Steady-State ALPS for Real-Valued Problems

Gregory S. Hornby
University Affiliated Research Center, U.C. Santa Cruz
NASA Ames Research Center, Mail Stop 269-3
Moffett Field, CA USA
gregory.s.hornby@nasa.gov

ABSTRACT

The two objectives of this paper are to describe a steady-state version of the Age-Layered Population Structure (ALPS) Evolutionary Algorithm (EA) and to compare it against other GAs on real-valued problems. Motivation for this work comes from our previous success in demonstrating that a generational version of ALPS greatly improves search performance on a Genetic Programming problem. In making steady-state ALPS some modifications were made to the method for calculating age and the method for moving individuals up layers. To demonstrate that ALPS works well on real-valued problems we compare it against CMA-ES and Differential Evolution (DE) on five challenging, real-valued functions and on one real-world problem. While CMA-ES and DE outperform ALPS on the two unimodal test functions, ALPS is much better on the three multimodal test problems and on the real-world problem. Further examination shows that, unlike the other GAs, ALPS maintains a genotypically diverse population throughout the entire search process. These findings strongly suggest that the ALPS paradigm is better able to avoid premature convergence then the other GAs.

Categories and Subject Descriptors

I.2.8 [**Artificial Intelligence**]: Problem Solving, Control Methods, and Search

General Terms

Algorithms, Performance, Reliability

Keywords

Age, Premature Convergence, Numerical Optimization, Evolutionary Algorithms

1. INTRODUCTION

Since the introduction of the initial Genetic Algorithm (GA) by Holland [6] a number of variations have been introduced with the goal of constructing a better optimizer [2, 5, 17]. While some improvements in performance have been achieved, premature convergence is still a problem. In the Genetic Programming (GP) community, the Age-Layered Population Structure (ALPS) Evolutionary Algorithm (EA) was introduced as a way of addressing, and significantly reducing, premature convergence [7]. ALPS differs from other EAs in that it segregates the population into multiple layers using a novel measure of age, and it reduces premature convergence by introducing a new group of randomly generated individuals into the bottom layer at regular intervals.

One advantage of ALPS is that it does not use any details of the representation and, thus, can be used with any sort of encoding scheme: GP programs, bit-strings, real-valued vectors of parameters, sequences of integers (such as for solving a scheduling or TSP problem), and anything else. Having been demonstrated to be effective on GP problems with GP representations [12, 16, 22], the main objective of this paper is to introduce the ALPS paradigm to the GA community by comparing ALPS against leading GA systems on real-valued, GA problems. To demonstrate that ALPS works well on real-valued problems we compare it against a basic GA, the Covariance Matrix Adaptation Evolution Strategy (CMA-ES) [5] and Differential Evolution (DE) [17] on five challenging, real-valued functions and on one real-world problem. Both CMA-ES and DE use information from the genotypes in the population to direct search and achieve improved performance. Intuitively we might expect that these domain specialized algorithms (CMA-ES and DE) will always outperform a domain independent algorithm (ALPS) but instead we find that ALPS is the better algorithm on multimodal problems. These results suggest that ALPS should be the GA of choice on hard, real-world problems.

The second objective of this paper is to present a steady-state variation of the original ALPS algorithm. One of the advantages of a Steady-State GA is that it does not have the synchronization issues of a Generational GA [19], and this can be quite beneficial for implementation on distributed systems such as Beowulf clusters. To implement a steady-state version of ALPS we needed to change those parts of the algorithm that depended on evolution happening in discrete generations. One of the changes was to method of measuring age and here we switched from updating ages at the end of each generation to basing age on a count of the number of evaluations divided by the population size. Another change we made was in the method by which individuals are moved up layers, and here we added checks to prevent replacement of recently moved individuals. We hope that this description of a steady-state version of ALPS will be useful to those for whom a steady-state EA is preferable to a generational one.

The rest of this paper is organized as follows. In Section 2 we discuss search algorithms and robustness. In Section 3 we describe the ALPS paradigm and then in Section 4 we present our steady-state version of ALPS. Sections 5 and 6 describe the experimental setup and test problems. Results of the experiments are given in Section 7 and we find that CMA-ES and DE outperform ALPS on

the uni-modal test problems but ALPS is much better on the three multi-modal test problems and on the real-world problem. Section 8 contains a discussion and thoughts on future directions, and our conclusion is in Section 9.

2. IMPROVING SEARCH

In the field of optimization algorithms, one type of algorithmic improvement is that of increasing the *speed* at which problems of solvable difficulty can be solved. This is shown in papers in which the comparison is on which algorithm can find the global optima of a benchmark problem in the fewest number of evaluations. Another type of algorithmic improvement is increasing the *robustness* of the algorithm. That is, either increasing the reliability of finding the global optima or being able to find better results then other algorithms. ALPS was developed to be a more robust EA, especially on hard problems, although not necessarily the fastest one on easy problems. Especially as the computing power of a typical processor continues to increase and computer clusters are becoming more common there is interest in improving robustness at the cost of being slower.

Combating *premature convergence* is one attribute of a robust optimization algorithm. GAs cannot generally be run effectively beyond some number of evaluations and this is because the population converges prematurely to an optimum and the further application of variation operators cannot generate viable individuals on a different (and better) optima. To reduce the premature convergence problem, many EA practitioners will try increasing the mutation rate, the mutation size and/or the population size. Increasing the mutation rate will keep diversity high and keep the population from converging quickly but it is just as likely to replace good alleles and building blocks as bad ones. Also, if the mutation size is too large then the mutation operator will not create offspring near its parent and be unable to explore narrow fitness peaks. Using a larger population simply increases the number of generations before it converges with the hope of increasing the likelihood of finding the fitness peak with global optima. But selecting the best population size can be a challenge: using too large of a population on a simple problem results in search taking much longer than necessary, and on difficult problems the minimum size needed may be too large to be feasible.

