Co-evolution of Fitness Maximizers and Fitness Predictors

Michael D. Schmidt
Computational Synthesis Laboratory
School of Electrical and Computer Engineering
Cornell University, Ithaca NY 14853, USA
mds47@cornell.edu

Hod Lipson
Computational Synthesis Laboratory
School of Mechanical and Aerospace Engineering
Cornell University, Ithaca NY 14853, USA
hod.lipson@cornell.edu

ABSTRACT

We introduce an estimation of distribution algorithm (EDA) based on co-evolution of fitness maximizers and fitness predictors for improving the performance of evolutionary search when evaluations are prohibitively expensive. Fitness predictors are lightweight objects which, given an evolving individual, heuristically approximate the true fitness. The predictors are trained by their ability to correctly differentiate between good and bad solutions using reduced computation. We apply co-evolving fitness prediction to symbolic regression and measure its impact. Our results show that the additional computational investment in training the co-evolving fitness predictors can greatly enhance both speed and convergence of individual solutions while overall reducing the number of evaluations. In application to symbolic regression, the advantage of using fitness predictors grows as the complexity of models increases.

Keywords

Genetic Programming, Symbolic Regression, Co-evolution, Fitness Prediction, Estimation of Distribution Algorithm

1. INTRODUCTION

The driving importance of genetic programming is to solve complex problems which cannot feasibly be solved directly or rigorously in real world applications. Genetic applications are extremely successful at intelligently searching an enormous search space of solutions. However, they are directly hindered by the cost of measuring each evolved individual's fitness. We address this restraint by introducing an estimation of distribution algorithm (EDA) which approximates fitness measurement to maintain evolutionary progress while maximizing evolutionary speed [15]. To achieve this goal, we introduce the concept of co-evolving *fitness maximizers* with *fitness predictors*.

Estimation of distribution algorithms generate evolutionary individuals intelligently by modeling the search space of the problem being solved [17]. In this paper, we effectively model the search space of the fitness maximizers via fitness prediction.

Fitness maximizers are the individual encodings the program is directly trying to optimize to solve a particular problem [14]. These individuals have a measurable fitness, but in many applications this fitness is prohibitively expensive to measure.

Fitness predictors, on the other hand, are lightweight objects which heuristically approximate the true fitness of a fitness maximizer. However, since nothing can be assumed about how to predict the fitness of an individual, the fitness predictors must also be evolved to generate accurate predictions.

The co-evolution of fitness predictors comes at a cost, but this cost is more than made up for by the rapid improvement of the fitness maximizers. In this paper, we demonstrate the application of fitness predictor co-evolution to symbolic regression.

In symbolic regression, functional expressions are evolved in order to fit the provided training data [14]. Very complex expressions require a large set of training data in order to characterize all features of the phenomenon. Ordinarily, the fitness of each expression is measured by finding the error at each of these data points [1]. By applying a fitness predictor, we drastically reduce the total error evaluations necessary.

Our fitness predictor in symbolic regression is a *sub-sample* of the provided training data. This sub-sample is evolved to accurately predict fitnesses of the current function population. The fitness predictor sub-samples allow for fitness differentiation in the fitness maximizer population, while reducing the total error calculations needed by several orders of magnitude.

2. RELATED WORK

Co-evolution is a genetic programming technique where two populations are evolved within the context of each other [12]. More precisely, the fitness of individuals in one population is in some way related to the individuals in the other. As a result, the evolution of one population influences the other by changing the fitness calculation [18]. Much research has been done on the use and application of co-evolution to enhance problem solving [2, 3, 4, 6, 11, 19, 20, 22, 23, 24]. We apply this work to our co-evolved fitness predictor algorithm.

We base our work on an estimation-exploration algorithm (EEA) setup. Unlike classical co-evolution [12], an EEA consists of three components: A population of estimators, a population of exploratory solutions, and a target hidden system. In this case the evolving fitness maximizers comprise the exploratory population, the fitness predictors are the estimation population, and the true fitness landscape is the target hidden system.

Symbolic regression is used in this paper to demonstrate the application of co-evolved fitness predictors. In symbolic regression, functional expressions are evolved in order to model training data by minimizing error [1, 14]. We again make strong use of previous research done in symbolic regression [8, 10, 13, 21].

Dolin, Brad, Bennett, and Rieffel apply co-evolution to symbolic regression in order to enhance performance [7]. In their research, populations of functions and sample points are co-evolved in direct competition with each other. An over-representative sample population is used so that extra attention is focused on data samples of high error. Our work takes a significantly different approach to co-evolving fitness samples. Instead of focusing on areas of error, we co-evolve predictors which characterize the training data as a whole. We also aim to drastically reduce the necessary evaluations by using an under-representative predictor sample.

