) argued that protected subtrees raise the representational level of the primitive language in a task-specific manner .

Recently, Angeline and Pollack (1993) offered a different form of self-adaptation in evolutionary programming as applied to finite state automata in which individual links and output symbols could be randomly "frozen," effectively negating any probability for mutation. The current investigation examines the potential for more gradually affecting the probability of mutating links and output symbols in finite state machines used for time series prediction.

3 EXPERIMENTS

The base-line method of evolutionary programming investigated was similar to that of Fogel et al. (1966) and refined in Fogel (1991). Each machine in the population was judged in terms of a fitness function which represents the cost or benefit of each possible error or correct prediction. Each machine received a tournament score based on its fitness relative to q other machines selected at random from the population (Fogel 1991); in each competition, if its fitness was equal to or greater than its opponent, it received a "win." Parents for the next generation were chosen by ranking the population based on the number of wins (instead of raw fitness) and selecting those individuals scoring in the top half. Each parent created a single offspring in accordance to specific mutation operations.

Five modes of mutation were used to create offspring: add a state, delete a state, change the initial state, change an output symbol, and change a next-state transition. The mutation operation selected a specific mode of mutation for any single manipulation of a machine uniformly across modes. The specific component to modify was chosen in accordance with a uniform distribution from the set of such components in the machine (e.g., if an output symbol was to be changed, each output symbol had an equal chance of being selected). The number of mutations per parent was given by a Poisson random variable with a rate of 3.0. The maximum number of states for any machine was set to 25 and the minimum number of states was set to three. Two self-adaptive evolutionary programs for finite state machines were examined: selective self-adaptation and multi-mutational self-adaptation.

Selective Self-Adaptation

In this method of self-adaptation, a mutability parameter was associated with each component of a finite state machine. For each mutation, a component was selected based on the relative value of its mutability parameters. Specifically, the probability that the ith component was selected was given by:

$$\frac{P(i)}{\sum P(k)} \tag{5}$$

where $P(i)$ is the mutability parameter for the ith component, and the summation is taken with the index k running over all components. Separate mutability parameters were maintained for each state (i.e., probability of deleting the state), each output symbol on a transition based on an input symbol, and each next-state transition. For example, if the chosen mutation was to delete a state, the mutability parameters associated with each state of the machine were used to determine the relative probability of deleting each state. Similarly, when the chosen mutation indicated changing an output symbol associated with a transition in the machine, the particular transition was chosen using the mutability parameters associated with the output symbols of the machine's transitions.

All mutability parameters for each machine were initially set to a minimum value of 0.001. Thus each component of any initial machine was equally likely to be selected for mutation at the beginning of any trial. Mutability parameters for components of states subsequently incorporated as a result of an add-state mutation were also set to the minimum value. The mutability parameters were themselves mutated in a similar fashion to Fogel et al. (1991), specifically

$$\sigma_i' = \sigma_i + \alpha \cdot N(0, 1) \tag{6}$$

where σ_i is the parent's mutability parameter for the *i*th component, σ_i' is the offspring's mutability parameter for the *i*th component, and α is a scaling factor equal to 0.01 in the following experiments. Any mutability parameter that fell below a minimum value of $\varepsilon = 0.001$ was reset to the minimum; no upper limit was imposed.

Multi-mutational Self-adaptation

In a similar manner as selective self-adaptation, multi-mutational self-adaptation associated a mutability parameter with each component of each machine. But in contrast, each mutability parameter designated the absolute probability of modification for that particular component. Thus the probability for each component to be mutated was independent of the probabilities of other components to be mutated, this offering greater diversity in the types of offspring machines that could be generated from a parent.