Another way to reduce premature convergence is to use some system for explicitly maintaining genotypic diversity in the population throughout an evolutionary run. Diversity of the population can be maintained by modifying the replacement strategy, such as with *preselection* [1], *crowding* [3], or *deterministic crowding* [15]. Another approach is the use of *sharing functions*, which modify the fitness of individuals based on their genotypic similarity [4]. Alternatively, the population structure can be modified, such as with spatially structured populations in which individuals have a location and are restricted to interacting with their neighbors. In addition, there are many newer methods which are variations of these basic approaches. While these methods work to varying degrees, ultimately, all such methods are limited to discovering solutions that are within the basin(s) of attraction of the initial population.

One way to try to escape the basin(s) of attraction of the initial population is to build a model of the fitness landscape, such as with CMA-ES. But this adds the challenge of being able to build a sufficiently good approximation. Another way of breaking out of the basin of attraction of the current population is by regularly introducing new individuals, with new genetic material, into the population and this is the approach that is taken by the Age Layered Population Structure (ALPS). In the next section we describe how ALPS works.

Table 1: Different systems for setting the age-limits for each age-layer and the corresponding maximum age in each layer for an `age-gap` of 1.

	Max age in layer						
Aging-scheme	0	1	2	3	4	5	6
Linear	1	2	3	4	5	6	7
Fibonacci	1	2	3	5	8	13	21
Polynomial (n^2)	1	2	4	9	16	25	49
Exponential (2^n)	1	2	4	8	16	32	64

3. ALPS PARADIGM

We developed the ALPS paradigm as a more robust EA that is adept at avoiding premature convergence [7]. This ability is most easily noticed when using a small population or when performing extremely long evolutionary runs. With ALPS, a novel measure of age is used to segregate individuals into different age-layers and then, at regular intervals, the youngest layer is replaced with randomly generated individuals. We first review the measure of age and then how ALPS manages individuals in its age layers.

3.1 Measuring Age

Over the years *age* has been used in various EAs to try to improve performance [8, 10, 11, 13]. In these systems all individuals, whether they are created randomly or through mutation or recombination, start with an age value of 1. After each generation in which an individual is kept in the population (eg. it is not changed through mutation or recombination) its *age* is increased by one. Thus in these systems *age* is a measure of how long a particular individual with the same set of alleles has been in the population.

With ALPS, *age* is a measure of how long an individual's family of genotypic material has been in the population. Randomly generated individuals, such as those that are created in the initial generation of a canonical EA run, start with an age of 1. Each generation that an individual stays in the population (such as through elitism) its age is increased by one. Individuals that are created through mutation or recombination take the age of their oldest parent and add one to it. For example, if individual $\texttt{Ind}_A$, age 23, and individual $\texttt{Ind}_B$, age 28, are selected as parents for recombination then their offspring, $\texttt{Ind}_C$, will be assigned an age of 29. At the end of the reproduction phase $\texttt{Ind}_A$ will have its age increased to 24 and $\texttt{Ind}_B$ will have its age increased to 29 and $\texttt{Ind}_C$ will keep its age of 29. In contrast, other systems would assign an age of 1 to $\texttt{Ind}_C$.

3.2 Generational ALPS

With the ALPS paradigm, the population is segregated into multiple age layers, with each layer having an upper age limit. The EA acts on each age layer somewhat independently of the others, with an exception being that parents can be selected from both the current layer and the layer below. When an individual is too old for its current layer, the algorithm tries to move it to the next layer up. Also, at regular intervals the bottom layer is replaced with a new sub-population of randomly generated individuals, each with an age of 1. We now describe the algorithm in more detail.

In setting up an ALPS run, the number of age layers and the age limits for each layer are parameters that are set by the user. Different systems can be used for setting these values, such as by using linearly, polynomially or exponentially increasing limits (Table 1). To keep the size of the population and number of layers manageable, and since there is generally little need to segregate individuals

which are within a few "generations" of each other, these values are then multiplied by an `age-gap` parameter. Also, there is no maximum age for the last layer and a single-layer version of ALPS operates exactly as the standard EA. For example, in a system with five layers, a polynomial aging-scheme and an age gap of seven the maximum ages for the five layers are: 7, 14, 28, 56, and ∞.

With an ALPS-EA, evolution occurs in each layer somewhat independently of the others. When selecting parents to create new individuals for a given layer, parents are selected from individuals in that layer as well as the previous one. Restricting the selection of parents in this way limits selection competition to those individuals of similar ages and prevents the older individuals from dominating the younger ones. Once individuals have been in the "bottom" layer of the population for as many generations as its age limit, all individuals in this layer are replaced with randomly created individuals. For example, with the example in the previous paragraph, the bottom layer is replaced with a new group of randomly created individuals every 7 generations.

When an individual, I_A, becomes too old for its current layer, or it is selected for replacement by a new individual, an attempt is made to move it up to the next highest layer. This attempt consists of finding an individual, I_B, in the next highest layer that is either too old for that layer or less fit than the individual being moved up (I_A). If such an individual I_B is found then I_A is moved up and takes individual I_B's place otherwise, if no such I_B exists, individual I_A is overwritten and discarded. If an individual I_B is replaced by an individual I_A in this manner then, in being replaced by I_A, I_B is tested to see if it can be moved up to the next higher layer. For example, consider a population with 10 layers of 25 individuals per layer. If individual 2 in layer 0, $I_{0,2}$, is replaced then it is tested against all individuals in layer 1. Assume that there is an individual, $I_{1,7}$, in layer 1 such that either $I_{0,2}$ has better fitness then $I_{1,7}$ or $I_{1,7}$ is too old for layer 1. In this case, first an attempt is made to move $I_{1,7}$ up to layer 2 – and if $I_{1,7}$ can replace an individual in layer 3, then an attempt is made to move that individual up to layer 4, and so on – then $I_{0,2}$ is placed in the population where $I_{1,7}$ used to be. If an individual cannot replace one in the next highest layer, then it is overwritten by the individual that replaces it.

