

# Co-evolving Fitness Predictors for Accelerating Evaluations and Reducing Sampling

Michael D. Schmidt and Hod Lipson

School of Computer Science, Cornell University  
mds47@cornell.edu, hod.lipson@cornell.edu

**Abstract.** We introduce an estimation of distribution algorithm that co-evolves fitness predictors in order to reduce the computational cost of evolution. Fitness predictors are light objects which, given an evolving individual, heuristically approximate its true fitness. The predictors are trained by their ability to correctly differentiate between good and bad solutions using reduced computation. We apply co-evolution of fitness predictors to symbolic regression and measure its impact. Our results show that a small computational investment in co-evolving fitness predictors greatly enhances both speed and convergence of individual solutions while reducing the computational effort overall. Finally we apply fitness prediction to interactive evolution of pen stroke drawings. These results show that fitness prediction is extremely effective at modeling user preference while minimizing the sampling on the user to fewer than ten prompts.

## 1 Introduction

Evolutionary computation is driven by the increasing demand to solve complex problems which cannot be solved directly or systematically using informed methods. In many real-world applications, measuring the fitness of an evolved individual can be computationally prohibitive, and is often the overwhelming bottleneck of the entire computational process. We address this restraint by introducing fast fitness estimators, which dynamically approximate true fitness of a candidate individual in order to maintain evolutionary progress while reducing number of true (full) evaluations and necessary training samples.

The proposed method is a coevolutionary form of an Estimation of Distribution Algorithm (EDA) [17]. Traditionally EDAs generate new individuals intelligently by learning properties of successful individuals in previous generations [19]. Here, co-evolving fitness predictors do not generate individuals, nor model properties of successful individuals; the co-evolving fitness predictors are evolved to estimate the fitness landscape (including low and high fitness locations) and in particular distinguish between good and bad individuals, using low-cost heuristics.

Since little can be assumed about what makes an accurate and low cost predictor for a particular problem (or it may change over time), fitness predictors must also be evolved, and are rewarded for their predictive performance. The co-evolution of fitness predictors comes at a cost, but this cost is more than made up for by the speedup introduced to the individual population.

We demonstrate and quantify this advantage by applying fitness predictor co-evolution to symbolic regression. In symbolic regression, functional expressions are evolved in order to fit and identify mathematical underpinnings of experimental data [16]. 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 evaluating the error at each of these data points [1]. By applying a fitness predictor, we drastically reduce the total number of error evaluations necessary.

Finally, we apply our co-evolved fitness predictor algorithm to interactive evolution of pen stroke drawings. A major challenge in interactive evolution is extracting user preferences with minimal demand on the user. We utilize co-evolved predictors in this domain to both permit uninterrupted individual evolution and also to design future prompts to the user. Fitness predictors hence become models of the user's preference which we exploit to intelligently design the most informative probes on the user.

In the evolution of pen stroke drawings, we find this technique to be highly effective at extracting preference models from very limited human interaction. Using only pair-wise preference questions, strategy and preference in pen stroke drawings are extracted in fewer than ten user probes. Our results show that the optimal questions to probe the user need not include drawings similar to the target drawing. Instead, the user models converge on trends in the user responses, thereby extrapolating strong preference for target drawings which the models are never actually trained to prefer.

## 2 Related Work

Co-evolution occurs when the fitness of an individual is defined by another individual. More formally, in a co-evolutionary process one individual can determine the relative ranking between two other individuals (in the same or in a separate population) [13,20]. Much research has been done on the use and application of co-evolution to enhance problem solving [11,28,6,12,21,22,24,26], with the main goal of controlling co-evolutionary dynamics that often result in lack of progress or progress in unanticipated directions [4,25,27]. Here we use a specific form of co-evolution which addresses many of these challenges [2,3].

The co-evolved fitness predictor algorithm is based on an estimation-exploration algorithm (EEA) setup. Unlike classical co-evolution [13], 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 individuals 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, symbolic functional expressions are evolved in order to model training data by minimizing error [1,16]. We again make strong use of previous research done in symbolic regression by applying proven encodings and applications [8,10,15,23].

Dolin et al [7] apply co-evolution to symbolic regression in order to enhance performance. In their research, populations of functions and sample points are co-evolved competitively with each other. An over-representative sample population is used so that extra attention is focused on data samples of high error at higher computational cost. 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.

In traditional interactive evolution, a human user is presented with one or more candidate individuals being evolved for selection. The human user directly performs selection and favored individuals are selected for propagation of offspring into the next generation [29, 30, 31].

A new area of research involves partial human interaction. In these algorithms, the user provides constraints on the problem throughout evolution to narrow the search space [32]. In this paper, fitness predictors become models of user preference. In this sense, the goal is to obtain optimal user models by using a more complex fitness predictor while at the same time accelerating the search for highly preferable individuals.

## 3 Fitness Prediction

### 3.1 Designing a Fitness Predictor

In this section, we lay out the basic elements of a fitness predictor as well as how it is evolved and used. The best predictor in the predictor population is used for all fitness calculations in the individual population. Therefore, the choice of the predictor encoding is very important. There are many ways to encode the fitness predictors. However, they must satisfy the following constraints.

- Predicted fitness of a candidate solution (possibly with bounds and confidence levels)
- Predict significantly faster than exact fitness calculation (tunable accuracy vs. speed)
- Be robust enough to differentiate fitness (or rank) between any pair of individuals

Though there are numerous encodings to choose from for any predictor application, there are several structures which naturally satisfy these conditions. A few are listed below.

- Neural and Bayesian Networks
- Decision Trees
- Functional Expressions
- Training Data Subsets

Choosing a simpler encoding is best for most applications. The simpler the predictor, the smaller the predictor search space becomes, allowing the co-evolution to focus more attention to evolving individuals rather than the fitness predictors. Since predictors are co-evolved, simple encodings can adapt to the convergence of the individual population to maintain progress.

A complex predictor can also be very useful, however. Complex predictors, such as multi-layered neural networks, can provide more accurate fitness predictions in the long term. In the short run however, complex predictors will be inaccurate and early generations of individuals can be poor.

### 3.2 Predictor Training

Fitness predictors must be co-evolved with the individual population to generate increasingly accurate predictions. To facilitate learning in the predictor population, a set of training data is maintained specifically for predictors. This training set is used to measure the fitness of the predictors themselves.

The predictor training data has a well defined structure. The predictor training set consists of copies of individuals from the individual population and their corresponding exactly measured (true) fitness. An example predictor training set is shown in Table 1.

**Table 1.** Example Predictor Training Data.

Individual	Exact Fitness
Copy of Individual 1 from generation 1	14.543
Copy of Individual 6 from generation 12	89.127
Copy of Individual 13 from generation 32	66.651
Copy of Individual 2 from generation 49	13.901

In order to populate the predictor training set with data, the exact fitness of a few select individuals must be calculated and stored periodically during co-evolution.