Many others have applied acceleration techniques to symbolic regression also. Eggermont and Hemert [9] apply the Stepwise Adaptive Weights method to symbolic regression. Their approach is very similar to Dolin's in that training points are given increasingly larger weights in fitness calculations where functions have high error. We compare our co-evolved predictor algorithm directly with Eggermont and Hemert's results in this paper.

3. FUNCTION MODEL

3.1 Function Encoding

In this paper, functional expressions are represented by a binary tree of mathematical operations. The operations can be unary operations such as abs(), exp(), and log(), or binary operations such as add(), mult(), and div(). The types of operations available are chosen differently for different experiments [1, 14]. Additionally, we specify a maximum depth for each tree to prevent extremely long functions from impacting performance.

Leaf nodes of the binary tree evaluate to terminal values. The terminal values consist of the function's input variables and the function's evolved constant values [10]. The number of constant values available to the function is chosen at runtime for each experiment.

3.2 Function Training Data

The function training data consists of a list of predefined known inputs and outputs. For the experiments done in this paper, the training data is simply a set of x and y pairs [8]. For example:

Training Data:

{ (x_0,y_0) (x_1,y_1) (x_2,y_2) (x_3,y_3) (x_4,y_4) (x_5,y_5) (x_6,y_6) (x_7,y_7) }

These known correct value pairs are used to determine each function's fitness, as described in the next section.

3.3 Function Fitness

The fitness of a symbolic function is determined by its mean absolute error from the set of predefined data points. These data points are loaded at runtime based on the experiment being conducted. We use a mean error rather than the sum of errors in order to reduce floating point overflow. To accomplish this, the mean fitness is calculated progressively using the following pseudo code.

```
For each [data point]
        $Fit = Fit/(1 + 1/n) – Err/(n+1)$
        n++
End
```

Most importantly, using a mean absolute error fitness metric also allows us to define a simple fitness predictor, which is described in Section 4.

3.4 Evolutionary Selection

For the experiments in this paper, we use two different types of selection for population generation: Deterministic Crowding (DC), and Truncation Selection (TS).

Deterministic crowding is our preferred selection technique, and is used in most experiments in this paper. The deterministic crowding method maintains good population diversity and tends to follow multiple divergent pathways to the final solution [5]. As a result, this method provides more accurate experimental results with fewer experiment repetitions.

We also implement Truncation Selection scheme for comparison with other researchers' results. The truncation selection method tends to converge very fast on simple data such as low order polynomials and is a well used and proven selection technique [6].

4. FITNESS PREDICTORS

Fitness predictors are the new element being explored in this paper. Essentially, they replace the fitness evaluations done in the function population training with less expensive predictions given any valid function.

4.1 Predictor Encoding

In application to symbolic regression, we choose the fitness predictor as simply a smaller sample of the predefined training data. An example of a training set of size eight and predictor size four is given below:

Function Training Data:

{ (x_0,y_0) (x_1,y_1) (x_2,y_2) (x_3,y_3) (x_4,y_4) (x_5,y_5) (x_6,y_6) (x_7,y_7) }

Predictor Encoding:

{ (x_1,y_1) (x_3,y_3) (x_6,y_6) (x_7,y_7) }

In this example, the predictor chooses to only evaluate the average error on the points 1, 3, 6, and 7. The predictor gives an estimate of the true fitness of the function by only evaluating on half of the training data. Using this predictor to train the fitness maximizers thereby cuts the number of function evaluations in half when evolving the function population.

There are many more ways to encode the fitness predictors. For example, an artificial neural network could be used whose inputs are the function's binary tree encoding, and the output is the predicted fitness. Different fitness predictor encodings are possible for any type of application, but they must always take a fitness maximizer as an input, and output a predicted fitness. We chose a sub-sample encoding for simplicity.

4.2 Predictor Training Data

4.2.1 Predictor Training Data Structure

The population of predictors has its own set of training data used to calculate the fitness of predictors themselves. The predictor training data consists of functions chosen from the function population and their true fitness measured on all training points. An example predictor training data is shown below:

```
Predictor Training Data:
{   [ F(x) = x^2 + e^sin(x),          True Fitness = -12.56 ]
    [ F(x) = 4.3x + sin(x + 1.2),     True Fitness = -82.56 ]
}
```

Only the most recent training functions are stored for predictor training. This optimizes the predictors for predicting fitness of the current function population, rather than all known functions or random functions.

The number of stored training items is important. The predictor training dataset must contain at least as many items as the number of sample points encoded by the predictor. This specification ensures an overdetermined system, where the predictor evolution progresses to a generally consistent solution, rather than solving special case scenerios.