4. STEADY-STATE ALPS

The proposed Steady-State version of ALPS (ALPS-SS) is much like the generational version of ALPS, with one main difference being in how age is calculated. With Generational ALPS, the age of all individuals is increased at the end of the generation. Since there are no explicit generations with ALPS-SS, we keep track of the number of evaluations. Age is then calculated by taking the number of evaluations in which an individual's genetic material has been around and dividing it by the size of the population. Randomly generated individuals store the number of evaluations that have been performed so far, and individuals created through mutation and recombination store the smallest value of their parents. The equation for calculating the age of an individual is:

$$age = 1 + (evals_{current} - evals_{created})/popsize \quad (1)$$

Where: $evals_{current}$ is the number of evaluations that have been performed so far; $evals_{created}$ is the number of evaluations that had been performed when the individual's genetic material was first created; and $popsize$ is the number of individuals in the population. A constant of 1 is added so that the age of randomly generated individuals is 1 at creation time.

Using this measure of age, ALPS-SS works as follows. The algorithm starts by configuring the age layers and then creating, and evaluating, an initial, random population. Once the initial population is created and evaluated, `ALPS-SS` enters its main loop which consists of iteratively selecting an index in the population array for which to create a new individual, creating the new individual, and then inserting it in the population. In more detail, this main loop consists of:

```
 1: procedure ALPSSS( )
 2:     while not done do
 3:         Select a target index, i.      ▷ Randomly or sequentially
 4:         if i is in bottom layer & in re-initialization mode then
 5:             Make random individ in the next index of Layer 0.
 6:             Evaluate it.
 7:             if Finished re-initializing Layer 0 then
 8:                 Turn off re-initialization mode.
 9:             end if
10:             Go back to the start of the main loop.  ▷ On line 7.
11:         end if
12:         Decide whether to do mutation or recombination.
13:         Select the appropriate number of parents.
14:         if No valid parents then
15:             Put in re-initialization mode.      ▷ In bottom layer
16:             Put a random individual in the first index.
17:             Evaluate it and go back to the start of the main loop.
18:         else if only one valid parent then
19:             Mutate parent to create child.
20:         else
21:             Make the child using either mutation or recomb.
22:         end if
23:         Evaluate the new individual.          ▷ Will put in slot i
24:         TryMoveUp(i)
25:         individual in slot i ← new individual.
26:     end while
27: end procedure
28: procedure TRYMOVEUP(i)
29:     j ← individ in next highest layer which i can replace.
30:     if Such a j exists then
31:         TryMoveUp(j)
32:         individual in slot j ← individual in slot i
33:     end if
34: end procedure
```

The above algorithm allows for different methods of selecting the parents (eg tournament selection or some form of roulette wheel selection). In addition, elitism can be added either to just the top layer or to all layers.

An additional change that is made to ALPS-SS is the method by which individuals are moved up. In generational ALPS, all individuals that are being replaced in one layer are moved up as a group at the end of the generation. This means that individuals being moved up cannot overwrite other individuals from their same layer that are also being moved up. To prevent an individual which is being moved up a layer from overwriting an individual that was also just recently moved up, ALPS-SS has an additional check that when moving an individual up it only replaces individuals that were moved more than n evaluations ago (where n is the total size of the population).

5. EXPERIMENTAL SETUP

The objective of these experiments is to compare the performance of the Steady-State ALPS (ALPS-SS) GA against mainstream algorithms in the GA community, and here we compare it against a basic GA (BGA), CMA-ES and DE. ALPS-SS was described in the previous section and we configure it as follows.[1] We

[1]Source code for ALPS is available at http://idesign.ucsc.edu

use 10 layers of 40 individuals, a fibonacci aging scheme with an age gap of 3, and an elitism of 5. New individuals are created using mutation or recombination, with equal probability. In selecting parents, the first parent is chosen through a tournament selection with a tournament size of 5 and the second parent (only needed for recombination) is chosen at random. With mutation, either 1-4 genes are randomly selected to be mutated or all genes are mutated. Genes are mutated by selecting a mutation size from 1% to 0.000001% (by randomly selecting one of $\times 10^{-2}$, $\times 10^{-3}$, ... $\times 10^{-6}$) of the difference between the minimum and maximum values for that gene and then this is used as the variance for a random number with a normal distribution which is the mutation amount for that gene. For recombination, a value for each gene i is selected at random from the range $[P_{1,i} + (P_{1,i} - P_{2,i}), P_{2,i}]$, using a uniform distribution.

The basic GA (BGA), CMA-ES and DE are implemented as follows. The BGA is implemented as a single layer of the ALPS with a population size of 400 and an elitism of 2, everything else is the same as with the ALPS algorithm. By keeping everything as similar as possible between ALPS-SS and the BGA we can see what advantages, if any, are given by using age layers.

For CMA-ES, we use the source code implemented by Hansen [5], with most of the default parameters.[2] We adjusted the standard deviations for the initial individuals to appropriate values for each problem domain (1.5 for the rotated Rosenbrock and F8F2 and 350 for the Griewangk, F101 and Rana problems). CMA-ES also has a restart feature and this was constrained to only two restarts for the Griewangk, Rosenbrock and F8F2 functions. Allowing more restarts on these functions resulted in CMA-ES spending the vast majority of its time performing its internal processing (modeling the function) and it took orders of magnitude more time to run then the other algorithms. Since CMA-ES outperforms the other algorithms on the Griewangk and Rosenbrock functions, this upper limit on the number of restarts is not a hindering constraint. On the other functions the maximum restart level allowed for CMA-ES was never reached and in all cases we configured the restarts to double the population size.