Care must be taken when choosing which individuals to store in the predictor training set since exact fitness measurement is expensive and the set strongly influences the progress of the predictors. In order to maximize predictor performance, a balance must be found between the following conditions.

- Maximize fitness diversity and structural diversity of individuals in the predictor training set
- Maintain a representative sample of the current individual population
- Address weaknesses in current predictors
- Maintain an over-determined set size

Satisfying the first condition promotes predictors which can robustly predict the fitness of a wide range of individuals. This is accomplished by choosing an individual which has the greatest statistical variance from the currently stored individuals.

The second condition, maintaining a representative individual sample, is important for predictor accuracy optimization. Using individuals which recently existed in the individual population allows the predictors to focus more attention toward predicting fitness of the future individual generations. This is accomplished by discarding old predictor training items and maintaining a constant sized training set.

The third condition is important to ensure predictors maintain their ability to distinguish between good and bad individuals. This is accomplished by using individuals which cause the most disagreement among predictors' predictions.

Finally, the fourth condition ensures that the predictor training set will contain enough items so that the predictor encoding cannot over-specialize. This specification ensures an over-determined system, where the predictor evolution progresses toward a generally consistent solution, rather than solving special case scenarios. For example, the set must be larger than the number of nodes in a neural network predictor, or larger than the number of data items in a training data subset predictor.

### 3.3 Predictor Fitness

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

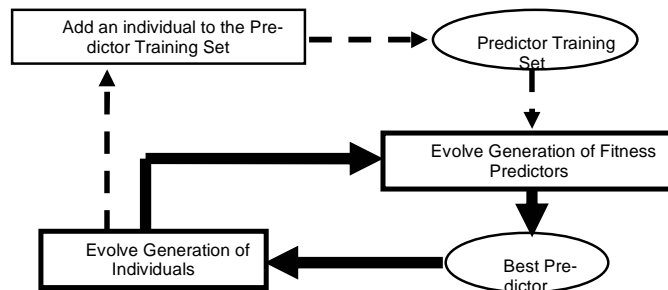
Individuals can be evolved to maximize their predicted fitness, or they can be evolved maximize their worst-case or expected fitness according multiple predictors. The use of several predictors, though costly, provides a fitness distribution that can be used to estimate fitness bounds when statistical confidence levels are needed.

Trainers are also individuals, but their role is not to maximize fitness, but to improve the predictor's accuracy. A low-fitness individual may be more useful for training a predictor to identify bad areas of the fitness landscape. A particularly useful individual is one for which the current predictors disagree in their predictions; such an individual has captured a weakness among the predictors. Evolving for disagreement addresses the first and third criteria cited above.

## 4 Co-evolving Fitness Predictors

### 4.1 Algorithm Overview

The algorithm presented in this paper evolves two populations simultaneously: the individuals and their fitness predictors. This section outlines the basics needed to implement co-evolution utilizing fitness predictors. Each iteration of our algorithm is called a cycle. In each cycle, both populations are evolved, and progress is made toward a final solution. The basic program flow is outlined in Figure 1.



**Fig. 1.** General co-evolution of individuals and fitness predictors algorithm. Two populations are evolved at once. The best fitness predictor is used for all fitness evaluations in the individual population. Individuals are saved for predictor training periodically.

Indicated by solid arrows in Figure 1, the main program flow consists of evolving the individuals and evolving fitness predictors. The best predictor from the fitness predictor population is used to evaluate all fitness values in the individual population. The dashed arrows indicate a periodic step that adds a new individual to the predictor training set. The frequency that this step can be done is determined by the cost of measuring the exact fitness of an individual. Since the main goal is to remove the cost of measuring the exact fitness, this step is done rarely - every few hundred cycles. Alternatively, this step could be done after the individual and predictor populations have reached a plateau for a certain number of generations. In cases where exact fitness measurement is extremely long, a measurement could always be running in the background and added whenever available.

## 5 Symbolic Regression

Symbolic Regression is usually done using genetic programming, where functional expressions are evolved to fit experimental data. The ultimate goal is to identify the mathematical underpinnings of a measured phenomenon.

For experiments in this paper, we represent functional expressions as a binary tree of primitive operations [16]. The operations can be unary operations such as  $\text{abs}()$ ,  $\text{exp}()$ , and  $\text{log}()$ , or binary operations such as  $\text{add}()$ ,  $\text{mult}()$ , and  $\text{div}()$ . The types of operations available are chosen differently for different experiments [1,7,9,8,14,16]. The terminal values available 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 as well.

The function training data consists of a list of inputs and their respective outputs [8]. For the experiments in this paper, we generate training data by sampling a specific target function at uniform intervals. We also take additional samples of the hidden target function to form a validation set for convergence testing of the final solution.

### 5.3 Fitness Measurement

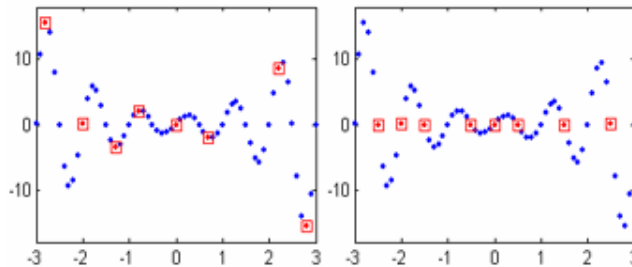
The fitness of a symbolic function is directly related to its error when evaluated on the experimental data points. For experiments in this paper, data samples are taken from a hidden target function and stored as training data. The fitness for individual functions is then the negative mean absolute error.

$$\text{Fitness}(f(x)) = -\frac{1}{N} \sum_{i=1}^N |y_i - f(x_i)|$$

Other fitness alternatives include the negative absolute error sum and negative standard error, or maximum error. We use the mean absolute error fitness metric because it allows us to define a simple training sample fitness predictor, which is described in the next section.

### 5.4 Sampling Predictor Encoding

In application to symbolic regression, we choose a fitness predictor which chooses a small subset of the available training data, and measures the mean error only on the subset. Effectively, this type of predictor evolves a sample of the full training data to adequately represent the hidden target function.



**Fig. 6 (a).** Example of a high performing training subset fitness predictor. Samples (highlighted in red) are well spread, and identify unique features of the data.

**Fig. 6 (b).** Example of a very poor training subset fitness predictor. Samples (highlighted in red) are well spread, but do not identify any distinct features of the data.

How well this type of predictor actually predicts fitness is determined by which training points it selects to represent the full training set. An example of good and bad predictors is shown in Figure 6. Figure 6 gives examples of good and bad training subset fitness predictors. In these figures, the full training set is shown in blue dots, and the samples chosen by the fitness predictor are highlighted in red squares.

The good predictor has well distributed sample set which identifies several minima and maxima regions of the data. The poor predictor, on the other hand, has well distributed samples but does not identify any features of the data. Evolving individuals using the poor predictor is likely to generate inaccurate individuals such as  $F(x)=0$ .