For each offspring, each mutability parameter was compared to the outcome of a uniform random variable on (0,1) (denoted U(0,1)). If the random number was lower than the mutability parameter, the appropriate mutation was executed. For example, mutation would delete each state for which the outcome of the U(0,1) fell below that state's mutability parameter. Similarly, each output symbol and next-state transition were mutated when the resampled U(0,1) fell below the associated mutability parameter. Multi-mutational self-adaptation also modified the mutability parameters using the same technique and standard deviation as with selective self-adaptation. Unlike selective self-adaptation in which the chosen standard deviation is of little importance because the probabilities of specific mutations are all relative to other mutations, the standard deviation of the Gaussian noise is extremely important under the multi-mutational approach. Given too large a variance, the mutability parameters can decrease the stability and potential evolvability of the resulting offspring. For the current study, the minimum value for a mutability parameter in multi-mutational self-adaptation was set to 0.005. Thus no component had less than a 1 in 200 chance of being modified at any time. If adding Gaussian noise to the parameter resulted in a value less than this minimum it was reset to 0.005. The maximum value for the parameter was set to 0.999. Initial values for the parameters of machines in the initial population were set to 0.005.

To offset the potential increase in the deletion rate of states in evolving machines, the probability of adding a state to an offspring was increased to

0.3. The chance of mutating the initial state of a machine in multi-mutational self-adaptation remained at 0.2.

Design

The above methods were tested on two simple prediction tasks. The first was offered in Fogel et al. (1966). A base string of symbols served as an initial observation from an environment. The environment was taken to be the string (101110011101)*. Fitness was assessed as the fraction of correct predictions made over all observed symbols. A new symbol was introduced into the environment every five generations (i.e., a complete iteration of mutation, competition, and selection). Ten symbols were provided as the initial set of observations. The second environment was taken to be the string (101100111000110010)*. For each environment, the population size was 100 machines and trials were executed over 750 generations. Each experiment consisted of 50 trials with the basic evolutionary program (i.e., all modes of mutation having equal probability, all specific components having equal probability), the selective self-adaptation method and the multi-mutational self-adaptation method.

Results

Figure 2 indicates the score of the best machine in the population at each generation averaged over all 50 trials with each of the three methods on the environment (101110011101)*. The curves demonstrate an asymptotic rise toward 100 percent correct, as the cyclic pattern in the environment is mapped by successively better finite state machines. But the rate of improvement across the three methods appears to favor the self-adaptive methods. Figure 3 shows the *t*-test score comparing both self-adaptive methods to the basic evolutionary programming. Grey areas under the graphs denote a difference in the means which returned a *P*-value less than 0.05. Although there appears to be significant evidence of an improvement with the self-adaptive methods, caution must be used when

Figure 2: The fraction correct of the best machine in the population at each generation averaged over all 50 trials with each method applied to the environment (101110011101)*. Both self-adaptive methods appear to be at least as efficient as, or mor e efficient than, the evolutionary pr ogram without self-adaptation on the chosen environment.

(a)

(b)

Figure 3: Consecutive *t*-test scores comparing the results of each self-adaptive method to the baseline results without any self-adaptation on the environment (101110011101)*. (a) No self-adaptation vs. selective self-adaptation (b) No self-adaptation vs. multi-mutational self-adaptation. Positive scores favor the method without self-adaptation while negative scores favor the self-adaptive methods. Grey areas under the graphs show generations where the *P*-value was less than 0.05. *T*-scores across generations are correlated and thus no definitive statistical conclusion can be firmly stated. The results do justify an expectation that further analysis will indicate statistically significant differences in favor of the self-adaptive methods.

interpreting these data because they represent a sequence of dependent trials. Figure 4 indicates the score of the best machine in the population at each generation averaged over all trials with each method on the environment (101100111000110010)*. The results are similar to those depicted in Figure 2. Figure 5 indicates the relevant *t*-scores comparing the self-adaptive methods with the base-line method for this more complex environment.

4 DISCUSSION

The practicality of evolutionary optimization algorithms can be significantly increased through the incorporation of self-adaptive parameters that determine how each parent will distribute future trials (Bäck and Schwefel 1993). Including such parameters frees the human operator from having to select mutation distributions (or genetic operators in genetic algorithms) ad hoc. The majority of efforts in self-adaptation have pertained to real-valued continuous function optimization problems

Figure 4: The fraction correct of the best machine in the population at each generation averaged over all 50 trials with each method applied to the environment (101100111000110010)*. Both self-adaptive methods appear to be at least as efficient as or mor e efficient than the evolutionary pr ogram without self-adaptation on the chosen environment.[1]

(Schwefel 1981; Bäck and Schwefel 1993; Fogel et al. 1991; Saravanan and Fogel 1994), but they can be extended to discrete combinatorial optimization problems such as the evolution of finite state machines for time series prediction.

