Chapter 8

AGE-FITNESS PARETO OPTIMIZATION

Michael Schmidt[1] and Hod Lipson[2,3]

[1]*Computational Biology, Cornell University, Ithaca, NY 14853, USA;* [2]*School of Mechanical and Aerospace Engineering, Cornell. University, Ithaca NY 14853, USA;* [3]*Computing and Information Science, Cornell University, Ithaca, NY 14853, USA.*

Abstract We propose a multi-objective method, inspired by the Age Layered Population Structure algorithm, for avoiding premature convergence in evolutionary algorithms, and demonstrate a three-fold performance improvement over comparable methods. Previous research has shown that partitioning an evolving population into age groups can greatly improve the ability to identify global optima and avoid converging to local optima. Here, we propose that treating age as an explicit optimization criterion can increase performance even further, with fewer algorithm implementation parameters. The proposed method evolves a population on the two-dimensional Pareto front comprising (a) how long the genotype has been in the population (age); and (b) its performance (fitness). We compare this approach with previous approaches on the Symbolic Regression problem, sweeping the problem difficulty over a range of solution complexities and number of variables. Our results indicate that the multi-objective approach identifies the exact target solution more often than the age-layered population and standard population methods. The multi-objective method also performs better on higher complexity problems and higher dimensional datasets - finding global optima with less computational effort.

Keywords: Symbolic Regression, Age, Fitness, Multi-objective

1. Introduction

A common problem in many applications of evolutionary algorithms is when the progress of the algorithm stagnates and solutions stop improving (Murphy and Ryan, 2007). Expending additional computational effort in the evolution often fails to make any substantial progress. This problem is known as premature convergence (Ryan, 1996; Louis and Rawlins, 1993).

A common method for dealing with premature convergence is to perform many evolutionary searches, randomizing and restarting the search multiple

R. Riolo et al. (eds.), *Genetic Programming Theory and Practice VIII,*
DOI 10.1007/978-1-4419-7747-2_8,

times (Auger and Hansen, 2005; Jansen, 2002). This approach can be wasteful however, as the entire population is repeatedly thrown out. There is also the difficulty of deciding when to restart, and the possibility that the converged population could continue improving with additional diversity.

One of the best performing methods in the genetic programming literature for addressing premature convergence is the Age-Layered Population Structure (ALPS) method (Hornby, 2006; Hornby, 2009a; Hornby, 2009b). ALPS uses a special notion of age - how long genotypic material has existed in the population - in order to partition the evolving population into age layers (Figure 8-1). Random individuals are constantly inserted into the youngest population layer, and layers evolve independently of others. As a result, the youngest layers, tend to be very diverse, and are allowed to evolve and mature before competing against the oldest and most fit solutions. This improves upon random restarts in that the best solutions are never thrown out and new solutions maintain diversity as they propagate up the layers. Implementation of the ALPS algorithm, however, requires new parameters, such as how to pick age layer cutoffs and how many solutions to keep in each layer, etc.

In this paper, we consider using the ALPS concept of age as a fundamental property in the evolutionary optimization. Rather than using age to partition the population into layers, we use age as an independent dimension in a multi-objective Pareto front optimization. In this context, a solution is selected for if it has both higher fitness and lower genotypic age than other solutions.

Does real evolution mirror Afpo? probably not, but interesting how difficult it is for evolution to work, perhaps in reality it is a much more complicated 'algorithm' than we give credit for.

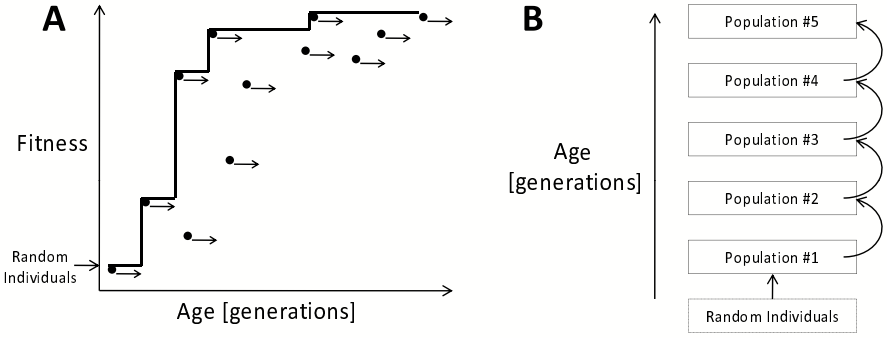


Figure 8-1. The two primary optimization methods compared. The Age-Fitness Pareto Population algorithm (A) considers a single population of individuals moving in a two-dimensional Age-Fitness Pareto space. Individuals are selected for if they simultaneously have higher fitnesses and lower age than other individuals. Ages increase every generation, or are inherited during crossover, and new random individuals are added with zero age. In the Age-Layered Population Structure (ALPS) algorithm, there are several layers of populations for each age group. New individuals are added to the youngest population, and individuals migrate to older populations as their age increases.

