Computationally Efficient Evolutionary Algorithms: Enhanced by On-line Machine Learning

Jong-Han Kim and Taehoon Kim

***Abstract*— An efficient evolutionary optimization algorithm of which the convergence is improved is proposed. A "machine" which learns the parameter-cost relations on-line is implemented inside the evolutionary algorithm, and the machine reuses the parameter-cost information as training sets to update the hypothesis functions. As the populations converge and regression accuracy improves, some portion of the cost evaluations are substituted with machine-learned regressions, and they are put into the selection process. This significantly reduces the computational load and running time, because the training/computation of the machine is much cheaper than the actual cost function evaluations. Also, this implies that the effective number of offsprings can be easily increased, which leads to improved convergence with little increase of the computational load. The improved convergence is shown by a simple numerical examples and a practical design problem.**

I. Introduction

Evolution-based optimization methods have a number of advantages over traditional hill-climbing (e.g., gradient descent, Newton's method) techniques. Unlike the gradient-based methods, evolutionary algorithms can easily escape from local minima, eventually converging toward the global minimum. Also, the evolutionary algorithms can be applied to any type of cost functions, since they do not require the gradient or any higher order information. For cost functions with discontinuities or peaky noise components, the gradient is not always well-defined or computed with poor numerical accuracy, which frequently results in failure of gradient-based algorithms[1].

However, evolutionary computation requires numerous cost function evaluations, which normally yields to heavy computational burdens. The cost functions encountered in practical optimization problems typically involve time-consuming computations such as numerical integration or large matrix inversion, therefore frequent evaluation of the cost function is not desirable. In standard evolutionary algorithms, a population of sample points evolves to select better individuals based on the fitness measure, by the processes called selection/reproduction. These processes inherently impose repetitive evaluations of the cost functions within the previously-visited domains, of which the ranges reduce as the population evolves toward next generations. However, conventional evolutionary algorithms use the computed cost information only for the fitness evaluation and dispose of them without storing.

We claim that this wastefulness can be improved by using machine learning techniques. A "machine" which learns the parameter-cost relations is implemented inside the algorithm. As the algorithm evaluates the cost function repetitively, the machine reuses those information as training sets to update the hypothesis functions. Since the region in which the cost is evaluated contracts around the optimum as the generation number increases, the regression performance around the optimum gradually improves as they evolve; i.e., the machine learns the large-scale macroscopic views of the cost functions in the early stages of the evolution, and the scale reduces as the population evolves, eventually achieving a very accurate approximation around the optimum in the local microscopic views. As the regression accuracy improves, some portion of the cost evaluations are substituted with machine-learned regressions, and will be put into the selection process. This significantly reduces the computational load and running time, because the training/computation of the machine is much cheaper than the actual cost function evaluations. The proposed algorithm is first shown using a simple numerical example, then applied to a practical autopilot design optimization problem.

II. Proposed Scheme

A. Evolutionary Algorithms

Typical evolutionary algorithms find optimal solutions by iterating the following steps.

Conventional Evolutionary Algorithm:

1) Initialization
2) Offspring generation (N_λ)
3) Cost evaluation
4) Fitness evaluation and selection for the next generation ($N_\mu + N_\lambda \longmapsto N_\mu$: number of parents)
5) Check termination condition

where N_μ and N_λ represent the number of parents and offsprings in each generation. Increasing N_λ typically leads to rapid convergence in a small number of generations, however, the computational demands per each generation increases linearly with N_λ, thus actual convergence rate (convergence per unit time, or convergence per number of function evaluations) does not reduce significantly. The proposed algorithm introduces a way to increase effective N_λ without increasing the computational load linearly.

This work was conducted as a term project for CS229 machine learning class. The authors are grateful to Prof. Min-Jea Tahk at KAIST for his guidance and discussions on evolutionary algorithms and applications.

