Benefits of a Periodic Selection Event in Evolutionary Strategy Algorithms

John Nicholson, *Student Member, IEEE*, and Mark White, *Member, IEEE*

***Abstract*— We compare common, fixed population size evolutionary strategies with a strategy incorporating a growing population size and intermittent selection events. In the latter strategy, the population size grows geometrically and selection free every generation. After a fixed number of generations, a selection event occurs which kills many of the individuals in the population and reduces the population size back to an initial value. The quality of solutions and speed of this algorithm are compared using four real-valued problem domains, to common evolutionary strategy algorithms based on (μ, λ) and $(\mu + \lambda)$, with promising results.**

I. Introduction

Two common evolutionary (CE) algorithms are compared against an algorithm based on a different method of selection – the selection event (SE). In the SE algorithm the population size is not constant, but grows geometrically every generation until the selection event occurs, at which time the population size contracts. The biological inspiration for this algorithm is that of the epidemic – exemplified by the effect of pesticides on insect populations. The typical crop-destroying insect has a short life-span, and consequently a population goes through many generations in a single season. Additionally, such insects are reproductively prolific, leading to a significantly expanded population over several generations.

In many evolutionary strategies, the population size is constant throughout the simulation. Some algorithms, such as Restart CMA [1], will adjust the population size on a run-by-run basis. In the "sustained population dynamics" simulation [2] and other similar simulations, there is no control over the population size at all; instead, time and space are simulated in a 2- or 3-dimensional environment, and when individuals interact with each other they may fight or reproduce, etc. In [3], Schwefel *et al* note that a Darwinian natural selection process resulting in a "normal surplus of births over deaths ... is neither reflected in GAs nor in EP". With the selection event algorithm, we begin to apply this natural phenomenon to computational optimization.

In §II we describe the different algorithms investigated in this paper. In §III we give a brief description of the different environments used to evaluate the performance of the algorithms. In §IV we provide and discuss the simulation results.

John Nicholson is a Ph.D. student in the School of Electrical and Computer Engineering, North Carolina State University, Raleigh, NC 27695-7911, USA. (email: jwnichol@ncsu.edu).

Mark White is with the School of Electrical and Computer Engineering, North Carolina State University, Raleigh, NC 27695-7911, USA. (email: ncsu@markwhite.name).

II. Algorithm Description

A. Commonalities between all algorithms

1) Gene type, parameters, and operations: All of the algorithms used in this paper use a real-valued gene similar to that described in D.B. Fogel's work on evolving a checkers player [4]. This gene consists of a real-value $value$, used as one of the values of $\mathbf{x}$, the input vector to the fitness function; a real-value σ, specifying the standard deviation of a normally-distributed random variable used to mutate $value$; and a real-value τ, used in the mutation of σ. The gene supports only the mutation operation. During mutation, σ is mutated first, then this new σ is used to mutate $value$. Lastly, all individuals have a genome consisting of d genes, where d represents the dimensionality of the optimization problem.

2) Initial value generation: The initial values of $value$, σ, and τ are calculated as shown in (1).

$$\begin{aligned} \sigma_0 &= \left(\frac{\kappa_{coverage}(\beta_{upper} - \beta_{lower})^d}{\lambda_0} \right)^{1/d} \\ \tau &= \kappa_\tau \sigma_0 \\ value &= \sigma_0 N(0,1) \in [\beta_{lower}, \beta_{upper}] \end{aligned} \quad (1)$$

where $\kappa_{coverage}$ is the coefficient of coverage, a scale factor which for all simulations in this paper has a value of 10.0; κ_τ is a scale factor to get τ from the same calculation as σ_0, in all simulations this value is 0.05; β_{lower} and β_{upper} are the lower- and upper-bounds for the particular problem environment; d is the number of dimensions for the problem environment; and $N(0,1)$ is an independent sampling of a standard Gaussian distributed random variable.