As in the ALPS method, random individuals are added into the population at each generation. Rather than flowing up the age layers, they flow through a two-dimensional space of fitness and age (Figure 8-1). Young solutions exist in the same population as the oldest and most fit, but persist because they are non-dominated in the two-dimensional objective space.

A key benefit of the proposed approach is that it does not require a population partitioning or structuring. For example it does not constrain intermediate layer sizes, the number of total layers, or layer partitions. These variations all exist within the larger Pareto space of the search, allowing the age-fitness distributions to vary dynamically.

Like ALPS, this approach makes no assumptions about the underlying solution representation. Therefore, it can be applied to nearly any evolutionary search problem to improve the optimization performance. In the next sections, we provide broader background on the premature convergence problem and describe the Age-Fitness Pareto Algorithm. We then describe our experiments and results, before concluding with discussion and final remarks.

2. Background

An essential component for avoiding premature convergence is maintaining diversity in the population throughout evolution. A deceptively-simple way to increase diversity in an evolutionary algorithm is to increase the mutation rate. Increasing the mutation rate however also increases the rate of deleterious mutations. Effectively, high mutation rate increases diversity but greatly inhibits the search performance. Instead, recent research has focused on alternative methods for maintaining diversity without heavily sacrificing the evolution performance.

One way to maintain diversity is using similarity-sensitive selection methods, such as fitness-sharing and crowding methods. Fitness-sharing penalizes solutions that are similar to existing solutions, whereas crowding methods replace existing solutions with similar ones. For example, in deterministic crowding (Mahfoud, 1995), individuals in the population are only removed from the population when replaced by direct similar descendent. In effect, these methods sustain multiple partially-independent evolutionary trajectories in one population simultaneously.

Another approach is to prevent the population from converging entirely. In Hierarchical Fair Competition (HFC) (Hu et al., 2002; Hu et al., 2005), the algorithm maintains sub-populations of various fitness ranges. Therefore, the algorithm always contains several individuals at different fitness values to enforce diversity.

A common technique to deal with premature convergence in practice is to simply perform many evolutionary searches (Auger and Hansen, 2005; Jansen,

2002), accepting that nearly all runs converge to local optima. In effect, each random restart explores a new set of evolutionary trajectories, starting from new random individuals.

All of these methods are known to reduce premature convergence and increase the chance of identifying global optima using existing evolutionary algorithm machinery (mutation, selection, fitness, etc).

Other research has looked at tracking solution age. Most notably is the Age-Layered Population Structure (ALPS) mentioned earlier (Hornby, 2006; Hornby, 2009a; Hornby, 2009b). Similar to HFC, the algorithm maintains sub-populations of individuals, but splits these sub-populations by the genotypic age of the solution. Therefore, very young solutions are not selected out by old and optimized solutions in the population.

Similar to random restarts, ALPS adds new random solutions into the lowest layer throughout evolution. As the evolution continues, the solutions age and are promoted to the next age group. In effect, ALPS performs many random restarts in parallel.

3. Algorithm

In this section we describe the details and reasoning behind the multi-objective Age-Fitness Pareto optimization algorithm.

Genotypic Age

Interestingly, the concept of genotypic age as used in ALPS has shown to be one of the best approaches for avoiding premature convergence and improving results (Hornby, 2006). Our goal in this paper is to develop this idea further by utilizing genotypic age as a fundamental search trait.

The age of a solution is generally measured in generations, or alternatively computational effort measured in fitness evaluations for steady-state algorithms (Hornby, 2009a; Hornby, 2009b).

All randomly initialized individuals start with age of one. With each successive generation that an individual exists in the population, the age is incremented by one. This alone measures the amount of time an individual has existed in the population. However, we are more interested in the age of the genotype.

To measure the age of the genotype, we need to pass on ages during crossover and mutation events. There are several options, such as taking the age of the most similar parent, taking the average age of the parents, etc. The best method reported in the literature (Hornby, 2009b), and the method we use, is to inherit the maximum age of the parents.

Therefore, the age is a measure of how long the oldest part of the genotype has existed in the population.

Age-Fitness Pareto Population

The Age-Fitness Pareto Population method uses a single population, in contrast to the population layers in the ALPS algorithm. The algorithm tracks the fitness of each individual as in a normal evolutionary algorithm, and also the maximum genotypic age described in the previous section.

The individuals in the population can be thought of as lying on a two-dimensional plane of age and fitness, as in Figure 8-1. The multi-objective optimization task is to identify the non-dominated Pareto front of the problem domain (Deb, 2001); here, the objectives are to maximize the fitness with minimum age.

The age is a special objective however. Age is constantly increasing for all individuals in each generation, either through staying in the population for multiple generations, or inheriting older genotypes in crossover. Like in the ALPS algorithm, new random individuals are added to the population every generation. We add a single new random individual each generation. More than one can be added per generation; however, one of these is likely to dominate the others in fitness, making it equivalent to adding this most dominant random individual each generation.