J.-H. Kim and T. Kim are with the Department of Aeronautics and Astronautics, Stanford University, CA 94305. {jonghank, taehoonk}@stanford.edu

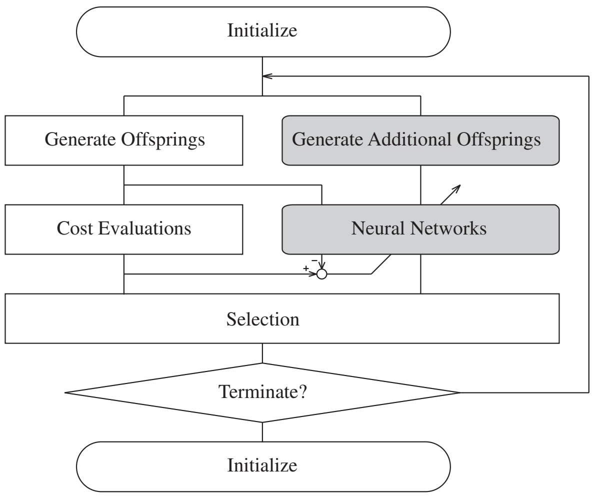

Fig. 1. Computationally efficient evolutionary algorithm (Shaded areas represent the additional processes)

B. Efficient Evolutionary Algorithm

In the proposed algorithm, we add a machine which learns the parameter-cost relations, and consequently several additional steps are needed to train it and use it. Given that the computational cost required by the machine training/computation is much cheaper than the cost evaluation (a reasonable assumption for practical optimization problems with complex cost functions), the load increased by these extra steps is assumed to be negligible. The flow chart is shown in Fig. 1.

Efficient Evolutionary Algorithm:

1) Initialization
2) Offspring generation (N_λ)
3) Cost evaluation *(and machine training)*
4) *Additional offspring generation ($N_{\lambda'}$)*
5) *Cost evaluation by additional offspring (by machine computation)*
6) Fitness evaluation and selection for the next generation ($N_\mu + N_\lambda + N_{\lambda'} \longmapsto N_\mu$)
7) Check termination condition

C. Multilayer Feedforward Neural Networks

Multilayer feedforward neural networks with a single hidden layer are implemented to learn and approximate the parameter-cost relations. The structures are shown in Fig. 2.

$$J(x) = [\ g(w_1^T z) \ \ g(w_2^T z) \ \ \cdots \ \ g(w_{N_{hu}}^T z) \ \ 1\ ]\ v$$

where $z = [x^T \ 1]^T$, $x \in \mathbb{R}^n$, $w_i \in \mathbb{R}^{n+1}$, $1 \le i \le N_{hu}$, $v \in \mathbb{R}^{N_{hu}+1}$, and $g(y) = 1/(1+e^{-y})$. N_{hu} represents the number of hidden units.

The neural networks are trained on-line as the evolutionary algorithm evaluates the cost function. The gradient descent back-propagation algorithm to minimize the error function $E = \sum_k \frac{1}{2}(y^{(k)} - J^{(k)})^2$ was used[2].

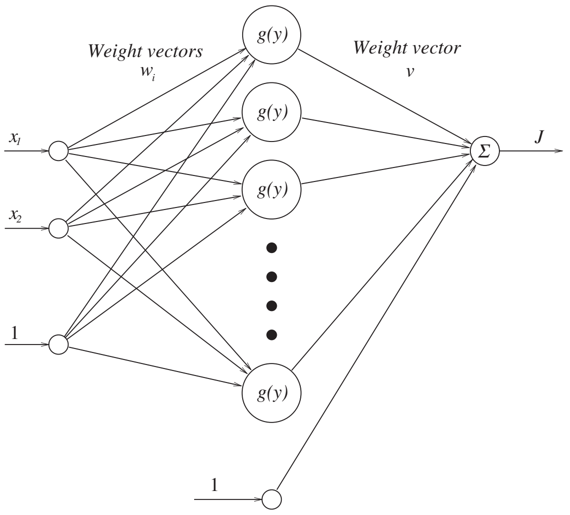

Fig. 2. Multilayer feedforward neural networks

Output layer update:

$$\begin{aligned} \delta &= -\partial E/\partial J^{(k)} = (y^{(k)} - J^{(k)}) \\ v_i &:= v_i + \mu\delta g(w_i^T z^{(k)}) && 1 \le i \le N_{hu} \\ v_{N_{hu}+1} &:= v_{N_{hu}+1} + \mu\delta && \mu\text{: learning rate} \end{aligned}$$

Input layer update:

$$\begin{aligned} \epsilon_i &= \delta v_i g'(w_i^T z^{(k)})) && 1 \le i \le N_{hu} \\ w_i &:= w_i + \nu\epsilon_i z^{(k)} && \nu\text{: learning rate} \end{aligned}$$

where the superscript (k) represents the k-th training set.

III. Numerical Example

The proposed optimization scheme is demonstrated in a simple numerical example.