4.2.2 Generating Predictor Training Data

New predictor training items are generated periodically during co-evolution. Generating a new item is viewed as an expensive operation since the true fitness of the chosen function requires evaluating the function on all training points.

To generate a predictor training data item, a fitness maximizer is chosen from the current population. The true fitness of this individual is evaluated and stored into the predictor population's training data set.

This choice of fitness maximizer is extremely important. In order to generate good predictors, the program must maintain a diverse set of fitness maximizers. This is done by calculating the *variance* of each function in the population from the currently stored training items on the points defined by the current best predictor. The most variant function is chosen since it offers the most new information to the predictor training data.

There are other training data selection methods which also maintain diverse training data. One alternative would be to choose the fitness maximizer which is getting the worst prediction. This method unfortunately requires the full fitness evaluation on every fitness maximizer and can be very expensive. The variance selection method described previously is effective and much less expensive.

4.3 Predictor Fitness

Since fitness predictors are being co-evolved, they also have their own fitness. The fitness predictor's fitness is how well it predicts the true fitness of the stored fitness maximizers. Specifically, the fitness is the mean absolute fitness prediction error of the stored predictor training items described in the previous section.

There are several alternative ways to measure this type of predictor's fitness, and hence its evolution. Other alternatives we considered were to have fitness related to the amount of disagreement created in the predictor training data, thereby spotting areas of uncertainty. Another technique would be to reward predictors based on the amount of error their data items measure.

These techniques still maintain the predictive qualities necessary since they still approximate the existing training data. However, they have very different co-evolutionary effects. We present the straight forward fitness matching technique in this paper for simplicity.

5. CO-EVOLVING FITNESS PREDICTORS

The algorithm presented in this paper evolves two populations at once, the fitness maximizers and the fitness predictors.

Each iteration of our algorithm is called a *cycle*. In each cycle, both populations are evolved, and progress is made toward the final solution. The basic program flow is outlined below:

```
1.  Load function training data
2.  Initialize predictor training data with f(x)=0
3.  Initialize the function and predictor populations
4.  Execute a cycle
    a.  Evolve predictor population
    b.  Evolve function population
    c.  Generate a predictor training data item
5.  Loop back to step 4
```

Not shown in this basic outline are minor controls which turn on and off any of the steps 4.a, 4.b, and 4.c in each cycle. These controls determine how much time is spent evolving predictors verses functions, and also when to generate new predictor training data.

The time spent evolving the predictors verses the functions is controlled by a ratio value referred to as the *predictor training ratio*. This value typically ranges between 0 and 0.5. For example, a value of 0.5 forces the number of function evaluations invested in both populations to be equal.

The second control determines when to generate a predictor training data item. Since this is an expensive step, this step is only executed every *predictor data period*. This period is typically in the range of 10 to 1000 cycles.

6. MEASURING PERFORMANCE

6.1 Function Evaluation Metric

In application to symbolic regression, we use the total number of function evaluations to measure performance. This metric is directly related to the execution time but is independent of encoding types, generations, processor speed, etc [7]. All other execution time is viewed as evolution implementation overhead, which could theoretically be removed.

6.2 Evaluation Budgeting

Note that co-evolving fitness predictors also comes at a cost of evaluations. For each generation of predictors, evaluations are spent to calculate the fitness predictors' own fitnesses.

Since we use function evaluations as our performance metric, we can easily decide how to appropriate all evaluations. The controls described in section 5 turn on and off the training of both populations in order to maintain the specified investment ratio of evaluations.

Since the search space of the fitness predictor is many orders of magnitude less than the search space for the fitness maximizers, we choose to invest only a small percent of evaluation in training the predictors. For all experiments described in the following section, we invest 5% of evaluations into training predictors, and the remaining train the functions themselves. Even with this small investment, the fitness predictors offer significant payoff which are described later.

7. EXPERIMENTATION

7.1 Untrained Sampling Comparison

The goal of this experiment is to show that predictor co-evolution is always at least as good as or better than untrained fitness samples. To do this, we compare the performance of the co-evolution predictor algorithm with three other training data sampling techniques.

7.1.1 Full Sampling

This algorithm uses the full set of training data provided at runtime. Error is measured on every data point in order to calculate the most accurate fitness possible. For the data modeled in this experiment, this is roughly ten times the number of data points used by the predictor.

This comparison gives an approximate idea of how significantly sampling a small subset of training data can improve the speed and performance in symbolic regression.

7.1.2 Static Random Sample

This algorithm uses a uniform random sample of training data for fitness calculation. The number of samples is identical to the size of the predictor's sample. This sample is chosen at runtime, and used throughout one run.