We used an implementation of Differential Evolution developed by Storn which has the latest features of a recent book on this method [17].[3] We used the default settings of this implementation, with the exception of setting the population size to 400 so as to match the population sizes of ALPS-SS and the basic GA and also since preliminary trial runs suggested that this population size was at least as good as smaller ones on these test problems. In addition, we configured the parameter value ranges appropriately for each test problem.

6. TEST PROBLEMS

To test the different algorithms we use five benchmark problems from the GA community. The first two benchmark problems are the rotated **Griewangk** and **Rosenbrock** (F2 in De Jong's test suite [3]). Of the test functions used in this paper these two are the easiest and, being quite popular, are found in many GA papers, such as Salomon's [18]. While the Rosenbrock function is non-separable, it is symmetric and unimodal with a long narrow ridge to the global optima that some algorithms find easy to follow. The Griewangk has been shown to become easier as the number of dimensions increase, becoming nearly uni-modal for 10 or more dimensions [21]. The other three test problems we use are more challenging multi-modal ones that were constructed by Whitley et al. [21] to address some of the shortcomings of previous work, and these are **F101**, **Rana** (also known as F102) and **F8F2** (which is built using their method of creating composite functions from the Griewangk (F8) and the Rosenbrock (F2) functions). Since rotating functions has been strongly advocated by GA practitioners [18, 20] we use rotated versions of these five problems.[4]

The parameter range being explored for each of these functions is as follows. For the Rotated Griewangk, Rotated F101 and Rotated Rana, each parameter was constrained to the range [-512.0, 511.0]. For the Rotated Rosenbrock and Rotated F8F2 each parameter was constrained to the range of [-2.048, 2.047].

The research in constructing nonsymmetric, non-separable, multimodal test functions is an attempt to create functions that resemble real-world functions, thus for the last problem we use an actual real-world problem: evolving an antenna design. One of the successes in the EC community is the evolution of an X-band antenna for NASA's ST-5 Mission [9, 14], which was a co-winner of the Gold Award at the first Human Competitive Competition at GECCO-04. In our version of this problem a genotype consists of a vector of 24 real-valued parameters which specify the XYZ coordinates of eight end points of a bent wire. This wire is the radiating element of an antenna which is to be optimized, and we use the Numerical Electromagnetics Code (NEC) to simulate each design.[5] For this problem all antennas start with a feed-element segment from (0.0, 0.0, 0.0) to (0.0, 0.0, 0.001) and then continue with eight segments through each of the eight XYZ coordinates specified by the genotype. Units for these antennas are in meters (m), and the antenna was constrained to fit in a box of $\pm 0.04m$ in the X and Y direction and 0-$0.041m$ in the Z (up) direction – this dimensional constraint approximates what might be given for commercial antennas at this frequency. We use the same cost function as Lohn et al. [14] for their GA implementation, although here we are optimizing for a single frequency (2106.0 MHz) whereas in their work they were optimizing for two frequencies. This function sums squared difference of gain values for those values below a given level (here we use 0.5 dBic):

$$cost = \sum_{\substack{0° < \phi < 360° \\ 0° < \theta < 90°}} (\text{gain}_{\phi,\theta} - T)^2 \quad \text{if gain}_{\phi,\theta} < T \qquad (2)$$

Here $\text{gain}_{\phi,\theta}$ is the gain of the antenna in dBic (right-hand polarization) at a particular angle, T is the target gain (0.5 dBic), ϕ is the azimuth, and θ is the elevation. The gain component of the cost function samples in 5° increments in the upper hemisphere: $0° \leq \theta \leq 90°$ and $0° \leq \phi \leq 360°$. Frequently NEC fails to evaluate an antenna – this happens with about one quarter of randomly generated antennas but with much lower frequency on offspring of successfully evaluated antennas – in which case a worst score of 1.0e+8 is returned.

7. EXPERIMENTAL RESULTS

In our experiments we performed 30 runs with each algorithm for the test problems listed in the previous section. With the antenna design problem, 24 real-valued parameters were used and for the other problems 20 real-valued parameters were used. Each run was for one million evaluations, although with CMA-ES it frequently stopped after much fewer evaluations on the Griewangk,

[2]Version 3.02.03.beta, which is available at: http://www.lri.fr/~hansen/cmaes_inmatlab.html

[3]Version 4.0 from http://www.icsi.berkeley.edu/~storn/code.html

[4]Source code for all five of these problems is available at: http://www.cs.colostate.edu/~genitor/functions.html