## 6 Sampling Reduction Experiment

### 6.1 Compared Control Algorithms

The goal of this experiment is to compare the predictor co-evolution algorithm with conventional symbolic regression algorithms. The co-evolved fitness predictor algorithm predicts fitness by measuring error on a subset of the training data. In conventional symbolic regression, fitness is measured using the full training set or random subsets of it. In this experiment, we compare the co-evolved predictor algorithm with three conventional symbolic regression techniques which are listed in Table 2.

The full sampling 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. The static random sample algorithm uses a uniform random sample of the 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 effect, the static subset is a static fitness predictor that is never trained. Likewise, no evaluations are invested in evolving the subset. It is simply a uniform random sample of the provided training data.

**Table 2.** Summary of the compared sampling algorithms.

Algorithm	Sample Size	Sample Selection
Co-Evolved Fitness Predictors	8	Evolved Subset
Static Random Sample	8	Random subset chosen at runtime
Dynamic Random Sample	8	Changing random subset
Full Sample	200	Use all training data

In the dynamic random sampling algorithm, a sample of the training data is used, which is also the same size of the fitness predictor's subset. This sample is not static however. Instead, the sample is continuously randomized during evolution. Effectively, the dynamic random sample is a random fitness predictor which is continuously changing, but not in any intelligent way.

### 6.2 Experimental Setup

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

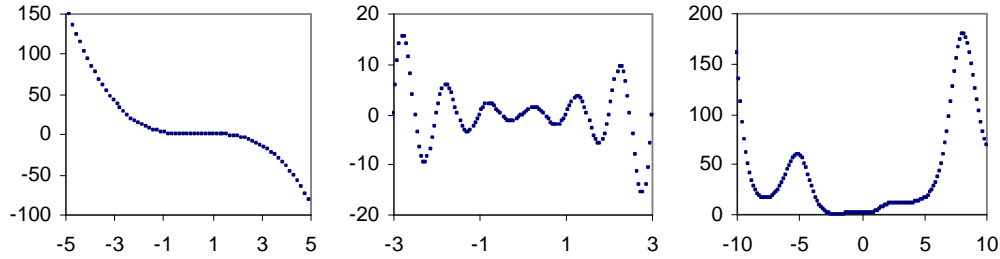
For each run, all parameters except those relating to the fitness calculation specific to each algorithm are held constant. We use a population size of 128, 8 fitness predictors, Deterministic Crowding selection, 0.1 mutation probability, and 0.5 crossover probability. The operator set is  $\{+, -, *, /, \exp, \log, \sin, \cos\}$  and the terminals are the input value and one evolved constant.

The algorithms in Table 2 are tested on three different target functions of varying difficulty. Each test is repeated 50 times, and the average fitness for each run is recorded over time. The training data is generated by sampling each target function uniformly over a given range. The target functions are listed in Table 4. These functions are plotted in Figure 7 on their respective data ranges.

**Table 4.** Target hidden functions.

Target Function	Training Data
$F_1(x) = 1.5 \cdot x^2 - x^3$	$x = [-5 : 0.05 : 5]$
$F_2(x) = e^{ x } \cdot \sin(2\pi \cdot x)$	$x = [-3 : 0.03 : 3]$
$F_3(x) = x^2 \cdot e^{\sin(x)} + x + 2 \cdot \sin\left(\frac{\pi}{4} - x^3\right)$	$x = [-10 : 0.1 : 10]$

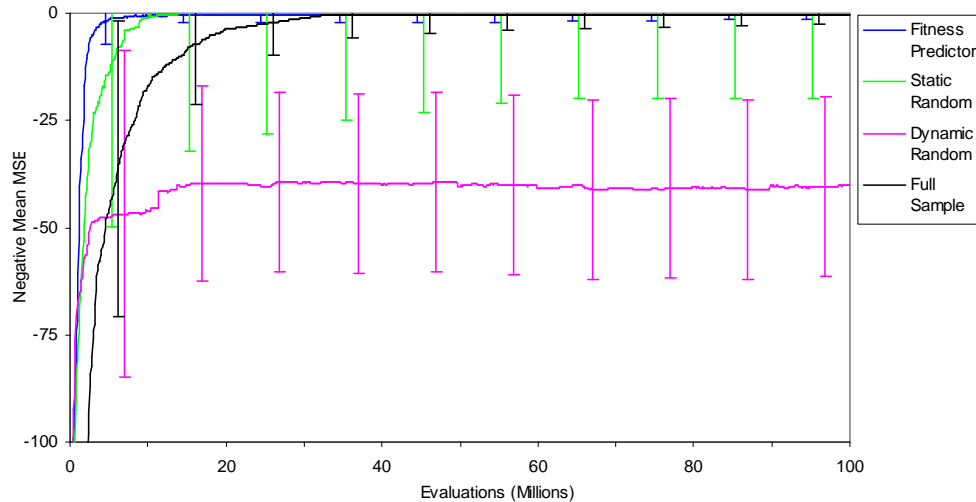
The first function  $F_1(x)$  is chosen as a simple example where all algorithms should do well. Since this function has very few features, the random sample is very effective representation of the training data. So the best that the predictor algorithm can do is to co-evolve a fairly uniform distribution 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 uniform random algorithm. The third function  $F_3(x)$  is chosen as a complex function with multiple local minima and maxima, but also has an additional polynomial frequency noise term is added. This noise term is expected to be very difficult to find.



**Fig. 7.** Training data samples from the hidden target functions experimented on. These hidden functions are listed in Table IV.

### 6.3 Comparison Results

The first function modeled is  $F_1(x)$ . The mean MSE for this function over 50 runs is plotted against the total number of function evaluations in Figure 8.



**Fig. 8.** Error results over evolutions for the four algorithms for  $F_1(x)$ . The conventional algorithms perform well, but the fitness predictor algorithm converges faster and with less deviation.

An example solution found during evolution is shown below. This solution is then simplified to show that it is indeed equivalent to the hidden target function  $F_1(x)$ .