In other words, this algorithm uses a static fitness predictor that is never trained. Likewise, no evaluations are invested in training the subset. It is simply a uniform random sample of the provided training data.

This comparison directly demonstrates the added payoff of co-evolving a predictor over simply choosing a smaller data sample of the same size.

7.1.3 Random Sampling

In this algorithm, a sample of the training data is used again which is also the same size of the fitness predictor's subset. This sample is not static however. Instead, the sample is randomized after each fitness evaluation.

This comparison is given to demonstrate the payoff of intelligently evolving a data sample. Randomly sampling allows the sample to change, however it is still not trained in any way.

7.1.4 Experiment Setup

This experiment thoroughly compares the four symbolic regression programs described: Fitness Predictor Sampling, Full Sample, Static Random Sample, and Random Sampling.

For each run, all parameters except those relating to the sampling technique are held constant. These control parameters are shown in Table 1.

Table 1: Experimental Control Parameters

Parameter	Control Value
Function Population Size	128
Predictor Population Size	5
Predictor Training Ratio	0.05
Predictor Samples	8
Selection	Deterministic Crowding
Mutation Probability	0.1
Crossover Probability	0.5
Operators	+, -, *, /, exp, log, sin, cos
Terminals	Input X, One Constants

These programs are tested on three different functions of varying difficulty. These functions are listed blow and also graphed in Figure 1. Each test is repeated 50 times, and the average fitness for each run is recorded over time.

$$F_1(x) = 1.5x^2 - x^3$$
$$F_2(x) = e^{|x|}*\sin(2pi*x)$$
$$F_3(x) = x^2*\exp(\sin(x)) + x + 2*\sin(pi/4 - x^3)$$

These functions are plotted in Figure 1 on their respective data ranges.

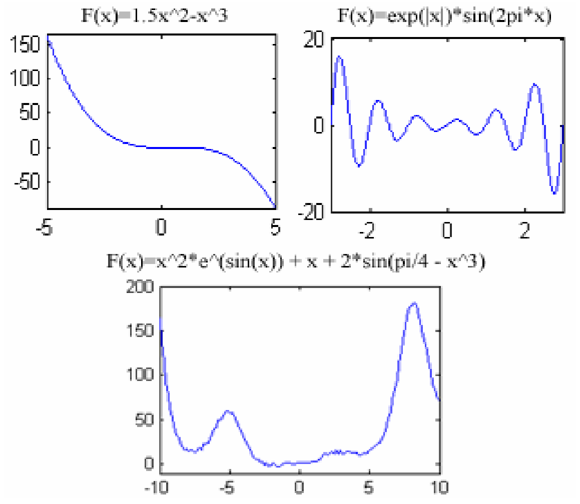

Figure 1: Experiment 7.1 Data Model

The first function $F_1(x)$ is chosen as a simple example where all algorithms should do well. Since this is a function with only one local minima, the random sample is very effective representation of the training data. So the best that the predictor algorithm can do is to co-evolve a fairly uniform distribution of samples.

The second function $F_2(x)$ is slightly more complex in that it has multiple peaks and valleys. This allows for much more intelligent predictors to be co-evolved, which more accurately represent the data than the uniformly random algorithm.

The third function F3(x) is chosen as a complex function with multiple local minima and maxima, but also an additional noise term is added. This noise term is very difficult to find without intelligent sampling.

7.2 Stepwise Adaptive Weights Comparison

In this experiment, the co-evolved fitness predictor program is compared with the Stepwise Adaptive Weights (SAW) program researched by Eggermont and Hemert [4].

The SAW algorithm uses the full training data, but adds an additional weight to each training item. The weight of each training item is updated each generation based on the error of each function in the population. Training points with high error on many functions in the population receive greater and greater weight in the fitness calculation for each generation.

Eggermont and Hemert compare two SAW techniques with the canonical genetic program by applying the SAW technique to symbolic regression. In their experiments, they model two polynomial functions which are shown graphed in Figure 2.

$F_1(x) = x^5 – 2x^3 + x$

$F_2(x) = x^6 – 2x^4 + x^2$

$F_3(x) = x^2*\exp(\sin(x)) + x + 2*\sin(pi/4 - x^3)$

These functions are both fairly simple, but they both have very distinct local maxima and minima on their given data ranges. The SAW program is very good at solving this data as opposed to the canonical GP due to the fact that the weights can pull intermediary solutions out of local maxima to by forcing importance on data features [4].

We show in this experiment that the co-evolved predictor program is also extremely effective at modeling this type of data.