[5]NEC2 is available at: http://www.si-list.net/swindex.html

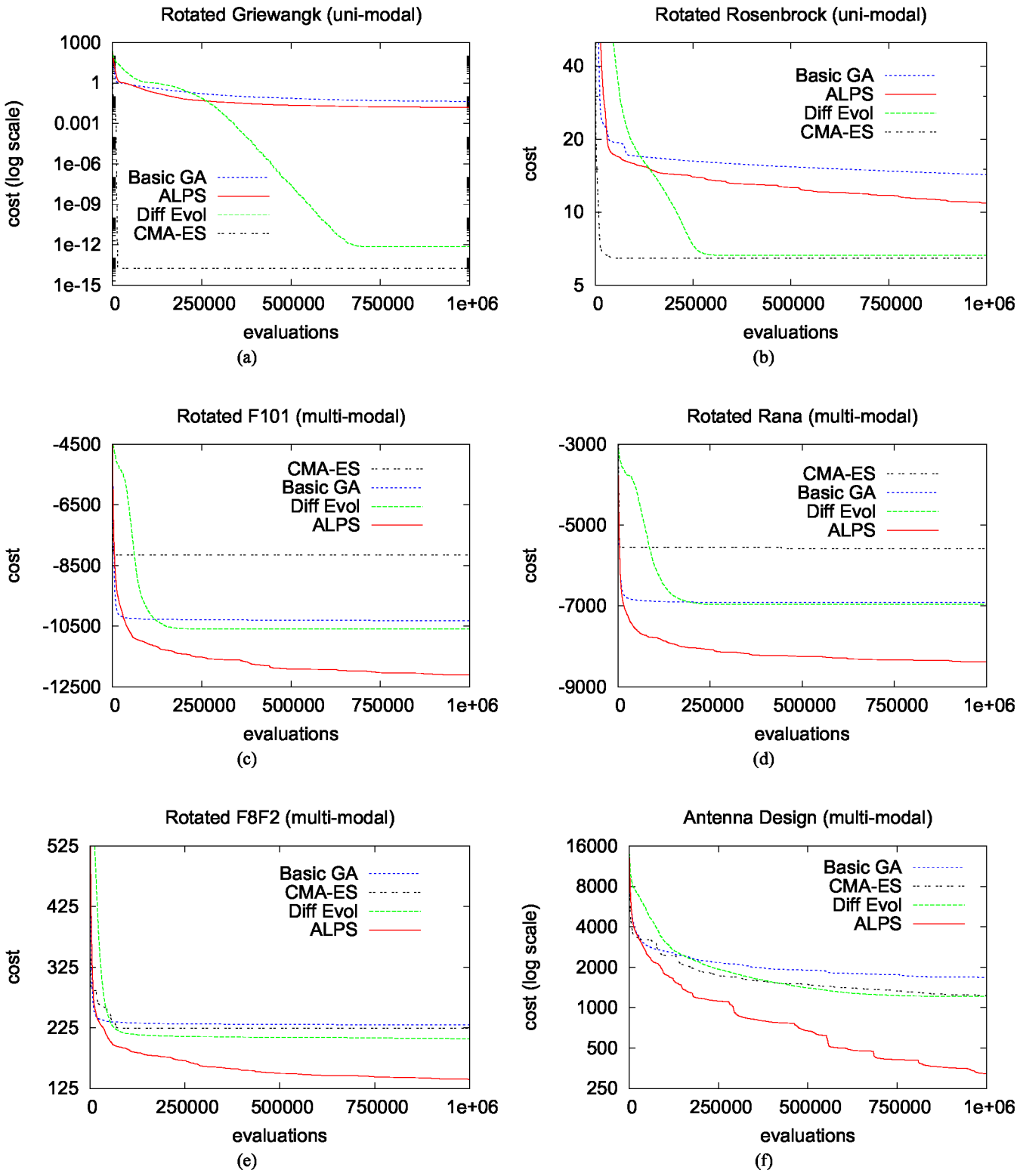

Figure 1: Performance plots for the different algorithms on the different test problems. Results are the average of 30 trials.

Rosenbrock and F8F2 functions. The goal for all six problems is to find the minimum value in the search space and so we use the terminology of optimizing "cost" instead of "fitness". Results for these runs are shown in the graphs in Figure 1, for which each line is the average over the 30 trials of the best individual found. We now go over results on each problem giving final results as average best $\pm$ the standard deviation.

The **Rotated Griewangk** function: ALPS: 1.542e-02$\pm$1.08e-02; BGA: 4.266e-02$\pm$2.77e-02; CMA-ES: 1.805e-14$\pm$2.58e-14; and DE: 6.991e-13$\pm$0.00e+00. Here, ALPS only outperforms the basic GA (BGA) with the difference between the two being highly significant ($P < 0.001$ using a two-tailed, Mann-Whitney test). Both CMA-ES and DE seem to solve the problem, with CMA-ES being the better of the two. From a performance plot on this problem (Figure 1(a)) it can be seen that CMA-ES finds the solution in only a few thousand evaluations whereas DE takes about 700,000 evaluations to find a solution. Interestingly, both ALPS and the BGA start off doing better than DE, but after around 100,000 evaluations they plateau and stop improving. Whitley et al. [21] have noted that at higher dimensions this problem becomes smoother and effectively uni-modal so these results indicate that both CMA-ES and DE do very well on relatively smooth, uni-modal problem whereas ALPS and the BGA have some shortcomings.

The **Rotated Rosenbrock** function: ALPS: 1.088e+01$\pm$2.12e+00; BGA: 1.429e+01$\pm$2.74e+00; CMA-ES: 6.453e+00$\pm$6.59e-01; and DE: 6.629e+00$\pm$0.00e+00. Performance is plotted in Figure 1(b). Again, ALPS and the BGA are quite mediocre – with the better performance by ALPS being highly significant ($P < 0.001$, using a two-tailed Mann-Whitney test) – although in this case they are both continuing to improve slowly. Again, CMA-ES finds a solution after a few thousand evaluations whereas it takes DE much longer. This problem is basically unimodal with a long (rotated) ridge to the optimum. It seems that both CMA-ES and DE can follow this ridge effectively whereas both ALPS and the BGA struggle to do so. Yet, none of the algorithms comes close to the global optima, which is less than 5.