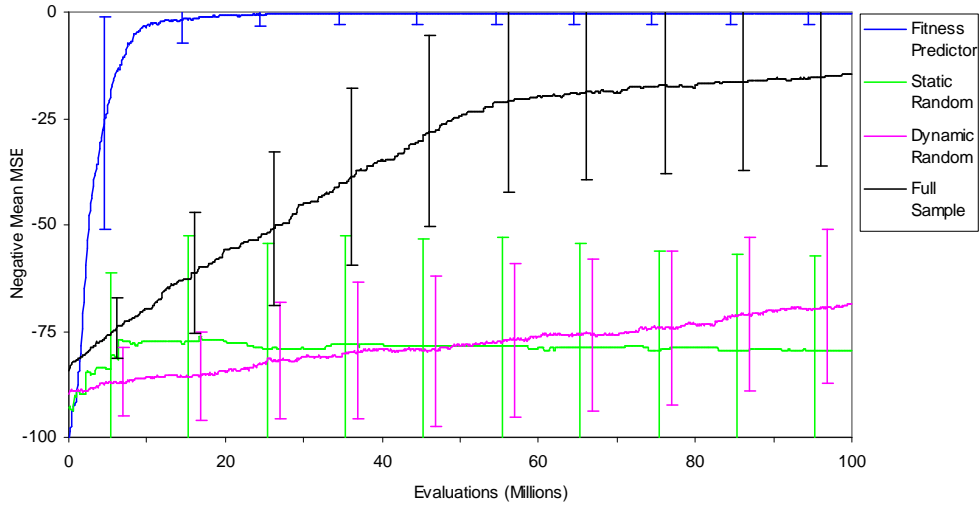
$$\begin{aligned}
 f_{soln}(x) &= (x - (x + \sin(231.954)) \cdot x) \cdot x \\
 &= (x - (x + 0.5) \cdot x) \cdot x \\
 &= (x - x^2 + 0.5x) \cdot x \\
 &= x^2 - x^3 + 0.5x^2 \\
 &= 1.5x^2 - x^3
 \end{aligned}$$

Figure 8 shows most of the algorithms converge onto the target function very quickly. Even for this simple target function, the co-evolved predictor algorithm is slightly faster than the static random algorithm which has the same data sample size. The full sample program uses the most accurate fitness calculation, and hence has the most stable and gradual progress. Interestingly, the dynamic random sampling algorithm performs very poorly. For this simple target function, the dynamic sample causes too much noise in the evolution and test runs either converge or diverge rapidly.

This result shows that the fitness predictor algorithm offers a small but significant speed up over the other algorithms on average for simple data. The fitness predictor algorithm maintains a predictor with a data subset stable enough for steady improvement, while being dynamic enough to improve slightly over a static random sample.

Figure 8 also shows standard deviation error bars for each algorithm over the evolution. For the 50 test runs, the fitness predictor algorithm has the smallest standard deviation as well. Not only does the fitness predictor algorithm find solutions faster, it is also considerably more reliable than the other algorithms

The second function modeled is  $F_2(x)$ . Again the mean MSE for this function over 50 repetitions is plotted against the total number of function evaluations in Figure 9.



**Fig. 9.** Average error results over during evolution for the four algorithms finding  $F_2(x)$ . The fitness predictor algorithm avoids local minima very effectively for this target function.

An example solution found during evolution is shown below. This solution is then simplified to show that it is indeed equivalent to the hidden target function  $F_2(x)$ .

$$\begin{aligned}
 f_{\text{soIn}}(x) &= \sin(x \cdot 6.28318) \cdot \exp(\log(\exp(\exp(\log(|x|)))))) \\
 &= \sin(x \cdot 2\pi) \cdot \exp(\log(\exp(|x|))) \\
 &= \sin(x \cdot 2\pi) \cdot \exp(|x|)
 \end{aligned}$$

Several interesting behaviors emerge for the different algorithms in Figure 9. For this slightly more complex target function, the co-evolved predictor algorithm performs roughly the same, the full and dynamic algorithms lag far behind, and the static algorithm fails to find any convergence.

The full sample algorithm makes slow and steady progress, but gets severely bogged down by local maxima solution at approximately 60 million evaluations. The two most frequent local maxima solutions are shown below.

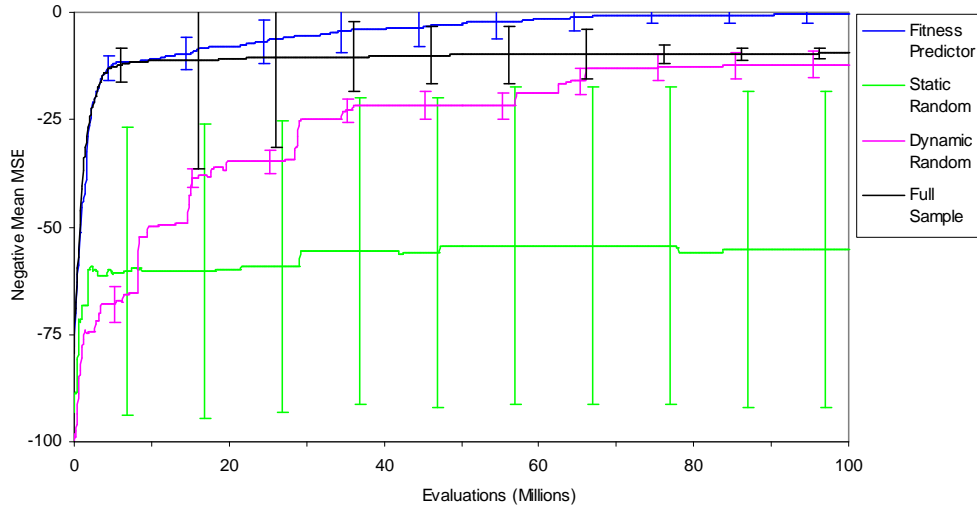
$$f_{\text{maximum}_1}(x) = C \cdot \sin(2\pi \cdot x) \quad f_{\text{maximum}_2}(x) = e^{\pm x} \cdot \sin(2\pi \cdot x)$$

Note that the local maxima either do not include the exponential amplitude, or do not take the absolute value of  $x$  in the exponent. The full sample algorithm tends to rapidly alternate between these two local maxima, and very rarely finds the absolute exponent.

The fitness predictor algorithm encounters these same local maxima but does not stagnate alternating between them. We observe that the predictor sample points tend to focus on one particular side of the y axis, with only a few points on the other once  $f_{\text{maxima2}}(x)$  is encountered. This type of subset allows the  $f_{\text{maxima2}}(x)$  individuals to survive multiple generations instead of alternating back to  $f_{\text{maxima1}}(x)$ . As a result, attention is strongly focused on the exponent, and the absolute value is much more likely to be found. The behavior of the predictor's sample points is described in more detail in the next section.

The standard deviation error bars once again show the fitness predictor algorithm to be significantly more reliable on average. Though the full sample algorithm starts with the smallest deviation, the high error early on shows only that these solutions are reliably poor. This result shows that the fitness predictor algorithm successfully avoids difficult local maxima by intelligently focusing on the exponent sign, where the other algorithms are severely hindered.

The third function tested is  $F_3(x)$ . Over 50 repetitions, the mean MSE is plotted against evaluations in Figure 10.



**Fig. 10.** Average error results over during evolution for the four algorithms finding  $F_3(x)$ . The conventional algorithms fail to converge, or get stuck at local minima.