3) Tournament selection: In these algorithms, tournament selection is used whenever some individuals need to be chosen in a non-deterministic manner. Tournaments are used because of their simplicity in implementation and generality with respect to problem environments [5, Ch.22 and Ch.24].

B. CE_α algorithm description

This algorithm is based on the well-known (μ, λ) ES [5, Ch.9]. The algorithm is modified to include the use of a tournament selection operator. Parents are selected from the current generation, $P^{(t)}$, by being randomly chosen to participate in a pair-wise (or binary) tournament with replacement. The individual that is the more-fit of the two becomes a parent, and both individuals are again eligible to be chosen randomly for another pair-wise tournament. The selected parent is then mutated, and the offspring placed into the next generation. This is done repeatedly until all members of the population of the next generation, $P^{(t+1)}$, have been generated.

C. CE_β algorithm description

This algorithm is the one used in D.B. Fogel's work evolving a checkers player [4], and is based on the $(\mu+\lambda)$ ES [5, Ch.9]. In this strategy, $P^{(t+1)}$ consists of the best half of the individuals in $P^{(t)}$, unmodified. Each of these individuals also produces a child that is created through mutation of all genes.

D. Selection Event (SE) algorithm description

There are two phases in this evolutionary algorithm: the expand phase and the collapse phase. In the expand phase, which occurs every generation, the population grows. During the collapse phase, which occurs every T_{se}[1] generations, a "tournament-to-the-death" is used to reduce the population size back to the initial value, $|P|^{(0)}$. In the tournament-to-the-death, the less-fit of two randomly chosen individuals is deleted, and the more-fit individual is put back into the population, and may again participate in another tournament-to-the-death.

The outline below is a step-by-step summary of the algorithm.

1) At time t, start with an initial population of $|P|^{(t)}$ individuals.
2) Expand Phase
   a) Each individual in $P^{(t)}$ is mutated $\mathcal{F}_\alpha$ times[2], with the resulting offspring placed into the next generation. Note that the parents are not placed into the next generation[3].
   b) Repeat the above step T_{se} times. When done repeating, you will have $|P|^{(t+T_{se})} = {\mathcal{F}_\alpha}^{T_{se}}|P|^{(t)}$.
3) Collapse Phase
   a) Perform pair-wise tournament-to-the-death to determine which individual is deleted from generation $t + T_{se}$. The more-fit of the two individuals is returned to the population.
   b) The above step is repeated $({\mathcal{F}_\alpha}^{T_{se}} - 1)|P|^{(t)}$ times, which returns the population size to $|P|^{(t)}$.
4) Steps 2-3 describe how the algorithm works for generations t through $t + T_{se}$. These steps are repeated for subsequent generations.

As an example, consider the following parameters: asexual reproduction rate $\mathcal{F}_\alpha = 3.0$, initial population size $|P|^{(0)} = 30$, selection event period $T_{se} = 3$. With these, the population size $|P|$ changes as seen in Table II. As can be seen from this example, $|P|$ temporarily expands to 810 in generation three, and then collapses back to 30 in the same generation. This paradigm also allows for $T_{se} = 1$, where $|P|$ temporarily expands and then collapses within every generation.

[1]Two new parameters are used in this algorithm, and are described in Table I.

[2]For the situations where $\mathcal{F}_\alpha$ is not a whole number, all individuals produce the whole part of $\mathcal{F}_\alpha$ children, and a pair-wise tournament selection with replacement strategy is used to produce the remainder of the individuals.

[3]There is no reason this need be the case. Indeed, the authors have done some simulations where parents were placed into the next generation. However, for the scope of this paper, the statement holds.

TABLE I

NEW PARAMETERS INTRODUCED BY THE SELECTION EVENT ALGORITHM.

Name	Type	Description
$\|P\|^{(0)}$	Integer	The initial population size.
T_{se}	Integer	The periodicity of the selection event.

TABLE II

EXAMPLE SHOWING HOW THE POPULATION SIZE CHANGES FOR THE SE ALGORITHM.