The age objective effectively protects young, low fit individuals from being replaced by old, high-fit individuals. Since the Pareto front identifies non-dominated solutions, each solution on the front can only be replaced or removed by the occurrence of a younger and fitter solution

Individuals that evolve to a local optimum and become trapped there are likely to stay there for many generations. In this case, new random individuals may converge to different optima with potentially higher fitnesses, thereby dominating and eventually replacing the older stagnant individual.

Pareto Tournament Selection

There are a number of ways to implement multi-objective evolution (Deb, 2001). In this paper, we use the simple random mating with tournament selection method (Kotanchek et al., 2009).

Each generation, we select random pairs of individuals, cross and mutate them probabilistically, and add them to the current population. Additionally, a new random individual is added to the population.

We specify a target population size - analogous to the population size in a traditional evolutionary algorithm. The goal of the selection is to remove dominated individuals from the population until the target population size is reached.

It is possible that the non-dominated set - the age-fitness Pareto front of the population - is larger than the target population size. In this case, our selection method does not remove any further individuals, and the population size may

remain larger than the target size. However, in our experiments, this was never the case. The population size always reached the target size.

The Pareto tournament selection method we used selects groups of k individuals randomly from the population. It then forms the Pareto front among those individuals, and discards any dominated individuals from the tournament. This is repeated until the population is less than or equal to the target population size again, or until the population is entirely non-dominated.

This selection method allows for the possibility for dominated solutions to persist in the population through successive generations. They simply need to survive by being non-dominated in smaller random Pareto tournaments. In our experiments we used a tournament size of two.

4. Experiments

In this section we detail our experimental methods to test the impact of treating age as a fundamental search optimization criterion.

We perform identical experiments on three algorithms: (1) the ALPS algorithm (Hornby, 2006), (2) the proposed Age-Fitness Pareto algorithm, and (3) the Deterministic Crowding algorithm (Mahfoud, 1995), a well established diversity-maintenance method.

We experimented on the Symbolic Regression problem. Symbolic regression (Koza, 1992) is the problem of identifying the simplest equation (Grunwald, 1999) that most accurately fits a given set of data. Symbolic regression has a wide range of applications, such as prediction (Korns, 2009), classifi-

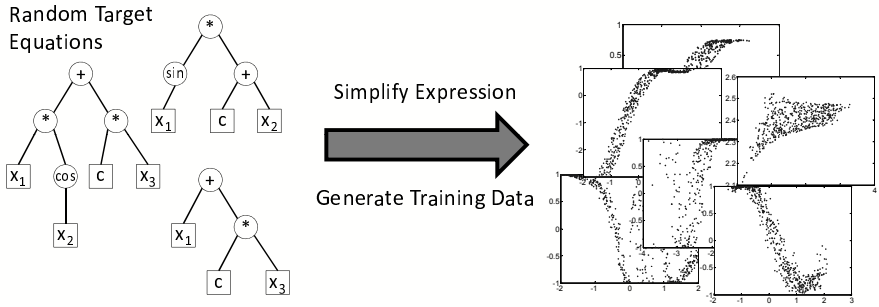


Figure 8-2. The generation of random test problems for symbolic regression. We start by picking a random number of inputs, between one and ten. We then generate a random equation using these inputs and simplify the equation before measuring its complexity (the number of nodes in the binary tree). We then generate a random training data set by sampling the input variables around the origin and evaluating the target equation on these data points. We then generate a validation data set in a similar fashion, but with a wider range around the origin to test if the solutions extrapolate to the exact solution.

cation (Doucette et al., 2009), modeling, and system identification. Recently, symbolic regression has been used to detect conserved quantities in data representing physical laws of nature (Schmidt and Lipson, 2009), as well as infer the differential equations underlying dynamical systems (Bongard and Lipson, ress).

We chose the Symbolic Regression problem because it is a common benchmark of genetic programming (Koza, 1992). Additionally, we can easily vary the problem complexity and the problem dimensionality in order to study and compare how the proposed method scales with problem complexity.

Symbolic Regression

Symbolic regression (Koza, 1992) is the problem, and classic application of genetic programming, of searching a space of symbolic expressions computationally to minimizing various error metrics for a particular data set. Both the parameters and the form of the equation are subject to search. In symbolic regression, many initially random symbolic equations compete to model experimental data in the most parsimonious way. It forms new equations by recombining previous equations and probabilistically varying their sub-expressions. The algorithm retains equations that model the experimental data well while abandoning unpromising solutions. After an equation reaches a desired level of accuracy, the algorithm terminates, returning the most parsimonious equations that may correspond to the intrinsic mechanisms of the observed system.

In symbolic regression, the genotype or encoding represents symbolic expressions in computer memory. Often, the genotype is a binary tree of algebraic operations with numerical constants and symbolic variables at its leaves (McKay et al., 1995; de Jong and Pollack, 2003). Other encodings include acyclic graphs (Schmidt and Lipson, 2007) and tree-adjunct grammars (Hoai et al., 2002). The fitness of a particular genotype (a candidate equation) is a numerical measure of how well it fits the data, such as the equation's correlation or squared-error with respect to the experimental data.