Self-adaptation on continuous representations allows for parents to continue to generate offspring in directions on the adaptive landscape (error surface) that have proved useful in the past. It essentially serves as a memory of previous trajectories; those that have worked well recently are reinforced while those that have not generated useful trial solutions are purged from the population. But "direction" is difficult to apply to discr ete representations. Although it might be useful in some particular real-valued continuous optimization problem to iteratively increase the value of a certain parameter (e.g., move toward increasingly greater values of *x* while searching for the minimum of *f(x)*), it is not, by analogy, useful to continue changing an output symbol or next-state transition if such changes have been of value in the past (cf. Lenat 1983).

Self-adaptation has proved useful on discrete structures (e.g., finite state machines) when a possibility for freezing parameters has been included (Angeline and Pollack 1993), prohibiting mutation to certain components and thereby maintaining informational gains held within the coding structure. Rather than mandating either the extreme of completely freezing parameters or the extreme of mutating all parameters with equal probability, the self-adaptive methods examined in the current investigation can transition between these extremes. In essence, the methods allow for a gradual freezing of useful input-output and next-state transitions.

The efficiency of any evolutionary optimization algorithm is dir ectly dependent on the shape of the adaptive landscape being searched and the mutation operations that are used to search the state space. It is crucial that there be a strong functional relationship between each parent and its offspring, while simultaneously offering the potential for nearly continuous

(a)

(b)

Figure 5: Consecutive *t*-test scores comparing the results of each self-adaptive method to the baseline results without any self-adaptation on the environment (101100111000110010)*. (a) Selective self-adaptation vs. no self-adaptation. (b) Multi-mutational self-adaptation vs. no self-adaptation. Positive scores favor the self-adaptive methods while negative scores favor the method without self-adaptation. See Figure 3 for a discussion of the interpretation of these data.

functional diversity (Fogel 1988; and others). This can often be accomplished by the use of zero mean multivariate Gaussian mutations on real-valued function optimization problems (Fogel and Atmar 1990; Bäck and Schwefel 1993; Fogel and Stayton 1994; and others). Maintaining functional links between parents and offspring when using discrete representations is more difficult. The pr oposed self-adaptive methods may provide a general mechanism for achieving this end. The preliminary results appear to indicate improved convergence rates when using either self-adaptive method as compared to failing to use any such method. More careful assessment of the statistical significance of these r esults, extensions to more complex environments and comparisons between the realized mutation variance (i.e., the mean number of imposed mutations per machine) remain for further investigation.

Footnotes

[1] The results for the multi-mutative and non-adaptive techniques in this experiment were incorrectly presented in Fogel et al. (1994). The current depictions in Figures 4 and 5 correct any discrepancies.

References

Angeline, P. J. and J. B. Pollack (1992). The Evolutionary Induction of Subroutines. In *Proceedings of the 14th Annual Conference of the Cognitive Science Society*, 236-241. Hillsdale, NJ: Lawrence Erlbaum Associates.

Angeline, P. J. and J. B. Pollack (1993). Evolutionary Module Acquisition. In *Proceedings of the Second Annual Conference on Evolutionary Programming*, eds. D.B. Fogel and W. Atmar, 154-163. La Jolla, CA: Evolutionary Programming Society.

Angeline, P. J. and J. B. Pollack (1994). Coevolving High-Level Representations. In *Artificial Life III*, ed. C.G. Langton, 55-71. Reading, MA: Addison-Wesley.

Bäck, T. and H.-P. Schwefel (1993). An Overview of Evolutionary Algorithms for Parameter Optimization. *Evolutionary Computation* 1: 1-24.

Bäck, T., F. Hoffmeister and H.-P. Schwefel (1991). A Survey of Evolution Strategies. In *Proceedings of the Fourth International Conference on Genetic Algorithms*, eds. R. K. Belew and L. B. Booker, 2-9. San Mateo, CA:Morgan Kaufmann.