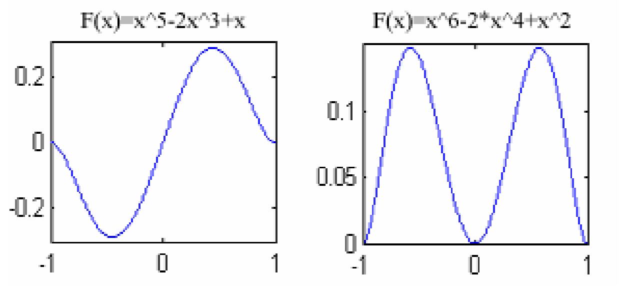

Figure 2: Experiment 7.2 Data Model

To make the most direct comparison possible, we changed a few of the controlled parameters in order to more closely match the parameters used by Eggermont and Hemert. The new control parameters are shown in Table 2.

Table 2: New Experimental Control Parameters

Parameter	Control Value
Function Population Size	100
Selection	Truncation Selection
Operators	+, -, *, /
Terminals	Input X, No Constants

Additionally, the MSE error calculations were changed to absolute error as used by Eggermont and Hemert. The absolute error is the sum of absolute error on each training point.

Eggermont and Hemert's SAW applications evolve a population of 100 functions for 1000 generations, and use 50 sample points. Assuming each function's fitness is evaluated once per generation on 50 points, the total evaluations done in the experiment is 5e6. Even though truncation selection is being used, we still calculate the fitness for each individual every generation, although it is not necessary.

The experiment is repeated 15 times for each function, and the minimum, maximum, median, mean, and standard deviation are compared.

8. RESULTS

This section shows the results obtained from performing the experiments described in section 7.

8.1 Uniform Sampling Comparison

The first function modeled is $F_1(x) = 1.5x^2 – x^3$. The mean MSE for this function over 50 repetitions is plotted against the total number of function evaluations in Figure 3.

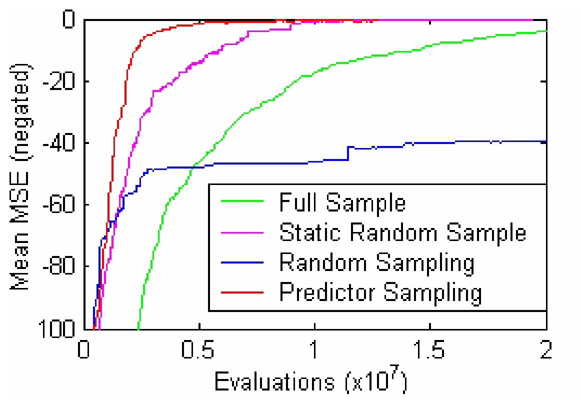

Figure 3: Mean MSE vs. Evaluations for F(x)=1.5x2 - x3

As Figure 3 shows, the co-evolved predictor program is slightly faster than the static random sample program. The full sample program uses the most accurate fitness calculation, and hence has the most stable and gradual progress. The random sampling program starts off the fastest, but tapers off very quickly.

The second function modeled is $F_2(x) = e^{|x|}*\sin(2pi*x)$. Again the mean MSE for this function over 50 repetitions is plotted against the total number of function evaluations in Figure 4.

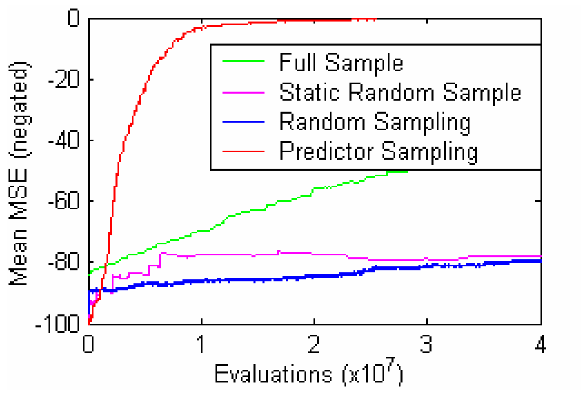

Figure 4: Mean MSE vs. Evaluations for F(x)=e|x|*sin(2*pi*x)

For this more complicated function, the co-evolved predictor program starts off the slowest, but quickly jumps far ahead of the other programs. The full sample program makes slow and steady progress. The static random sample and random sampling programs fail to find any convergence. These two programs tend to evolve linear functions which minimize error, but do not closely follow any of the data.

The third function tested is $F_3(x)$. Over 50 repetitions, the mean MSE is plotted against evaluations in Figure 5.