A point mutation can randomly change the type of the floating-point operation (for example, flipping an add operation to a multiply or an add to a system variable), or randomly change the parameter constant associated with that operation (if it is used). The crossover operation recombines two existing equations to form a new equation. To perform crossover, we select a random location in the genotype, and copy all operation and parameter values to the left of this point from the first parent and remaining operations and parameters to the right from the second parent.

Generating Test Problems

We measured the performance of each algorithm on randomly generated test problems. To generate a random problem in symbolic regression, we simply need a random target equation to find and a set of data corresponding to that equation for the fitness error metric.

We experimented by varying two characteristics of the random symbolic regression problems: (1) the dimensionality of the data (i.e. the number of variables in the data set, and (2) the complexity of the target function (i.e. the size of the equation's binary parse tree). Both of these characteristics factor in to the problem's difficulty. Increasing dimensionality increases the base set of possible variables the equation may use, while increasing complexity increases the chances of coupled nonlinear features.

The first step in our random test problem generation is to randomly sample the dimensionality of the problem. We picked a random number of variables ranging between one and ten.

Next, we generated a random equation which can use any of these variables. We generated a random equation in the same fashion that we generate random individuals in the evolutionary algorithm.

Many randomly generated equations may have compressible terms. For example, $f(x) = 4.211 + 0.93x^2 + 1.23$ is equivalent to $f(x) = 0.93x^2 + 5.441$. Therefore, we perform a symbolic simplification on the randomly generated equation in order to get an accurate measure of the target equation's complexity. We measure complexity of the problem as the total number of nodes in the binary tree representation of the equation. For example, the complexity of the equation mentioned above is seven.

We repeated this step as necessary in order to get a uniform distribution of problem complexities. We continued generating and simplifying equations in order to uniformly cover the problem complexities ranging between 1 and 32.

Next, we randomly sampled the input values of the equation 500 times. These variables were sampled from a normal distribution around the origin, with standard deviation of two. The equation was then evaluated on these variables in order to get the target output value. Several examples of training data are shown in Figure 8-2.

Finally, we also generated a separate validation data set of 500 points. The validation data set was created in the same fashion as the training data set, however the input variables were sampled with a standard deviation of three. By using a broader input sampling, we can use the validation dataset to test whether solutions extrapolate in their predictions to unseen data.

We also use this to measure the percent of times the algorithms find the exact solution - if the algorithm achieves near zero error on the extrapolated validation dataset. Since we are not adding any noise to the dataset, we expect

the algorithms to reach zero error on the generated data, if the exact solution is in fact found.

Measuring Performance

We tested each algorithm on 1000 randomly-generated symbolic regression problems. Each evolutionary search was performed on a single quad-core computer.

Evolution was stopped if the algorithm identified a zero error solution on the validation data set (i.e. less than $10 \sup -3$ normalized mean absolute error), or when the algorithm reached one million generations. Throughout each search, we log the best equation, its fitness (i.e. normalized mean absolute error) on the training and validation sets, its complexity, and the total computational effort. We measure computational effort as the total equation evaluations performed in fitness calculations.

The fitness of the normalized mean absolute error is normalized using the standard deviation of the target output values. The normalized fitness allows comparing fitnesses between evolution runs and detecting convergence to the exact target solution more easily. In all figures, we show the fitness on the validation data set (i.e. the normalized mean absolute error on the validation data).

Algorithm Settings

We used the symbolic regression algorithm described in (Schmidt and Lipson, 2006; Schmidt and Lipson, 2008) as the basis for our implementation. We simply swap out the population representation and selection for the three compared algorithms.

In the ALPS algorithm we use exponential scaling of the layer limits, with a base multiplier of 20. The maximum age for a layer i is given by the equation $20 * 2i$. This was chosen such that there could be up to a maximum of 16 layers should the evolution reach the maximum of one million generations.

We performed selection in the ALPS populations using Deterministic Crowding (Mahfoud, 1995). The number of individuals in each layer was calculated as the target population size of 256 divided by the current number of layers. One new individual was added to the first layer each generation. The Age-Fitness Pareto algorithm used a target population size of 256. One new random individual was added to the population each generation. Finally, the deterministic crowding method uses a population size of 256. One new random individual is added to the population each generation by replacing the worst ranked solution. For all algorithms, crossover probability was 0.75 and mutation probability was 0.01.

5. Results

This section summarizes the experimental results comparing the three algorithms: (1) the ALPS algorithm, (2) Age-Fitness Pareto algorithm, and (3) the Deterministic Crowding algorithm with randomized individuals.

Fitness and Convergence

Our first observation is that the fitness versus the computational effort of each algorithm are similar (Figure 8-3). On average, the ALPS algorithm has the lowest error early on while the Age-Fitness Pareto algorithm has the highest error. This difference, however, does not appear to be significant due to the overlapping standard errors.