The **Rotated F101** function: ALPS: -1.211e+04$\pm$3.90e+02; BGA: -1.031e+04$\pm$8.71e+02 CMA-ES: -8.157e+03$\pm$8.76e+02; and DE: -1.059e+04$\pm$6.72e+02. ALPS has much better search performance then other three algorithms and this difference is highly significant ($P < 0.001$, using a two-tailed, Mann-Whitney test). Performance is plotted in Figure 1(c). On this non-separable, non-symmetric, highly multi-modal function, ALPS and the BGA have similar performance for the first few thousand evaluations and then the BGA levels off and ALPS keeps finding better and better results and outperforms the other algorithms. DE takes just over 100,000 evaluations to catch up to the BGA and then ends up with slightly better results – although the difference is not statistically significant ($P \geq 0.05$). CMA-ES converges very quickly to a mediocre optima and has the worst performance of all. Interestingly, even though ALPS needs roughly 90,000 evaluations to find a solution better than the other algorithms, it is still continually improving for the full one million evaluations whereas the other algorithms effectively stop improving after less than 200,000 evaluations.

The **Rotated Rana** function: ALPS: -8.385e+03$\pm$3.18e+02; BGA: -6.917e+03$\pm$5.99e+02; CMA-ES: -5.587e+03$\pm$6.54e+02; and DE: -6.960e+03$\pm$5.29e+02. The difference between ALPS and the other algorithms is highly significant ($P < 0.001$, using a two-tailed, Mann-Whitney test). Performance is plotted in Figure 1(d), and is quite similar to that on the F101 function. Again both the BGA and DE converge at similar cost values, with the difference between insignificant ($P \geq 0.05$). Again, this shows that ALPS is the better algorithm on a highly multimodal problem.

The **Rotated F8F2** function: ALPS: 1.400e+02$\pm$1.73e+01; BGA: 2.302e+02$\pm$5.45e+01; CMA-ES: 2.245e+02$\pm$3.80e+01; and DE: 2.067e+02$\pm$3.96e+01. The difference between ALPS and the other algorithms is highly significant ($P < 0.001$, using a two-tailed, Mann-Whitney test). Performance is plotted in Figure 1(e) and this time the algorithms are more tightly clustered. The BGA and CMA-ES start the fastest but both then level off and are quickly passed by first DE and then by ALPS. Here the BGA, CMA-ES and DE all have similar average results and the differences between the three of them is statistically insignificant ($P \geq 0.05$). Unlike the other three algorithms, ALPS continues finding noticeably better results even after one million evaluations. A third example that ALPS is the better algorithm on highly multimodal problems.

The **Antenna** opt. problem: ALPS: 3.226e+02$\pm$6.30e+02; BGA: 1.683e+03$\pm$5.38e+02; CMA-ES: 1.235e+03$\pm$1.75e+02; and DE: 1.211e+03$\pm$2.33e+02. The difference between ALPS and the other algorithms is highly significant ($P < 0.001$, using a two-tailed Mann-Whitney test). Performance is plotted in Figure 1(f). Whereas the previous five problems are artificially constructed test functions that were designed to be challenging and to mimic features of real-world problems, this is an actual real-world problem. On this problem not only does ALPS find better solutions than the other algorithms, but even after one million evaluations it still seems to be improving faster then the other algorithms. What has happened it that whereas the other algorithms have tended to converge on mediocre solutions with Cost scores between 900 and 1300, the majority of solutions found with ALPS are between 0.0 and 20.0, with a few runs still having their best solution above 900. If ALPS is run longer then these other runs will also converge at solutions close to 0.0.

Overall, ALPS is the better algorithm on the multi-modal test problems and the real-world problem and CMA-ES and DE are the better functions on the unimodal problems. On the two unimodal test problems (Rotated Griewangk and Rotated Rosenbrock) CMA-ES very quickly finds the best value with DE being slower and covering to a slightly worse value. Interestingly, ALPS starts faster then DE and is the better algorithm over the first 200,000 evaluations. On the three multi-modal test problems (Rotated F101, Rotated Rana and Rotated F8F2) and on the real-world and the real-world problem ALPS is by far the better algorithm.

8. DISCUSSION

One of the reasons put forth as to why ALPS is better at avoiding premature convergence is that it does a good job of maintaining genotypic diversity in the population. This idea can be tested by tracking the standard deviation in the values for each gene over the course of an evolutionary run, and this is plotted for F8F2 in Figure 2. This graph shows that with CMA-ES genotypic diversity drops to a negligible amount almost immediately, which matches its near-immediate plateau in improvements (Figure 1(c)). With DE, genotypic diversity drops at a constant rate for the first 300,000 evaluations and then levels off at roughly 0.015. In this case, the population has stopped improving after roughly 225,000 evaluations. With the BGA, genotypic diversity drops to an average standard deviation of roughly 2.3 in 20,000 evaluations and then stays at this level for the rest of the evolutionary run. Similarly, most of the performance improvements achieved with the BGA are in the first 20,000 evaluations. In contrast to the other three algorithms, ALPS starts with and keeps an average standard deviation of roughly 300 throughout the entire evolutionary run: there is no loss of genotypic diversity. More importantly, even though it is maintaining a high level of genotypic diversity, it is also outperforming the other algorithms (Figure 1(c)).

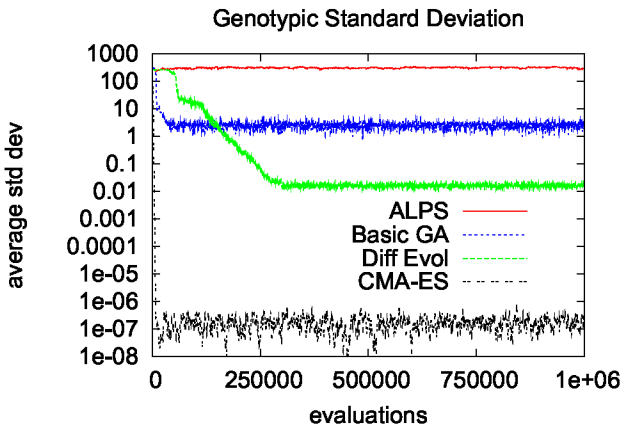

Figure 2: A graph of the genotypic diversity maintained by each algorithm for a single run on F101. Diversity is calculated by taking the standard deviation of each gene's values over the previous 400 evaluations and then the average of these 20 standard deviations is plotted.