Generation	0	1	2	3	4	5	...
$\|P\|$	30	90	270	810→30	90	270	...
Phase	E	E	E	E →C	E	E	...

III. PROBLEM ENVIRONMENTS

Several different real-valued problem environments were used to evaluate algorithm performance, where performance is meant in two different ways: as optimality of solution, and as the number of function evaluations performed during the search.

All problem environments were taken from [6], though with a different purpose than outlined in that reference. Here, the environments are used to provide a cursory examination of the performance of the new SE algorithm in four different landscapes. All functions have a single global optima, which the algorithms are attempting to find. Since we are only interested in the relative performance of the SE and CE algorithms, the value of this optima is of no concern. The problems are briefly illustrated and described in Figure 1.

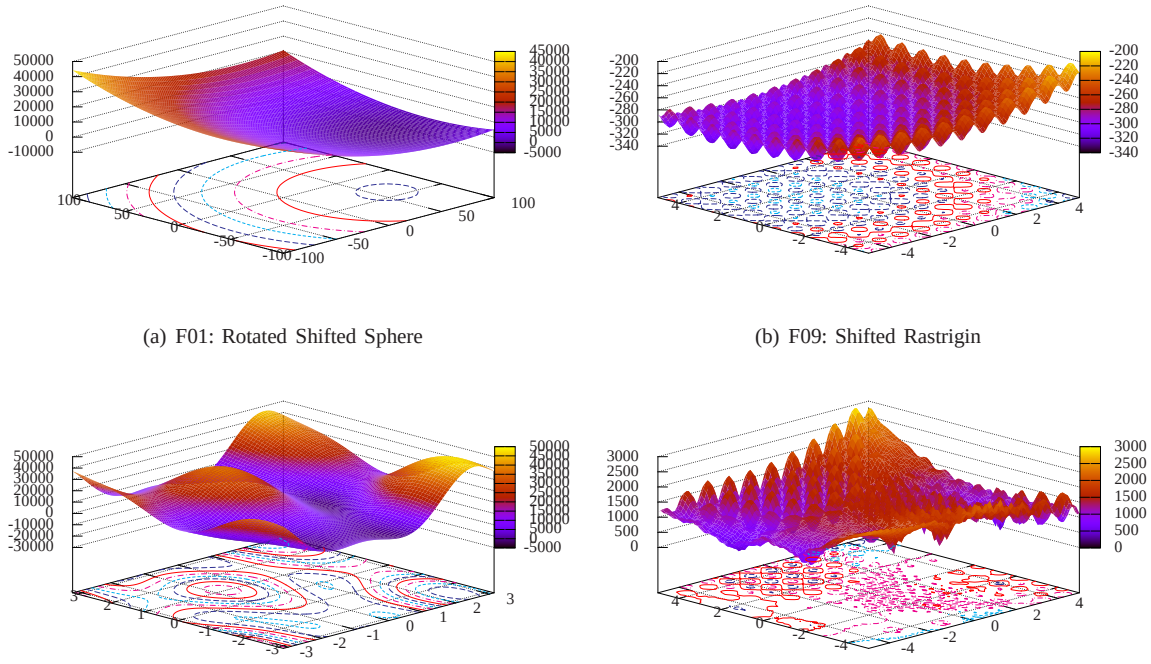
IV. RESULTS

For the selection event algorithm, simulations were run with several different parameter settings to obtain a broader view of its performance. A nomenclature was developed to distinguish these parameter sets on the graphs:

$$[T_{se}]SE[|P|^{(0)}], [\mathcal{F}_\alpha]$$

where the values within the [ ]'s denote the parameters that were varied. For all of the parameter sets evaluated, the population grew from 30 individuals to around 810 individuals, though at different growth rates. In order to compare all three algorithms, the CE algorithms had a population size of 810 individuals. For each of the environments, 12 runs were performed using random seeds for each run.