Later into the evolutionary search, all algorithms converge to similar fitness values. This suggests that the algorithms are reaching common local optima. The deterministic crowding method does perform worse here as it is the last to converge on to this trend. Near the end however, the average fitnesses are very similar, as most runs for all algorithms do converge to the exact solution.

Figure 8-3 also shows the rate that each algorithm identifies the exact target solution. Here we have differences and non-overlapping standard errors for each algorithm.

The ALPS algorithm again has the highest exact solution rate early on in evolution. All algorithms show the standard s-shaped convergence rates where computational effort increases greatly for the hardest of the test problems.

Late in the searches, the algorithms begin to plateau at different rates of finding the exact solution. The Age-Fitness Pareto algorithm performed the best, finding the exact solution approximately 5% more often than the ALPS algorithm.

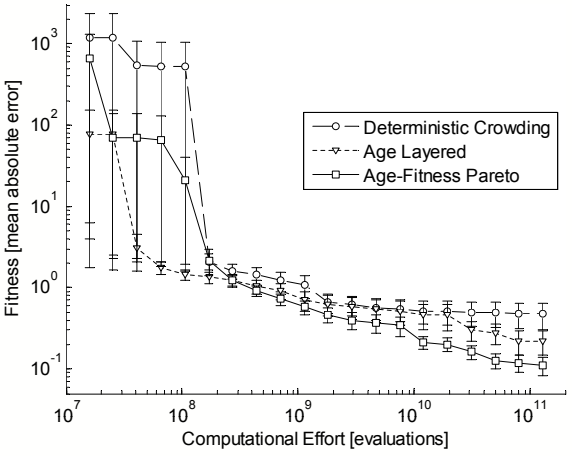
Importantly, Figure 8-3(a) further demonstrates that the hardest problems solved by ALPS were solved by the Age-Fitness Pareto algorithm using a third of the computational effort.

The deterministic crowding algorithm, with the added randomized individual per generation, performed worst of the three algorithms. Here, deterministic crowding identified the exact target solution approximately 5% less often than the ALPS algorithm, and approximately 10% less often than the Age-Fitness Pareto algorithm.

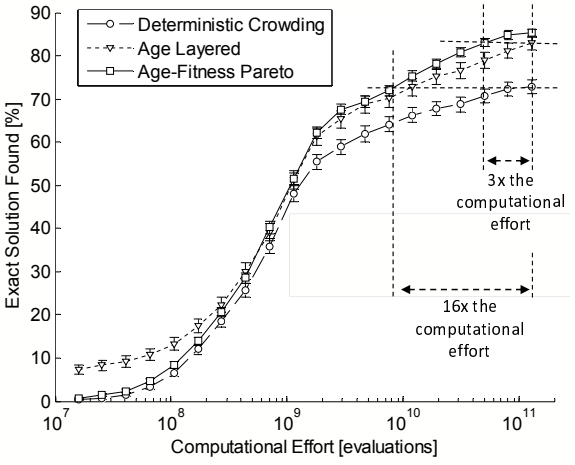
Computational Effort

We also compared the total computational effort each algorithm required to find the exact target solution. Here, we look at the computational effort versus the complexity of the target solution, and the dimensionality of the datasets for each evolutionary search (Figure 8-4).

In the computational effort versus the target solution complexity, we notice that ALPS performs the best for the simplest test problems, requiring the least computational effort to identify the exact solution. As the complexity and