A. Introductory Example

The first example is a single-variable optimization problem. This function has its global minimum at $x^* = 0$, with $f(x^*) = 0$. Fig 3 shows the function on $-10 \le x \le 10$. Since this function has many local minima, any gradient-based algorithm may not be a good tool for global optimization.

$$\text{minimize } f(x) = 1 - e^{-x^2/20}\cos(2x)$$

Approximation performances are shown by snapshots at several generations, in Fig. 3 (1st generation) and Fig.4 (11th and 21st generation). In both plots, solid curves are obtained by using neural networks, and dotted are the actual cost function curves.

Fig. 5 compares the convergence histories of both conventional and proposed evolutionary algorithms based on 50 runs each. For both of algorithms, $N_\mu = 10$ and $N_\lambda = 30$ are chosen as the evolurionaty algorithm (EA) setup parameters, and $N_{\lambda'} = 30$ and $N_{hu} = 10$ are chosen additionally for the proposed algorithm (which effectively doubles the offspring population). We can observe that the proposed algorithm

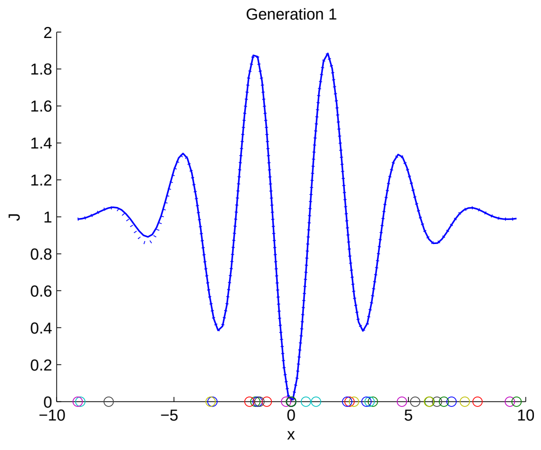

Fig. 3. Cost function $f(x) = 1 - e^{-x^2/20}\cos(2x)$

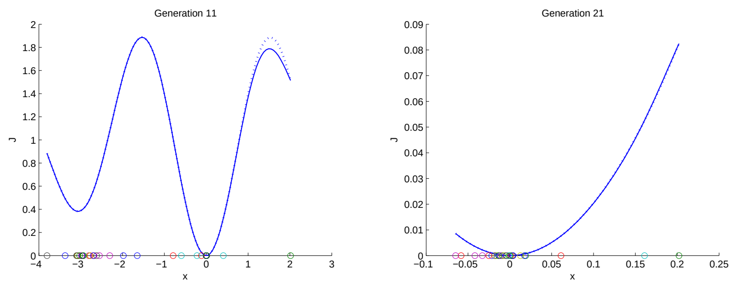

Fig. 4. Snapshots at the 11th and 21st generation

converges earlier than the original algorithm, while some runs of the original algorithm failed to converge to the global minimum.

For this toy example, a function evaluation is just computing simple explicit functions, and does not require heavy computational load; it is even cheaper than training/computing the machine. Therefore, actual time to convergence is increased by applying the new technique. However, in case of practical problems with time-consuming cost functions, the proposed algorithm will reduce the actual time to convergence. We will see this in the next chapter.

IV. Application - Autopilot Design

A practical autopilot design problem is chosen as the application problem.

A. F-15 Longitudinal Dynamics

Fig. 7 shows the normal acceleration control loop of an F-15 fighter.[3] The design parameters are the amplifier gain S_{amp}, and the rate gyro gain S_{rg}.

TABLE I

Numerical data for a cruise flight condition

h(m)	C_L	C_{L_α}	$C_{L_{\delta_e}}$	C_D
6,096	0.24	4.17	0.4	0.05
U(m/sec)	C_{D_α}	C_{m_α}	$C_{m_{\delta_e}}$	$\frac{c}{2U}C_{m_q}$
189.63	0.35	-0.29	-0.5	-0.0512