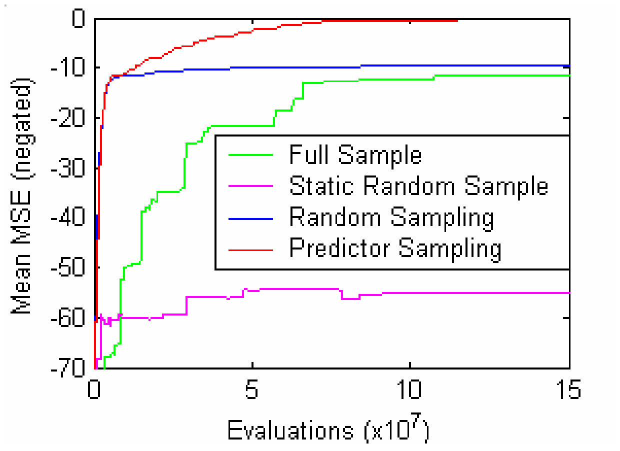

Figure 5: Mean MSE vs. Evaluations for $F(x)= x^2 * \exp(\sin(x)) + x + 2 * \sin(\pi/4 - x^3)$

In this experiment, the predictor and random sampling programs converge onto the modeled function without the noise term very rapidly. The predictor program is the only one able to find the noise term leading to a successful fit. The full sample makes steady progress, however it is also severely bogged down when finding the noise term. The static random sampling program is unable to find any convergence as it is unable to represent all features of the graph while remaining static.

8.2 Stepwise Adaptive Weights Comparison

In this experiment, the co-evolved fitness predictor program is executed on the experiments performed by Eggermont and Hemert. The GP, GP-CSAW, and GP-PSAW results were obtained from Eggermont and Hemert [4]. The final column in Tables 2 and 3 are the co-evolved fitness predictor program results, where the abbreviation GP-FP is used for brevity. Tables 2 and 3 show the results of modeling functions $F_1(x)$ and $F_2(x)$ respectively.

Table 2: $F_1(x)=x^5-2x^3+x$ Results

	GP	GP-CSAW	GP-PSAW	GP-FP
Median	3.763e-7	3.693e-7	3.669e-7	3.027E-15
Mean	0.1161	0.08323	0.06513	0.034139
Std. Dev.	0.2547	0.1803	0.1856	0.13222
Minimum	2.324e-7	1.704e-7	2.114e-7	1.841E-15
Maximum	1.034	0.6969	1.111	0.5120775

Table 3: $F_2(x)=x^6-2x^4+x^2$ Results

	GP	GP-CSAW	GP-PSAW	GP-FP
Median	1.888E-07	1.824E-07	1.899E-07	1.743E-15
Mean	0.06963	0.061	0.05084	0.0408403
Std. Dev.	0.2258	0.1741	0.1418	0.15817
Minimum	1.135E-07	1.048E-07	1.013E-07	1.504E-15
Maximum	1.73	1.161	0.5467	0.612605

These results are further summarized in the bar graph shown in Figure 6.

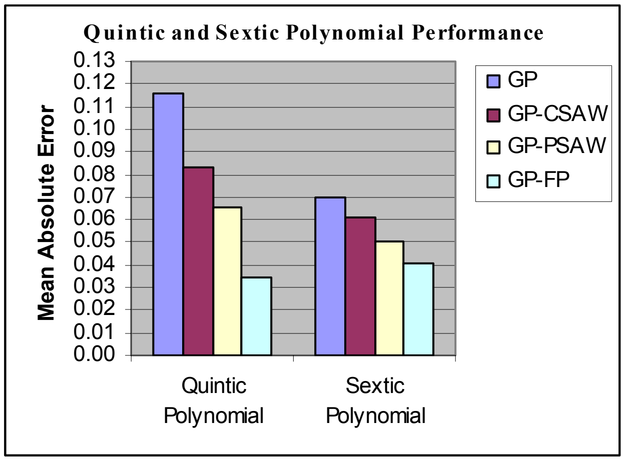

Figure 6: SAW Comparison Results

As shown in Figure 6, the co-evolved fitness predictor program achieves the lowest mean absolute error after using the same number of function evaluations for both experiments.

9. ANALYSIS AND CONCLUSIONS

In this section, we discuss our observations and conclusions drawn from the experiments conducted. We also identify important characteristics of the fitness predictors in each experiment, and identify their impact on the fitness maximizer evolution.

9.1 Sample Selection Conclusions

An interesting observation of the predictors is that they identify the important data regions without including points at local maxima and minima. Training points at the local maxima and minima tend to overestimate the error, and thus lead to bad predictors, and may lead to total annihilation of the best fitness maximizer. To demonstrate the behavior of the fitness predictors, the frequency of the training points used is shown below in Figure 7.