Figure 2 contains four graphs – one for each problem environment. Within a graph, each data-point represents one run from one of the eleven algorithm variations. All runs were ended after the same predetermined number of generations, which for all simulations was 6000 generations. Figure 2 illustrates both how consistently a particular algorithm concluded its search, and is useful when comparing the quality of solutions achieved by the algorithms. From these plots, we see that the SE algorithms outperform the CE algorithms with respect to the optimality of solution criteria, in all four environments.



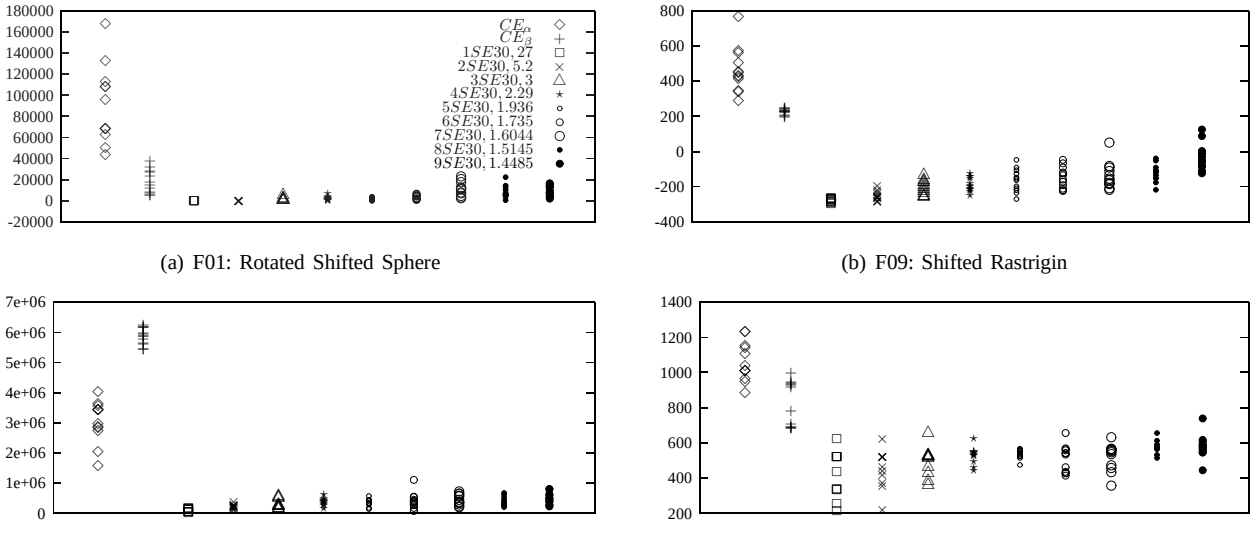
(a) F01: Rotated Shifted Sphere

(b) F09: Shifted Rastrigin

(c) F12: Schwefel 2.13

(d) F15: Hybrid

Fig. 1. Lower-dimension plots of test problem environments. 1(a) illustrates a simple unimodal function of 100 dimensions. 1(b) illustrates a multi-modal function of 50 dimensions. 1(c) illustrates a multi-modal function of 50 dimensions. 1(d) illustrates a multi-modal function of 50 dimensions, made by combining the Rastrigin, Weierstrass, Griewank, Ackley, and Sphere functions.



(a) F01: Rotated Shifted Sphere

(b) F09: Shifted Rastrigin

(c) F12: Rotated Shifted Sphere

(d) F15: Shifted Rastrigin

Fig. 2. Simulation results for the four problem environments. Each data-point represents the population's average fitness at the end of one simulation run.

We also plot the population's average fitness against the number of individuals evaluated[4], as seen in Figure 3 and Figure 5. These plots show how the average fitness changes as the simulation progresses. Of critical importance to note from these figures is that in order to have an equal number of individuals evaluated by each algorithm, the number of generations had to be made unequal. Recall from §II-B that CE_α creates (and hence evaluates) 810 individuals every generation, and from §II-C that CE_β creates 810 individuals initially and 405 individuals each subsequent generation. This means that in order to evaluate the same number of

[4]The number of individuals evaluated is proportional to the number of function evaluations performed, with the constant of proportionality varying by environment.