Bremermann, H. J. (1962) Optimization through Evolution and Recombination. In *Self-Organizing Systems*, eds. M.C. Yovits, G.T. Jacobi, and G.D. Goldstine, 93-106. Washington D.C.:Spartan Books.

Fogel, D. B. (1988). An Evolutionary Approach to the Traveling Salesman Problem. *Biological Cybernetics* 60: 139-144.

Fogel, D. B. (1991). *System Identification through Simulated Evolution: A Machine Learning Approach to Modeling*. Needham, MA: Ginn Press.

Fogel, D. B. (1994). Asymptotic Convergence Properties of Genetic Algorithms and Evolutionary Programming: Analysis and Experiments. *Cybernetics and Systems* 25: 389-407.

Fogel, D. B. (1995). *Evolutionary Computation: Toward a New Philosophy of Machine Intelligence.* Piscataway, NJ: IEEE Press.

Fogel, D. B. and J.W. Atmar (1990). Comparing Genetic Operators with Gaussian Mutations in Simulated Evolutionary Processes Using Linear Systems. *Biological Cybernetics* 63: 111-114.

Fogel, D. B., L.J. Fogel and J.W. Atmar (1991). Meta-Evolutionary Programming. In *Proceedings of the 25th Asilomar Conference on Signals, Systems and Computers*, ed. R.R. Chen, 540-545. San Jose, CA: Maple Press.

Fogel, D. B., L.J. Fogel, W. Atmar and G.B. Fogel (1992). Hierarchic Methods of Evolutionary Programming. In *Proceedings of the First Annual*

Conference on Evolutionary Programming, eds. D.B. Fogel and W. Atmar, 175-182. La Jolla, CA: Evolutionary Programming Society.

Fogel, D. B. and L.C. Stayton (1994). On the Effectiveness of Crossover in Simulated Evolutionary Optimization. *BioSystems* 32: 171-182.

Fogel, L. J. (1962). Autonomous Automata. *Industrial Research* 4: 14-19.

Fogel, L. J. (1964). On the Organization of Intellect. Doctoral Dissertation, UCLA.

Fogel, L. J., A. J. Owens and M. J. Walsh (1966). *Artificial Intelligence thr ough Simulated Evolution*, NY: John Wiley.

Fogel, L. J., D. B. Fogel and P. J. Angeline (1994). A Preliminary Investigation on Extending Evolutionary Programming to Include Self-Adaptation on Finite State Machines. *Informatica* 18: 387-398.

Fraser, A. S. (1957). Simulation of Genetic Systems by Automatic Digital Computers. I. Introduction. *Australian Journal of Biological Sciences* 10: 484-491.

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

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

Lenat, D. B. (1983). The Role of Heuristics in Learning by Discovery: Three Case Studies. In *Machine Learning*, eds. R.S. Michalski, J.G. Carbonell, T.M. Mitchell, 243-306. Palo Alto, CA:Tioga Publishing.

Rechenberg, I. (1965). Cybernetic Solution Path of an Experimental Problem. Royal Aircraft Establishment, Library Translation 1122, Farnborough, Hants, U.K.

Rechenberg, I. (1994). personal communication, Technical University of Berlin, Germany.

Saravanan, N. and D.B. Fogel (1994). Learning Strategy Parameters in Evolutionary Programming: An Empirical Study. In *Proceedings of the Third Annual Conference on Evolutionary Programming*, eds. A.V. Sebald and L.J. Fogel, 269-280. River Edge, NJ: World Scientific Publishing.

Schaffer, J. D. and A. Morishima (1987). An Adaptive Crossover Distribution Mechanism for Genetic Algorithms. In *Proceedings of the Second International Conference on Genetic Algorithms*, ed. J.J. Grefenstette, 36-40. Hillsdale, NJ: Lawrence Erlbaum.

Schwefel, H.-P. (1965). Kybernetische Evolution als Strategie der Experimentellen Forschungin der Strömungstechnik. Diploma Thesis, Technical University of Berlin.

Schwefel, H.-P. (1981). *Numerical Optimization of Computer Models*, Chichester, U.K: John Wiley.

A.V. Sebald (1991) personal communication, UCSD.