An example solution found during evolution is shown below. This solution is then simplified to show that it is indeed equivalent to the hidden target function  $F_3(x)$ .

$$\begin{aligned}
 f_{\text{soln}}(x) &= (((\exp(\log(x \cdot x) + \sin(x))) + 8921.34) - 8921.34) + x) - \sin(((x \cdot ((x - 8921.34) + \\
 &\quad 8921.34)) \cdot (8921.34 + (x - 8921.34))) + 8921.34)) / (8921.34 - 8921.34 / 8921.34)) \\
 &= x^2 \exp(\sin(x)) - \sin(x^3 + 8921.34) / (1/2) + x \\
 &= x^2 \exp(\sin(x) + x - 2 \sin(x^3 - \pi/4)) \\
 &= x^2 \exp(\sin(x)) + x + 2 \sin(\pi/4 - x^3)
 \end{aligned}$$

Figure 10 shows an interesting local maxima obstacle for this experiment. Both the fitness predictor and full sample algorithms quickly converge to this local maximum at approximately 10 million evaluations.

$$f_{\text{maximum}}(x) = x^2 \cdot e^{\sin(x)} + x$$

Note that this local maximum represents the general shape of the data and is only missing the high frequency noise term. The full sample algorithm converges to  $f_{\text{maximum}}(x)$ , but makes no progress beyond the general shape. As expected, this noise term is very difficult for all algorithms to find.

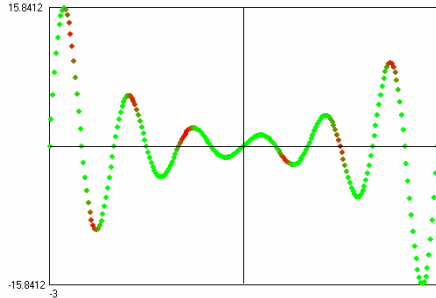
The fitness predictor algorithm is the only one able to progress out of the local maxima to a convergent solution for this experiment. We observe that after finding the general shape, general shape functions saturate the individual population. This causes the fitness predictor population to focus attention specifically on the remaining noise term.

These results indicate that the fitness predictor algorithm is able to solve very complex target functions once again by achieving more evolution and progressing out of difficult local maxima. The co-evolution of the fitness predictor data sample allows stronger focus on the high frequency noise term, where the other algorithms do not.

The fitness predictor algorithm's standard deviation is smallest near the local maxima solution and the convergent solution. Between, the deviation is slightly higher due to different runs converging at different times. In contrast, the full sample algorithm begins with high deviation, and slowly narrows after reaching the local maxima solution. This indicates the full sample can very reliably find the local maxima approximation, but is extremely unlikely to jump to a convergent solution even with further evolution.

## 6.4 Fitness Predictor Behavior

In the previous results section, several comments regarding the fitness predictors data sample behavior were made. Here we discuss in greater detail how predictors are able to focus evolutionary attention in order to progress out of local maxima solutions. To observe the behavior of predictor data samples, we plot the frequency of data points used immediately after convergence on the target function  $F_2(x)$  in Figure 11.



**Fig. 11.** Training point utilization by subset fitness predictors modeling  $F_2(x)$ . The predictor samples interesting regions of the data using a combination of easy to model points and difficult points.

An interesting observation made from Figure 11 is that sample points do not focus entirely on peaks and valleys of the data. High frequency data points are well distributed over the data set, however many lie along side the peaks and valleys.

We call this behavior the gentle guidance effect which fitness predictors tend to exhibit. If all sample points focus on the peaks and valleys of the data, the data subset would represent the max error or worst case fitness estimate since it only includes the high amplitude data points. 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, and also prevents alternation between multiple local maxima solutions.

Fitness prediction forces the data subset to contain a combination of data points which individuals are currently modeling very well as well as points which they are not. Effectively, this boosts evolutionary attention on a small number of points which are not being modeled well while keeping past progress.

## 8 Leading Research Comparison

In this section, we compare the co-evolved fitness predictor algorithm with recent symbolic regression technique publications. In each case we duplicate the experimental setup as closely as possible and compare with the most competitive results.

In order to duplicate experiments, we modify our previous control parameters such as population size, operator set, terminal set, and selection method to match the compared algorithm as closely as possible given details provided.

We compare the co-evolved predictor algorithm by stopping evolution based on the number of function evaluations the compared algorithm made during the experiment. Since each compared algorithm

differs in fitness calculation, we assume that each individual's fitness is measured by evaluating it on the training set each generation. Likewise, we force the co-evolved predictor algorithm to calculate fitness for every individual every generation also, even though different selection algorithms do not require it.

## 8.1 Compared Algorithms

We compare the co-evolved fitness predictor algorithm with four leading symbolic regression techniques: Stepwise Adaptive Weights (SAW), Grammar Guided Genetic Programming (GGGP), Tree-Adjunct Grammar Guided Genetic Programming (TAG3P), and Co-evolution with Tractable Shared Fitness [9, 14, 7]. Where possible, we compare with the best results provided by these algorithms and match performance metrics as closely as possible.

## 8.2 Comparison Results

The comparison results for these algorithms are summarized in Table 6.

**Table 6.** Symbolic regression research comparison summary.

Algorithm	Target Function	Test Type	Results	Co-Evolved Predictor Results
PSAW	$F(x) = x^5 - 2x^3 + x$	Percent Convergences <sup>†</sup>	85.9%	93.9%
	$F(x) = x^6 - 2x^4 + x^2$	Percent Convergences <sup>†</sup>	81.8%	86.9%
GGGP	$P2, P3, P4, P5^*$	Percent Convergences <sup>††</sup>	92%, 64%, 48%, 28%	100%, 86%, 62%, 52%
	$F(x) = \cos(2x)^{**}$	Percent Convergences <sup>††</sup>	20%	76%
TAG3P	$P2, P3, P4, P5^*$	Percent Convergences <sup>††</sup>	100%, 100%, 96%, 84%	100%, 86%, 62%, 52%
	$F(x) = \cos(2x)^{**}$	Percent Convergences <sup>††</sup>	36%	76%
Co-ev Tractable	$F(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}}$	Maximum Evaluations <sup>†††</sup>	3.384e7	2.107e7

\* P3, P4, P5 etc. refer to polynomials ( $x^3+x^2+x$ ,  $x^4+x^3+x^2+x$ ,  $x^5+x^4+x^3+x^2+x$ , ...)

\*\* The operator set does not include the  $\cos()$  function, a trigonometric identity must be found

† The percent of successful convergences from 99 test runs

†† The percent of successful convergences from 50 test runs

††† The maximum number of evaluations before convergence for 100 test runs

As shown in Table 6, the improvement provided by co-evolved fitness prediction in symbolic regression rises with the complexity of the problem. On simple target functions such as the polynomials, other optimizations perform better. On more difficult problems such as finding trigonometric or Gaussian data, the co-evolved fitness prediction algorithm offers significant improvement.

The original goal of the fitness prediction algorithm was not to provide a symbolic regression optimization technique, but rather develop a general approach reducing the computational cost of evolution. An exciting realization is that co-evolution of fitness predictors could be applied in tandem with any of the compared techniques for even further enhancement.