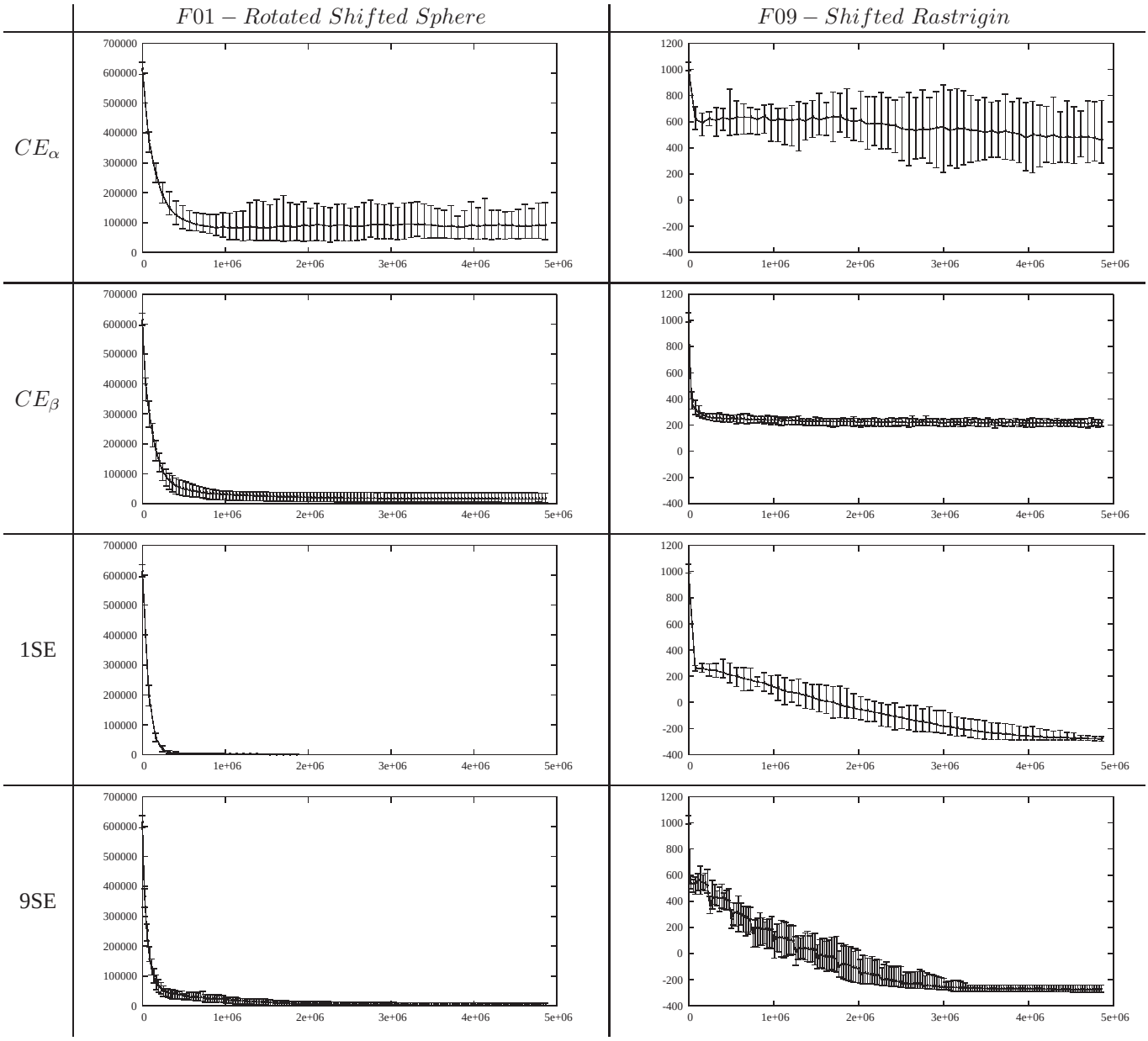


Fig. 3. Simulation results for the Rotated Shifted Sphere and Shifted Rastrigin problem environments. These plots show the population average fitness against the number of individuals evaluated. The plots include error bars for the population's average fitness value, showing the best-average, the average-average, and worst-average of the 12 simulation runs.