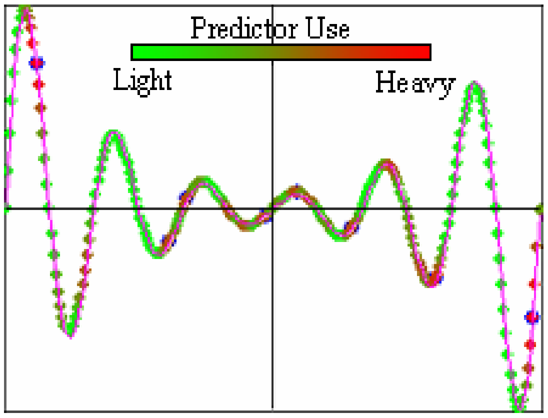

Figure 7: Predictor Data Point Use

Notice that the evolved predictors tend to use only one or two data points at local minima and maxima, and then distribute the remaining points on the sides of local curves.

From this observation, the fitness predictors offer gentle guiding of the functions in the population. Since the predictors are evolved only using recent functions found in the function population, predictors' sample points include a combination of both points that the functions are modeling well and a few points where many functions are missing.

The effect of gentle guiding supports the high fit individuals to survive but strongly encourages alterations which offer improvement. This prevents extinction of functions which are modeling certain regions very well.

This effect is most prominent in the modeling of $F_3(x)$ F shown in Figure 1 of Section 7.1. We observe that often early in evolution, a function $e^x*\sin(c*x)$ is evolved which is missing the absolute value operation to give it the correct form $e^{|x|}*\sin(c*x)$. Without the absolute value, the function closely matches the right hand side of the graph where $x > 0$, but is nearly flat on the $x < 0$ side. Ordinarily this intermediate function would have a very low fitness and be replaced. However, the predictors allow the function to survive since it is modeling the $x > 0$ side so well.

In contrast, the SAW program gives heavy weight to training points which have high error. This induces rapid revolution away from the current functions in the population. Based on the results shown in figure 6, the gentler is much more successful.

9.2 Performance Gain Conclusions

A primary goal of the co-evolution of fitness predictors is to reduce the number of evaluations needed to evolve the fitness maximixer population. As Figure 3 in section 8.1 shows, reducing the training data sample size does indeed accomplish this. Both the static random sample program and co-evolving fitness predictors program strongly outperform the full sample program. The fitness predictor program performs the best for this simple model because it is able to always evolve predictors that are uniformly distributed, where the static random sample maybe have a randomly poor distribution.

The performance gain is also very apparent in the SAW comparison experiment. In addition to the gentle guidance effect of the fitness predictors, the reduced number of evaluations allows the co-evolved fitness predictor program to reach solutions faster and with greater probability.

The investment of evaluations to co-evolve the predictors has a strong return. On all functions experimented in Section 7.1, the co-evolved predictor program always outperforms the untrained equal sized random sample program.

The benefit of co-evolved predictors also increases with the complexity of the data. For the simple function $F_1(x)$ modeled in Figure 3, the predictor program evolves slightly faster than the static random sample program. For the more complex functions $F_2(x)$ and $F_3(x)$ shown in Figures 4 and 5, the predictor program performs increasing faster than the untrained sample programs.

9.3 Avoiding Local Maximum Conclusion

The co-evolution of fitness predictors is very effective at avoiding local maxima functions. Although the predictors have a gentle guiding effect, local maxima fitness solutions are avoided due to the fitness predictor training data using only the most recent data. If the function population becomes dominated by a local maxima solution, the predictor training data becomes dominated with similar functions also. As a result, the gentle guiding becomes gradually more forceful and dynamic.

As described earlier, the predictors will have points that the function matches exactly, and a few points that have error. In the local maxima situation, the low error points jump around randomly since they do not change any predictions on the dominated training data. This effect generates very poor predictors in the short run, and allows mutated children to enter the population.

In addition to the local maxima domination effect, simply having fewer sample points in the predictors helps prevent local maxima functions. The functions have more freedom to model the predictor sample in different ways. As a result, the smaller predictor sample permits more divergent evolution. Since predictors change during co-evolution however, divergence is controlled in the long run to model the full training data.

The advantage of the co-evolved predictor program progressing from local maxima is most apparent in experiment modeling $F_3(x)$ shown in Figure 5. In this experiment only the co-evolved fitness predictor program is able to find the noise term successfully. The random sampling and full sample programs reach the local maxima of fitting the basic shape of the data without the noise term fairly quickly. However, they are unable to progress further. The co-evolved predictor program is able to intelligently focus on the noise variation at this difficult point, and gradually pick up on its polynomial frequency.

10. FUTURE WOK

There are many alternate training techniques and enhancements possible for fitness predictors in symbolic regression. We plan to implement and evaluate the characteristics of different predictor evolution techniques as described in Sections 4.1, 4.2, and 4.3.