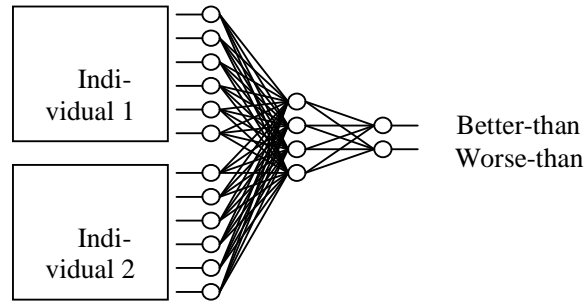
## 10 Co-evolved Fitness Prediction in Interactive Evolution

In this section, we apply the fitness predictor algorithm to interactive evolution of pen stroke drawings. Since fitness has very weak meaning in this domain, we design the predictors to simply compare two individuals and output which is more preferred.

### 10.1 Comparator Predictors and User Models

A simple comparator takes two individuals as input, and outputs three cases: better-than, equal, and worse-than. When comparing individuals however, we are only interested in their better-than or worse-

than outcome. Therefore, the interface of the comparison predictor takes two individuals, and outputs two confidence values for better-than and worse than. This structure is shown below in Figure 14.



**Fig. 14.** The structure of a neural network comparator predictor user model.

The comparator predictor effectively models the complex subjective preferences of the user. Prompts to the user in our experiments consist of two images, and asking which one is more preferred. These prompts are then stored as predictor training data.

## 10.2 Evolving Pen Stroke Drawings

In this experiment we evolve drawings produced by a series of closed pen strokes. Each individual encodes each coordinate drawn to on a 32 by 32 pixel image. In our experiments we predefine the number of strokes for each drawing, but this could easily be evolved as well in future work. The search space for these types of drawings increases exponentially with the number of pen strokes,  $(W*H)^N$  for images of width  $W$ , height  $H$ , and  $N$  pen strokes. The large search space makes the discovery of preferable individuals an excellent application for an evolutionary algorithm.

## 10.3 Evolution Setup

We use standard genetic programming to evolve individuals in this experiment with the exception that all selection and fitness comparisons are performed using the trained comparator model ensemble. Individuals are not given explicit fitness values but are instead ranked using the averaged comparator population.

We use Deterministic Crowding for selection because it is a natural fit for a pair-wise comparator. The Deterministic Crowding method maintains population diversity through child-parent elitism and tends to follow multiple divergent pathways to the final solution [15]. This results in more variety in user prompt selection and better generalization of the comparator fitness landscape.

## 10.4 Discovering Square Shape Preference

### 10.4.1 Square Drawings

The first experiment conducted tests the ability of algorithm to identify and infer a user's preference for "square-like" drawings from initially random pen drawings. Each individual is encoded as a series of four pen strokes. In this experiment we make two basic algorithm comparisons with random search and local search techniques. These comparisons gauge how effectively our algorithm reduces the interactive cost with the user to discover the target drawing.

### 10.4.2 Random Search Comparison

To quantify the difficulty of this problem we calculate the probability of finding a "square-like" drawing through random search. Note that the square is uniquely determined by two of its vertices. Given that a "square-like" drawing can have any orientation or size, and we allow the 3rd and 4th vertices to have some noise of four pixels, the probability and expected random individuals observed,  $T$ , to find such a drawing is calculated below.

$$P_{square} = \frac{(\#loc_1) \cdot (\#loc_2) \cdot (\#loc_3) \cdot (\#loc_4)}{(\#pixels) \cdot (\#pixels) \cdot (\#pixels) \cdot (\#pixels)}$$

$$P_{square} \approx \frac{(32^2) \cdot (32^2) \cdot (4^2) \cdot (4^2)}{(32^2) \cdot (32^2) \cdot (32^2) \cdot (32^2)} \approx 0.024\%$$

$$\therefore E(T_{square}) \approx 4096$$

### 10.4.3 Local Search Comparison

An alternative to interactive evolution is an interactive local search algorithm. In this technique, the user is given a random individual and asked to fine tune parameters individually. In pen stroke drawing this corresponds to adjusting the x and y coordinates of each pen stroke vertex to desired locations. This technique can be viewed as very simplistic partial interactive evolution where the use effectively constrains each parameter individually.

The local search algorithm we compare with can be thought of as a perfect algorithm for transforming a random drawing into the target drawing. The user is assumed to be an oracle that makes perfect choices to optimally tweak parameters with the minimum number of prompts.

Here we calculate a lower bound on the expected number of local adjustment steps necessary to shape a random pen stroke drawing to a square shape. Recall that a square is determined by two vertices. So we begin by calculating the expected diagonal of the square from two random points on the 32 by 32 pixel grid.

$$\langle \Delta x \rangle = \langle \Delta y \rangle = \frac{\sum_{x_1=0}^{31} \sum_{x_2=0}^{31} |x_1 - x_2|}{32^2} = \frac{10912}{32^2} \approx 10$$

$$\langle d \rangle = \sqrt{\langle \Delta x \rangle^2 + \langle \Delta y \rangle^2} \approx 15$$

Next, the expected mean of these two points, and all random vertices is the center point of the grid (x=15, y=15). This means that the two remaining points are expected to be at the center of the final square. Hence, they must each move half a diagonal,  $\langle d \rangle / 2$ . In the easiest case, these points move only along an individual axis. In the worst case, they move diagonally, stepwise on the grid. The expected number of steps per vertex is calculated below.

$$\langle steps \rangle = \frac{1}{16} \cdot \sum_{x=0}^{15} x + \left\lceil \sqrt{15^2 - x^2} \right\rceil \approx 18$$

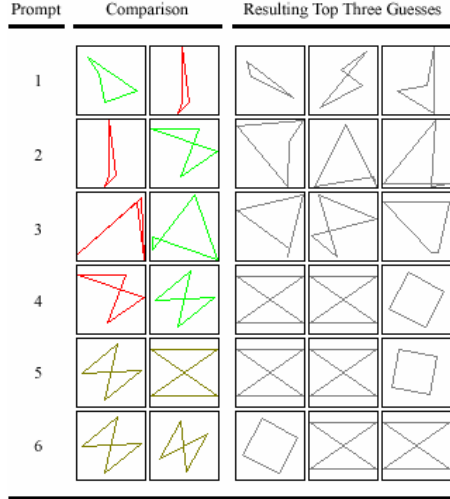
Therefore, the lower bound on the total number of expected local updates to the x and y coordinates of the two remaining vertices is approximately 36. Note that this is a lower bound approximation on the expected minimum number of user prompts required for a perfect local search algorithm to form a pen stroke drawing of a square at any orientation. An expert user is required to identify a desired diagonal and then isolate the remaining vertices accordingly.

### 10.4.4 Results

Figure 15 shows a standard run for a square drawing using the comparison predictor algorithm. The user has a specific strategy to prefer shapes with parallel sides and right angles consistent with a square. The user's preferred drawing for each comparison is shown in green, non-preferred drawings are shown in red, and drawing deemed to be equivalent are shown in dark yellow.