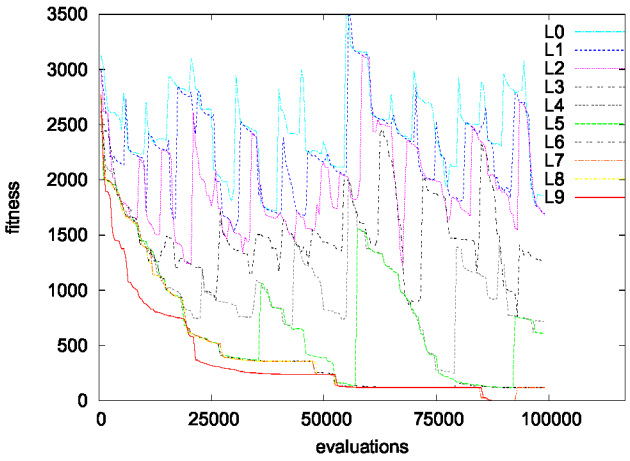

Figure 3: Plots of the cost of the best individual in each layer for an evolutionary run with ALPS.

Tracking the cost of the best individual in each age layer of ALPS can give more insight as to how the use of its age layers leads to a more robust algorithm. The graph in Figure 3 plots the best cost in each layer of a steady-state ALPS run. These cost values oscillate in cycles that correspond to the age limits of the age layers. Arcs connecting different layers can be seen and these show an evolutionary line of genetic material moving up through the layers. For example, for the first 20000 evaluations, the best cost in layers L3 through L8 are roughly the same and this is likely some family of closely related individuals evolving through the population. At around 20000 evaluations, their descendants are passed down into Layer 9 and they take over as the new best individuals in the population. Again, another arc starting in Layer 0 at 25000 evaluations moves up through the layers until at around 55000 evaluations it passes from Layer 5 to layers 6 through 9 and this line of individuals takes over as the best in the population. Finally, sometime between 40000 to 50000 evaluations, offspring from the individuals in Layer 0 start evolving to be better and better and their offspring move up the layers until at around 85000 evaluations they become the best individuals in the population and solve the function. This shows how ALPS uses the different age layers to allow groups of individuals to explore new and different parts of the fitness landscape and thereby reduce the problem of premature convergence.

While this implementation of ALPS does a good job at maintaining diversity and optimizing on multimodal problems it does a poor job of climbing the optima of a unimodal problem. It is possible that by using a better optimization algorithm within each layer that better search performance can be achieved. DE always outperformed our implementation of a basic GA, which suggests that our BGA is fairly mediocre. Since our implementation of ALPS uses the BGA for each layer, a hybridized ALPS which uses Differential Evolution at each layer (instead of a BGA) may have the strengths of both algorithms: the ability to avoid prematurely converging on the wrong local optima (from ALPS) with a strong ability to find the best point on a given local optima (from DE). The implementation of a hybrid ALPS-DE GA is one direction for future work.

9. CONCLUSION

When the Age-Layered Population Structure (ALPS) was introduced, it was shown to work well on a particular design problem with a GP-style representation [7]. More recent work has found it to work well with other GP systems and different problems [12, 16, 22]. One of the main interests in this paper was determining how well ALPS would do on optimizing vectors of real-valued parameters. For this we compared it against two of the more successful GA algorithms, CMA-ES and Differential Evolution, both of which are specialized to work on vectors of real-values. In our comparison we found that CMA-ES and Differential Evolution outperformed ALPS on the two unimodal test problems but that ALPS was the better algorithm on the three multimodal test problems and on the real-world antenna design problem.

By monitoring the standard deviation of gene values, we found that ALPS keeps genotypic diversity fairly constant at a high value throughout an evolutionary run whereas other algorithms lose diversity, which they never regain. An analysis into the movement of individuals across the age layers of the ALPS algorithm shows that it is successful at introducing new individuals into the population and allowing them to evolve to their potential. By better combining exploration (by segregating individuals into layers) with exploitation (optimizing within a layer) the ALPS paradigm better avoids prematurely converging to the wrong local optima and, consequently, is a better algorithm on multimodal problems. Since many real-world problems of interest are multimodal, our results strongly suggest that ALPS is worth considering for challenging real-world problems.

A second contribution of this paper is our description of a steady-state version of the ALPS algorithm. Necessary for this steady-state implementation was the development of a way for calculating age that does not rely on explicit generations. Also, it was necessary to prevent individuals being moved up layers from replacing other individuals that had recently moved up. This version of ALPS is likely to be of interest to those who need, or prefer, a steady-state algorithm rather than a generational one.

A shortcoming with ALPS was in how poorly it performed on the unimodal test problems. Since the BGA also performs poorly then ALPS on these problems this suggests that by improving the type of GA used by ALPS in each layer would result in better optimization performance on these unimodal problems. For example, Differential Evolution was much better than a basic GA on the unimodal problems so one way of improving ALPS would be to use DE inside each layer instead of the BGA. Not only might such a

hybridized ALPS-DE EA be better on the unimodal test functions but an improved hill-climbing ability may also result in better performance on multi-modal problems.