individuals, the CE_β algorithm has to run for twice as many generations. Similarly for the SE algorithms, the growth rate $\mathcal{F}_\alpha$ and selection event periodicity T_{se} determine how many individuals are created and evaluated. From these plots we see that (1) the CE_β algorithm has the fastest algorithmic speed, though it also tends to prematurely converge, and that (2) the growth rate parameters for SE have a small effect on the speed of solution[5].

Note that in the SE plots, the population's average fitness "bounces" – see Figure 4 for a close-up view of the data.

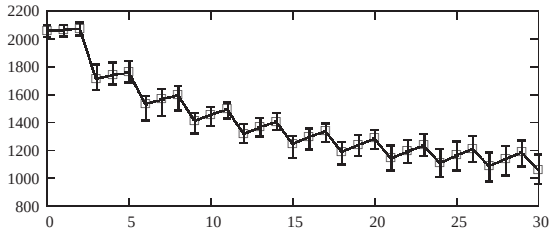


Fig. 4. Close-up view of a simulation run showing how the fitness for the SE algorithms change generation to generation. This view is of the $3SE30,3$ algorithm in the hybrid environment.

[5]Though only the fastest and slowest growth rates simulated are shown in these figures, the intermediate growth rates do not show significantly different results.

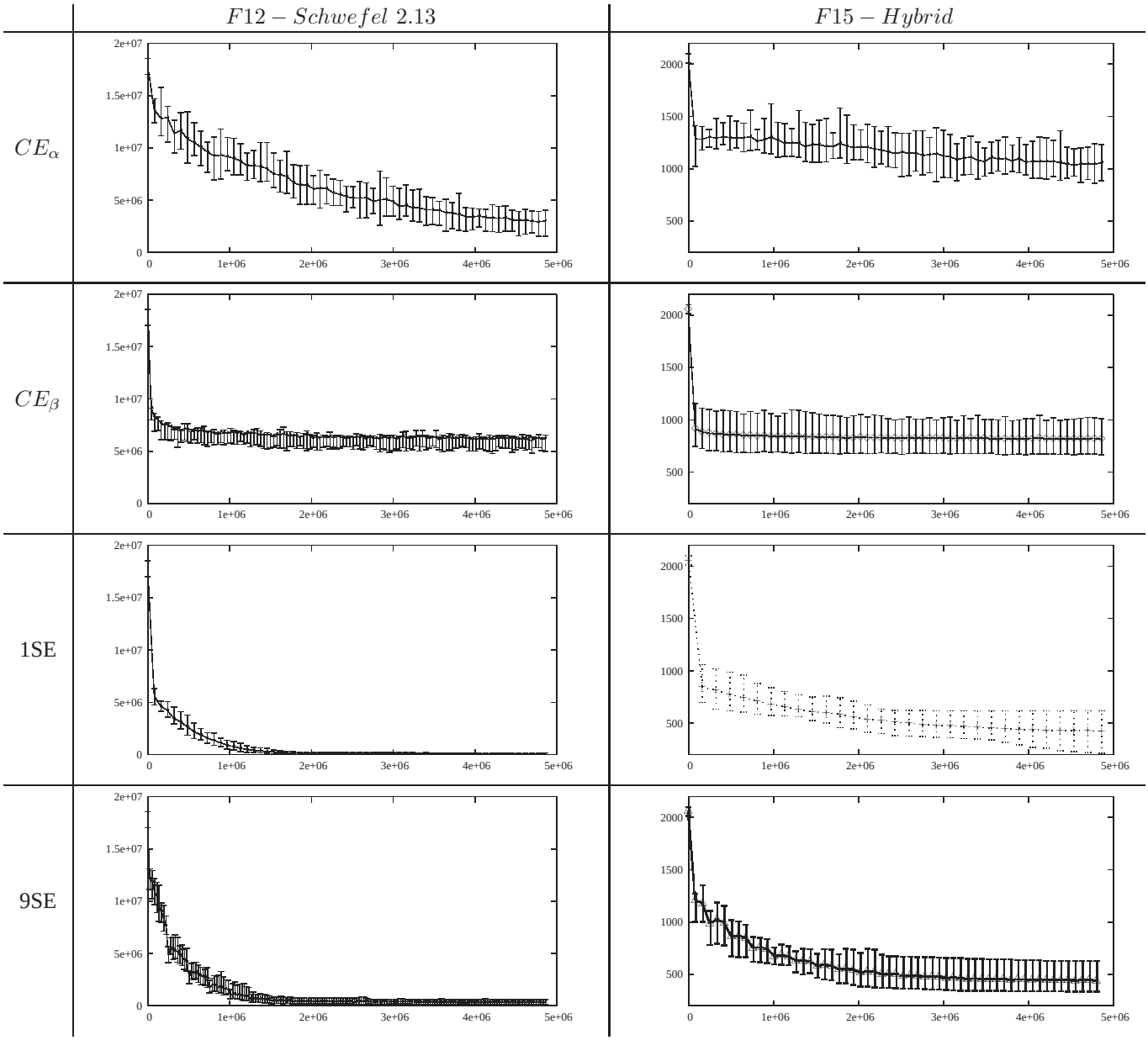


Fig. 5. Simulation results for the Schwefel 2.13 and Hybrid problem environments. These plots show the population average fitness against the number of individuals evaluated. The plots include error bars for the population's average fitness value, showing the best-average, average-average, and worst-average of the 12 simulation runs.

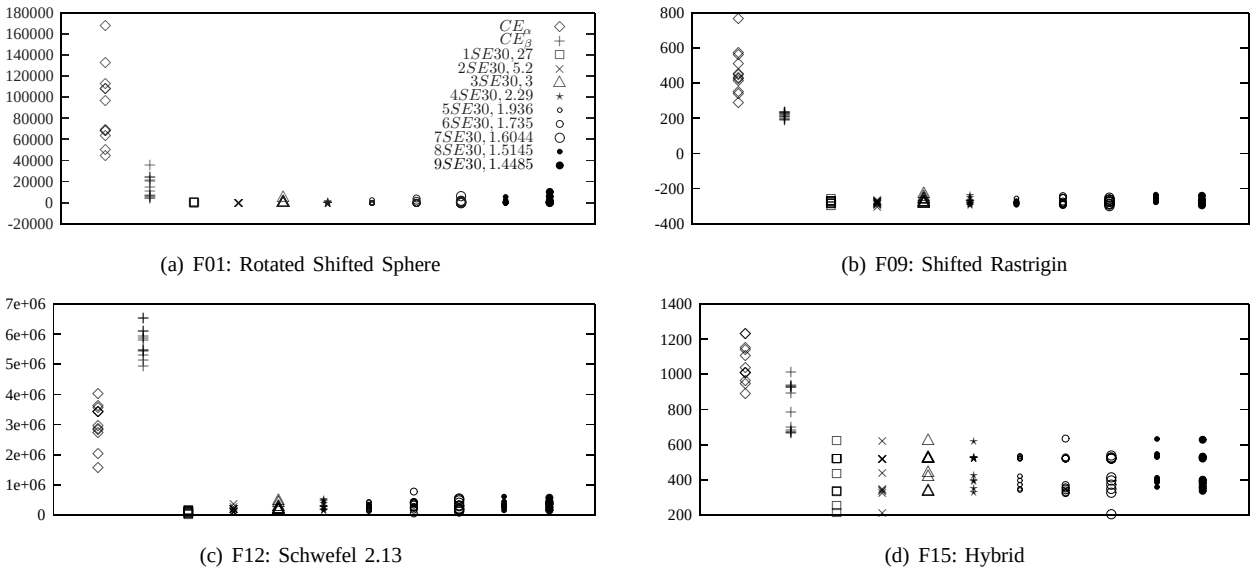
This is a result of the expand and collapse cycles: as the population expands, the average fitness slowly increases[6] since most of the mutations have a negative effect on fitness, at least initially and before subsequent mutations having a positive effect on fitness can be realized; when the population collapses, many of the less-fit individuals die, causing the average fitness to decrease.