We also plan to apply predictor co-evolution to other evolutionary computation disciplines and evaluate their utility. In particular, we would like to develop more robust predictors which can integrate with many genetic applications in addition to symbolic regression.

11. REFERENCES

[1] Augusto, Douglas A., and Helio J.C. Barbosa. "Symbolic Regression via Genetic Programming." VI Brazilian Symposium on Neural Networks (SBRN'00), 01: 22-01, 2000.

[2] Bongard J. C., and H. Lipson "Nonlinear System Identification using Co-Evolution of Models and Tests", IEEE Transactions on Evolutionary Computation, 2004.

[3] Bongard J., and H. Lipson "Managed challenge alleviates disengagement in co-evolutionary system identification." In Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'05), 2005.

[4] Cliff D., and G.F. Miller "Tracking The Red Queen: Measurements of adaptive progress in co-evolutionary simulations." European Conference on Artificial Life, 1995.

[5] Crow, James F., and Motoo Kimura. "Efficiency of Truncation Selection." Proceedings of Natl Acad Sci USA 1979, 76:396-399.

[6] De Jong, E.D., and J.B. Pollack "Ideal evaluation from coevolution." Evolutionary Computation, 2004, 12(2):159–192.

[7] Dolin, Brad, Forrest Bennett, and Eleanor Rieffel. "Co-evolving an Effective Fitness Sample: Experiments in Symbolic Regression and Distributed Robot Control." Proceedings of the 2002 ACM Symposium on Applied Computing, 2002

[8] Duffy, J., and J. Engle-Warnick. "Using Symbolic Regression to Infer Strategies from Experimental Data." S-H. Chen eds., Evolutionary Computation in Economics and Finance, Physica-Verlag. New York, 2002.

[9] Eggermont, J., and J.I. van Hemert. "Stepwise Adaptation of Weights for Symbolic Regression with Genetic Programming." Proceedings of EuroGP, 2001.

[10] Ferreira, Cândida. "Function Finding and the Creation of Numerical Constants in Gene Expression Programming." Advances in Soft Computing – Engineering Design and Manufacturing, Springer-Verlag, 2003: 257-266.

[11] Ficici, S.G., and J.B. Pollack. "Pareto optimality in coevolutionary learning." In Advances in Artificial Life: 6th European Conference (ECAL 2001), 2001: 316–327.

[12] Hillis, W. D. "Co-evolving improves simulated evolution as an optimization procedure." In Langton, C. et al. (Eds.), Artificial Life II. Addison Wesley, 1992.

[13] Keijzer, Maarten. "Improving Symbolic Regression with Interval Arithmetic and Linear Scaling." In Proceedings of the Sixth European Conference on Genetic Programming, Springer, Essex UK., 70-82, 2003.

[14] Koza, J.R. Genetic Programming: On the Programming of Computers by Means of Natural Selection. Cambridge, MA: The MIT Press, 1992.

[15] Larrañaga, Pedro, and José A Lozano. Estimation of Distribution Algorithms: A New Tool for Evolutionary Computation. Norwell: Kluwer Academic Publishers, 2001.

[16] Mahfoud, S.W. "Niching Methods for Genetic Algorithms." Ph.D. Thesis, University of Illinois at Urbana-Champaign, 1995.

[17] Naga, P. Laura, and J.A. Lozano. "Special Issue on Estimation of Distribution Algorithms." Evolutionary Computation. 13.1 (2005): v-vi.

[18] Olsson, B. "Co-evolutionary search in asymmetric spaces." Information Sciences, 2001, 133:103–125

[19] Potter, M.A., and K.A. De Jong "Cooperative coevolution: An architecture for evolving coadapted subcomponents." Evolutionary Computation, 2000, 8(1):1–29

[20] Rosin, C. D., and R.K. Belew. "New methods for competitive coevolution." Evolutionary Computation, 1997, 5(1):1–29.

[21] Soule, T. and Heckendorn, R.B. "Function Sets in Genetic Programming." GECCO-2001: Proceedings of the Genetic and Evolutionary Computation Conference, Morgan Kaufmann, 190, 2001.

[22] Stanley, K.O., and R. Miikkulainen. "Competitive coevolution through evolutionary complexification." Journal of Artificial Intelligence Research, 2004, 21:63–100

[23] Watson, R.A. and J.B. Pollack. "Coevolutionary dynamics in a minimal substrate." In L. Spector and E.D. Goodman et al, editors, Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001), pages 702–709, San Francisco, CA, 2001. Morgan Kaufmann.

[24] Zykov, V., J. Bongard, and H. Lipson "Co-evolutionary variance guides physical experimentation in evolutionary system identification." In The 2005 NASA/DoD Conference on Evolvable Hardware, 2005.