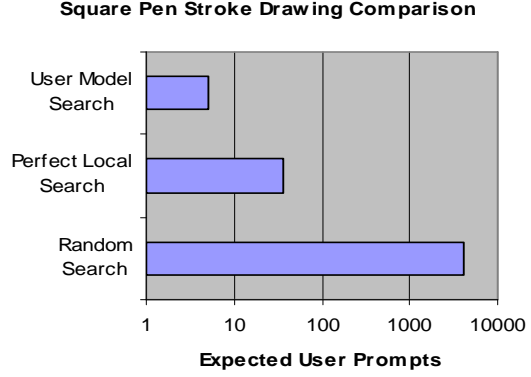
First, notice that none of the comparisons shown to the user involve a square. Based upon the several non-square comparisons, the comparator model is able to infer that the user is likely to prefer a square shape. If in fact the user was not knowingly seeking squares but giving feedback on some unknown preference, the algorithm would predict and identify box-like solutions they are likely to favor.

The algorithm successfully found a square in its top three guesses after four user prompts, and ranked a square as its top guess after six prompts. This is a vast improvement over the random search which is expected to require 4096 user interactions before finding an approximate square.



**Fig. 15.** The prompts given to the user and the resulting top three guesses over six iterations. No user prompts contain square shapes, but the algorithm identifies a square after four prompts.

Figure 16 shows the comparison with the comparison predictor algorithm with the perfect local search and random search algorithms. The logarithmic scale shows that the user comparator model makes significant improvement over the perfectly performing local search and vastly reduces the search cost over random search.



**Fig. 16.** The number of user prompts expected between the compared algorithms to find a square shape.

## 10.5 Discovering Star Shape Preference

### 10.5.1 Star Drawings

In this experiment, the algorithm evolves drawings with six pen strokes. Here we evaluate the algorithm ability to identify a preference for star-shaped drawings. In this experiment we make two basic algorithm comparisons with random search and local search techniques. These comparisons gauge how effectively our algorithm reduces the interactive cost with the user to discover the target drawing.

### 10.5.2 Random Search Comparison

To approximate the number of star shapes possible, we split the drawing space into six regions: a center where no vertices can be, and five surrounding regions for each vertex, all of equal area. Note that a five point star is uniquely determined, by region in this case, but three of its vertices. The probability and expected random individuals observed,  $T$ , to find an approximate star shaped drawing is therefore:

$$P_{star} = \frac{(\#loc_1) \cdot (\#loc_2) \cdot (\#loc_3) \cdot (\#loc_4) \cdot (\#loc_5)}{(\#pixels) \cdot (\#pixels) \cdot (\#pixels) \cdot (\#pixels)}$$

$$P_{star} \approx \frac{5}{3888} \approx 0.128\%$$

$$\therefore E(T_{star}) \approx 777.6$$

### 10.5.3 Local Search Comparison

As found earlier, the expected distance between two random vertices on the 32 by 32 pixel grid is approximately 15. To simplify this calculation we make an approximation that the expected locations of the other three vertices lay at the center of the star. The radius of this star with line length 15 is easily

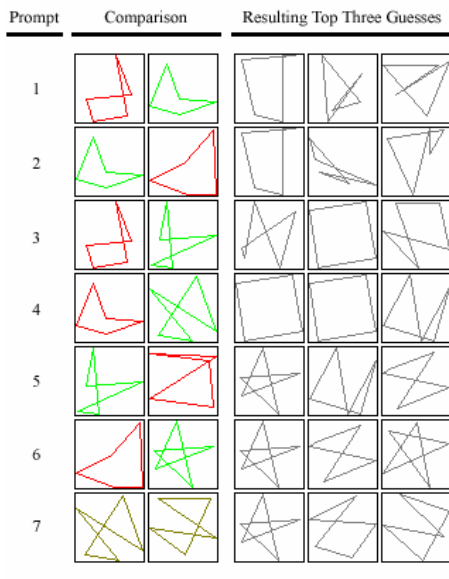
calculated to be approximately 8. The expected number of steps, moving both in x and y is calculated below.

$$\langle steps \rangle = \frac{1}{9} \cdot \sum_{x=0}^8 x + \left\lceil \sqrt{8^2 - x^2} \right\rceil \approx 9$$

Therefore, the lower bound on the total number of expected local updates to the x and y coordinates of the three remaining vertices is approximately 27. Note that this is a lower bound approximation on the expected minimum number of user prompts required for a perfect local search algorithm to form a pen stroke drawing of a square at any orientation. An expert user is required to identify an optimal starting edge and then isolate the remaining vertices accordingly.

#### 10.5.4 Results

Figure 17 shows a standard run to evolve an approximate star shaped drawing. The user has a general strategy when answering comparison prompts to prefer shapes with multiple sharp pointed corners such as a star shape has. The user's preferred drawing for each comparison is shown in green, non-preferred drawings are shown in red, and drawings deemed to be equivalent are shown in dark yellow.

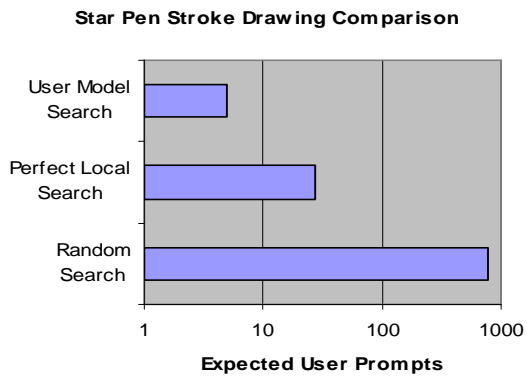


**Fig. 17.** The prompts given to the user and the resulting top three guesses over seven iterations. A star shaped preference is identified after five user prompts.

Notice in Figure 17 that a very well formed star is derived as the top predicted shape after five user prompts. No prior prompts required a star shape. Instead the comparator model inferred the star shape as an optimum solution given the user responses favoring shapes with multiple sharp edges.

The next three prompts answered by the user are shown to demonstrate the comparator model is stable and continues to favor the consistent star-like shapes with further input. Therefore, the algorithm has likely converged on the star shape preference.

Figure 18 shows the comparison with the comparison predictor algorithm with the perfect local search and random search algorithms. The logarithmic scale shows that the user comparator model makes significant improvement over the perfectly performing local search and vastly reduces the search cost over random search.



**Fig.18.** The number of user prompts expected between the compared algorithms to find the target star shape.

## 11 INFERRING A FITNESS LANDSCAPE

### 11.1 Discovering Clock Drawing Preference

In this experiment we want to visualize the fitness landscape being learned by the comparator model. To do this, we modify our pen stroke drawing individual to have only a single pen stroke originating from the center of the drawing area. The search space of the individuals is now only a single coordinate,  $x$  and  $y$ . We can then display relative fitnesses for all possible individuals in a 3D surface.

For this experiment, the single pen stroke encoding is considered to be a clock hour hand. The user is then asked to prefer clocks drawn where the time is closer to some time of day that they prefer. The user for the experiment chooses to prefer clock drawings where the hand points to either 3:00 or 7:00. Therefore, this experiment has two equally favorable global maxima solutions.