Figure 6 is a reproduction of the earlier Figure 2, showing the simulation results for the simulations where an equal number of individuals were evaluated. Within a graph, each data-point represents one run from one of the eleven algorithm variations. All runs were ended after the same predetermined number of individuals (for these simulations, almost five million individuals). From these plots, we see both further evidence that the SE algorithms provide more optimal solutions than the CE algorithms on these problems, and that the growth rate parameters do not greatly effect the optimality of solution found.

[6]Recall that these are all minimization problems

V. Conclusion, Discussion, and Future Work

This paper has introduced a new algorithm, which simulation results have shown performs well in a wide range of problem environments when compared to two well-known algorithms. However, much remains to be investigated. Primarily, why does this algorithm produce more optimal results than the CE algorithms? We believe the improved perfor-



(a) F01: Rotated Shifted Sphere

(b) F09: Shifted Rastrigin

(c) F12: Schwefel 2.13

(d) F15: Hybrid

Fig. 6. Simulation results for the problem environments. Each data-point represents the population's average fitness at the end of one simulation run.

mance is due to (1) allowing mutations to build upon one another before they must show an improvement in fitness (in other words, temporarily allowing neutral or even deleterious mutations), (2) the intense selection pressure of the collapse phase of the algorithm directing the evolution of the solution, and (3) that advantageous adaptations of σ, particularly those adaptations that occur in the early stages of a population expansion, will be more easily "detectable" and therefore survive the selection process. These questions and hypotheses remain the authors' primary research focus.

Additionally, some simple modifications to the algorithm are potentially interesting. First, implementing some selection during generations that previously were expand-only, such as producing eleven children and killing off the least fit child every generation. Second and perhaps more interesting, implementing an in-family selection component, where some of the collapse selection is performed within an ancestral line, as opposed to the current implementation of selection across the entire population.

REFERENCES

[1] A. Auger and N. Hansen, "A restart cma evolution strategy with increasing population size," in *Proceedings of the IEEE Congress on Evolutionary Computation*, 2005.

[2] N. Brodu, "Environmental fitness for sustained population dynamics," in *Proceedings of the IEEE Congress on Evolutionary Computation*, 2005.

[3] H. Schwefel, G. Rudolph, and T. Back, "Contemporary evolutionary strategies," in *Advances in Artificial Life*, F. M. et al, Ed. Berlin: Springer, 1995, pp. 893–907.

[4] K. Chellapilla and D. Fogel, "Evolving neural networks to play checkers without relying on expert knowledge," *IEEE Transactions on Neural Networks*, vol. 10, no. 6, pp. 1382–1391, 1999.

[5] T. Back, D. Fogel, and T. Michalewicz, Eds., *Evolutionary Computation 1: Basic Algorithms and Operators*. Institute of Physics Publishing, 2000.

[6] P. Suganthan, N. Hansen, J. Liang, K. Deb, Y. Chen, A. Auger, and S. Tiwari, "Problem definitions and evaluation criteria for the cec 2005 special session on real-parameter optimization," Nanyang Technological University, Tech. Rep., May 2005.