(a) **Error over time**

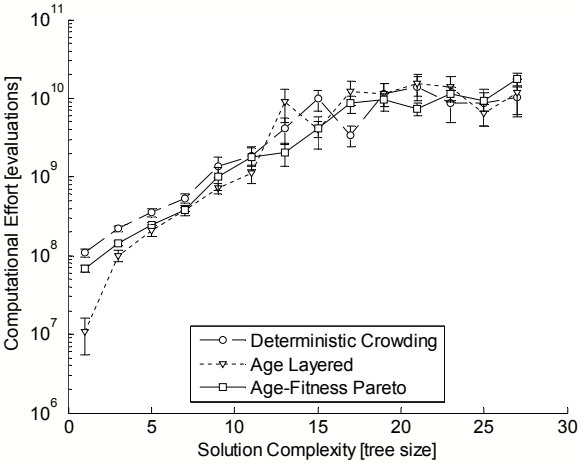


(b) **Convergence over time**

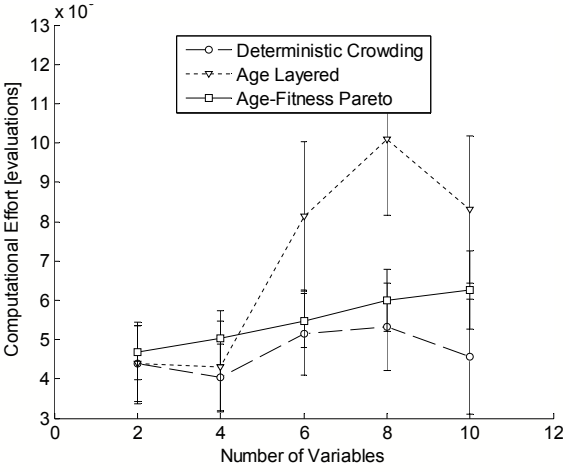
Figure 8-3. The fitness and convergence rate to the exact solution of the compared algorithms versus the total computational effort of the evolutionary search. The fitness plotted (a) is the normalized mean absolute error on the validation data set. Fitness is normalized by the standard deviation of the output values. Convergence to the exact solution (b) is percent of the trials which reach epsilon error on the validation data set. The error bars indicate the standard error.

difficulty of the problem increases, the computational effort approaches the Age-Fitness computational effort.

Deterministic crowding required the most computational effort for the simplest test problems. Interestingly, deterministic crowding does better relatively as the solution complexity increases. This is likely due to the fact that it is not



(a) **Time of convegence over complexity**



(b) **Time of convergence over number of inputs**

Figure 8-4. The computational effort required when the exact solution was found versus the target equation complexity (a) and the number of variables in the dataset (b). Each algorithm found the exact solution with different frequencies; these plots show the computation effort for when the algorithms did find the exact solution. The error bars indicate the standard error.

finding the exact solution as much as the other algorithms. In effect, it is only finding the exact solution for the easiest of the high complexity target solutions.

This same effect may also arise for the ALPS trends with complexity. The Age-Fitness Pareto algorithm holds the middle amount of computational effort to find the exact solution across almost all solution complexities. However, it also found the exact target solution most often as found in Figure 8-3.

When looking at the computational effort used to find the exact solution versus the number of variables in the data set, we can see a similar trend. These values are most-likely dominated by the most difficult target solutions.

Therefore, ALPS showed the most computational effort but still found the exact solution with a similar rate to the Age-Fitness Pareto algorithm. Deterministic crowding required less computational effort, but identified the exact solution much less often.

It is also interesting to note that there was a much stronger dependence on the target solution complexity than the number of variables in the dataset.

Solution Bloat

Finally, we looked at the amount of solution bloat experienced by each algorithm over the course of the evolutionary searches in Figure 8-5.

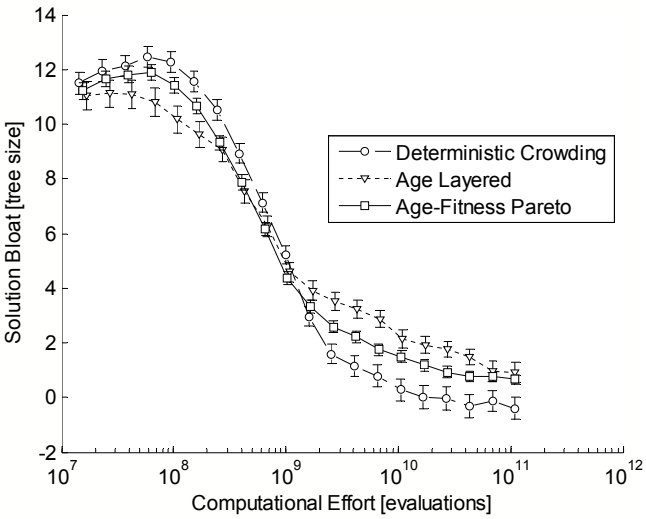


Figure 8-5. Solution bloat over the course of the evolutionary search. Solution bloat is defined as the binary tree size of the best individual in the population minus the binary tree size of the target solution. The error bars indicate the standard error.

We define bloat as the binary tree size of the best solution in the population minus the binary tree size of the target solution. Therefore, the most bloated solutions have positive bloat values, and overly simple solutions have negative bloat values.

In these results, all algorithms started with high amount of bloated solutions early on in the evolutionary searches. On average, the bloat decreased as the search progressed, and the algorithm converged toward exact solutions. Our only method of bloat control comes from coevolution (Schmidt and Lipson, 2008), where changing fitness biases the search toward simpler solutions.

Interestingly, the deterministic crowding algorithm dropped the most in solution bloat. This suggests that the algorithm is under-fitting - it is stagnating at simple local optima.

In contrast, the ALPS and Age-Fitness Pareto algorithms have similar, more-complex solutions on average, which converge toward slightly bloated solutions.

On average, ALPS was the least bloated early on in the evolutionary searches, but bloated the most as the searches progressed.

6. Discussion

The results in the previous section show several interesting differences and similarities between the three compared algorithms.

Primarily, we found that the Age-Fitness Pareto algorithm performed best overall, identifying the exact target solution most often and earlier (Figure 8-3). The ALPS algorithm was close behind, and the deterministic crowding algorithm performed the worse, finding the exact solution least often.