### 11.2 The Fitness Landscape of a Comparator

To determine if the comparator accuracy learns the dual solutions in this experiment we need some way to calculate a fitness landscape from a comparator model. Therefore, we need to define a fitness calculation for a single individual using the binary comparator.

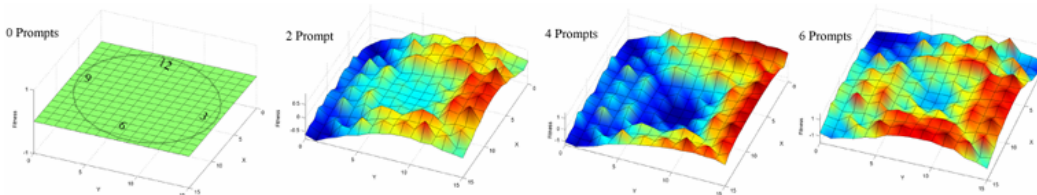
For this experiment, the search space for all clock drawings is the total number of pixels,  $32^2$ . For this small search space, it is feasible to list all possible drawings and compare them with a particular drawing. We calculate an effective fitness by taking the average confidence of better-than and worse-than outcomes when compared with all possible  $32^2$  clock drawings.

$$Fitness = \frac{1}{32^2} \sum_x \sum_y Better(Ind(x, y)) - Worse(Ind(x, y))$$

In this expression,  $Better()$  and  $Worse()$  are the two confidence outputs of the comparison predictor.

### 11.3 Landscape Results

Figure 19 shows four fitness landscapes of the comparator model over six user prompts. At zero prompts, the fitness landscape is entirely flat.



**Fig.19.** The clock time fitness landscapes learned over six user prompts. The final landscape yields very gradual peaks to the two optimal solutions.

Notice that the fitnesses on the  $xz$ -plane resemble an arch which peaks at approximately the clock hand 3:00 position. Correspondingly, the  $yz$ -plane fitnesses exhibit the same phenomenon although slightly less accurately near the 7:00 clock hand position. This final resulting landscape shows the comparator has inferred a very friendly fitness landscape for the evolutionary search. Very gradual gradients exist near the target clock hand locations that should be easily descended.

It's interesting to note that the comparator landscape also exhibits a few other medium fitness clock hand areas such as near the 9:00 position. This is a weakly favored region that the algorithm shows some probability for preference in.

## 12 Conclusion

In summary, we have introduced the use of co-evolution of fitness predictors as a dynamic estimation of distribution algorithm. We have developed a general co-evolutionary model for applying fitness prediction to any genetic program in order to reduce the computation cost of evolution.

When applied to symbolic regression, this approach is found to be extremely effective over standard methods for both speeding up target function convergence and avoiding local maxima solutions. Co-evolved fitness prediction not only accelerates evolutionary progress, but does so more reliably with smaller deviation.

Our comparisons with other symbolic regression research show that fitness prediction offers similar improvement for simple target functions, and very strong improvement for complex target functions. Moreover, the co-evolution of fitness predictors is a general enhancement technique which could theoretically be combined with these techniques for even further improvement.

When applied to the interactive co-evolution of user models, the algorithm is found to be extremely efficient at extracting preference models and resulting solutions from very limited human interaction. Using only pair-wise preference questions, strategy and preference in pen stroke drawings are extracted by the comparison predictors in fewer than ten user probes.

In comparison to the perfect local search algorithm, where the user essentially draws their exact preference explicitly, the interactive algorithm requires roughly ten times fewer total user input. Furthermore, the prompts to the user are presented in much simpler binary preference questions.

Finally, our results show that the optimal questions to probe the user for predictor training need not include drawings similar to the target drawing. Instead, the user models converge on trends in the user responses, thereby extrapolating strong preference for target drawings which the models are never actually trained to prefer.

## 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., (2004) Solution Concepts in Coevolutionary Algorithms, Ph.D. Thesis, Computer Science Dept., Brandeis University
12. 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.
13. Hillis, W. D. "Co-evolving improves simulated evolution as an optimization procedure." In Langton, C. et al. (Eds.), Artificial Life II. Addison Wesley, 1992.
14. Hoai, N.X., R.I. McKay, D. Essam, and R. Chau. "Solving the symbolic regression problem with tree-adjunct grammar guided genetic programming: comparative results." Evolutionary Computation, 2002. CEC 2002, Vol 2. May 2002, 1326-1331.
15. 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.
16. Koza, J.R. Genetic Programming: On the Programming of Computers by Means of Natural Selection. Cambridge, MA: The MIT Press, 1992.

17. Larrañaga, Pedro, and José A Lozano. Estimation of Distribution Algorithms: A New Tool for Evolutionary Computation. Norwell: Kluwer Academic Publishers, 2001.
18. Mahfoud, S.W. "Niching Methods for Genetic Algorithms." Ph.D. Thesis, University of Illinois at Urbana-Champaign, 1995.
19. Naga, P. Laura, and J.A. Lozano. "Special Issue on Estimation of Distribution Algorithms." Evolutionary Computation. 13.1 (2005): v-vi.
20. Olsson, B. "Co-evolutionary search in asymmetric spaces." Information Sciences, 2001, 133:103-125
21. Potter, M.A., and K.A. De Jong "Cooperative coevolution: An architecture for evolving coadapted subcomponents." Evolutionary Computation, 2000, 8(1):1-29
22. Rosin, C. D., and R.K. Belew. "New methods for competitive coevolution." Evolutionary Computation, 1997, 5(1):1-29.
23. 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.
24. Stanley, K.O., and R. Miikkulainen. "Competitive coevolution through evolutionary complexification." Journal of Artificial Intelligence Research, 2004, 21:63-100
25. 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.
26. 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.
27. Luke S., Wiegand R.P. (2002) When Coevolutionary Algorithms Exhibit Evolutionary Dynamics, GECCO-02 Workshop on Understanding Coevolution, New York, NY.
28. Wiegand, R.P., (2004) "Analysis of Cooperative Coevolutionary Algorithms." Ph.D. thesis, George Mason University.
29. Dawkins, R. "The Blind Watchmaker," W. W. Norton, New York, 1987.
30. Poli, R. "Genetic programming for image analysis." In Koza, J. R., Goldberg, D. E., Fogel, D. B., and Riolo, R. L., editors, Genetic Programming 1996: Proceedings of the First Annual Conference, pages 363-368, Stanford University, CA, USA. MIT Press, 1996.
31. Poli, R., S. Cagnoni. "Genetic programming with user-driven selection: Experiments on the evolution of algorithms for image enhancement" presented at the 2nd Annu. Conf. Genetic Programming, J. R. Koza et al., Eds., San Francisco, CA, 1997.
32. Takagi, H. "Interactive evolutionary computation: fusion of the capabilities of EC optimization and human evaluation." Proc. IEEE 89: 1275-1296, 2001.