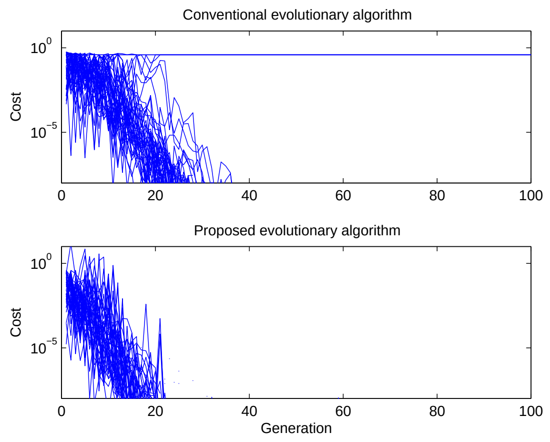

Fig. 5. Convergence histories of conventional vs. proposed algorithm

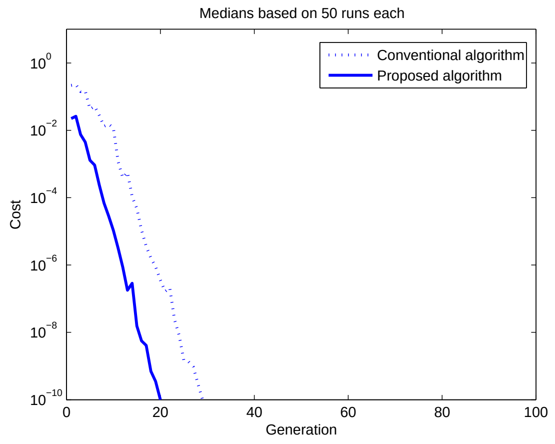

Fig. 6. Convergence histories (median)

Transfer function

$$\frac{\dot{\theta}}{\delta_e} = \frac{\frac{mU}{Sq}C_{m_{\delta_e}}s - C_{m_{\delta_e}}C_{z_\alpha} + C_{m_\alpha}C_{z_{\delta_e}}}{\left(\begin{array}{c}\frac{mUI_y}{(Sq)^2c}s^2 - (\frac{I_y}{Sqc}C_{z_\alpha} + \frac{mU}{Sq}\frac{c}{2U}C_{m_q})s \\ +(\frac{c}{2U}C_{m_q}C_{z_\alpha} - \frac{mU}{Sq}C_{m_\alpha})\end{array}\right)}$$

$$\frac{a_z}{\delta_e} = \frac{\frac{I_y}{Sqc}C_{z_{\delta_e}}s^2 - \frac{c}{2U}C_{m_q}Cz_{\delta_e}s + C_{m_{\delta_e}}C_{z_\alpha} - C_{m_\alpha}C_{z_{\delta_e}}}{\frac{180g}{\pi U}\left(\begin{array}{c}\frac{mUI_y}{(Sq)^2c}s^2 - (\frac{I_y}{Sqc}C_{z_\alpha} + \frac{mU}{Sq}\frac{c}{2U}C_{m_q})s \\ +(\frac{c}{2U}C_{m_q}C_{z_\alpha} - \frac{mU}{Sq}C_{m_\alpha})\end{array}\right)}$$

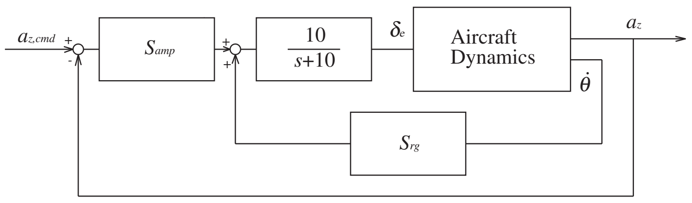

Fig. 7. Normal acceleration control loop for F-15 aircraft

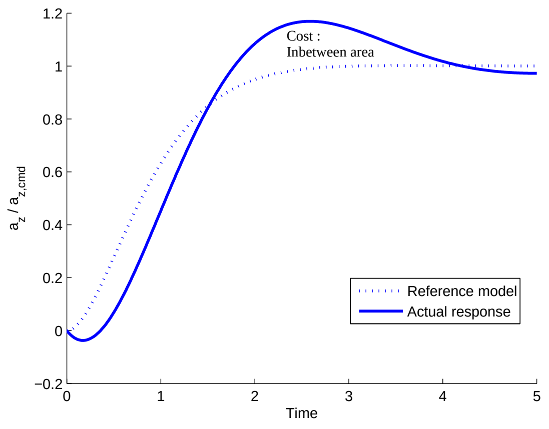

Fig. 8. Cost function for certain S_{amp} and S_{rg}

B. Cost Function

The desired reference model was set by a second order model with $\omega_n = 2(rad/s)$ and $\zeta = 0.9$.

$$\frac{a_{z,ref}}{a_{z,cmd}} = \frac{\omega_n^2}{s^2 + 2\zeta\omega_n s + \omega_n^2}$$