The deterministic crowding algorithm used a randomized individual each generation, similar to the Age-Fitness Pareto and ALPS algorithms. However, it still performed significantly worse than the other algorithms. This suggests that the performance improvement is not coming solely from increased diversity through random individuals. Therefore, the genotypic age is playing an important role.

We also found that the performance was most affected by the target solution complexity. The performance trend with the number of variables was much weaker.

On average, the deterministic crowding algorithm experienced the least bloat, suggesting that it could be under-fitting, stagnating at low complexity local optima. The ALPS and Age-Fitness Pareto algorithms instead tended toward slightly bloated solutions on average, which may reflect their higher performance overall.

Another trend we noticed was that the ALPS algorithm found the exact solution with the least computational effort on the simplest test problems, and found the exact solutions most often early in the evolutionary search. One possible

explanation is that ALPS gets the most benefit from its very first population layers. However, the computational efficiency declines as more and more age layers are maintained. In contrast, the Age-Fitness Pareto algorithm can discard entire ranges of ages should a low age and fit solution arise in the population.

There is also the possibility that the layer sizes and partitions could be optimized further for the random target test problems. However, it is unclear if this could affect the longer term trends on the most difficult and complex test problems.

One final difference of the Age-Fitness Pareto algorithm is that it does not introduce any new evolutionary parameters to the algorithm or addition of multiple population structures. Instead, it simply adds an additional search objective.

7. Conclusion

In summary, we proposed a multi-objective algorithm inspired by the ALPS algorithm (Hornby, 2006) for addressing premature convergence in evolutionary algorithms - a common problem experienced in evolutionary computation where the evolutionary search stagnates at local optima solutions.

Previous research has shown that genotypic age can be used to substantially improve performance in genetic programming by preventing high-fitness and old genotypes from defeating newer and more diverse genotypes. The Age-Fitness Pareto algorithm considers this concept of genotypic age as a fundamental trait in the evolutionary search which can be optimized as an additional objective. Selection favors non-dominated solutions with high fitness and low genotypic age.

Results on randomly generated symbolic regression problems indicate that this approach finds the exact target solution more often than previous methods over a range of target problem complexities and dataset dimensions.

Finally, this approach can be readily incorporated into other evolutionary algorithms, as it makes no assumptions about the problem or solution representations. Additionally, it does not require structuring the multiple sub-populations. It simply adds an additional search objective of genotypic age.

Acknowledgment

This research is supported by the U.S. National Science Foundation (NSF) Graduate Research Fellowship Program, NSF Grant ECCS 0941561 on Cyber-enabled Discovery and Innovation (CDI), and the U.S. Defense Threat Reduction Agency (DTRA) grant HDTRA 1-09-1-0013.

References

Auger, Anne and Hansen, Nikolaus (2005). A restart CMA evolution strategy with increasing population size. In *IEEE Congress on Evolutionary Computation*, pages 1769–1776. IEEE.

Bongard, J. and Lipson, H. (in press). Automated reverse engineering of nonlinear dynamical systems. *Proceedings of the National Academy of Sciences of the United States of America*.

de Jong, Edwin D. and Pollack, Jordan B. (2003). Multi-objective methods for tree size control. *Genetic Programming and Evolvable Machines*, 4(3):211–233.

Deb, Kalyanmoy (2001). *Multi-Objective Optimization Using Evolutionary Algorithms*. Wiley, Chichester, UK.

Doucette, John, Lichodzijewski, Peter, and Heywood, Malcolm (2009). Evolving coevolutionary classifiers under large attribute spaces. In Riolo, Rick L., O'Reilly, Una-May, and McConaghy, Trent, editors, *Genetic Programming Theory and Practice VII*, Genetic and Evolutionary Computation, chapter 3, pages 37–54. Springer, Ann Arbor.

Grunwald, P. (1999). Model selection based on minimum description length.

Hoai, N. X., McKay, R. I., Essam, D., and Chau, R. (2002). Solving the symbolic regression problem with tree-adjunct grammar guided genetic programming: The comparative results. In Fogel, David B., El-Sharkawi, Mohamed A., Yao, Xin, Greenwood, Garry, Iba, Hitoshi, Marrow, Paul, and Shackleton, Mark, editors, *Proceedings of the 2002 Congress on Evolutionary Computation CEC2002*, pages 1326–1331. IEEE Press.

Hornby, Gregory (2006). ALPS: the age-layered population structure for reducing the problem of premature convergence. In Cattolico, Mike, editor, *Genetic and Evolutionary Computation Conference, GECCO 2006, Proceedings, Seattle, Washington, USA, July 8-12, 2006*, pages 815–822. ACM.

Hornby, Gregory S. (2009a). Steady-state ALPS for real-valued problems. In Rothlauf, Franz, editor, *Genetic and Evolutionary Computation Conference, GECCO 2009, Proceedings, Montreal, Québec, Canada, July 8-12, 2009*, pages 795–802. ACM.