Finally, since ALPS does not use any knowledge about the type of representation being used, it can be used to evolve not only GP type encodings, but also bit strings, vectors of real-valued parameters, ordered lists of numbers, or any other representation used in the evolutionary community. We would expect that the advantages shown here would carry over to EAs using other types of representational schemes and for other problems. Comparing ALPS to the best systems for evolving bit strings or solving scheduling problems is another direction for future work.

Acknowledgements

This material is supported in part by the National Science Foundation's Creative-IT grant 0757532.

10. REFERENCES

[1] D. J. Cavicchio. *Adaptive Search using simulated evolution*. PhD thesis, University of Michigan, Ann Arbor, 1970.

[2] M. Clerc and J. Kennedy. The particle swarm-explosion, stability, and convergence in a multidimensional complex space. *IEEE Transactions on Evolutionary Computation*, 6:58–73, 2002.

[3] K. A. DeJong. *Analysis of the Behavior of a Class of Genetic Adaptive Systems*. Dept. Computer and Communication Sciences, University of Michigan, Ann Arbor, 1975.

[4] D. E. Goldberg and J. Richardson. Genetic algorithms with sharing for multimodal function optimization. In J. J. Grefenstette, editor, *Proc. of the Second Intl. Conf. on Genetic Algorithms*, pages 41–49. Lawrence Erlbaum Associates, 1987.

[5] N. Hansen. The CMA evolution strategy: a comparing review. In J. Lozano, P. Larranaga, I. Inza, and E. Bengoetxea, editors, *Towards a new evolutionary computation. Advances on estimation of distribution algorithms*, pages 75–102. Springer, 2006.

[6] J. H. Holland. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor, Mich., 1975.

[7] G. S. Hornby. ALPS: The age-layered population structure for reducing the problem of premature convergence. In M. K. et al., editor, *Proc. of the Genetic and Evolutionary Computation Conference, GECCO-2006*, pages 815–822, Seattle, WA, 2006. ACM Press.

[8] G. S. Hornby, M. Fujita, S. Takamura, T. Yamamoto, and O. Hanagata. Autonomous evolution of gaits with the sony quadruped robot. In Banzhaf, Daida, Eiben, Garzon, Honavar, Jakiel, and Smith, editors, *Proc. of the Genetic and Evolutionary Computation Conference*, pages 1297–1304. Morgan Kaufmann, 1999.

[9] G. S. Hornby, J. D. Lohn, and D. S. Linden. Computer-automated evolution of an X-band antenna for NASA's Space Technology 5 mission. *Evolutionary Computation*, accepted with revisions.

[10] A. Huber and D. A. Mlynski. An age-controlled evolutionary algorithm for optimization problems in physical layout. In *International Symposium on Circuits and Systems*, pages 262–265. IEEE Press, 1998.

[11] J.-H. Kim, J.-Y. Jeon, H.-K. Chae, and K. Koh. A novel evolutionary algorithm with fast convergence. In *IEEE International Conference on Evolutionary Computation*, pages 228–29. IEEE Press, 1995.

[12] M. F. Korns and L. Nunez. Profiling symbolic regression-classification. In R. L. Riolo, T. Soule, and B. Worzel, editors, *Genetic Programming Theory and Practice VI*, Genetic and Evolutionary Computation, chapter 14, pages 215–229. Springer, Ann Arbor, 15-17May 2008.

[13] N. Kubota, T. Fukuda, F. Arai, and K. Shimojima. Genetic algorithm with age structure and its application to self-organizing manufacturing system. In *IEEE Symposium on Emerging Technologies and Factory Automation*, pages 472–477. IEEE Press, 1994.

[14] J. Lohn, G. Hornby, and D. Linden. Evolutionary antenna design for a NASA spacecraft. In U.-M. O'Reilly, T. Yu, R. L. Riolo, and B. Worzel, editors, *Genetic Programming Theory and Practice II*, chapter 18, pages 301–315. Springer, Ann Arbor, 2004.

[15] S. W. Mahfoud. Crowding and preselection revisited. In R. Männer and B. Manderick, editors, *Parallel Problem Solving from Nature, 2*, pages 27–36. North-Holland, 1992.

[16] T. McConaghy, P. Palmers, G. Gielen, and M. Steyaert. Genetic programming with reuse of known designs. In R. L. Riolo, T. Soule, and B. Worzel, editors, *Genetic Programming Theory and Practice V*, Genetic and Evolutionary Computation, chapter 10, pages 161–186. Springer, Ann Arbor, 17-19May 2007.

[17] K. Price, R. Storn, and J. Lampinen. *Differential Evolution - A Practical Approach to Global Optimization*. Springer, 2005.

[18] R. Salomon. Reevaluating genetic algorithm performance under coordinate rotation of benchmark functions; a survey of some theoretical and practical aspects of genetic algorithms. *BioSystems*, 39(3):263–278, 1996.

[19] D. Whitley and J. Kauth. Genitor: A different genetic algorithm. Technical Report CS 88-101, Colorado State University, 1988.

[20] L. D. Whitley, M. Lunacek, and J. N. Knight. Ruffled by ridges: How evolutionary algorithms can fail. In K. D. et al., editor, *Proc. of the Genetic and Evolutionary Computation Conference, Vol. II*, LNCS 3103, pages 294–306, New York, 2004. Springer-Verlag.

[21] L. D. Whitley, S. B. Rana, J. Dzubera, and K. E. Mathias. Evaluating evolutionary algorithms. *Artificial Intelligence*, 85(1-2):245–276, 1996.

[22] A. Willis, S. Patel, and C. D. Clack. GP age-layer and crossover effects in bid-offer spread prediction. In *Proceedings of the 10th annual conference on Genetic and Evolutionary Computation Conference*, Atlanta, GA, July 12-16 2008.