The two parameters (S_{amp} and S_{rg}) are to be optimized so that the response signal is as close to the reference signal as possible. For faithful implication of this objective, the optimization problem is defined as follows[4]. A graphical interpretation is presented in Fig. 8.

$$\text{minimize } J(S_{amp}, S_{rg}) = \int_{t_i}^{t_f} |a_z(t) - a_{z,ref}(t)|\, dt$$

C. Improved Convergence

The evolution histories of the parameters by a conventional algorithm are presented in Fig. 9. Note that some runs failed to converge to the global optimum, because of the small offspring population ($N_\mu = 10$ and $N_\lambda = 30$). The evolution by the proposed algorithm (with $N_{\lambda'} = 30$ and $N_{hu} = 10$) are shown in Fig. 10, where all the trial runs converged to the correct global optimum. We can observe that the convergence is significantly improved by the proposed algorithm, with little increase of computational load. For this specific problem, less than 10% of computational load is additionally required for neural network training and computation. However this number depends on the complexity of cost functions, and will be much smaller for problems with more complicated cost functions.

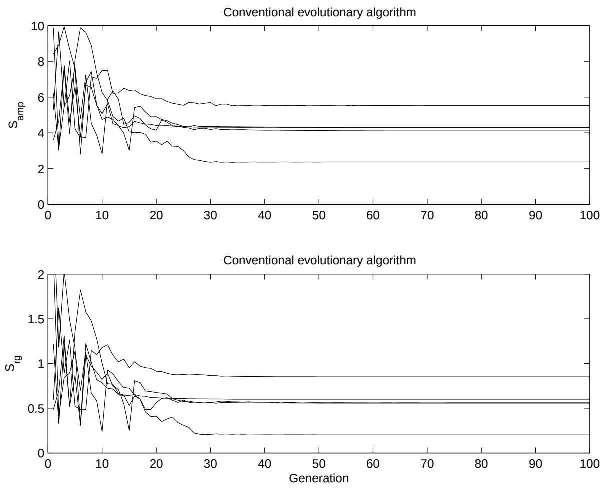

Fig. 9. Evolution histories for a conventional algorithm

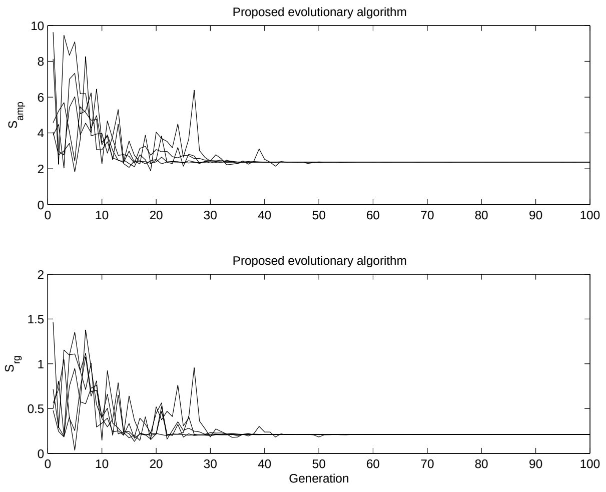

Fig. 10. Evolution histories for the proposed algorithm

V. Conclusion

Computationally efficient evolutionary algorithms are developed. Neural networks are implemented inside the evolutionary algorithm, learning the parameter-cost relations. Online training leads the approximation accuracy to improve as the populations evolve. Then additional offspring populations whose cost values are computed by the neural networks are generated. Because the computational load additionally required for these processes is usually much less than that required for the actual cost evaluations in practical optimization problems, these proceses increase the effective offspring populations with little increase of computational load.

The proposed algorithm was applied to a numerical example and a practical autopilot design problem. It was demonstrated to improve convergence characteristics without severely increasing computational load, compared to a conventional algorithm.

References

[1] T.Bäck, *Evolutionary Algorithms in Theory and Practice*, Oxford University Press, 1996.
[2] S.Haykin, *Neural Networks - A Comprehensive Foundation*, 2nd ed., Prentice Hall, 1999
[3] J.H. Blakelock, *Automatic Control of Aircraft and Missiles*, 2nd ed., Wiley, 1991.
[4] C.S.Park and M.J.Tahk, "A Co-evolutionary Minimax Solver and its Application to Autopilot Design," *Preceeding of AIAA Guidance, Navigation, and Control Conference*, Boston, USA, pp.408-415, Aug. 1998.