Hornby, Gregory S. (2009b). A steady-state version of the age-layered population structure EA. In Riolo, Rick L., O'Reilly, Una-May, and McConaghy, Trent, editors, *Genetic Programming Theory and Practice VII*, Genetic and Evolutionary Computation, chapter 6, pages 87–102. Springer, Ann Arbor.

Hu, Jianjun, Goodman, Erik D., Seo, Kisung, Fan, Zhun, and Rosenberg, Rondal (2005). The hierarchical fair competition (HFC) framework for sustainable evolutionary algorithms. *Evolutionary Computation*, 13(2):241–277.

Hu, Jianjun, Goodman, Erik D., Seo, Kisung, and Pei, Min (2002). Adaptive hierarchical fair competition (AHFC) model for parallel evolutionary algo-

rithms. In Langdon, W. B., Cantú-Paz, E., Mathias, K., Roy, R., Davis, D., Poli, R., Balakrishnan, K., Honavar, V., Rudolph, G., Wegener, J., Bull, L., Potter, M. A., Schultz, A. C., Miller, J. F., Burke, E., and Jonoska, N., editors, *GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference*, pages 772–779, New York. Morgan Kaufmann Publishers.

Jansen, Thomas (2002). On the analysis of dynamic restart strategies for evolutionary algorithms. In Merelo Guervós, Juan Julián, Adamidis, Panagiotis, Beyer, Hans-Georg, Fernández-Villacañas, José-Luis, and Schwefel, Hans-Paul, editors, *Parallel Problem Solving from Nature - PPSN VII*, pages 33–43, Berlin. Springer.

Korns, Michael F. (2009). Symbolic regression of conditional target expressions. In Riolo, Rick L., O'Reilly, Una-May, and McConaghy, Trent, editors, *Genetic Programming Theory and Practice VII*, Genetic and Evolutionary Computation, chapter 13, pages 211–228. Springer, Ann Arbor.

Kotanchek, Mark E., Vladislavleva, Ekaterina Y., and Smits, Guido F. (2009). Symbolic regression via GP as a discovery engine: Insights on outliers and prototypes. In Riolo, Rick L., O'Reilly, Una-May, and McConaghy, Trent, editors, *Genetic Programming Theory and Practice VII*, Genetic and Evolutionary Computation, chapter 4, pages 55–72. Springer, Ann Arbor.

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

Louis, Sushil J. and Rawlins, Gregory J. E. (1993). Syntactic analysis of convergence in genetic algorithms. In Whitley, L. Darrell, editor, *Foundations of Genetic Algorithms 2*, pages 141–151, San Mateo. Morgan Kaufmann.

Mahfoud, Samir W. (1995). *Niching methods for genetic algorithms*. PhD thesis, Champaign, IL, USA.

McKay, Ben, Willis, Mark J., and Barton, Geoffrey W. (1995). Using a tree structured genetic algorithm to perform symbolic regression. In Zalzala, A. M. S., editor, *First International Conference on Genetic Algorithms in Engineering Systems: Innovations and Applications, GALESIA*, volume 414, pages 487–492, Sheffield, UK. IEE.

Murphy, Gearoid and Ryan, Conor (2007). Manipulation of convergence in evolutionary systems. In Riolo, Rick L., Soule, Terence, and Worzel, Bill, editors, *Genetic Programming Theory and Practice V*, Genetic and Evolutionary Computation, chapter 3, pages 33–52. Springer, Ann Arbor.

Ryan, Conor (1996). *Reducing Premature Convergence in Evolutionary Algorithms*. PhD thesis, University College, Cork, Ireland.

Schmidt, Michael and Lipson, Hod (2007). Comparison of tree and graph encodings as function of problem complexity. In Thierens, Dirk, Beyer, Hans-Georg, Bongard, Josh, Branke, Jurgen, Clark, John Andrew, Cliff, Dave, Congdon, Clare Bates, Deb, Kalyanmoy, Doerr, Benjamin, Kovacs, Tim, Kumar, Sanjeev, Miller, Julian F., Moore, Jason, Neumann, Frank, Pelikan, Martin,

Poli, Riccardo, Sastry, Kumara, Stanley, Kenneth Owen, Stutzle, Thomas, Watson, Richard A, and Wegener, Ingo, editors, *GECCO '07: Proceedings of the 9th annual conference on Genetic and evolutionary computation*, volume 2, pages 1674–1679, London. ACM Press.

Schmidt, Michael and Lipson, Hod (2009). Distilling free-form natural laws from experimental data. *Science*, 324(5923):81–85.

Schmidt, Michael D. and Lipson, Hod (2006). Co-evolving fitness predictors for accelerating and reducing evaluations. In Riolo, Rick L., Soule, Terence, and Worzel, Bill, editors, *Genetic Programming Theory and Practice IV*, volume 5 of *Genetic and Evolutionary Computation*, chapter 17, pages –. Springer, Ann Arbor.

Schmidt, Michael D. and Lipson, Hod (2008). Coevolution of fitness predictors. *IEEE Transactions on Evolutionary Computation*, 12(6):736–749.